NASA-GB-001-96

Software Program

Software Management Guidebook

November 1996

NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION

WASHINGTON, DC

Software Program

Software Management Guidebook

November 1996

NATIONAL AERONAUTICS AND
SPACE ADMINISTRATION

WASHINGTON, DC

Foreword

This document is a product of the National Aeronautics and Space Administration (NASA) Software Program, an Agency-wide program to promote continual improvement of software engineering within NASA. The goals and strategies for this program are documented in the *NASA Software Strategic Plan*, July 13, 1995.

Additional information is available from the NASA Software IV&V facility on the World Wide Web site http://www.ivv.nasa.gov/.

Table of Contents

Page

Figures

Page

Tables

Page

1. Introduction

1.1 Background

The objective of every National Aeronautics and Space Administration (NASA) software engineering project is to provide, to the customer, a software product that is engineered to satisfy the customer's requirements, within determined cost, schedule, and quality guidelines.

The term *software engineering* encompasses new development, modification, reuse, re-engineering, maintenance, and all other activities resulting in software products. Throughout NASA, organizations engineer software products that cover a wide spectrum of characteristics (Reference 1):

- Application domains include flight and embedded software, mission ground support software, general support software, science analysis software, research software, and administrative and Information Resources Management (IRM) software.
- Target operating environments encompass PC-based, mainframe-based, workstation-based, and client/server-based solutions.
- Product sizes range from only a few thousand lines of code to more than a million.
- Cost and cycle-time requirements vary.
- Desired end-product qualities—reusability, commercialization, and consequences of software failure (from minor inconvenience to loss of a mission or loss of life)—also vary.

One significant lesson learned from many years of software engineering throughout NASA is that no single solution can solve every problem. No one life-cycle model, analysis and design method, testing method, product evaluation method, or degree of formality for documents and reviews is appropriate for all NASA software projects. To accommodate these variations, each project must tailor its software process to acknowledge customer requirements and constraints; goals and objectives for cost, cycle time, and product qualities; and management's tolerance for risk. Such tailoring is the responsibility of the project's software manager.

1.2 Purpose

The purpose of this NASA *Software Management Guidebook* is twofold. First, this document defines the core products and activities required of NASA software projects. It defines life-cycle models and activity-related methods but acknowledges that no single life-cycle model is appropriate for *all* NASA software projects. It also acknowledges that the appropriate method for accomplishing a required activity depends on characteristics of the software project.

Second, this guidebook provides specific guidance to software project managers and team leaders in selecting appropriate life cycles and methods to develop a tailored plan for a software engineering project.

1.3 Scope

This handbook addresses the engineering of software products, where those products either (1) comprise a software system for which this handbook governs the overall engineering effort or (2) are part of a hardware-software system for which this handbook governs only the software portion.

System engineering management issues are outside of the scope of this guidebook. Section 2.2.4 places the software life cycle in the context of the system life cycle, and Appendix Appendix D. discusses the system-level considerations required of the software manager and team members.

This book also does not cover *acquisition* of software products; it covers development and maintenance.

1.4 Overview

1.4.1 Organization

Chapter 2 of this guidebook summarizes the common elements of the overall NASA software engineering process. Chapters 3 through 6 describe the NASA software engineering process in somewhat more detail, including summary descriptions of required activities and products, and recommended methods for performing those activities. Chapter 7 discusses running and then closing out software projects.

Appendix Appendix A. is a glossary of software engineering terms that every software project manager should understand. Appendix Appendix B. provides guidance for building software components with reusability in mind. Appendix Appendix C. provides more detailed guidance for incorporating non-developed items (NDIs) into software products. Appendix Appendix D. lists additional considerations when the software under development is only part of a larger system.

1.4.2 Terminology

Several terms have specific meanings in this document. These terms are used consistently to emphasize and ensure understanding of the flexibility built into the software process common requirements. Please refer to the glossary in Appendix Appendix A. for definitions and discussion of the following key sets of terms:

- Development, maintenance, enhancement
- Life-cycle models, phases, activities, methods
- Software configuration items (CIs), systems
- Documentation, record

1.4.3 Notation

Table 1–1 explains the use of icons in this guidebook.

Table 1–1. Use of Icons

!!	A double exclamation mark highlights related tips that have been shown effective on NASA programs.
✓	A check mark highlights required software engineering activities.
✂	A pair of scissors highlights software process tailoring information; that is, methods and techniques proven to be effective on NASA programs and recommended for use in performing a particular activity.

2. Software Engineering Process Requirements and Infrastructure

This chapter summarizes the common aspects of the overall NASA software engineering process. The first two sections discuss key activities and products that are expected of NASA software projects. The third section discusses the roles and responsibilities of personnel at various organizational levels. The final section introduces the concept of a process asset library (PAL).

2.1 General Requirements

Every software project must meet a number of general requirements in carrying out the detailed required activities, for example

- The software team uses systematic, documented methods for all software engineering activities. These methods are described or referenced in the project's software plan.
- The software team applies standards for representing requirements; design; code; test plans, procedures, and results; and other software products. These standards are described or referenced in the software plan.
- During the course of the project, the software team identifies and evaluates NDIs, including commercial-off-the-shelf (COTS), government-off-the-shelf (GOTS), and reusable software products, as well as software products not created by project personnel (for example, by other NASA or contractor personnel), for use in fulfilling the project requirements. The scope of the search and the criteria to be used for evaluation are as described in the software plan. Software products that meet the criteria are used where practical. Incorporated software products must meet applicable data rights and licensing requirements. Appendix Appendix C. discusses an approach for developing systems that comprise predominantly NDI components.
- During the course of the project, the software team identifies opportunities for developing software products for reuse and evaluates the benefits and costs of these opportunities. Opportunities that provide cost benefits and are compatible with the customer's objectives are identified to the customer. The software requirements might also state that the software team develop software products specifically for reuse.
- The software team identifies critical software CIs (or portions thereof) and develops and implements a strategy that addresses the following critical issues:
  - System resource utilization: critical system resource capacities or constraints are imposed on the final product
  - Safety: failure of the software could lead to a hazardous state
  - Security: failure could lead to a breach of system security
  - Privacy: failure could lead to a breach of system privacy
  - Other critical characteristics

The software team develops an appropriate strategy for such software, including tests and analyses, to ensure that the requirements, design, implementation, and operating procedures for the identified software minimize or eliminate the potential for hazardous or compromising conditions. The software team records the strategy in the software plan, implements the strategy, and produces evidence, as part of required software records, that the defined strategy has been carried out successfully.

- The software team records rationale that will be useful to the software operations and maintenance (O&M) organization for key decisions made in specifying, designing, implementing, and testing the software. The rationale includes trade-offs considered, analysis methods, and other criteria used to make the decisions. The rationale is recorded in documents, code comments, or other media that are transferable to the software O&M organization.

2.2 Specific Required Activities and Products

The software manager establishes a project software engineering process that is based on the NASA software process and consistent with the software requirements. The NASA software process comprises three categories of activities:

1. Management
2. Technical
3. Software process improvement

Figure 2–1 illustrates the relationships among the activities; Table 2–1 summarizes the activities and the primary products generated as a result of performing each activity.

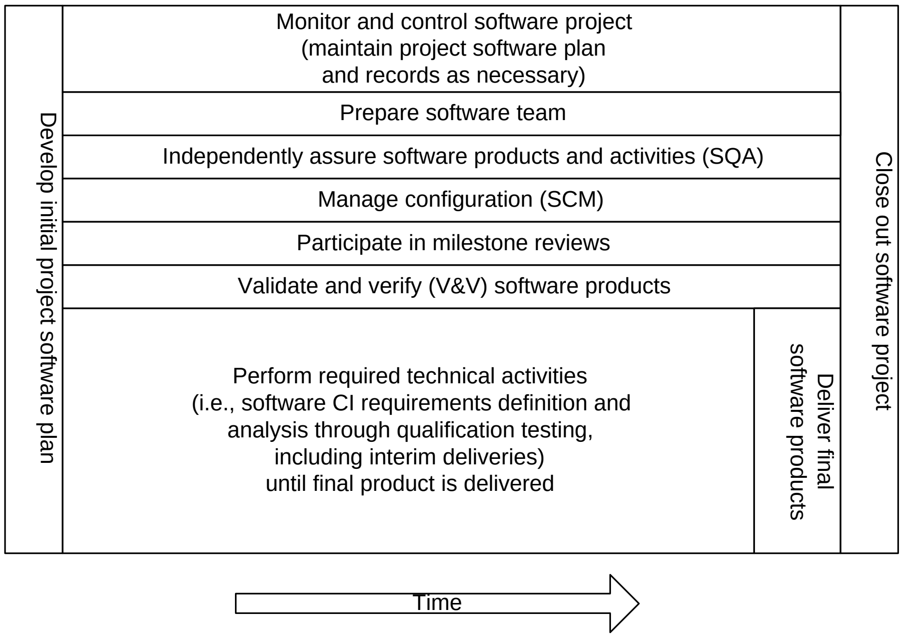
Develop initial project software plan
Monitor and control software project
(maintain project software plan
and records as necessary)
Prepare software team
Independently assure software products and activities (SQA)
Manage configuration (SCM)
Participate in milestone reviews
Validate and verify (V&V) software products
Perform required technical activities
(i.e., software CI requirements definition and
analysis through qualification testing,
including interim deliveries)
until final product is delivered
Deliver final software products
Close out software project
Time

Figure 2–1. Required Software Process Activities

Table 2–1. Required Activities, Products, and Roles

Activity	Primary Products	Key Roles
Software project planning	• Software plan • Project planning review	• Software manager • Software team leader • Software QA representative
Software CI requirements definition and analysis	• Software CI requirements specification • V&V records for requirements definition and analysis products • Software requirements milestone review	• Software requirements analyst • Software QA representative
Software CI design	• Software CI design specification • V&V records for design products • Software design milestone review	• Software design architect • Software detail designer • Software QA representative
Software CI implementation and testing	• Unit-level design • Implementation test plans, procedures • Implemented, integrated, tested software • V&V records for implementation and testing products • Qualification test readiness milestone review	• Software implementer • Software unit tester • Software integrator and tester • Software QA representative
Software CI qualification testing	• Qualification test plan, procedures • Qualification tested software • V&V records for qualification testing products	• Software qualification tester • Software QA representative
Preparation for software delivery	• Executable software • Software source files • Version description • As-built software description • Software user's guide	• Software configuration manager • Rest of software team • Software QA representative
Software project close-out	• Software project history • Software project lessons learned and recommendations for improvement • Software project close-out data	• Software manager • Software QA representative
Software product validation and verification (V&V)	• Software product V&V records	• Software team • Software QA representative
Software configuration management	• Software configuration management plan (part of the software plan) • Controlled software products • Software configuration management records	• Software configuration manager • Rest of software team • Software QA representative
Software quality assurance	• Software quality assurance plan (part of the software plan) • Software quality assurance records	• Software QA representative
Milestone reviews	• Milestone reviews	• Software manager • Rest of software team • Software QA representative
Software team preparation	• Training records	• Software manager • Rest of software team • Software QA representative
Project monitoring and controlling	• Management indicators • Project status reviews	• Software manager • Software QA representative
Software process improvement	• Software technology study plans and study results • Defect causal analysis recommendations	• Entire software team • Software QA representative
System-level considerations	• System and operations concept, operational scenarios • System requirements specification • System design specification • Hardware and software CI integration and test plan, procedures • System qualification test plan, procedures • Qualification tested system • V&V records for qualification testing products	• Entire software team • Software QA representative

2.2.1 Management Activities

The following are required management-related activities:

- Software project planning
- Software team preparation
- Software project monitoring and control
- Software project close-out

2.2.2 Technical Activities

The following are required technical activities, each of which produces one or more specific software products. They may overlap, may be applied iteratively, may be applied differently to different elements of software, and are not necessarily performed in the order listed. (Remember that *activities* are not synonymous with *phases*. Refer to the Glossary (Appendix Appendix A.) for definitions and discussion regarding *activities* versus *phases*.)

- Software CI requirements definition and analysis
- Software CI design
- Software CI implementation and testing
- Software CI qualification testing
- Preparation for software delivery

The following are required support activities that are performed in conjunction with each of the above technical activities:

- Software product validation and verification (V&V)
- Milestone reviews
- Software configuration management (SCM)
- Software quality assurance (SQA)

2.2.3 Software Process Improvement Activities

Every software project presents an opportunity to study and improve the software process. One mechanism used in the NASA software process improvement program is to study the application of new technologies.[1] Process studies are conducted any time an unproven life-cycle or activity-related method is selected by the software manager. The NASA *Software Process Improvement Guidebook* (Reference 2) describes the approach NASA uses to study and understand the effects of new technologies on software products and processes.

2.2.4 System-Level Considerations

Figure 2–2 illustrates the concept of a system life cycle. This section clarifies some terms related to the system life cycle.

[1] A "new" technology is one that has not been proven to be effective in practice in a particular NASA application area or domain. It may have been proven effective elsewhere, however.

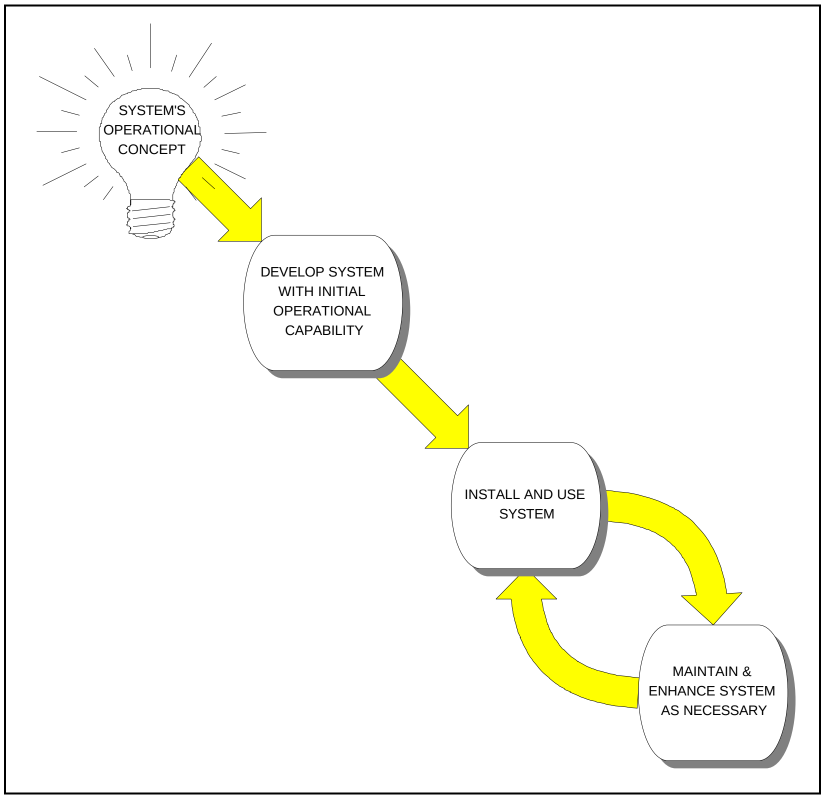
SYSTEM'S OPERATIONAL CONCEPT
DEVELOP SYSTEM WITH INITIAL OPERATIONAL CAPABILITY
INSTALL AND USE SYSTEM
MAINTAIN & ENHANCE SYSTEM AS NECESSARY

Figure 2–2. System Life Cycle

In this guidebook, the term *system* refers to the operational entity that the organization is responsible for developing, maintaining, or enhancing. That is, if the organization is responsible for developing several software and hardware CIs *and* is responsible for integrating them into an operational entity, then the collection of those CIs is the system. If, however, the organization is responsible for developing a single software CI, which may be integrated into (for example) a ground support system by a different NASA organization, then the software CI itself is the system referred to in this guidebook.

Some of the systems that NASA organizations develop, or maintain and enhance, include multiple CIs. There may be a mix of software CIs and hardware CIs, or the system may be only hardware or only software. This guidebook discusses the activities associated with the development and with the maintenance and enhancement of the software CI elements of a system, but also includes very high-level summaries of the relevant system-level activities to help the reader understand the context in which the software effort may take place. When the system comprises only software CIs, then most of the system- and CI-level products are one and the same.

Development (see Figure 2–3) is the creation and installation of an operational system that meets an initially defined set of system requirements. Once the system is operational, subsequent changes are considered *maintenance* or *enhancement* (see Figure 2–4). Many of the maintenance and enhancement activities are the same or similar to those used in development. The NASA software process applies equally to development and to maintenance and enhancement efforts. (The figures reflect the software emphasis of this guidebook.)

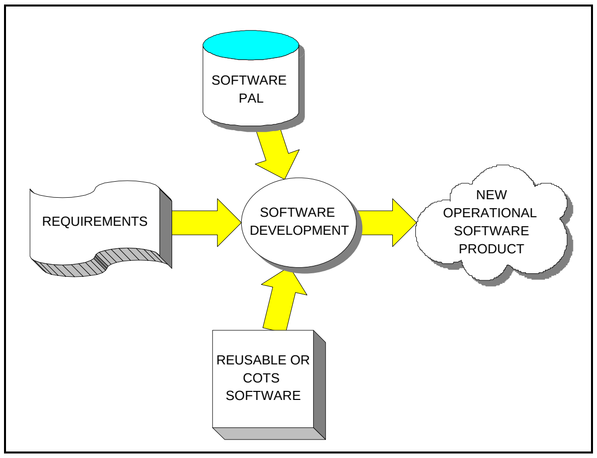

Figure 2–3. Software Development Context

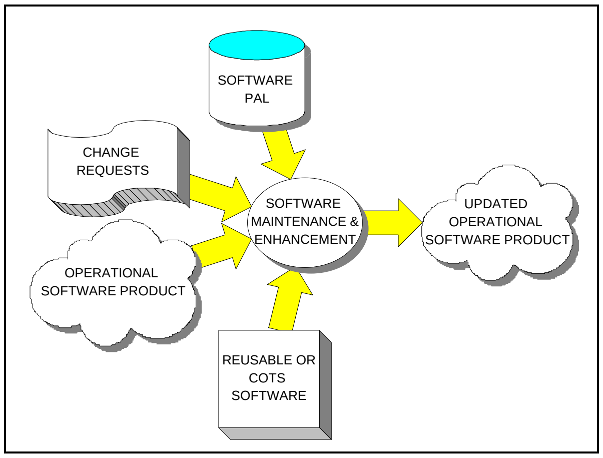
SOFTWARE PAL
CHANGE REQUESTS
OPERATIONAL SOFTWARE PRODUCT
SOFTWARE MAINTENANCE & ENHANCEMENT
UPDATED OPERATIONAL SOFTWARE PRODUCT
REUSABLE OR COTS SOFTWARE

Figure 2–4. Software Maintenance or Enhancement Context

When the software being engineered is a part of a larger system, additional system-level considerations may need to be taken into account. Appendix Appendix D. addresses those considerations.

2.3 Software Process Responsibilities

The subsections that follow summarize the basic responsibilities for maintaining and using the NASA software process at the various organizational levels. Table 2–2 provides examples of software process-related products at each level.

Table 2–2. Sampling of Software Products at Each Organizational Level

Level	**Products**
1 Headquarters, IV&V Facility, and Software Working Group	• NASA *Software Strategic Plan* • This *Software Management Guidebook* • Other agency-wide software engineering (management, development, and assurance) guidebooks, plans, policies, and standards • Domain guidance • Independent in-progress assessments of high-profile development projects' activities and products • Software engineering training
2 Center and Intra-Center Elements (Directorates, Divisions)	• Software quality assurance plan (to include SQA procedures), which is referenced by the software plan and is administered by a center-level software quality assurance organization that is independent of the individual projects • Approach for reviewing and approving projects' software plans • Approach for developing and approving lower level standards, procedures, and plans
3 Branches and Software Projects	• Software engineering (management and development) standards and processes • Software plan • PAL and other related software assets • Project-related software training • Software products resulting from applying the process defined by the plan

2.3.1 Level 1: NASA Headquarters, IV&V Facility, and Software Working Group

The *NASA Software Strategic Plan* (Reference 3), completed in July 1995, complements the Agency-wide strategic vision and mission statements while focusing on software within NASA. This plan was developed by the NASA Software Working Group (SWG) under the auspices of the NASA Software Independent Validation and Verification (IV&V) Facility at Fairmont, West Virginia, sponsored by the Office of Safety and Mission Assurance (OSMA) at Headquarters (HQ). The goals and implementation strategies of the *NASA Software Strategic Plan* address (1) defining and improving software engineering processes (including processes for management, development, and quality assurance), (2) transferring software product and process technologies, and (3) maintaining a core competency in software. Those three elements comprise the NASA Software Program.

The NASA Software Working Group is the implementation vehicle for the NASA Software Program. The role of the SWG is as follows:

- Define, refine, and implement the goals of the *NASA Software Strategic Plan*
- Provide guidance to all NASA software-related activities

- Ensure that available software processes are disseminated

The SWG has one or two representatives from each center and is chaired by a member of the IV&V Facility.

A 1993 survey of NASA Centers (Reference 1) found that NASA does not have a common set of software standards that is used across the Agency in the manner of the Department of Defense's military standard (Reference 4). One focus of the NASA SWG is to produce guidebooks and supporting training on the basis of NASA-wide experience in key areas such as project management, assurance, risk management, and software process improvement (see Section 2.4 for examples).

2.3.2 Level 2: Center and Intra-Center Elements

For the most part, individual centers have given directorates, divisions, and offices the responsibility for developing their own standards and common processes. According to the 1993 survey (Reference 1), two NASA centers (the Marshall Space Flight Center (MSFC) and the Jet Propulsion Laboratory (JPL)) have written software development standards that are baselined at the center level and include a formal waiver process. The other centers have software standards and processes implemented at a lower organizational level. Center-level activities typically focus on defining approaches for plan review and approval. For critical mission systems, however, the SQA organization at each center has the responsibility to ensure that, throughout the life cycle of the project, software engineering activities are performed and software products are prepared in accordance with the software project's software plan.

2.3.3 Level 3: Branches and Software Projects

Branch managers and software project managers share responsibility for developing and approving lower level standards, procedures, and plans for implementing the software process in their areas. They are responsible for the following:

- Identifying, developing, and maintaining those lower level standards, procedures, and plans that are unique to a particular development effort and those that are common to families of software products
- Providing training for their development efforts
- Establishing and maintaining local PALs that contain locally specific process assets

Lower level software engineering process functions define, develop, and implement needed software process assets that are not provided in a higher level NASA PAL. They are also responsible for defining, developing, and implementing a software process improvement program that supports the needs of the projects and branches and for providing software-related information required by higher level measurement and process improvement programs.

Each project must plan its own specific approach for accomplishing the software work assigned to it. The approach must be documented in a software plan and must comply with the local and higher level process requirements.

2.4 Process Assets

The software team uses process assets from an experience-based PAL. A PAL is a compilation of NASA estimating and planning models, historical data, life-cycle models, activity definitions, product standards and templates, and examples of good practices that are available to a software project for developing, maintaining, and implementing its defined process. A PAL may be implemented at any and all organizational levels, as appropriate, within a NASA organization responsible for systems development.

An organization’s PAL typically contains the following products:

- Higher level NASA process assets
- All of the local organization’s software process definition documents
- Software-related guidebooks, handbooks, and white papers, as they become available
- Recommended activity and method definitions and product standards applied within the organization
- Training material related to the organization’s software process
- Plans and results from software process studies
- Other assets that the organization determines to be applicable

This *Software Management Guidebook* and other NASA headquarters-level software-related products (for example, the NASA *Software Measurement Guidebook* (Reference 5) and the NASA *Software Process Improvement Guidebook* (Reference 2)) are included in every NASA PAL. Every PAL must also include copies of NASA Management Instruction (NMI) 2410.10B and other software-related NMIs.

NASA’s *Software Management Guidebook* is a primary source of guidance to the manager in preparing the project’s software plan (see Section 3.2) and selecting appropriate elements from the PAL:

- Life-cycle models (for example, waterfall, iterative refinement, spiral)
- Milestone reviews (for example, requirements reviews, design reviews)
- Products and product formats (for example, degree of formality, packaging)
- Engineering methods (for example, structured approach, object-oriented approach, the Cleanroom method)
- Product V&V methods (for example, peer review methods, testing methods)
- Product control methods (for example, degree of formality)

Using these assets, each software manager prepares a project-specific software plan that defines the selected life-cycle model, methods, tools, and product standards the software team will use. Managers may also use their organizations’ PALs (of varying degrees of formality and structure) as sources of more detailed software process assets that have been tailored for specific application domains. This common tailoring approach allows each software project to draw on proven, successful methods appropriate to satisfy their customers’ needs in a predictable, efficient, and cost-effective manner.

3. The Software Project's Process

This chapter summarizes the software project's process, which includes required activities from developing a software plan that meets the unique needs of each customer through delivery of the final software product and close-out of the project. This chapter also briefly discusses the project software plan, which documents the project's process.

3.1 The Five-Step Project Process

Figure 3–1 is a summary of the five key steps required to take a software engineering project from inception through final delivery and close-out. Subsequent chapters of this guidebook expand on each of the steps. Table 3–1 maps the five steps to the applicable section of this guidebook

Table 3–1. Mapping the Five-Step Project Process to This Guidebook

Project Process Step	**Handbook Section**
STEP 1	Chapter 4: Beginning to Plan the Project: Understanding the Scope of Work
STEP 2	Chapter 5: Defining the Technical Approach
STEP 3	Chapter 6: Finishing the Software Plan—Defining the Management Approach
STEP 4	Section 7.1: Managing the Project
STEP 5	Section 7.2: Closing Out the Project

STEP 1 BEGIN TO PLAN THE PROJECT. UNDERSTAND THE SCOPE OF THE WORK. (Chapter 4)

- Ascertain the customer's requirements and constraints from the software requirements specifications, discussions with the customer, etc.
- Ascertain the customer's goals and objectives (overall cost, schedule, and product qualities) primarily from discussions with the customer and higher levels of NASA.
- Understand management's "risk tolerance level" (both the project team's management and the
- Reach agreement with the customer regarding the products to be delivered and the goals for each product with respect to product cost, schedule, and qualities.
- Document the customer's requirements and constraints, goals and objectives, and management's risk tolerance level in the project's software plan. Document the products to be delivered in the

STEP 2 DEFINE THE TECHNICAL APPROACH THAT BEST ACHIEVES STEP 1. (Chapter 5)

- Incorporate appropriate lessons learned from similar or related software projects.
- Select an appropriate life-cycle model.
- Populate the life-cycle model with appropriate technical activities, methods and techniques, and products.
- Identify any software process improvement activities to be conducted (that is, variations from recommended approaches or required standards).
- Review the technical process using the organization-defined review process.
- Iterate until all stakeholders (project team and customer) are satisfied with the technical process.
- Document the project's technical process in the project's software plan.

STEP 3 FINISH THE SOFTWARE PLAN. DEFINE THE MANAGEMENT APPROACH. (Chapter 6)

- Define and document the management approach to support the technical approach developed in Step 2.
  - Establish the software project's organizational structure.
    Estimate and schedule the work.
  - Identify and plan for team training.
  - Identify and plan for risks.
  - Select measures to facilitate monitoring and controlling the project.
- Review the estimates, schedule, and plan using the organization-defined review procedure.
- Iterate among Steps 1 through 3 until all stakeholders are satisfied with the plan.

STEP 4 RUN THE PROJECT (EXECUTE THE PROJECT'S SOFTWARE PLAN). (Section 7.1)

- Use the life-cycle milestone reviews and appropriate training to help ensure that project personnel understand the products, activities and methods, responsibilities, etc. at each new life-cycle phase.
- Monitor and control the project based on the software plan.
- Review project status periodically with higher levels of management, the customer, and the software team.
- Review the software plan regularly (for example, in conjunction with significant changes, deviations from the software plan, milestone events) and update as needed and as appropriate.
- Maintain necessary project records.

STEP 5 CLOSE OUT THE PROJECT. (Section 7.2)

- Complete the software project history.
- Submit project close-out data.
- Communicate lessons learned to other parts of the organization.

Figure 3–1. The Five-Step Project Process

3.2 Documenting the Project's Process—The Software Plan

Activity Requirements

ü

Planning the Software Project

Objective. Develop a comprehensive plan, based on best practices proven on NASA projects, that will guide the software team through the entire software engineering effort.

Key elements, roles, and responsibilities. The software manager develops and records the software plan for conducting the activities required by this guidebook and by other software-related requirements. Three sets of significant activities (the first three steps of the five-step project process) need to be performed to plan a software project (shaded area on Figure 3–2):

1. Understand the scope of the work (Chapter 4)
2. Define the optimal technical approach (Chapter 5)
3. Define the corresponding management approach (Chapter 6)

(The next three sections of the guidebook provide details on each of these three important sets of activities.)

The independent SQA representative prepares the software quality assurance portion of the software plan and the project's software configuration manager prepares the software configuration management portion of the software plan. This planning is consistent with system-level planning. Software plans are reviewed and approved in accordance with organization-defined procedures, as are significant changes to previously approved plans.

The development and recording of planning and engineering information are intrinsic parts of the software process and are to be performed regardless of whether the information is required as part of the deliverables. The organization's software plan documentation standard serves as a checklist of items to be covered in the planning or engineering activity. To enhance the usability of the information, portions of the plan may be bound or maintained separately. Examples include separate documents for SQA and SCM plans.

Primary products. The primary product from this activity is as follows:

- Project software plan

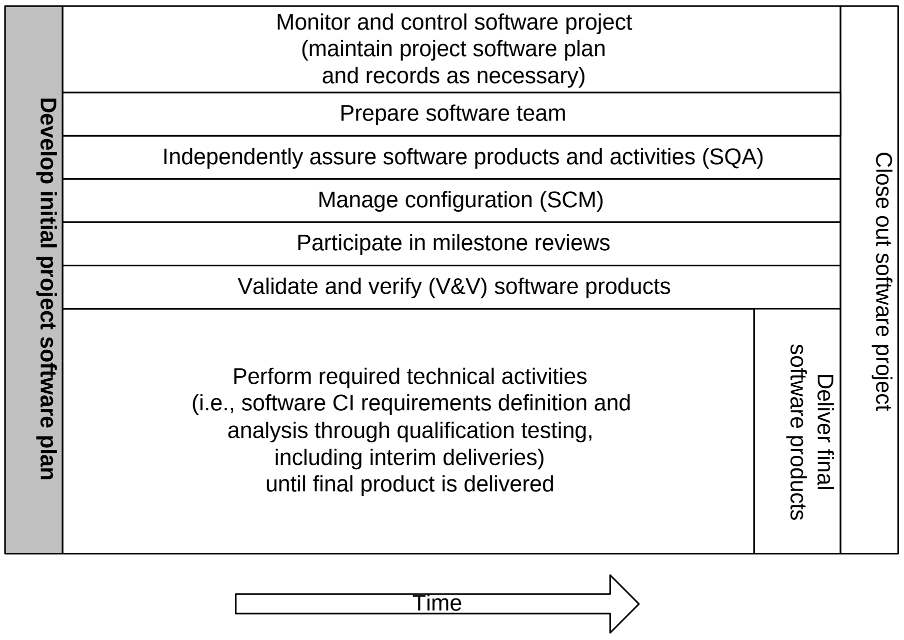
Develop initial project software plan
Monitor and control software project (maintain project software plan and records as necessary)
Prepare software team
Independently assure software products and activities (SQA)
Manage configuration (SCM)
Participate in milestone reviews
Validate and verify (V&V) software products
Perform required technical activities (i.e., software CI requirements definition and analysis through qualification testing, including interim deliveries) until final product is delivered
Deliver final software products
Close out software project
Time

Figure 3–2. Planning the Software Project

Every software project has its own software process. The project's software plan documents the process and provides a disciplined approach to organizing and managing a software project. The existence of a plan does not guarantee project success; the key to successful software management is generating and maintaining a realistic, usable plan and then *following* it. Following the plan involves not only maintaining the plan itself, but also performing activities to measure progress and performance against the plan. Use the plan to assist in recognizing danger signals, and take early and appropriate actions to solve problems.

This guidebook does not mandate a specific format for a software plan; however, every PAL should include good examples of software management plans and a reusable template for such a plan. An excellent example of such a template is the *Reusable Software Management Plan* developed by the Software Assurance Technology Center at the Goddard Space Flight Center (GSFC) (Reference 6). It includes an on-line help tool for tailoring the text to an individual software project. Another excellent example is provided in the NASA Software Engineering Laboratory's *Manager's Handbook for Software Development* (Reference 7). Depending on the specific development environment, items may be arranged differently or new material may be added. In nearly all cases, a project's software plan will be contained in more than one physical document, because many of the plan's components will be common to multiple projects (for example, a QA or CM plan). However, the project's software plan must contain references to all required topics that are packaged separately.

By completing the initial plan early in the life cycle, the manager becomes familiar with the essential aspects of the specific software engineering effort:

- Scope and requirements of the project
- Overall schedule and milestones
- Staff requirements

Each manager defines the project's technical approach, driven by the customer's requirements, constraints, goals, and objectives; management's risk tolerance level; and the target and development or maintenance environments. The technical approach is described in the software plan and should concentrate on information unique to, or tailored for, a specific project. Simply reference applicable documents that contain *standard* policies, guidelines, standards, and procedures to be applied; do not restate that information in detail. Begin to write the plan as soon as information about the project's definition and scope is available. The plan should be available within the first 30 to 60 days of the project, except for information that will not be available until later in the life cycle (indicate who will supply any missing information and when it will be provided). Distribute copies of the plan to all levels of project management, to the client, and to the software team.

The following additional points are included here for completeness and to ensure common understanding. While developing the plan and running the project

- Incorporate lessons learned from other similar or related software projects. Read software project history reports from those projects.
- Use sound judgment. One significant lesson learned from performing software engineering on NASA projects is that no single life-cycle model, analysis and design method, testing method, degree of formality of documents and reviews, etc. is appropriate for every NASA software project. Use of sound, professional judgment is expected with respect to decisions related to such topics.
- You can always do more. This guidebook defines the minimum required software activities and products, not the precise set of things to do. If a manager or project team feels that something additional would be useful, they are empowered to do so.
- Remember that you are not alone in this planning effort; you are not the first nor will you be the last to plan a new or different software project. Consult with other software managers for their opinions, advice, and ideas.

Figure 3–3 illustrates the inputs to the process of preparing a tailored software plan:

- The customer's requirements and constraints as well as his or her goals and objectives
- The risk tolerance levels of the project manager and of the project's organizational management
- Reusable process assets from local and higher level PALs
- The NASA *Software Management Guidebook* (this document)
- The NASA *Software Measurement Guidebook* (Reference 5)
- The NASA *Software Process Improvement Guidebook* (Reference 2)

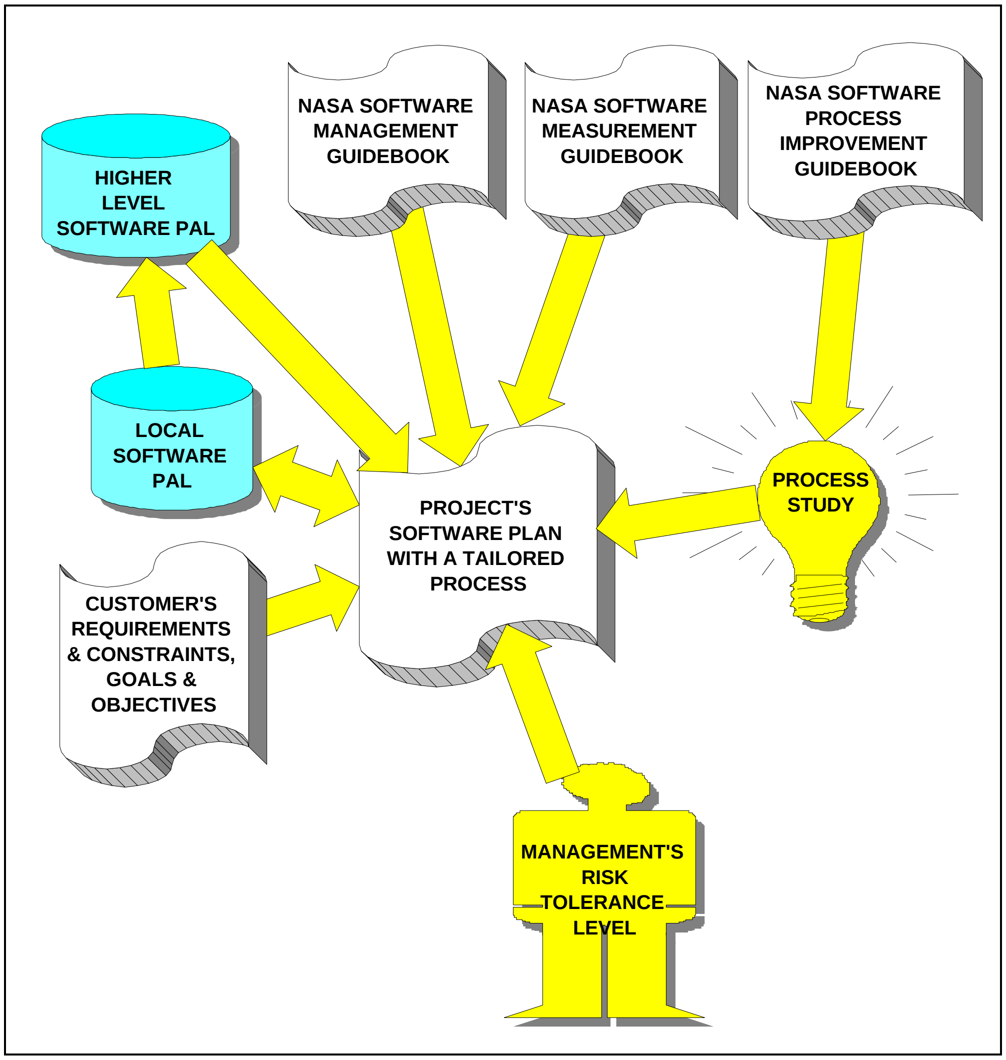
NASA SOFTWARE MANAGEMENT GUIDEBOOK
NASA SOFTWARE MEASUREMENT GUIDEBOOK
NASA SOFTWARE PROCESS IMPROVEMENT GUIDEBOOK
HIGHER LEVEL SOFTWARE PAL
LOCAL SOFTWARE PAL
PROJECT'S SOFTWARE PLAN WITH A TAILORED PROCESS
PROCESS STUDY
CUSTOMER'S REQUIREMENTS & CONSTRAINTS, GOALS & OBJECTIVES
MANAGEMENT'S RISK TOLERANCE LEVEL

Figure 3–3. Tailoring the Project's Software Process

Software Measurement Guidebook presents information on the purpose and importance of measurement for

1. Understanding and modeling the software engineering process
2. Aiding in the management of software projects
3. Guiding improvement in software engineering processes

It discusses the specific procedures and activities of an organizational measurement program and the roles of the people involved. The guidebook also clarifies the role that measurement can and must play in the goal of continual, sustained improvement for all software production and maintenance efforts.

The *Software Process Improvement Guidebook* is a companion document that describes how a software project uses a new technology (that is, one that has not yet been proven anywhere within the organization) in a controlled fashion and quantitatively assesses the effect of the technology on the products generated and the process used.

4. Beginning to Plan the Project: Understanding the Scope of Work

This chapter describes activities that are required to begin planning the project. Software project planning need not wait, indeed must not wait, for receipt of the final software requirements to begin the project planning process. It should begin as soon as project personnel become aware of the intent of the customer to initiate a new software effort, and it should be coordinated, as much as possible, with other NASA groups responsible for related requirements.

Activity Requirements

ü

Understanding the Scope of Work

Objective. Record the project's understanding of the scope of the work to be performed. This activity forms the basis for the technical and management approaches.

Key elements, roles, and responsibilities. The software manager records, in the software plan, his or her understanding of the following:

- The customer's requirements, constraints, and contributions
- The customer's goals and objectives
- Management's risk tolerance level
- Products to be delivered to the customer and their characteristics

Primary products. The primary product from this activity is as follows:

- Required portions of the project's software plan

The rest of this chapter provides guidance in accomplishing this required activity.

4.1 Ascertaining Customer Requirements and Constraints

Ascertain the customer's requirements and constraints from the software requirements specifications, discussions with the customer, and any other appropriate means. The customer may have only a general idea of the software requirements; determination of detailed requirements may follow later or may even be part of the project. (The same applies to characteristics of products to be delivered.)

NASA experience has been that the more the end user of the software product is involved throughout the product's engineering life cycle, the more likely the expected product will result. The customer, however, is not always the end user of the software product. When this is the case, work with the customer to encourage that the end user be as involved throughout as much of the product's life cycle as possible. This involvement is particularly crucial during the requirements analysis and design activities early in the life cycle. It is also important to try to get end-user involvement at milestone reviews.

!!

TIPS

- Try to help the customer prepare the software requirements by reviewing drafts. In this way, the software manager can provide insight, suggest alternatives, point out implications of particular wording, etc., before the requirements statement is finalized.
- When detailed requirements are not well-defined or well-understood, consider using the joint application development (JAD) workshop technique, as described in Section 5.2.1, to help the customer refine the requirements.
- In some organizations, the project software staff defines the requirements as an early part of the project. In those cases, involve the customer as a co-author, or at least as a reviewer, before the requirements statement is finalized.

4.2 Ascertaining Customer Goals and Objectives

During meetings with the customer, ascertain the customer's organizational goals and specific project objectives regarding overall cost, schedule, and product qualities. Also understand the goals of higher levels of the organization. The software manager and customer use the goals to determine and agree on the appropriate robustness desired of product, formality and level of detail and polish in documents, milestone review presentations, and so on. The software manager defines, in the project's software plan, the specific approach that best addresses the application domain, the objectives established for the project, and the size of the effort. Table 4–1 provides examples of the types of objectives that might be established for a project.

Table 4–1. Sample Project Objectives

Objective	**Examples**
Cost	• Minimize cost to develop • Minimize cost to maintain
Schedule	• Deliver by fixed date (for example, 90 days before launch)
Product Qualities	• Maximize reusability • Maximize robustness • Maximize freedom from defects • Maximize performance (for example, response time) • Maximize maintainability or extensibility

4.3 Understanding Management's Risk Tolerance

To minimize costs or cycle time, it is sometimes necessary to introduce risk. Understand what level of risk tolerance is acceptable to the management of both the development or maintenance organization and the customer's organization, and include a risk management approach in the software plan that accommodates that level of risk. (See Section 6.3.)

An approach for continuous risk management is explained in detail in the Software Engineering Institute's (SEI's) technical report *Continuous Risk Management Guidebook* (Reference 8), recently adopted by the NASA Software Program for use throughout the Agency.

4.4 Understanding Products to be Delivered and Their Characteristics

Reach agreement with the customer regarding the products to be delivered and the goals for each product with respect to its cost, schedule, and qualities. Remember that initially the customer may have only a general idea of the product requirements; determination of specific products may follow later or may even be part of the project.

4.4.1 Documentation

The amount of time and effort expended in producing a document can vary considerably. (Refer to the glossary for the definition of *documentation*.) Factors that affect document production include the following:

- Audience for the document—points of view and levels of experience of the readers
- General format
- Level of detail
- Degree of formality

For deliverable documents, the following additional factors apply:

- Paper vs. electronic delivery (if paper, the number of copies per delivery)
- Number of interim deliverable versions

!!

TIPS

- To retain control, avoid using the staff of a technical publications department on early drafts of documents and on documents that will not be delivered to the customer.
- Give outlines or early drafts to the customer for review.

4.4.2 Software Product Releases

Use interim software releases (as opposed to the "big bang" approach) either to satisfy early operational needs of the customer or as a risk mitigation technique (for example, to ensure the feasibility of high-risk requirements). Integrating and testing the system in parts helps to localize erros and reduce debugging time. Early releases also help to build the customer's confidence level, as well as that of management, that the project is on track.

4.4.3 Milestone Reviews

Milestone reviews and associated review material should also be treated as deliverable products. That is, discuss their goals with the customer in terms of responsibilities, cost, schedule, and qualities. Many of the factors that affect producing documentation-oriented products also apply to milestone reviews. (See Section 5.2.9 for details.)

5. Defining the Technical Approach

Chapter 4 described the first step in the planning process: understanding the scope of the work to be performed. The next step is to define a technical approach that best accomplishes the work, which is the subject of this chapter. It includes selecting an appropriate life-cycle model along with corresponding activities, methods, and products.

Activity Requirements

ü

Defining the Technical Approach

Objective. Document the technical approach that the software team will use to accomplish the software work identified in Chapter 4.

Key elements, roles, and responsibilities. The software manager defines the project's technical approach by selecting an appropriate life-cycle model and methods for performing required activities, and appropriate packaging techniques for required products. To facilitate developing each project's technical approach and to avoid reinventing the wheel, this guidebook contains a summary of effective, proven,[2] recommended life-cycle models and methods from which the software manager can choose to satisfy the required common activities.

If, however, the manager believes that a different, unproven method would be more appropriate, then he or she is encouraged to try it out in a controlled fashion by planning a process study (see Section 2.2.3). Such a study allows the manager to capture data about the use of the alternative approach and to measure its overall effect on the software product and process.

If the software manager feels that additional activities or products would be useful, he or she is expected to apply sound, professional judgment in decisions related to such topics.

Each project's tailored process (see   ) is defined in its project-specific software plan.

Primary products. The primary product from this activity is as follows:

- Technical approach portions of the project's software plan

Although the following descriptions of process requirements and tailoring guidelines are extensive, a new project manager should not feel overly intimidated by the length of this chapter. If your new project is similar to one that another project manager has managed previously, then visit your PAL or seek out that manager and reuse the appropriate tailoring from the earlier project. Complete the applicable additional process tailoring and you will be ready to estimate and plan your project based on the specifics of your process. After you have defined the technical approach for your project, everything you need to know to complete the management approach and then run the project is provided in the following two chapters, which are much shorter.

[2] "Proven" means used successfully within NASA.

5.1 Selecting an Appropriate Life-Cycle Model

This section identifies the life-cycle models recommended for use on NASA software projects. A life-cycle model comprises one or more phases (for example, a requirements definition phase, a design phase, a test phase). Each phase is defined as the time interval between two scheduled events. For example, in the waterfall life-cycle model, the design phase is defined as the period between the software specification review and the critical design review (CDR).

Within each phase, one or more activities are executed. For example, during the waterfall model's design phase, the design activity is performed; the test planning activity for qualification testing may be done at the same time. In most cases, activities neither begin nor end precisely at the phase boundaries; rather, they overlap adjacent phases, as illustrated in Figure 5–1.

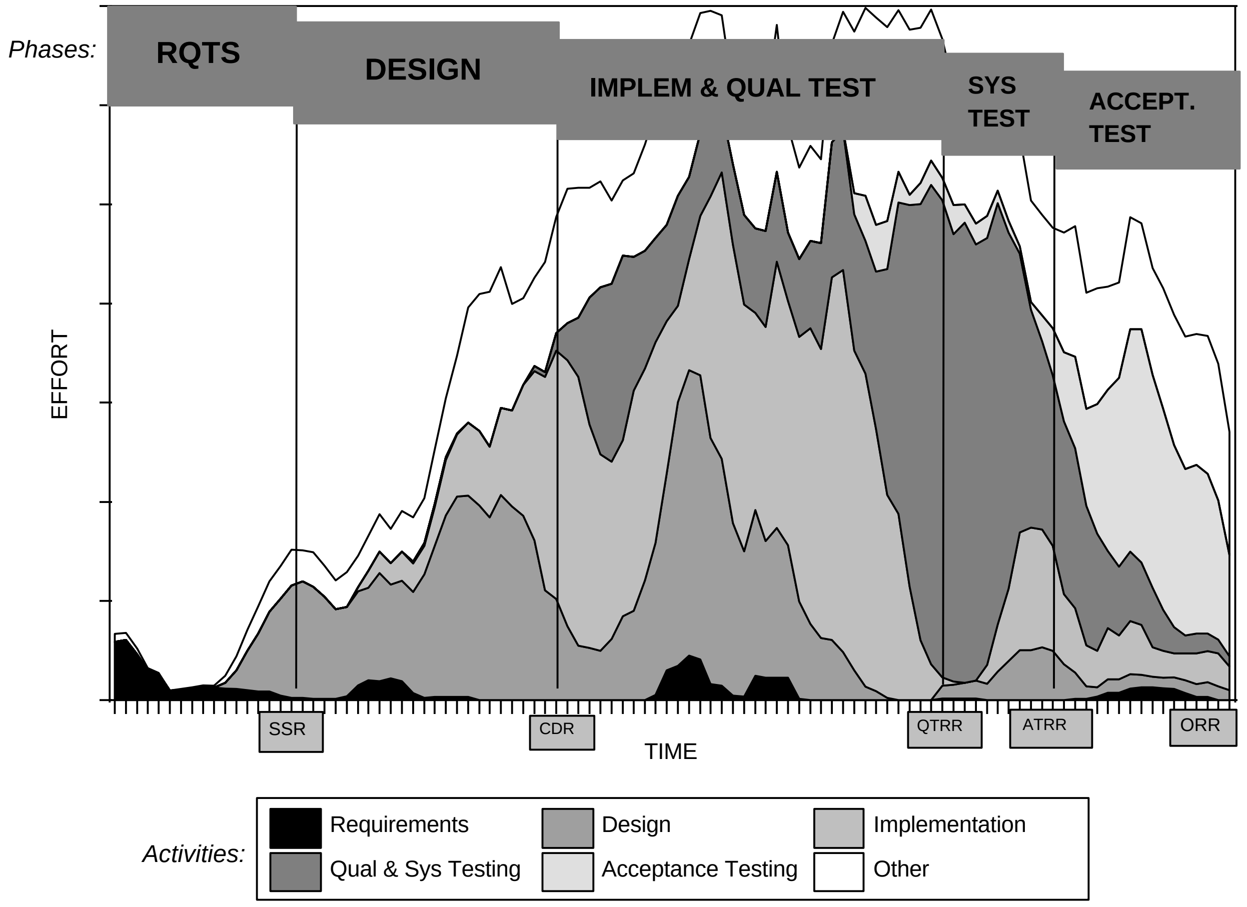

Figure 5–1. Phases and Activities

Various methods (or techniques) may be used in the performance of an activity. For example, object-oriented design is one proven design method; structured design is another.

This document does not mandate any particular software life-cycle model, and the order of activities described here is not intended to conform to any particular model. Few specific methods are mandated for required activities. These decisions are left to the software manager, who selects an appropriate life-cycle model and activity-related methods and defines them in the project's software plan. This chapter contains guidance on selecting an appropriate, recommended life-cycle model and methods for many activities.

For convenience, Table 5–1 provides the definitions (see the Glossary) of several important terms used extensively in this section.

Table 5–1. Defining a Life Cycle

Term	Definition
Software life cycle	"The period of time that begins when a software product is conceived and ends when the software is no longer available for use" (Reference 9). A life cycle is typically divided into life-cycle phases.
Life-cycle model	A framework on which to map activities, methods, standards, procedures, tools, products, and services (for example, waterfall, spiral).
Life-cycle phase	A division of the software effort into non-overlapping time periods. Life-cycle phases are important reference points for the software manager. Multiple activities may be performed in a life-cycle phase; an activity may span multiple phases.
Activity	A unit of work that has well-defined entry and exit criteria. Activities can usually be broken into discrete steps.
Method	A technique or approach, possibly supported by procedures and standards, that establishes a way of performing activities and arriving at a desired result.

Five life-cycle models are summarized in the following subsections. These models are recommended on the basis of NASA's experience applying them successfully at various centers. Development is addressed before maintenance, and the development life-cycle models are ordered from the simplest and most familiar to what may be the most complex and least familiar.

- Waterfall development life-cycle model
- Incremental development life-cycle model
- Evolutionary development life-cycle model
- Package-based development life-cycle model
- Legacy system maintenance life-cycle model

5.1.1 Waterfall Development Life-Cycle Model

Table 5–2 summarizes the life cycle defined by the waterfall development model.

Table 5–2. Summary of Waterfall Development Life-Cycle Model

Summary description and discussion	The waterfall (single-build) life-cycle model is essentially a once-through-do-each-step-once approach. Simplistically, determine user needs, define requirements, design the system, implement the system, test, fix, and deliver the system (Reference 10). This model is illustrated in Figure 5–2. Major products and milestone reviews for this life-cycle model are summarized in Table 5–3.
Advantages	• Well-studied, well-understood, and well-defined • Easy to model and understand • Easy to plan and monitor • Many management tools exist to support this life-cycle model
Disadvantages	• Most if not all requirements must be known up front • Does not readily accommodate requirements changes • Product is not available for initial use until the project is nearly done
Most appropriate when ...	• Project is similar to one done successfully before • Requirements are quite stable and well-understood • The design and technology are proven and mature • Total project duration is relatively short (less than a year) • Customer does not need any interim releases

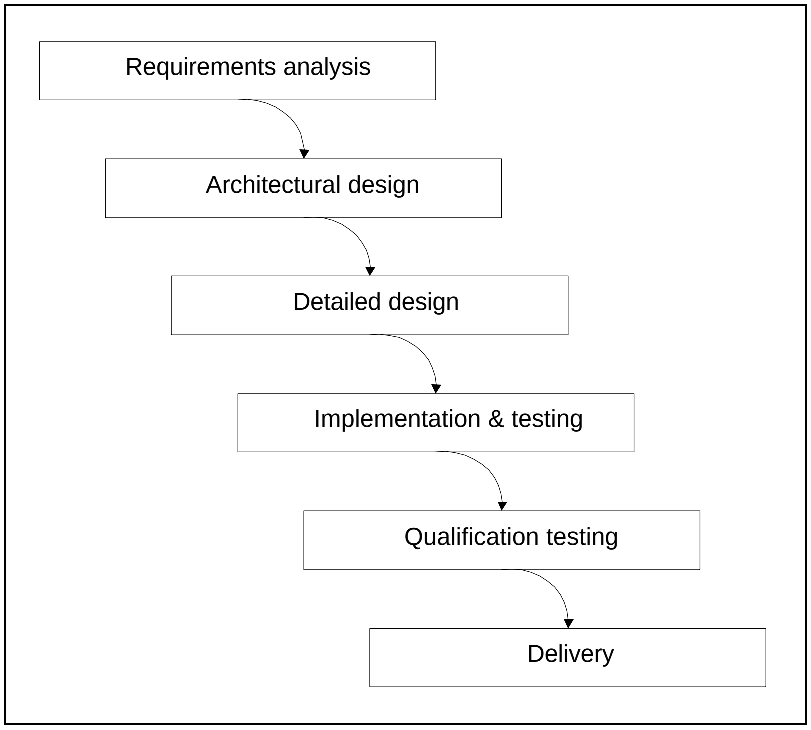

Figure 5–2. Waterfall Development Life-Cycle Model

Table 5–3. Products and Milestone Reviews for the Waterfall Development Life-Cycle Model

Life-cycle phase	**Major products**	**Milestone reviews**
Project planning	• Software plan	None
Requirements definition and analysis	• Software requirements specification (SWRS)	Software Specification Review (SSR)
Architectural design	• Software design specification (SWDS), preliminary • Qualification test plan • Preliminary user's guide	Preliminary Design Review (PDR)
Detailed design	• Software design specification (SWDS), detailed	Critical Design Review (CDR)
Implementation and testing	• Unit-level design • Implemented, tested software • Qualification test procedures • Draft user's guide	Qualification Test Readiness Review (QTRR)
Qualification testing	• Qualification-tested software • Qualification test report • Final user's guide • As-built software description	Acceptance Test Readiness Review (ATRR)

5.1.2 Incremental Development Life-Cycle Model

Table 5–4 summarizes the life cycle defined by the incremental development model.

Table 5–4. Summary of Incremental Development Life-Cycle Model

Summary description and discussion	The incremental (multi-build) life-cycle model determines user needs and defines a subset of the system requirements, then performs the rest of the development in a sequence of builds. The first build incorporates part of the planned capabilities, the next build adds more capabilities, and so on, until the system is complete (Reference 10). This model is illustrated in Figure 5–3. Major products and milestone reviews for this life-cycle model are summarized in Table 5–5.
Advantages	• Reduces risks of schedule slips, requirements changes, and acceptance problems • Increases manageability • Interim builds of the product facilitate feeding back changes in subsequent builds • Interim builds may be delivered before the final version is done; this allows end users to identify needed changes • Breaks up development for long lead time projects • Allows users to validate the product as it is developed • Allows software team to defer development of less well understood requirements to later releases after issues have been resolved • Allows for early operational training on interim versions of the product • Allows for validation of operational procedures early • Includes well-defined checkpoints with customer and users via reviews
Disadvantages	• Like the waterfall life-cycle model, most if not all requirements must be known up front • Sensitive to how specific builds are selected • Places products (particularly requirements) under configuration control early in the life cycle, thereby requiring formal change control procedures that may increase overhead, particularly if requirements are unstable
Most appropriate when ...	• Project is similar to one done successfully before • Most of the requirements are stable and well-understood; but some TBDs may exist • The design and technology are proven and mature • Total project duration is greater than one year or customer needs interim release(s)

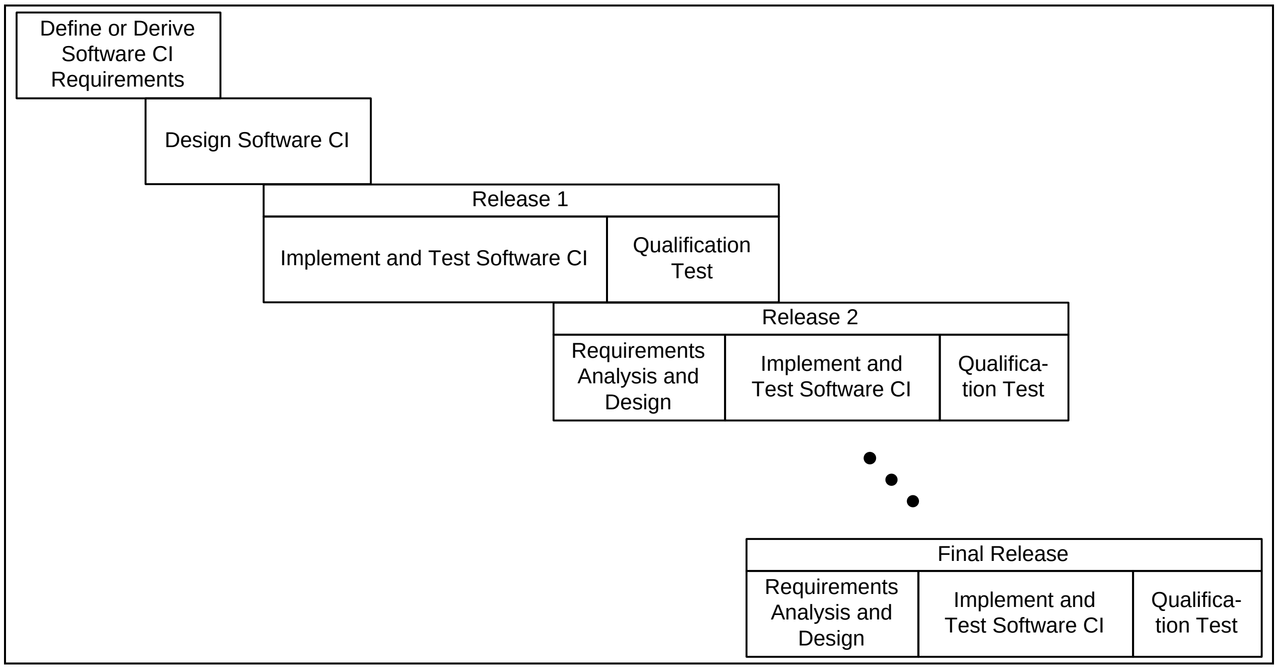
Define or Derive Software CI Requirements
Design Software CI
Release 1
Implement and Test Software CI
Qualification Test
Release 2
Requirements Analysis and Design
Implement and Test Software CI
Qualifica-tion Test
Final Release
Requirements Analysis and Design
Implement and Test Software CI
Qualifica-tion Test

Figure 5–3. Incremental Development Life-Cycle Model

Table 5–5. Products and Milestone Reviews for the Incremental Development Life-Cycle Model

Life-cycle phase	Major products	Milestone reviews
Project planning	• Software plan	None
Requirements definition and analysis	• Software requirements specification (SWRS)	Software Specification Review (SSR)
Architectural design	• Software design specification (SWDS), preliminary • Qualification test plan • Preliminary user's guide	Preliminary Design Review (PDR)
Detailed design	• Software design specification (SWDS), detailed through at least the first build	Critical Design Review (CDR)
	• Software design specification (SWDS), detailed through at least the next build	Build Design Review (BDR)
Implementation and testing	• Unit-level design • Implemented, tested software • Qualification test procedures • Draft user's guide	Qualification Test Readiness Review (QTRR)
Qualification testing	• Qualification-tested software • Qualification test report • Final user's guide • As-built software description	Acceptance Test Readiness Review (ATRR)

5.1.3 Evolutionary Development Life-Cycle Model

Table 5–6 summarizes the life cycle defined by the evolutionary development model.

Table 5–6. Summary of Evolutionary Development Life-Cycle Model

Summary description and discussion	Like the incremental development model, the evolutionary life-cycle model also develops a system in builds, but differs from the incremental model in acknowledging that the user needs are not fully understood and not all requirements can be defined up front. In the evolutionary approach, user needs and system requirements are partially defined up front, then are refined in each succeeding build. The system evolves as the understanding of user needs and the resolution of issues occurs. Prototyping is especially useful in this life-cycle model. (The evolutionary development life-cycle model is sometimes referred to as a spiral development model, but it is not the same as Boehm's spiral model (Reference 11). This model is also sometimes referred to as a prototyping life-cycle model, but it should not be confused with the prototyping technique defined in Section 5.2.1.) This life-cycle model is illustrated in Figure 5–4. Major products and milestone reviews for this model are summarized in Table 5–7.
Advantages	• Not all requirements need be known up front • Addressing high risk issues (for example, new technologies or unclear requirements) early may reduce risk • Like the incremental life-cycle model, interim builds of the product facilitate feeding back changes in subsequent builds • Users are actively involved in definition and evaluation of the system • Prototyping techniques enable developers to demonstrate functionality to users with minimal of effort • Even if time or money runs out, some amount of operational capability is available
Disadvantages	• Because not all requirements are well-understood up front, the total effort involved in the project is difficult to estimate early. Therefore, expect accurate estimates only for the next cycle, not for the entire development effort. • Less experience on how to manage (progress is difficult to measure) • Risk of never-ending evolution (for example, continual "gold plating") • May be difficult to manage when cost ceilings or fixed delivery dates are specified • Will not be successful without user involvement
Most appropriate when ...	• Requirements or design are not well-defined, not well-understood, or likely to undergo significant changes • New or unproved technologies are being introduced • System capabilities can be demonstrated for evaluation by users • There are diverse user groups with potentially conflicting needs

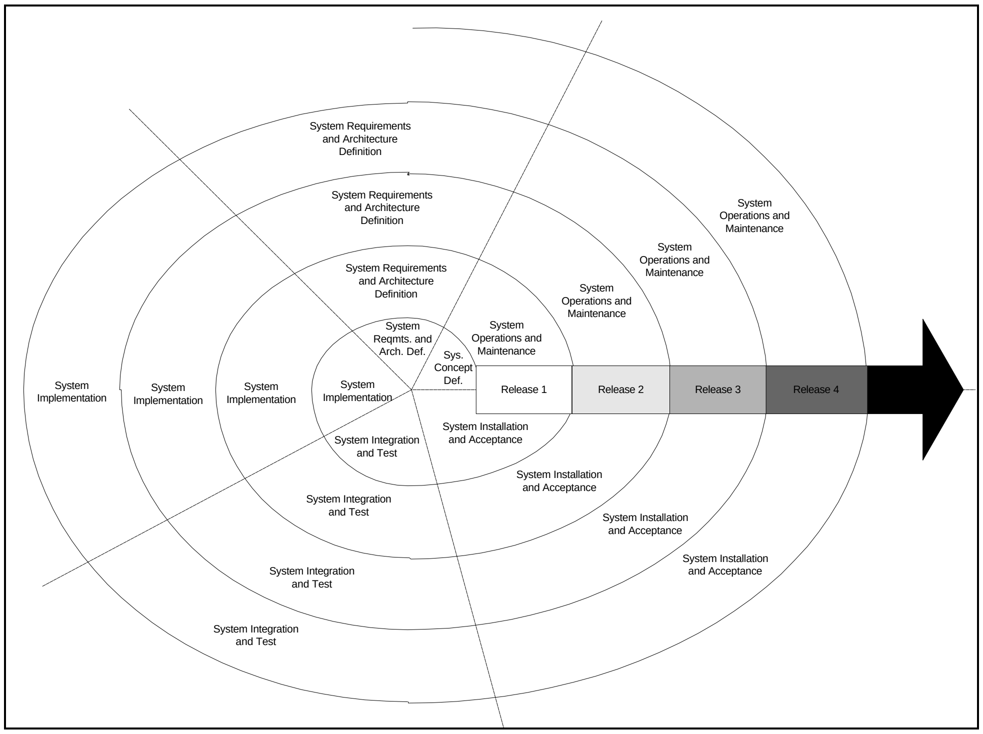
System Requirements and Architecture Definition
System Requirements and Architecture Definition
System Requirements and Architecture Definition
System Reqmts. and Arch. Def.
Sys. Concept Def.
System Operations and Maintenance
System Operations and Maintenance
System Operations and Maintenance
System Operations and Maintenance
System Implementation
System Implementation
System Implementation
System Implementation
Release 1
Release 2
Release 3
Release 4
System Installation and Acceptance
System Installation and Acceptance
System Installation and Acceptance
System Installation and Acceptance
System Integration and Test
System Integration and Test
System Integration and Test
System Integration and Test

Figure 5–4. Evolutionary Development Life-Cycle Model

Table 5–7. Products and Milestone Reviews for the Evolutionary Development Life-Cycle Model

Life-cycle phase	Major products	Milestone reviews
Concept definition	• Initial System Development Plan to be updated in later phases	System Concept Review (SCR)
Requirements and architecture definition	• Preliminary requirements document • Architectural design document containing the infrastructure plus the architecture of each release as it evolves • Requirements traceability map	Combined System Requirements Review (SRR) and System Design Review (SDR)
Implementation	• Evolutionary Implementation Plan • Timebox plan for each timebox (see Table 6–3) • Software product baseline combining new, reused, and off-the-shelf products • Updated requirements traceability map • Draft user documentation	Timebox assessments Qualification testing after all timeboxes for the release have been completed
Integration and test	• System test procedures • Integrated, tested software • Qualification test report • Final user documentation	Acceptance Test Readiness Review (ATRR)
Installation and acceptance	(Although these system life-cycle phases are shown in Figure 5–4 for completeness, they are not discussed here, or in the corresponding tables for the other life-cycle models, because they are out of the scope of the software life cycle.)	
Operations and maintenance		

5.1.4 Package-Based Development Life-Cycle Model

Table 5–8 summarizes the life cycle defined by the package-based development model.

Table 5–8. Summary of Package-Based Development Life-Cycle Model

Summary description and discussion	The package-based development life-cycle model is used for system development based largely on the use of commercial-off-the-shelf and Government off-the-shelf products and reusable packages (Reference 12). Typically, some custom software development is needed to provide interfaces among the NDIs. This model is illustrated in Figure 5–5. Major products and milestone reviews for this life-cycle model are summarized in Table 5–9.
Advantages	• Lower cost than developing equivalent functionality from scratch • Cycle time also often lower than developing equivalent functionality from scratch • Improves confidence in quality of the end product (since quality of NDIs is already known)
Disadvantages	• May result in compromises between desired functionality and functionality provided by NDIs • Maintainability may be more of a challenge because source of NDIs may not be the same NASA organization (for example, requires third party to make changes, raises SCM issues when NDI vendor releases updated versions)
Most appropriate when ...	• A significant portion of the functionality of a system can be provided by NDIs

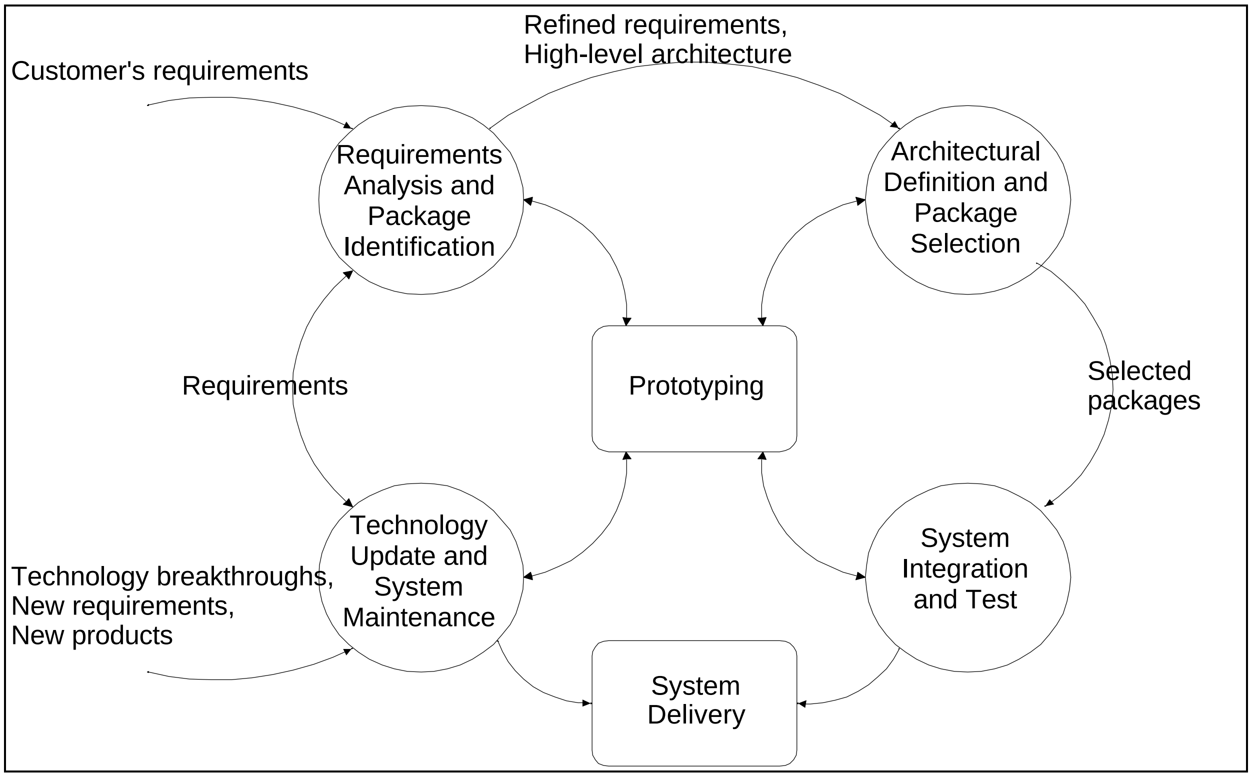
Refined requirements,
High-level architecture
Customer's requirements
Requirements Analysis and Package Identification
Architectural Definition and Package Selection
Requirements
Prototyping
Selected packages
Technology Update and System Maintenance
System Integration and Test
Technology breakthroughs,
New requirements,
New products
System Delivery

Figure 5–5. Package-Based Development Life-Cycle Model

Table 5–9. Major Products and Milestone Reviews for the Package-Based Development Life-Cycle Model

Life-cycle phase	**Major products**	**Milestone reviews**
Requirements Analysis and Package Identification	• System Development Plan • Requirements • Strawman high-level architecture • Candidate packages	System Requirements Review (SRR)
Architectural Definition and Package Selection	• Modified requirements • System architecture • Final packages	System Design Review (SDR)
System Integration and Test	• Delivered system	User demonstrations Operational Readiness Review (ORR)
Technology Update and System Maintenance	• Enhanced system	User demonstrations

5.1.5 Legacy System Maintenance Life-Cycle Model

Table 5–10 summarizes the life cycle defined by the legacy system maintenance model.

Table 5–10. Summary of Legacy System Maintenance Life-Cycle Model

Summary description and discussion	The legacy system maintenance release life-cycle model is used to apply fixes or minor enhancements to an operational system. (Use a waterfall or incremental life-cycle model for major enhancements.) Selected and sometimes abbreviated activities performed in the software *development* life cycles are also performed during maintenance. The legacy system maintenance life-cycle model is similar in nature to the waterfall life-cycle model; the primary difference is that the architectural design has already been established (Reference 10). This model is illustrated in Figure 5–6. Major products and milestone reviews for this life-cycle model are summarized in Table 5–11.
Most appropriate when ...	Maintenance release comprises only fixes and minor enhancements.

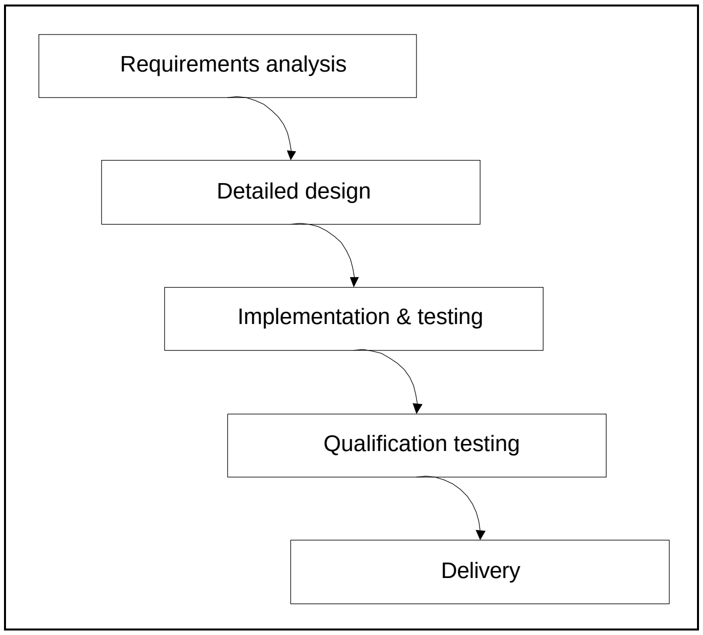

Figure 5–6. Legacy System Maintenance Life-Cycle Model

Table 5–11. Products and Milestone Reviews for the Legacy System Maintenance Life-Cycle Model

Life-cycle phase	Major products	Milestone reviews
Release planning	• Release contents agreement	Release Contents Review (RCR)
Requirements definition and analysis	• Release requirements specification	Release Requirements Review (RRR)
Design	• Release design specification	Release Design Review (RDR)
Implementation and testing	• Unit-level design • Implemented, tested software • Qualification test plan and procedures • Draft user's guide updates	Release Qualification Test Readiness Review (RQTRR)
Qualification testing	• Qualification-tested software • Qualification test report • Final user's guide updates • As-built software description updates	Acceptance Test Readiness Review (ATRR)

5.2 Selecting Appropriate Activities, Methods, and Products

After an appropriate life-cycle model has been selected, it must be populated with software engineering activities, methods, techniques, and products that will help achieve the goals and objectives established for the project. The following subsections present implementation guidance for each required activity (see Table 2–1) and identify recommended methods for performing those activities.

Two logical groups of activities are described in this section:

1. Activities that are performed to produce a specific software product (shaded in Figure 5–7, Primary Software Engineering Activities)
   - Software CI requirements definition and analysis (Section 5.2.1)
   - Software CI design (Section 5.2.2)
   - Software CI implementation and testing (Section 5.2.3)
   - Software CI qualification testing (Section 5.2.4)
   - Preparation for software delivery (Section 5.2.5)
2. Activities that are performed in support of each of the first group of activities (shaded in Figure 5–8, Software Engineering Support Activities and described beginning in Section 5.2.6)
   - Software product validation and verification (Section 5.2.6)
   - Software configuration management (Section 5.2.7)
   - Software quality assurance (Section 5.2.8)
   - Milestone reviews (Section 5.2.9)

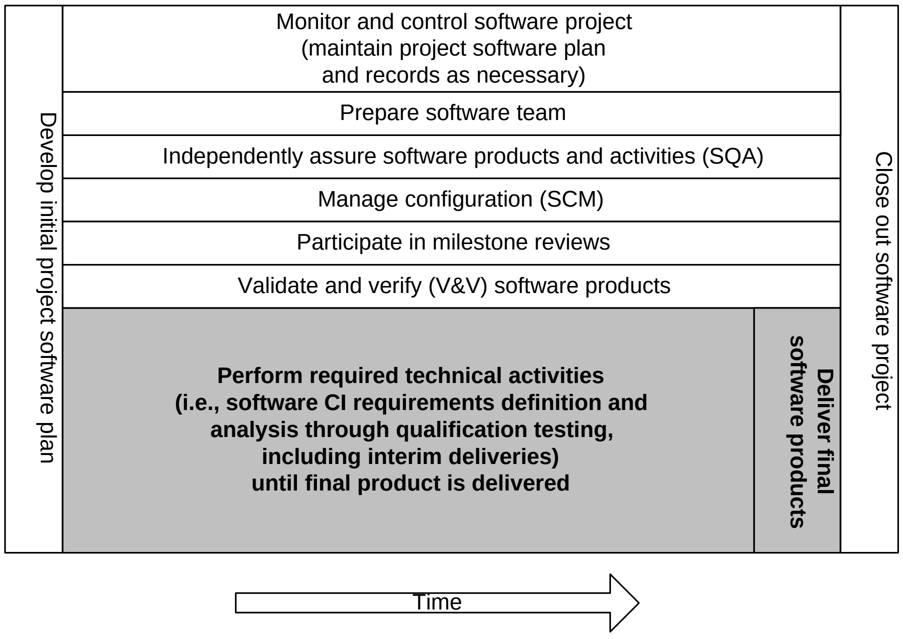
Develop initial project software plan
Monitor and control software project (maintain project software plan and records as necessary)
Prepare software team
Independently assure software products and activities (SQA)
Manage configuration (SCM)
Participate in milestone reviews
Validate and verify (V&V) software products
Perform required technical activities (i.e., software CI requirements definition and analysis through qualification testing, including interim deliveries) until final product is delivered
Deliver final software products
Close out software project
Time

Figure 5–7. Primary Software Engineering Activities

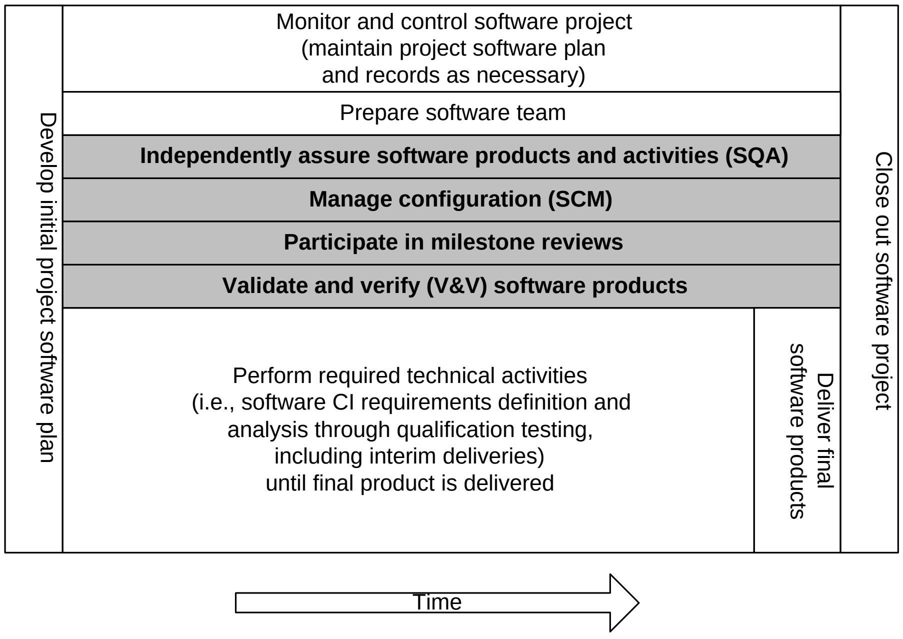
Develop initial project software plan
Monitor and control software project (maintain project software plan and records as necessary)
Prepare software team
Independently assure software products and activities (SQA)
Manage configuration (SCM)
Participate in milestone reviews
Validate and verify (V&V) software products
Perform required technical activities (i.e., software CI requirements definition and analysis through qualification testing, including interim deliveries) until final product is delivered
Deliver final software products
Close out software project
Time

Figure 5–8. Software Engineering Support Activities

5.2.1 Software CI Requirements Definition and Analysis

Activity Requirements

Software CI Requirement Definition and Analysis

Objective. Software requirements form the foundation for all subsequent design and implementation activities. They also form the basis for the customer's acceptance criteria. The quality of the software requirements directly affects the success of the project; therefore, the objective of this activity is to ensure that high-quality software requirements specifications are made available to the software designers, implementers, and testers.

Key elements, roles, and responsibilities.

- When no requirements are provided to the software team, software requirements analysts (sometimes referred to as system engineers) elicit them from the customer, end users, and others as appropriate.
- When higher level requirements are provided (for example, system-level requirements), the analysts examine the requirements for completeness, consistency, and understandability, then derive the software requirements from them.
- When detailed requirements for the software are provided, the analysts examine those requirements for completeness, consistency, and understandability.
- The software requirements analysts record the software requirements to be met by each software CI, the methods to be used to ensure that each requirement has been met, and the traceability between the software CI requirements and higher level requirements (if any). The result includes all applicable items in the software CI requirements specification (SWRS) documentation standard.
- Conduct the following support activities in conjunction with this activity:
  - Validate and verify products of this activity, intermediate and final, in accordance with Section 5.2.6
  - Finalize the products of this activity by holding milestone reviews in accordance with Section 5.2.9
  - Control products of this activity in accordance with Section 5.2.7

Primary products. The primary products from this activity are as follows:

- Software CI requirements specification
- Product V&V records for requirements definition and analysis products
- Milestone review of software requirements

Recommended methods

- Structured requirements analysis (Table 5–12)
- Object-oriented requirements analysis (Table 5–13)

Table 5–12. Structured Requirements Analysis Method

Summary Description and Discussion	Structured requirements analysis is a method of analyzing and specifying the requirements of a product from a functional point of view. Structured analysis includes the use of data flow diagrams, data dictionaries, structured English, decision tables, and decision trees to develop a structured requirements specification.
Advantages	Familiar to most software requirements analysts
Disadvantages	It is sometimes difficult to transform data flow diagrams from this activity into structure charts in the Software CI Design activity
Most appropriate when ...	• Structured techniques will be used throughout the effort • The problem to be solved is well-understood
Key products	Software requirements specification (SWRS), including the following: • Classification of requirements by clarity level (fully defined, needs clarification, or ambiguous) and by category (functional, performance, operations, or programmatic) • Data flow diagrams (DFDs) • Function specifications • Data dictionaries • Identification and definition of external interfaces • Traceability matrices (maps software requirements to higher level system requirements, if applicable)

Table 5–13. Object-Oriented Requirements Analysis Method

Summary Description and Discussion	Object-oriented requirements analysis is a method of analyzing and specifying the requirements of a product in terms of the objects that the system is modeling and operations that pertain to those objects.
Advantages	• There is empirical evidence within NASA that use of object-oriented technology facilitates reuse • Felt by some NASA practitioners to be a more natural, intuitive approach than traditional structured approaches, resulting in products that are more maintainable and modifiable
Disadvantages	• Can be difficult for personnel with a structured analysis background • Unless a recognized object-oriented method is chosen (for example, Booch, OMT), computer-aided software engineering (CASE) tool support is limited or nonexistent
Most appropriate when ...	• Object-oriented techniques will be used throughout the effort • Reusability, maintainability, or modifiability of the products developed is an important objective (The consensus of NASA personnel who have applied object-oriented methods is that object-oriented technology is applicable in most situations.)
Key products	Software requirements specification (SWRS), including the following: • Classification of requirements by clarity level (fully defined, needs clarification, or ambiguous) and by category (functional, performance, operations, or programmatic) • Entity relationship diagrams, data flow diagrams, and state transition diagrams • Identification and definition of external interfaces • Traceability matrices (maps software requirements to higher level system requirements, if any)

Recommended techniques

- Prototyping (Table 5–14)
- Joint application development (JAD) workshops (Table 5–15)

Table 5–14. Prototyping Technique

Summary Description and Discussion	A prototype is an early experimental model of a system, system component, or system function that contains enough capabilities for it to be used to establish or refine requirements, or to validate critical design concepts. A prototype is not an early operational version of a system. It does not contain all required system support functions; is not meant to be as reliable or robust as an operational system; and is seldom constrained by stringent performance, safety, security, or operational requirements.
Advantages	• Prototyping is useful to clarify unclear requirements, to obtain buy-in on user interface characteristics, to gain experience when a new technology is being applied, or to evaluate alternative designs when major performance or reliability issues are unresolved. • Users get to see early versions of product functionality and provide feedback without a lot of time and effort on the part of the developers and testers.
Disadvantages	• Prototyping is sometimes inappropriately used in an attempt to avoid performing proven software engineering activities like peer reviews, testing, configuration management, and documentation. • Users may interpret a prototype to be a finished product and not recognize (or accept) that additional work is required to develop an operational product. • Prototypes, even when used appropriately, have a tendency to become operational products.
Most appropriate when ...	• Requirements are unclear, when the user interface is crucial, when a new technology is being applied, or when major performance or reliability issues are unresolved.
Key products	• Prototype development plan • A prototype of the product to be developed • Prototype summary report

Table 5–15. JAD Workshop Technique

Summary Description and Discussion	JAD is a facilitated workshop technique designed to bring together principal stakeholders to solve a well-defined problem (for example, producing a product of one of the recommended requirements analysis methods) in order to produce a well-defined product or set of products (for example, a traceability matrix).
Advantages	• Includes people with authority to make decisions • Increases visibility into needs of customer and users, and concerns of developers • Facilitates discussing alternatives, and advantages and disadvantages of each • Reduces risk that decisions made by workshop participants will be changed later, thus reducing costs and cycle time due to less rework • Relies on effective group dynamics, resulting in increased synergism among stakeholders
Disadvantages	Impossible if principal stakeholders are not available, not given the authority, or not backed by their management
Most appropriate when ...	• Cycle time must be shortened • There are a number of interfacing organizations that are all stakeholders • Participants with knowledge and authority are available (or will be made available) • There is adequate time for preparation, execution, and follow up • An experienced facilitator and appropriate facilities are available
Key products	• Workshop notes • Action item lists • Intended end products of JAD workshop (requirements specifications in this case) • Post-JAD management briefing

!!

TIPS

- When the system is essentially all software, consider creating the equivalent of a *System & Operations Concept* document (including a discussion of operational scenarios) as an introduction to the software requirements specification.
- Record and track to closure requirements questions and TBD requirements.
- Include software qualification testers in the process of analyzing software requirements; they can review the requirements from a testability point of view.
- NASA has had considerable success using team inspections (see Table 5–22 and Reference 13) for V&V of requirements products.

5.2.2 Software CI Design

Activity Requirements

ü

Software CI Design

Objective. The objective of the software CI design activity is to translate the software CI requirements into a form that can be implemented in software.

Key elements, roles, and responsibilities. This activity produces two basic levels of design: architectural or high-level design, and detailed design.

- Prepare the architectural design.
  - Software design architects define and record design decisions; that is, decisions about the software CI's behavioral design and other decisions that affect the selection and design of the software components included in the software CI. The result includes all applicable items in the software CI design section of the software CI design specification (SWDS) documentation standard.
  - Software design architects define and record the architectural design of each software CI (the software components included in the software CI, their interfaces, and a concept of execution among them) and the traceability between the software components and the software CI requirements. The result includes all applicable items in the architectural design and traceability sections of the SWDS documentation standard. Software components may consist of other software components and may be organized into as many levels as are needed to represent the software CI architecture. For example, a software CI may be divided into three software components, each of which is divided into additional software components, and so on.
- Prepare the detailed design. Software detail designers develop and record a description of each lowest level software component, referred to as a *unit-level* component, or simply *unit*. The result includes all applicable items in the detailed design section of the SWDS documentation standard.
- Conduct the following support activities in conjunction with this activity:
  - Validate and verify products of this activity, intermediate and final, in accordance with Software Product Validation and Verification, Section 5.2.6.
  - Finalize the products of this activity by holding milestone reviews in accordance with Milestone Reviews, Section 5.2.9.
  - Control products of this activity in accordance with Software Configuration Management, Section 5.2.7.

Primary products. The primary products from this activity are as follows:

- Software CI design specification
- Product V&V records for design products
- Milestone review of software design

Recommended methods

- Structured design (Table 5–16)
- Object-oriented design (Table 5–17)

Table 5–16. Structured Design Method

Summary Description and Discussion	Structured design is a method of designing a product from a functional point of view. Structured design includes the use of structure charts, structured English, data flow diagrams, data dictionaries, decision tables, and decision trees to develop a structured design specification.
Advantages	Familiar to most software designers
Disadvantages	There is empirical evidence within NASA that components designed using structured methods are less reusable and less maintainable than those using object-oriented methods It is sometimes difficult to transform data flow diagrams from the Software CI Requirements Definition and Analysis activity into structure charts in this activity
Most appropriate when ...	Product being engineered is one-of-a-kind and is expected to have a relatively short life time (for example, up to a few years)
Key products	Software design specification (SWDS), including the following: • Data flow diagrams (DFDs) • Structure charts • Function specifications • Data dictionaries • Traceability matrices (maps software design components to software requirements)

Table 5–17. Object-Oriented Design Method

Summary Description and Discussion	Object-oriented design is a method of designing a product in terms of the objects that the system is modeling and operations that pertain to those objects.
Advantages	• There is empirical evidence within NASA that use of object-oriented technology facilitates reuse • Felt by some NASA practitioners to be a more natural, intuitive approach than traditional structured approaches, resulting in products that are more maintainable and modifiable
Disadvantages	• Can be difficult for personnel with a structured design background • Unless a recognized object-oriented method is chosen (for example, Booch, OMT), CASE tool support is limited or nonexistent
Most appropriate when ...	Reusability, maintainability, or modifiability of the products developed is an important objective (The consensus of NASA personnel who have applied object-oriented methods is that object-oriented technology is applicable in most situations.)
Key products	Software design specification (SWDS), including the following: • Refined object diagrams • Traceability matrices (maps software design components to software requirements)

Recommended techniques

The same techniques recommended in Section 5.2.1 (Software CI Requirements Definition and Analysis) are often useful in defining and evaluating alternative software CI designs.

Tailoring Guidance for the Design Activity

When there is a high level of design reuse, the architectural design probably already exists and need not be considered a separate element of the design activity. However, be sure to highlight changes from the reused architectural design.

TIPS

- When requirements are especially volatile, increase the rigor of traceability between requirements and design. If there is a change in requirements, then the affected design areas can be identified quickly.
- Include experts in the implementation language, operating system, and other specialties as appropriate in reviewing the design; they each can provide important input regarding their areas of expertise.
- NASA has had considerable success using team inspections (see Table 5–22 and Reference 13) for V&V of design products.

5.2.3 Software CI Implementation and Testing

This activity consists of two essential elements:

1. Implementation and unit testing
2. Integration and testing

5.2.3.1 Software Implementation and Unit Testing

Activity Requirements

ü

Software Implementation and Unit Testing

Objective. The objectives of software implementation and unit testing are (1) to convert the software design into computer programs and computer databases and (2) to test the software at a low level (unit level). (Units may be screens, file formats, reports, etc., as well as source code-oriented components. They also include NDI components.)

Key elements, roles, and responsibilities. This activity consists of two fundamental steps: implementation and unit testing.

- Implementation. Software implementers develop and record software corresponding to each software unit identified in the software CI design. This activity includes, as applicable, designing the unit, coding computer instructions and data definitions, building databases, populating databases and other data files with data values, and performing other activities needed to implement the design. Software units in the design might or might not have a one-to-one relationship with the code and data entities (routines, procedures, databases, data files, and so forth) that implement them or with the computer files that contain those entities. For example, Ada units may comprise more than one code entity. New and modified unit designs and new and modified hand-coded software units undergo inspection in accordance with Section 5.2.6, Software Product Validation and Verification.
- Unit testing. Unit testing verifies a unit's logic, computations, functionality, and error handling. Newly developed or extensively modified units undergo unit testing. Conduct unit testing only after the code has been certified. Units may be tested in isolation or in conjunction with other units. (Define, in the software plan, the overall unit testing approach to be used.) A test plan written by the unit tester (usually the same person as the unit implementer) provides informal guidelines. The unit tester prepares the test data and any necessary drivers or stubs, and then executes the test plan to verify logic paths, error conditions, and boundary conditions. The unit tester verifies each unit's designed functions, internal code paths, and error handling in accordance with the unit test approach defined in the test plan. Software implementers and software unit testers analyze the results of unit testing and record the test and analysis results in appropriate product V&V record files. Software implementers revise the software as necessary, and software unit testers perform necessary retesting; they also update other software products (for example, unit-level designs) as needed, based on the results of unit testing. When the unit tester is the implementer, a peer usually reviews test results for accuracy and completeness, then certifies the tested unit.

  COTS, GOTS, reused, and other NDI software products also undergo some level of V&V and preparation for integration, as defined in the software plan. (Refer to Appendix Appendix C. for more specific guidance in testing or evaluating NDIs.)

Primary products. The primary products from this activity are as follows:

- Unit-level design revisions
- Unit-level test plans, procedures
- Unit-tested software
- Product V&V records for implementation and unit testing products

Recommended testing methods

Because software testing is a key element in several required software engineering activities and because different testing methods are appropriate in different situations, the proven testing methods are summarized in one section of this handbook, Section 5.2.6, Software Product Validation and Verification.

Tailoring Guidance for Unit Testing

The formality and rigor of unit testing will vary depending on the unit's complexity and criticality. Units that are especially complex or critical may need to be tested in isolation, using test drivers and stubs. Otherwise, the testing may be conducted on a collection of related units, perhaps in conjunction with integration testing. Select the level of formality and rigor that is most appropriate and cost-effective for the project as a whole or for various parts of the system.

!!

TIPS

- Use path coverage testing for control units; use functional testing (black box and white or clear box) for algorithmic units.
- For high-reuse systems, concentrate unit testing activities on new and modified components.

5.2.3.2 Software Integration and Testing

Activity Requirements

ü

Software Integration and Testing

Objective. The objectives of software integration and testing are (1) to integrate the software components, which have been tested at a low level (unit level), into the software product at increasingly higher levels of integration and (2) to V&V the resulting software product.

Key elements, roles, and responsibilities. Integration and testing means integrating the software corresponding to two or more software components, testing the resulting software aggregate to ensure that it works together as intended, and continuing this process until all planned software in each software CI is integrated and tested. Integration testing (also referred to as *module*, *string*, and *thread* testing) verifies the internal integrity of a collection of logically related units (that is, a module), checks the module’s external interfaces with other modules, data files, external input and output, etc., and verifies the designed functions of logical groups of components. (Some integration and testing may take place during unit testing. The requirements outlined in this subsection are not meant to duplicate those activities.)

- Although a formal test plan is sometimes not required, integration testing is more carefully controlled than unit testing, thus at least an informal test plan is needed. Software integrators and testers establish test plans in terms of inputs, expected results, and V&V criteria. The test plans cover all aspects of the software CI architectural design. Software integrators and testers perform integration and testing in accordance with the integration test plans and procedures.
- During integration testing, the software is slowly built up by adding a few units at a time to a core of modules that have already been integrated. Integration testing is usually performed by the implementers responsible for the components being integrated. Software implementers and software integrators and testers analyze the results of integration and testing, and record the test and analysis results in appropriate product V&V record files. Software implementers revise the software as necessary, and software integrators and testers perform necessary retesting. They update other software products (for example, software design documents) as needed, based on the results of integration and testing.
- The last stage of this testing is software implementation team-internal software CI testing, which approximates software CI qualification testing.
- Conduct the following support activities in conjunction with this activity:
  - Finalize the products of this activity by holding milestone reviews in accordance with Section 5.2.9, Milestone Reviews.
  - Control products of this activity in accordance with Section 5.2.7, Software Configuration Management.

Primary products. The primary products from this activity are as follows:

- Integration and test plans, procedures
- Integrated, tested software
- Product V&V records for integration and testing products
- Milestone review of readiness for qualification testing

Recommended integration methods

- Top-down method (Table 5–18)
- Bottom-up method (Table 5–19)
- Functional path method (Table 5–20)

Table 5–18. Top-Down Method

Summary Description and Discussion	Integration follows a top-down approach, where lower level modules are added to the top-level driver, level by level, until all components have been integrated and tested.
Advantages	Integrated, tested system can be used as drivers (does not need drivers for higher level calling routines)
Disadvantages	Needs stubs for lower-level called components until the real components become available
Most appropriate when ...	• Architectural design is broad and shallow (for example, transaction processing) • Emphasis is on high-level interfaces • Using high-level NDIs • Beginning to integrate • There is risk or uncertainty regarding the architectural design

Table 5–19. Bottom-Up Method

Summary Description and Discussion	Integration follows a bottom-up approach, where low-level components are integrated and tested with other low-level components, which are then integrated and tested with other low-level components, and so on until all components have been integrated and tested.
Advantages	Does not need stubs for lower level called components
Disadvantages	Needs drivers for higher level calling routines until the real components become available
Most appropriate when ...	• Architectural design is narrow and deep (for example, scientific systems) • Working on utility components • Working on standalone algorithmic components • Using low-level NDIs • Beginning to integrate

Table 5–20. Functional Path Method

Summary Description and Discussion	An end-to-end functional path (or thread) is constructed, to which other modules are then added, until all components have been integrated and tested.
Advantages	• Tests functions or objects in context • Helps assure that functional requirements are being addressed • Requires fewer drivers, since the only driver required is the one needed to initiate the thread
Disadvantages	Harder to ensure complete testing coverage
Most appropriate when ...	• Making enhancements to an existing system • Verifying end-to-end data flows • Integration has progressed to the point where there is an existing structure to build into • Architectural design is object-oriented

Recommended testing methods

Refer to Section 5.2.6, for a discussion of recommended testing methods.

Tailoring Guidance for Integration Testing

Just as described for unit testing, the formality and rigor of integration testing will vary depending on the module’s complexity and criticality. Modules that are especially complex or critical may need to be tested in isolation, using test drivers and stubs. Otherwise, the testing may be conducted on a collection of related modules. Select the level of formality and rigor that is most appropriate and cost-effective for the project as a whole or for various parts of the system.

!!

TIPS

For high-reuse systems, concentrate integration testing activities on new and modified components.

5.2.4 Software CI Qualification Testing

Activity Requirements

Software CI Qualification Testing

Objective. Software CI qualification testing independently verifies that software CI requirements have been met and that the software contains its assigned functionality. (This testing is similar to testing performed in the final stage of software integration and testing.)

Key elements, roles, and responsibilities. Software qualification testers are responsible for this activity. They are not the same persons who performed detailed design or implementation of a particular software CI, although the designers and implementers may contribute to the process, for example, by contributing test plans that rely on their knowledge of the software CI's internal implementation.

- Software qualification testers define and record the test preparations, test plans, and test procedures to be used for software CI qualification testing and the traceability between the test plans and the software CI requirements. They prepare or acquire the test data needed to carry out the test plans and notify the client in advance of the time and location of software CI qualification testing.
- Software qualification testers perform software CI qualification testing of each software CI in accordance with the software CI test plans and procedures.
- Software implementers and software qualification testers analyze the results of software CI qualification testing and record the test and analysis results (including software problem reports) in appropriate product V&V record files.
- Software implementers revise software as necessary. Software qualification testers conduct necessary retesting and update other software products as needed, based on the results of software CI qualification testing.
- Conduct the following support activities in conjunction with this activity:
  - Finalize the products of this activity by holding milestone reviews in accordance with Section 5.2.9.
  - Control products of this activity in accordance with Section 5.2.7.

Primary products. The primary products from this activity are as follows:

- Qualification test plan, procedures
- Qualification tested software
- Product V&V records for software qualification testing products

Recommended methods

Refer to Section 5.2.6 for a discussion of recommended testing methods.

5.2.5 Preparing for Software Delivery

Activity Requirements

ü

Preparing for Software Delivery

Objective. Package the software and accompanying documentation for delivery to either the customer (if the software CI is the final product) or to the NASA group responsible for integrating software and hardware CIs into a higher level system.

Key elements, roles, and responsibilities. The software configuration manager prepares the software delivery package for delivery to either the customer (if the software CI is the final product) or to the NASA group responsible for integrating software and hardware CIs into a higher level system. The QA representative usually provides an additional check that the package is complete and accurate. The software delivery package includes a delivery letter with version description, the software, and associated documentation, including all applicable items in the software delivery package documentation standard. (If software is developed in multiple releases, the software plan identifies the software to be delivered in each release.)

- Prepare the version description. The software configuration manager prepares the version description, which includes the following components:
  - List of requirements met by the delivered software
  - Version identification of software and documentation to be delivered
  - Build instructions
  - Special operating instructions
  - List of resolved problem reports and change reports
  - List of unresolved problem reports
- Prepare the software. The software configuration manager prepares the source files and executable software to be delivered, including any batch files, command files, data files, or other software files needed to regenerate, install, and operate the software on its target computer(s).
- Prepare the as-built software design documentation. Produce this information if the customer requires that it be delivered or if software team personnel will maintain the software CI. The software team updates the design specification of each software CI to match the as-built software. The information includes all applicable items in the SWDS documentation standard.
- Prepare the software user's information. Produce this information if the customer requires that it be delivered or if software team personnel will operate the software CI. The software team identifies and records information needed by users of the software (persons who will operate the software and persons who will make use of its results). The information includes all applicable items in the software user's guide documentation standard.

Activity requirements continued on next page

Activity requirements continued

Requirements for documentation may be satisfied by substituting commercial or other manuals that contain the required information. The documentation identified in this section is normally developed in parallel with software engineering and is ready for use in software CI testing.

- Conduct the following support activity in conjunction with this activity:
  - Control products of this activity in accordance with Section 5.2.7, Software Configuration Management.

Primary products. The primary products from this activity is a software delivery package that includes the following:

- Delivery letter with version description
- The software (executable software, software source files)
- Accompanying documentation (as-built software design documentation, software user’s guide)

5.2.6 Software Product Validation and Verification

Note: Recall that this is the first of four sections describing the group of support engineering activities shown in Figure 5–8. (See the beginning of Section 5.2.)

Activity Requirements

ü

Software Product Validation and Verification

Objective. Independently evaluate software products to ensure they meet all requirements imposed on them (functional and performance requirements as well as product standard requirements).

Key elements, roles, and responsibilities.

- Software team members perform in-process V&V of the software products generated in carrying out the requirements of this guidebook. Software products are evaluated against their specifications or requirements at each stage of the product's evolution (see 21) to ensure that *the right product is being built and the product is being built right.* The two primary methods used for software product evaluation are peer reviews (for paper-oriented products) and testing (for executable products). The software products to be evaluated are identified in this guidebook in conjunction with each of the engineering activity descriptions.
- The persons responsible for evaluating a software product are not the persons who developed the product; although the persons who developed the software product may be involved, for example, as participants in a walkthrough of the product before, or as a part of, its inspection.
- Software team members prepare records of each software product V&V and the project maintains those records for the life of the project. Software product V&V records include (but are not limited to) inspection and certification records (when the V&V method is inspection) and problem and test reports (when the V&V method is testing). Problems in software products under configuration control at the project level or higher are handled as described in Section 5.2.7, Software Configuration Management.

Primary products. The primary products from this activity are software product V&V records, including the following minimum information:

- Identification of the product evaluated
- Product V&V method used
- Criteria used in the V&V
- Problems identified in the product (These are variously referred to as SPRs (system or software problem reports), STRs (system or software trouble reports), DRs (discrepancy reports), and IDRs (internal DRs).)
- Final resolution of the problems
- Certification that the product satisfies V&V criteria

Table 5–21. Software Product V&V Summary

Software product ...	... Is evaluated against (at least) the following requirements or design specification ...	... Usually using the following product V&V method(s)
Software requirements specification (SWRS)	System requirements allocated to the software CI	Walkthroughs; document reviews; inspection
Software design specification (SWDS)	SWRS	Walkthroughs; document reviews; inspection
Unit-level design specifications	SWDS (detailed design)	Inspection (per unit design certification criteria)
Unit-level code	Unit-level design specifications	Inspection (per unit code certification criteria)
		Unit testing (per unit test plan)
Integrated sets of units (modules) up to the CI level	SWDS (detailed design)	Integration testing (also known as module, string, thread testing) (per integration test plan)
Software CI build	Initially, based on SWDS; later, evolving toward SWRS	Build qualification testing (BQT) (per BQT plan)
Software CI release	Initially, based on SWRS; later, evolving toward system requirements allocated to the software CI	Formal (release) qualification testing (FQT) (per FQT plan)
Software delivery to customer	System requirements allocated to the software CI	Acceptance testing (AT) (per AT plan; when the system is a software system, FQT *is* the AT)

Recommended methods

- Peer review methods
  - Inspection method (Table 5–22)
  - Walkthrough method (Table 5–23)
  - Document review method (Table 5–24)
  - Demonstration method (Table 5–25)
- Testing methods
  - Functional (black box) testing (Table 5–26)
  - Structural (white or clear box) or coverage (statement, branch, or path) testing (Table 5–27)
  - Statistical testing (Table 5–28)
  - Regression testing (Table 5–29)
  - Cleanroom (Table 5–31)

Table 5–22. Inspection Method

Summary Description and Discussion	Inspections are well-defined peer reviews that are intended to verify correctness, quality, and compliance with requirements and standards. There are two types of inspection: One-on-one and team. A one-on-one inspection relies on a single inspector; team inspections include two or more. (However, studies conducted within NASA as well as other parts of the software industry clearly show that two or more inspectors are far more effective in discovering defects than a single one (Reference 14).) The primary use of inspections is to find defects; secondary uses include exposing team members to details of other parts of the system, helping increase awareness of the importance of paying attention to details when creating products, helping more junior personnel identify more subtle defects, and increasing the skills of junior personnel by exposing them to the techniques used by more senior staff members. Product certification is used as input to the project’s progress measurement process. Defects found by inspections (which are recorded on inspection and certification records) are used as input to the defect causal analysis process.
Advantages	Very thorough
Disadvantages	Sometimes misapplied—either focuses only on minor defects, such as formatting issues, or done as a formality to certify a product after informal peer reviews have already been performed and no further defects are known.
Most appropriate when ...	• Overall development time is tight • Time allocated to higher level testing is tight • Product quality goals are high
Key products	• Inspection and certification records (software product V&V records) • Inspection data collection forms

Table 5–23. Walkthrough Method

Summary Description and Discussion	Walkthroughs are primarily a method for communicating information to team members. Walkthroughs are not intended to be a defect-finding tool; however, obvious problems are sometimes identified during a walkthrough.
Advantages	Quick, effective, relatively informal method for sharing information with peers.
Disadvantages	Sometimes used in place of inspection method (walkthroughs are not nearly as thorough as inspections; defects are likely to escape detection until later in the life cycle, when effort to repair is greater).
Most appropriate when ...	Information needs to be communicated to others.
Key products	• Comments from team members • Questions to be answered • Action items

Table 5–24. Document Review Method

Summary Description and Discussion	Document reviews are peer reviews of a finished document that are intended to verify completeness, correctness, consistency, quality, and compliance with standards
Advantages	Enables review of all parts of a document in context with each other
Disadvantages	Review is not performed until the document is complete (or nearly complete)
Most appropriate when ...	A complete document must be reviewed
Key products	Review and certification records (software product V&V records)

Table 5–25. Demonstration Method

Summary Description and Discussion	Demonstrations are most frequently used in conjunction with prototypes. The purpose of a demonstration is to solicit feedback from another group (for example, end users of the product).
Advantages	• Facilitates early evaluation of product characteristics in terms of product's look and feel • Encourages customer and end user participation
Disadvantages	• Effective demonstrations require careful preparation (that is, they take extra time and effort) • May lead customer and end users to believe that product is (almost) done • Evaluation may occur later in phase than would occur using other methods • Can lead to requirements growth in terms of nice-to-have features
Most appropriate when ...	Alternative approaches need to be evaluated (for example, user interface techniques)
Key products	• Prototypes • Demonstration summary, results, or agreements

Table 5–26. Functional Testing Method

Summary Description and Discussion	In functional testing, sometimes referred to as black box testing, verifying functionality and correct interfacing of software components is the focus. The tester does not need to understand the internal design of the software component(s) under test, nor how it is implemented—simply *what* the software is expected to do. Functional testing is often organized based on threads of software components that accomplish a higher level function, and is sometimes based on operational scenarios. NASA software testers have found tests based on operational scenarios to be very effective at uncovering errors.
Advantages	• Not overly time-consuming • Ensures that functional requirements are tested • Can be performed by testers without detailed knowledge of design of the software
Disadvantages	• Less likely to detect defects in parts of software that are not frequently executed • Identifies symptoms of defects, not necessarily their causes • Robustness of product is not established
Most appropriate when ...	A primary objective of the testing is to verify • Satisfaction of requirements • Correct interfacing of components
Key products	• Test plans and procedures, including traceability to design or requirements specification (see Table 5–21) • Test records (software product V&V records)

Table 5–27. Structural or Coverage Testing Method

Summary Description and Discussion	Structural testing, sometimes referred to as clear box or white box testing, is testing in which verification of correct implementation of the software design is the focus. The tester must understand the internal design of the software component(s) under test. Coverage testing is a specific form of structural testing that is designed to ensure that each statement or logic path or software component is executed at least once.
Advantages	• Increases confidence in structure of design and correct implementation of the design • More likely (than functional testing) to identify causes of problems
Disadvantages	• Very time-consuming • May miss aspects of functionality and of the "big picture" • Requires in-depth understanding of software internals • Sometimes difficult to force certain conditions
Most appropriate when ...	• A primary objective of the testing is to verify correct implementation of the software design • A primary objective of the testing is to verify thorough exercise of all the software paths • Lower level component testing is being performed (that is, unit level or integration level) • Control-oriented or safety-critical components are being tested
Key products	• Test plans and procedures, including traceability to design or requirements specification (see Table 5–21) • Test records (software product V&V records)

Table 5–28. Statistical Testing Method

Summary Description and Discussion	Statistical testing is testing based on a detailed assessment of expected usage profiles of characteristics of the software that are especially important to the customer. Such characteristics include assessing which software components will be executed most often, which components can cause catastrophic results if defects are present, etc. Statistical testing is usually based on operational scenarios.
Advantages	Concentrates testing effort on parts of the software that are most important to the customer
Disadvantages	• Requires an in-depth understanding of how the software will be used operationally • Increases possibility that defects will remain undiscovered in less frequently exercised parts of the software
Most appropriate when ...	• Testing efficiency is at a premium • Testing period is at a minimum
Key products	• Test plans and procedures, including traceability to design or requirements specification (see Table 5–21) • Test records (software product V&V records)

Table 5–29. Regression Testing Method

Summary Description and Discussion	Regression testing is retesting previously tested software after some kind of change has been made. Changes may have been made in the software itself, or in other software or hardware with which the software interfaces. The purpose of regression testing is to verify that the changes have not adversely affected previously tested software. Regression testing does not usually include rerunning *all* test cases that were originally used, but instead a predefined subset of the test cases that are selected based on criteria established by the project team and the customer.
Advantages	• Promotes confidence in an evolving product • Ensures that obvious defects have not been introduced into the product
Disadvantages	If regression test sets are not chosen carefully, • Considerable effort can be expended if testing is overly exhaustive • Defects can go undetected if testing is too superficial
Most appropriate when ...	Changes are made to software that has undergone higher levels of testing (that is, integration level or qualification level)
Key products	Test records (software product V&V records)

One large NASA organization has found over the past few years that the methods summarized in Table 5–30 have been used most often for unit-level, integration-level, and qualification-level testing.

Table 5–30. Testing Methods vs. Testing Levels

Testing Method	**Unit Level Testing**	**Integration Level Testing**	**Qualification Level Testing**
Functional		✓	✓
Structural	✓	✓	
Coverage	✓	✓	
Statistical		✓	✓
Regression		✓	✓

The Cleanroom method (Reference 15) is an alternative approach that has been used successfully on some NASA projects (Reference 16).

Table 5–31. The Cleanroom Method

Summary Description and Discussion	The Cleanroom method provides an alternative to traditional testing, with a goal of *preventing* software errors rather than *detecting* them. Developed at IBM in the late 1980s, Cleanroom relies on human discipline and intellectual control to build quality into the final product instead of on computer-aided program debugging to detect and remove errors
Advantages	Successful application of the Cleanroom methodology can significantly increase software quality and reliability, decrease test and debug time, and minimize rework efforts
Disadvantages	May not be applicable to large projects
Most appropriate when ...	• Coders and testers can be split into two separate teams • Software system size is less than approximately 50,000 lines of code

Tailoring Guidance for Software Product V&V

Select or adjust the product V&V methods based on the customer's goals and objectives for cost, schedule, and product qualities. For example,

- If high robustness or reliability is called for
  - Increase the amount of peer review and testing
  - Increase the degree of independence of the evaluators from the developer
- If less robustness or reliability is acceptable
  - Have peer reviews and testing focus on those portions of the system that are expected to have high use or are especially critical

Refer to Appendix Appendix C. for guidelines for evaluating NDIs. (Examples of procedures for V&V can be found in Reference 17.)

!!

TIPS

- Make NDI V&V results reports available for other software projects; submit them to your organization's PAL.
- Use product certification as the completion criterion for progress measurement.
- Store software product V&V records using an approach that facilitates easy retrieval and data collection

5.2.7 Software Configuration Management

Activity Requirements

ü

Software Configuration Management

Objective. Ensure that the integrity of a software product is known and preserved through its development and maintenance.

Key elements, roles, and responsibilities. Software configuration management in each build takes place in the context of the software products and controls in place at the start of the build.

Configuration Identification—The software requirements analysts and design architects work with the software configuration manager to identify software CIs, identify the subordinate entities (or categories of entities) to be placed under configuration control, and assign a project-unique identifier to each software CI and subordinate entity to be placed under configuration control. These entities include the software products to be developed or used by the software team. The identification scheme is geared to the level at which entities (for example, computer files, electronic media, documents, software units) will actually be controlled. The identification scheme includes a description or specification of the version, revision, or release status of each entity.

Configuration Control—The software configuration manager establishes and implements procedures defining

- The conditions under which each identified entity (or category of entity) is initially placed under control
- The levels of control through which each entity must pass (for example, author control, project control, customer control)
- The packaging, storage, handling, and delivery of controlled software products
- The mechanisms used to track problems reported in, and changes requested to, controlled software products
- The steps to be followed to request authorization for changes, to process change requests, to track changes, to distribute changes, and to maintain past versions
- The persons or groups with authority to authorize changes and to make changes at each level (for example, software implementer, software manager, client)

Changes that affect an entity under customer control are proposed to the customer in accordance with established procedures.

Configuration Status Accounting—The software configuration manager maintains records of the configuration status of all entities that have been placed under configuration control at the project level or higher. These records are maintained for the life of the project. They include, as applicable, the current version, revision, or release status of each entity, a record of changes to the entity since it was placed under configuration control, and the status of problem reports and change requests that affect the entity.

Activity requirements continued on next page

Activity requirements continued

Configuration Audits—For software CI releases that are to become operational, the software configuration manager or QA representative conducts two types of audits of the software CI after it has finished software CI qualification testing but before delivery:

- Physical configuration audit—this audit verifies that the software conforms to its technical documentation and does not contain unauthorized changes
- Functional configuration audit—this audit verifies that the software CI meets all the requirements allocated to it

(Planning for SCM is included as part of the software project planning activity. The SCM plan (approach) may be physically included in the project's software plan or may be packaged separately. The software configuration manager works with the software manager in creating and maintaining the SCM plan.)

Primary products. The primary products from this activity are as follows:

- Software configuration management approach (part of the software plan), which includes identification, control, status accounting, and audits
- Controlled software products
- Software configuration management records

Tailoring Guidance for the SCM Activity

Levels of Control—There are two fundamental levels of software product change control: *baseline control* (or configuration management) and *local control*. Apply the appropriate level and type of control to verified products—intermediate as well as delivered, new as well as changed from previously controlled versions.

Baseline Control. Some software products, for example, the software requirements, design, and code, should have baselines established at predetermined points. These baselines are reviewed and agreed on with the customer, and serve as the basis for further development. Baselines are typically established in conjunction with milestone reviews, such as SSR (software requirements), CDR (software design), and following software CI qualification testing and physical configuration audit (PCA) or functional configuration audit (FCA) (code). Apply a rigorous change control process to baselined items.

Local Control. Some software products, such as the software plan, may not need to be placed under baseline control, but still need to be controlled locally. This implies that the version of the product in use at a given time (past or present) is known (that is, version control), and changes are incorporated in a controlled manner (that is, change control). For example, software plans are typically placed under local control, but not baseline control.

The specific change control approach used by a software project will vary based on many factors. On some projects, the customer is responsible for some or all SCM functions; in other cases, the software team is responsible; and sometimes the responsibilities are split (for example, requirements documents might be controlled by the customer, but design documents might be under local control until the CI is delivered). Similarly, the configuration review function for a large project might require a formal configuration control board (CCB) while the CCB for a four-person development project might be simply the software project manager and a customer representative.

Each project’s product control approach is defined in the SCM portion of its software plan and specifies which products are to be placed under baseline control and which will be under local control. The approach defines what products will be controlled, under what conditions each kind of product is initially placed under control, who controls it and where, and what has to occur to change it.

Configuration Audits—It may not be necessary to perform FCAs or PCAs at CI level, they may be done at higher level (for example, system or release). Nor is it necessary to perform FCAs and PCAs separately, they may be done together on a system level. (Reference 17 includes examples of procedures for FCAs and PCAs.)

Requirements Management—While a software product’s requirements are only one of several subordinate entities that must be controlled, they are emphasized here because everything that the software team does is based on those requirements. Poor management of requirements has caused problems in the past.

If the customer does not require or maintain any form of software requirements specification, the software team should document and control (to whatever degree of formality is deemed appropriate) the product’s requirements as they understand them and as they are being implemented. The customer should be given a copy of these software requirements to help ensure that the software team and the customer have a common understanding of the basis for the end software product. The requirements should be kept up to date in a controlled fashion.

!!

TIPS

- Identify products to be controlled early, but avoid placing products under control too soon.
- Generally, conduct PCAs and FCAs in conjunction with each other.
- The more builds or releases, the greater the importance of change control.
- The more activities that are going on in parallel (especially build or releases), the greater the importance of change control.

5.2.8 Software Quality Assurance

Activity Requirements

ü

Software Quality Assurance

Objective. Independent SQA is performed throughout the duration of the project to provide confidence to management that approved processes are being followed and that high-quality products are being produced.

Key elements, roles, and responsibilities. SQA is performed by a QA representative who is organizationally independent of the software project and has the skills, responsibility, authority, and organizational freedom to permit objective software product and process evaluations. (The quality of the software product is the responsibility of the entire software team.[3])

The QA representative does the following:

- Ensures that each activity identified in the software plan is performed in accordance with the plan
- Ensures that each software product identified in the software plan is prepared and undergoes software product V&V and corrective action as defined in the plan
- Prepares records of software quality assurance activities and maintains those records for the life of the project
- Provides recommendations for process and product improvement to the software manager and software team

(Planning for SQA is included as part of the software project planning activity. The SQA plan or approach may be physically included in the project's software plan or may be packaged separately. The QA representative works with the software manager to create and maintain the SQA plan. The QA representative performs SQA activities in accordance with the SQA plan.)

Primary products. The primary products from this activity are as follows:

- SQA approach (part of the project's software plan)
- SQA records

Recommended Methods

- Auditing, monitoring, and assessing performance of activities and qualities of products

[3] In some environments, the term *SQA* is used to refer to more than "independent assurance of software products and process"; that is, it sometimes includes peer reviews, testing, etc. This guidebook, however, restricts the SQA activity to include only independent assurance; peer reviews, testing, etc. are included under the Software Product V&V activity.

5.2.9 Milestone Reviews

Activity Requirements

ü

Milestone Reviews

Objective. Jointly, that is, with customer and software team participation, (1) make a final review of a completed product as a basis for its approval, (2) review readiness to proceed to the next phase of engineering, and (3) identify project-level issues and initiate corrective actions.

Key elements, roles, and responsibilities. Milestone reviews are typically associated with life-cycle phases and are usually used to conclude one phase and transition to the next. They are not intended to serve as product V&V; product V&V is assumed to have been performed throughout preparation of the product(s) being reviewed at the milestone review. The software manager and customer plan, and the software team takes part in, milestone reviews. Candidate milestone reviews are listed in Table 5–32. There are two distinct aspects to milestone reviews: technical and management.

Technical aspect—The key role in the technical aspect of milestone reviews is fulfilled by persons who have technical knowledge of the software products that are the subject of the review. Their objectives are as follows:

- Review final software products to establish an SCM baseline
- Provide insight and obtain feedback on the technical effort; surface and resolve technical issues
- Assess project status; identify and discuss near- and long-term risks regarding technical issues
- Agree on mitigation strategies for identified risks, within the authority of the technical participants
- Identify risks and issues to be addressed by management (that is, those discussed under Management aspect, below)
- Ensure ongoing communication between customer and software team technical personnel

(It is assumed that the customer’s technical representatives have reviewed the subject products in advance.)

Management aspect—The key role in the management aspect of milestone reviews is fulfilled by persons with authority to make cost, schedule, and resource allocation decisions. Their objectives are as follows:

- Inform upper management about project status, directions being taken, technical agreements reached, and overall status of evolving software products
- Resolve issues that could not be resolved by technical participants
- Agree on mitigation strategies for near- and long-term risks that could not be resolved by technical participants
- Identify and resolve management-level issues and risks not raised within technical aspect
- Obtain commitments and customer approvals needed for timely accomplishment of the project

Activity requirements continued on next page

Activity requirements continued

Primary products. The primary products from this activity are as follows:

- The milestone review event itself
- Action items resulting from the review (for example, develop a strategy to mitigate a risk identified during the review)
- Approved product(s)
- Approval to proceed to next phase

Table 5–32. Candidate Milestone Reviews

Review	Objective **These reviews are held to provide management with the information necessary to assess progress and execute appropriate corrective action, if required, regarding:**
Concept or contents reviews (SCRs, RCRs)	The operational concept for a software system or the content of a release.
Requirements reviews (SRRs, RRRs)	The specified requirements for a software system, subsystem, or release and to establish the functional baseline.
System design reviews (SDRs)	One or more of the following: • The system- or subsystem-wide design decisions • The architectural design of a software system or subsystem and to establish the system design baseline.
Software specification reviews (SSRs)	The specified requirements (that is, SWRS) for a software CI and to establish the CI-allocated baseline. (SSRs may also be referred to as SRRs when the system is essentially all software.)
Software design reviews (PDRs, CDRs, BDRs, RDRs)	One or more of the following: • The software CI-wide design decisions • The architectural design of a software CI • The detailed design (that is, SWDS) of a software CI or portion thereof (such as a database or an upcoming build) and to establish the CI development baseline.
Test readiness reviews (QTRRs, RQTRRs, ATRRs)	One or more of the following: • The status of the software test environment • The test cases and test procedures to be used for software CI qualification testing or system qualification testing • The status of the software to be tested The reviews that follow software CI qualification testing, PCA, and FCA, but precede the next stage of testing (that is, either system-level integration and testing or system installation and acceptance testing) are held to establish the CI product baseline.
Operational readiness reviews (ORRs)	One or more of the following: • The readiness of the software for installation at operational sites • The user and operator manuals • The software product specifications • The software version descriptions • The status of installation preparations and activities • The status of transition preparations and activities, including transitioning the software development environment (if applicable) to the maintenance organization and to establish the operational baseline.

Recommended Methods

- Meetings (Table 5–33), for example, the milestone review may be held as the final element of a JAD workshop (see Table 5–15)
- Presentations (Table 5–34)
- Demonstrations (Table 5–35)

Table 5–33. Meetings

Summary Description and Discussion	Meetings come in a wide variety of sizes (for example, length, depth of coverage, number of attendees) and degrees of formality. This variation can be of considerable advantage to the software manager in that he or she can, in conjunction with the customer, establish the most cost-effective structure for the meeting.
Advantages	• Usually most efficient and least costly method • Easily scheduled and planned • More conducive to greater interaction of participants than presentations
Disadvantages	• Degree of formality may not be acceptable to the customer • Sometimes difficult to control
Most appropriate when ...	Agreeable to the customer
Key products	• Action items • Approved products(s) • Approval to proceed to the next phase

Table 5–34. Presentations

Summary Description and Discussion	Presentations, like meetings, come in a wide variety of sizes and degrees of formality. Usually, however, additional effort is expended in preparing materials designed expressly for the presentation.
Advantages	Involve a larger audience than meetings
Disadvantages	• Involve a larger audience than meetings • Usually more formal than meetings and, therefore, require more preparation effort; for example, may require producing presentation-quality slides, holding dry runs • Less conducive to interaction of audience than meetings
Most appropriate when ...	Required by the customer
Key products	• Action items • Approved products(s) • Approval to proceed to the next phase

Table 5–35. Demonstrations

Summary Description and Discussion	Demonstrations are used to display the current state of the software product or prototypes of the product
Advantages	Instill level of confidence that the project is on the right track
Disadvantages	In early life-cycle phases, users may interpret the product to be finished and not recognize (or accept) that additional work is required to develop an operational product
Most appropriate when ...	The user interface is crucial, or when a new technology is being applied
Key products	• Action items • Approved products(s) • Approval to proceed to the next phase

Tailoring Guidance for the Milestone Reviews Activity

Milestone reviews may be conducted incrementally, dealing at each review with a subset of the listed items or a subset of the system or software CI(s) being reviewed.

NMI 7120.4 (Reference 18) establishes management policies and responsibilities for major system program and projects. The companion NASA Handbook (NHB) 7120.5 (Reference 19) indicates that only PDRs and CDRs are required of all major programs.[4] The candidate milestone reviews discussed in this guidebook are recommended, but not required, for all software development and maintenance projects, not only for major programs.

[4] NMI 7120.4 defines the scope of a “major” program or project.

!!

TIPS

- Make an effort to understand the customer’s preferred audience for the review, and the perspective from which the review material will be presented.
- Attempt to have representatives of the end users of the product participate in milestone reviews.
- Ensure that the method and degree of formality chosen for the milestone review are appropriate for the product(s) being reviewed.
- Minimize the amount of material created especially for the review; it adds cost and time to the project. Instead, use appropriate portions of the products being reviewed.
- Review presentation material with the customer in advance.
- Dry-run presentations. This will help gauge timing, identify loose ends, familiarize the presenters with the material, and help prepare the presentation team for potential issues that might be raised during the actual review.
- Dry-run demonstrations for the same reasons.
- Consider holding walkthroughs of the product for the customer during the engineering process (that is, throughout each life-cycle phase before its milestone review) to ensure his or her awareness of the product’s evolution well before the milestone review. (The milestone review should not be the first time that the customer has reviewed the product.)

6. Finishing the Software Plan—Defining the Management Approach

The preceding two chapters described how to understand the scope and characteristics of the work to be performed, and how to define an appropriate technical approach to perform the work. This chapter describes the steps needed to define the management approach for the project and to document and review the project's software plan. (As recommended in Chapter 4, planning should be coordinated, as much as possible, with other groups responsible for related software efforts.)

Activity Requirements

ü

Defining the Management Approach

Objective. Document the approach that the software manager will use to manage the software engineering effort.

Key elements, roles, and responsibilities. The software manager defines the project's management approach by doing the following:

- Establishing the software project's organizational structure
- Estimating and scheduling the work
- Planning other activities
- Reviewing the software plan

To facilitate the development of each project's management approach and to avoid reinventing the wheel, this guidebook contains a summary of effective, proven, recommended methods from which the software manager can choose to satisfy the required common activities.

If, however, the manager believes that a different method would be more appropriate, then he or she is encouraged to try it in a controlled fashion. That is, the manager includes a process study of applying the alternative approach to quantitatively assess its overall effect on the software product and process. (See Table 6–5.)

Whenever the software manager feels that additional activities or products would be useful, he or she is expected to apply sound, professional judgment in decisions related to such topics.

Each project's management approach is defined in its software plan.

Primary products. The primary product from this activity is as follows:

- Remaining management approach topics in the project's software plan

The rest of this chapter provides guidance in accomplishing this required activity.

6.1 Establishing the Software Project's Organizational Structure

Activity Requirements

ü

Establishing the Organizational Structure

Objective. Define an effective organizational structure to carry out the selected technical approach.

Key elements, roles, and responsibilities. Once the technical approach has been defined, the software manager must define an appropriate organizational structure to carry out the technical approach. Figure 6–1 illustrates a typical NASA software project structure. It depicts the basic roles that are required on a software project. Record in the software plan a description of the planned organizational structure.

Primary products. The primary product from this activity is as follows:

- Description of software project structure

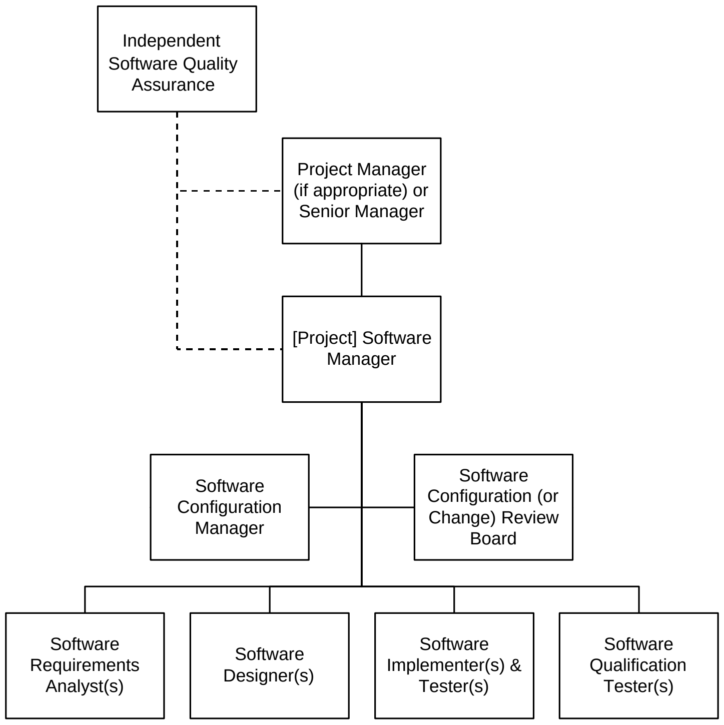
Independent Software Quality Assurance
Project Manager (if appropriate) or Senior Manager
[Project] Software Manager
Software Configuration Manager
Software Configuration (or Change) Review Board
Software Requirements Analyst(s)
Software Designer(s)
Software Implementer(s) & Tester(s)
Software Qualification Tester(s)

Figure 6–1. Typical Software Project Organization

Tailoring guidance

While Figure 6–1 illustrates a typical NASA software engineering organization, it does not imply that different persons are responsible for each role. Quite the contrary, very often a single individual may assume two or more roles on a typical software project (though care must be taken when assigning personnel to roles in which independence is required). These roles may also be filled by personnel from outside of the software team's own organization (for example, customer personnel may be responsible for certain roles).

6.2 Estimating and Scheduling the Work

Activity Requirements

ü

Estimating the Work

Objective. Develop estimates of the size, effort, and schedule duration for the software work.

Key elements, roles, and responsibilities. The software manager estimates product size, effort, and duration in accordance with local standards. Key software team members are often consulted in this activity. Table 6–1 summarizes the three periods for which the software manager develops estimates. Record these estimates in the software plan.

Primary products. The primary product from this activity is as follows:

- Estimates of software product size, effort, and schedule duration.

Table 6–1. Three Levels of Estimates and Plans

Period planned	Purpose of estimate and plan
Project, start to finish	• Provide customer with an estimate and plan for the full software effort. • Ensure thorough high-level analysis of the full software effort.
Upcoming build	Ensure thorough detailed analysis of the work to be accomplished in the next build.
Fiscal year	Develop plans that reflect the customer’s budget for the fiscal year.

Tailoring Guidance

Local standards for estimating size, effort, and duration are based on models derived from historical records of similar projects developed by the same organization. The basis of a size estimate for software projects, for example, may be actual counts of lines of codes, function points, database transactions, or whatever unit is appropriate for the application domain.

An example of local standards for estimating can be found in the *Manager’s Handbook for Software Development* (Reference 7) of the NASA Software Engineering Laboratory (SEL). The SEL derived local estimation standards for all software developed or maintained within the Flight Dynamics Division at GSFC. The SEL’s estimation models are based on the Constructive Cost Model (COCOMO) (Reference 20) but are tailored to the local environment and based on analysis of measurement data collected from software projects since 1976. A detailed explanation of the derivation of the SEL estimation models can be found in the *Cost and Schedule Estimation Study Report* (Reference 21). Further examples are available in the NASA *Software Measurement Guidebook* (Reference 5).

!!

TIPS

The best estimate is still just an estimate. Expect to have to re-estimate as a result of monitoring the status of the project (see Section 7.1.2).

Activity Requirements

ü

Scheduling the Work

Objective. Develop a schedule for performing the software engineering activities, identifying when specific products will be delivered, and including a profile of the staffing levels required to perform the work.

Key elements, roles, and responsibilities. Table 6–1 summarizes the three periods for which the software manager develops schedules. Scheduling for a system or software CI that is to be developed in multiple builds includes

- Overall planning for the project
- Detailed planning for the current build
- Planning for future builds to a level of detail consistent with the information available

The first step in software build planning is to lay out a series of one or more builds and to identify the objectives of each build. The next step is to schedule the activities in each build. The customer may set forth general milestones and have the software team provide specifics or may provide specific schedules.

Record in the software plan, the milestone plan and events list, the work breakdown structure, and the staffing plan.

Primary products. The primary products from this activity are as follows:

- Milestone plan and events list
- Work breakdown structure
- Staffing profile

Recommended techniques

- Mini-Waterfall
- Timeboxes

Table 6–2. Mini-Waterfall

Summary Description and Discussion	This is the traditional technique for planning builds. That is, each build includes some emphasis on analyzing the requirements allocated to the build, usually includes detailed design of the portion of the software that will be implemented in the build, implementation and testing of the software, and qualification testing of the build.
Advantages	Technique is familiar to most managers
Disadvantages	Basically the same as those described under Waterfall Life-Cycle Model (Table 5–2) but not as significant in impact; that is, the requirements to be implemented in the build must be known before beginning the build.
Most appropriate when ...	The requirements to be implemented in the build are quite stable and well-understood

Table 6–3. Timeboxes

Summary Description and Discussion	A timebox refers to a subdivision of a build created to achieve a unit of production that is manageable in size, complexity, staffing, and duration. It is a planning construct that controls functionality delivered by establishing fixed resource and time budgets. That is, when the allocated time and effort have been expended, it is time to move on to the next timebox.
Advantages	Helps avoid “gold plating”
Disadvantages	The product resulting from the timebox might not be complete in its own right
Most appropriate when ...	• Functionality is not well-understood or is likely to change considerably • Using the evolutionary development life-cycle model • Developing prototypes • Cycle time is critical

Tailoring Guidance for Planning Builds

- Avoid treating all software CIs as though they must be developed in lock-step, reaching key milestones at the same time. Allowing software CIs to be on different schedules can result in more optimal development.
- Similarly, avoid treating software components within a CI as though they must be developed in lock-step, all designed by a certain date, implemented by a certain date, etc. Flexibility in the scheduling of software components can also be effective.
- The required software engineering activities need not be performed sequentially. Several may be taking place at one time, and an activity may be performed continually or as needed throughout a build or across multiple builds. The activities in each build should be laid out in the manner that best suits the work to be done.

!!

TIPS

- If early builds are devoted to prototyping, developing throw-away software to arrive at a system concept or assist in defining system requirements, it may be appropriate to forego certain formalities, such as coding standards, that will be imposed later on the deliverable software. If the early software will be used later, such formalities may be appropriate from the start. These decisions are project dependent and should be recorded in the software plan.
- Consider making the first build a simple one to get the project off to a good start; that is, to establish confidence in the software team and the customer.

6.3 Planning Other Activities

Activity Requirements

ü

Planning Other Activities

Objective. Several other elements must be addressed in the project's software plan:

- Team training
- Risk management
- Technical performance measurement
- Process improvement initiatives
- Management measures

Key elements, roles, and responsibilities.

Plan for Team Training—The software manager is responsible for determining the project-specific training needs of his or her team with respect to the application area, new technologies to be employed, etc. Training is usually planned to be delivered just in time to support the team's activities. For example, if a new design method is to be used for the first time, training in that method should be scheduled just prior to commencing the design activity. Record training plans in the software plan.

Planning to manage risks—The software manager is responsible for performing risk management in accordance with local standards. Record the risk management approach in the software plan. (See Reference 8.)

Planning to monitor technical performance measures—There are sometimes situations in which a software CI has specific performance requirements (explicitly defined or implicitly determined), such as maximum processor or memory utilization, minimum response time, etc. Technical Performance Measurement (TPM) enables the software team to monitor and fine tune the product to meet its performance requirements, thus mitigating the risks associated with such requirements. In such cases, the software manager integrates TPM into the software engineering process in accordance with local standards. Record the TPM approach in the software plan.

Planning for software process improvement activities—Every software project presents an opportunity to study or improve the software process. Process studies may be conducted any time a life-cycle or activity-related method other than those recommended in this guidebook is selected by the software manager. These studies enable the software manager to assess the impact of the new technology on the overall product and process, or gain a more in-depth understanding of regularly used technologies. (See Section 2.2.3.) The software manager documents the planned use of a study in the software plan.

Selecting measures to facilitate monitoring and controlling the project—The software manager defines a set of software management indicators to facilitate monitoring the progress of the project. Table 6–4 lists required software engineering activities and summarizes measures that have proven worthwhile for monitoring those activities on NASA software projects. Record in the software plan the measures to be used, and how they will be collected and analyzed.

Activity requirements continued on next page

Activity requirements continued

Primary products. The primary products from this activity are as follows:

- Training plan
- Risk management plan
- TPM plan
- Process improvement plan
- Management measurement plan

Table 6–4. Required Activities and Related Measures

Activity	Cost	Schedule	Defects	Other
Software project planning	✓[a]			
Software CI requirements definition and analysis	✓	✓	✓	Risks, TPM[c]
Software CI design	✓	✓	✓	Risks, TPM[c]
Software CI implementation and testing	✓	✓	✓	Risks, TPM[c]
Software CI qualification testing	✓	✓	✓	Risks, TPM[c]
Preparing for software delivery				
Software project close-out				
Software product V&V	✓			
Software configuration management	✓[b]			
Software quality assurance	✓[b]			
Milestone reviews		✓		
Software team preparation				
Project monitoring and controlling	✓[a]			
Software process improvement	✓	✓		
System-level considerations	✓	✓		

Notes:

a Some NASA organizations typically collect managers' *total* effort regarding project planning and managing the project (as opposed to specific elements within these activities), because they know (quantitatively) that this amount of effort is a small fraction of the total software engineering effort.

b Typically, only *total* effort and schedule are collected and analyzed.

c If applicable

Recommended process improvement method

- Process studies (Table 6–5) (Reference 2)

Table 6–5. Process Studies

Summary Description and Discussion	A process study is a method by which the software manager can objectively determine the impact of introducing a new technology into the software engineering process. The study consists of identifying the objective of the change (for example, reduce cost of development), identifying the new technology, recording the baseline characteristic to be improved (for example, development cost of like products), and identifying the measures to demonstrate whether the objective has been attained. The project team applies the new technology to its work. The resulting measures are compared with the baseline characteristics, thus quantifying the effect of the new technology on the product or process. The results are recorded and made available for use by others.
Advantages	Provides an objective means of assessing a new software technology.
Disadvantages	None
Most appropriate when ...	A new technology is being used.
Key products	• Process study plan • Process study report

!!

TIPS — Team training

- Work with your organization’s training committee to determine what training techniques are available and will best meet the software team’s needs.
- Have local experts train in their specialty areas.
- Have QA personnel conduct training. They are well versed in your processes and product standards.
- For training in group-oriented activities (for example, testing, inspections), have the software team go through the training as a group. Include in the group, personnel who are already experienced in the subject; this allows those people to relate the training material specifically to the project.

!!

TIPS — TPM

Consider planning to develop prototypes to facilitate actual measurement of certain performance-related aspects of the software product. For example, when performance requirements are levied on a database-intensive software product, plan to develop a simple prototype driver to learn where the performance limits of the database package lie.

!!

TIPS — Management Measures

- There's always *some* effort associated with data collection, analysis, and reporting. Collect, analyze, and report only data that are worthwhile to the project, the organization, and NASA.
- Work with your QA representative and process improvement personnel to determine an appropriate set of measures and data collection approach.
- Ensure from the start that you know what data will be required at project close-out and put appropriate mechanisms in place to facilitate providing them.

6.4 Reviewing the Software Plan

Activity Requirements

ü

Reviewing the Software Plan

Objective. Ensure that stakeholders are aware of, and have an opportunity to review, the software plan.

Key elements, roles, and responsibilities.

- Review the plan using the locally defined review process.
- Iterate until stakeholders are satisfied with the estimates and plan.
- Distribute copies of the plan to stakeholders (that is, to team members, and appropriate team and customer management) and use the plan to run the software project.

Primary products. The primary product from this activity is as follows:

- Project's software plan

7. Running the Project

Chapters 3 through 6 discussed how to develop a plan to facilitate running the project. This chapter focuses on the software manager's activities during the execution of the project. Those activities are shown in the shaded portions of Figure 7–1:

- Using the software plan to guide the project in its engineering efforts
- Closing out the project.

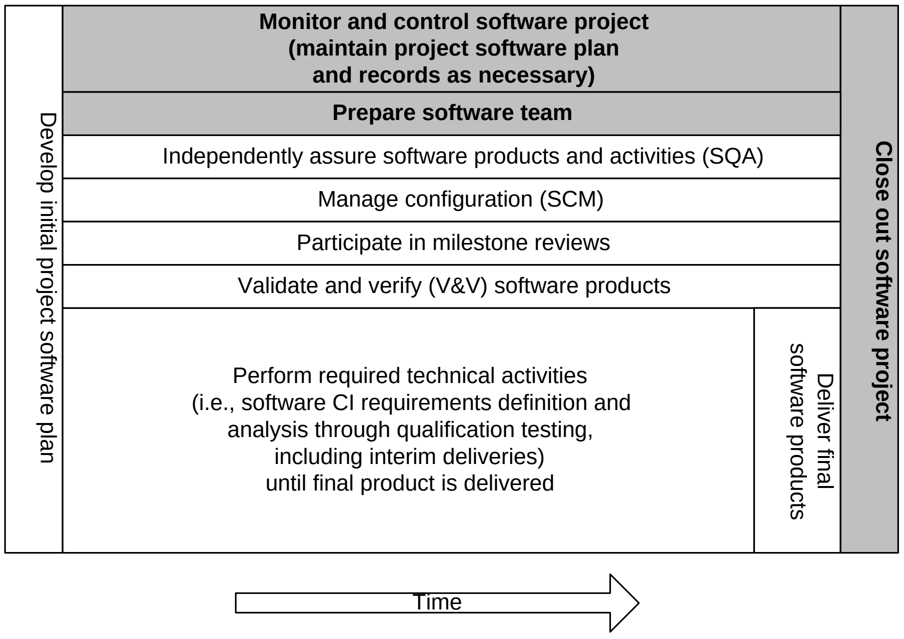
Develop initial project software plan
Monitor and control software project (maintain project software plan and records as necessary)
Prepare software team
Independently assure software products and activities (SQA)
Manage configuration (SCM)
Participate in milestone reviews
Validate and verify (V&V) software products
Perform required technical activities (i.e., software CI requirements definition and analysis through qualification testing, including interim deliveries) until final product is delivered
Deliver final software products
Close out software project
Time

Figure 7–1. Running the Project

7.1 Managing the Project

Activity Requirements

ü

Managing the Project

Objective. Software managers are expected to complete their projects on schedule and within a allowable percentage of the baseline budget. The manager uses an integrated set of management techniques to aid in accomplishing this objective.

Key elements, roles, and responsibilities. Managing the project includes the following key elements:

- Preparing the software team
- Monitoring and controlling
- Communicating with the stakeholders
- Maintaining the project software plan
- Keeping records

Subsequent subsections provide details on each of these elements.

Primary products. The primary product from the activity is as follows:

- Management products (details in following subsections)

7.1.1 Preparing the Software Team

Activity Requirements

ü

Preparing the Software Team

Objective. Ensure the software team is prepared for upcoming activities.

Key elements, roles, and responsibilities. The software manager ensures the software team is prepared for upcoming activities by acquiring training as needed and by holding phase orientation meetings.

The software manager ensures delivery of project-specific training based on the training plan recorded in the software plan. The phase orientation meetings ensures that the software team is aware of and understands the products that will be created during each life-cycle phase (or particular major activity), as well as the activities, methods, procedures, product standards, and support tools that will be used to generate the products.

Primary products. The primary product from the activity is as follows:

- Training records

7.1.2 Monitoring and Controlling the Project

Activity Requirements

ü

Monitoring and Controlling the Project

Objective. The software manager uses software management indicators to help monitor the status of the project and to initiate corrective action in a timely fashion.

Key elements, roles, and responsibilities. The software manager monitors the project by comparing actual measured values with planned or expected values (that is, those measures identified in Section 6.3). When variations between actual performance and the plan become excessive, the manager initiates appropriate corrective actions. (See Reference 5 for examples.)

Managing Effort, Schedule, and Quality

- Collect actual effort expended on software engineering activities using team members' time accounting data. Use software product V&V records (certification records specifically) as a reliable, consistent tool to support monitoring product completion.
- Use software product V&V records as a reliable, consistent tool to support monitoring product quality.

By monitoring progress, productivity, and quality, the software manager will discover whether or not the plan is effective. Although measurement data alone are not sufficient for gauging the plan's effectiveness, they can help the manager to perform some assessment.

Managing Risks—Risk management requires that software managers identify, analyze, and plan to mitigate risks. Review and reassess risks regularly. When new risks are identified, analyze them immediately and plan for mitigation. Close out risks that no longer exist.

Managing Critical Technical Performance Measures—Monitor critical technical performance measures in accordance with local standards.

Primary products. The primary products from this activity are as follows:

- Status records
- Corrective actions

!!

TIPS

Appoint a release leader to coordinate and support the development of a release throughout its implementation, testing, and transition to the client. Coordinating release-specific activities through a single point of contact enables the software team to stay focused on the task at hand; that is, finishing the release.

7.1.3 Communicating with Stakeholders

Activity Requirements

ü

Communicating with Stakeholders

Objective. The effective use of status reports and meetings can facilitate timely communication among the various stakeholders in a software project to ensure that there are no surprises.

Key elements, roles, and responsibilities. The software manager is responsible for communicating with the stakeholders. Table 7–1 lists recommended status reports and meetings and their purposes.

Primary products. The primary products from this activity are as follows:

- Status reports
- Meetings

Table 7–1. Recommended Status Reports and Meetings

Mechanism	Purpose
Status reports	
Weekly progress reports	To ensure regular communication of recent activities and outstanding issues
Monthly status reports	To ensure regular review and reporting of progress and cost
Meetings	
Periodically with the project team members	• To communicate or discuss issues and changes • To provide a forum for questions and answers • To strengthen team cohesiveness
Periodically with the customer	• To communicate or discuss issues and changes • To maintain insight into customer's "hot buttons"
With interfacing groups as needed	To communicate or discuss issues and changes

!!

TIPS—Meetings

- Hold status meetings regularly, not just when there is a problem.
- Keep lists of action items and review them at each meeting
- Follow up with meeting minutes to all meeting attendees

!!

TIPS—Status

Post progress and quality status for all to see. Numerous software project managers regularly post their projects' progress (planned vs. actuals) and quality (expected vs. observed) status for their teams to see. When (for example) a project is behind schedule, posting the status often encourages the team to catch up or even get ahead of the plan.

!!

TIPS—Product Handovers

Any time a product is handed over from one group to another, internal groups as well as external, hold walkthroughs of the product in which the supplier briefs the consumer. Two-way walkthroughs are even more comprehensive, where the supplier briefs consumer; then consumer briefs supplier as to understanding, often raising questions. Examples of product handovers are shown in Figure 7–2.

System requirements allocated to SW CI	Software CI requirements definition team
Software CI requirements	Designers
Software design	Implementers
Tested code components	SW CI integrators and testers
Tested software CI	System integrators and testers
Tested system	Customer

Figure 7–2. Product Handovers

7.1.4 Maintaining the Software Plan

Activity Requirements

ü

Maintaining the Software Plan

Objective. The software plan will be a successful management aid only to the extent that it is followed. Expect the plan to change and evolve as the project continues, so that throughout the development effort it provides the following:

- Documentation of the current project organization (that is, roles and responsibilities)
- A baseline reference against which to measure and compare actual project performance and activities
- Detailed clarification of the technical and management approach being used

Key elements, roles, and responsibilities. The software manager is responsible for periodically reviewing and assessing the effectiveness of the software plan. At the end of each phase or build, review and revise if necessary the estimates of project size, effort, and schedule, and update this information in the plan. (Do not, however, delete earlier estimates. They provide a record of the planning process that will be needed for the software project history and show which estimation techniques were successful and should be used again.)

An effectively maintained software plan documents the current strategy for the software effort. Because it provides a uniform characterization of the project, it can be invaluable if changes occur in team leadership. Ensure that the plan is realistic, otherwise continual modifications will be required to reflect actual decisions or experiences. While the plan should be revised as needed to reflect the current engineering environment, particularly regarding schedules and organizational relationships, significant revisions should not be considered routine document maintenance. For example, major shifts in technical approach or methodologies should occur only if absolutely necessary.

When significant changes are determined to be necessary, the plan should be updated and those updates should be communicated to all personnel who hold copies of the plan. Copies of the plan should be provided to all personnel affected by the change.

Primary products. The primary product from this activity is as follows:

- Up-to-date software plan

7.1.5 Keeping Project Records

Activity Requirements

ü

Keeping Project Records

Objective. Basic records are kept to facilitate tracking critical issues, to provide valuable historical information for others to learn from, and to meet software requirements.

Key elements, roles, and responsibilities. The software manager is responsible for keeping the following basic project records:

Action items—record and track action items

- From meetings with the client
- From milestone reviews
- From meetings with upper management
- From deficiencies found by audits
- Regarding outstanding requirements and design issues and questions

Software project history—At the end of each life-cycle phase (at a minimum) prepare a brief summary of the phase. Record such topics as what went well, what didn't, what surprises sprang up, what should have been done differently, etc. This will form the foundation for the final software project history report.

Primary products. The primary product from this activity is as follows:

- Project records

!!

TIPS

- Consider maintaining a daily or weekly diary that includes (among other things):
  - Actual completion dates
  - Personnel changes
  - When significant changes or problems were first discovered or discussed
  - When significant breakthroughs or brainstorms occurred
- Consider recording end-of-life-cycle phase summaries right in the software plan. This is a simple means of almost automatically producing the software history, right from the start.

7.2 Closing Out the Project

Activity Requirements

ü

Closing Out the Project

Objective. Finalize historical records of the software project for use by other software projects, and process groups within the organization.

Key elements, roles, and responsibilities. The software manager prepares a software project history that includes all applicable items in the organization's software project history documentation standard, and software project close-out data that characterizes both the final product generated and the process used to accomplish the work. Forward a copy of the software project history and the software close-out data to the organization responsible for including it in the PAL.

Primary products. The primary products from this activity are as follows:

- Software project history (including software project lessons learned and recommendations for improvement)
- Software project close-out data

!!

TIPS—Preparing the Project Software History Report

- Consider involving interested software team members in completing the project's software history report.
- Also consider involving the QA representative and a member of a process group. They can help analyze and package the results of any significant technology study that was conducted on the project.

!!

TIPS—Sharing Lessons Learned

Upon completion of every software project, there’s a wealth of information that is vital to the rest of the organization and needs to be shared. The following is a list of several mechanisms that can facilitate sharing that information. Consider using more than one of these mechanisms to ensure a broad dissemination of this invaluable information.

- Lunch time seminars
- Managers’ and staff meetings
- Newsletters

Appendix A. Glossary

Acceptance. An action by the customer (or an authorized representative) by which the customer assumes ownership of software products as partial or complete fulfillment of software requirements.

Activity. A unit of work that has well-defined entry and exit criteria. Activities can usually be broken into discrete steps.

Adapted unit. An existing unit that changes substantially (more than 25 percent of its content is changed, added, or deleted). Its origin is usually external to the project. (Contrast with *new unit, converted unit, transported unit*.)

Architecture. The organizational structure of a system or software CI, identifying its components, the component interfaces, and a concept of execution among the components.

Build. A version of software that meets a specified subset of the requirements that the completed software will meet. (See also *release*.)

Certification. Written confirmation that a work product has been evaluated (for example, inspected or tested) and any defects found by that evaluation process have been satisfactorily resolved.

Configuration item (CI). System component (for example, hardware CI or software CI) that is developed or purchased, controlled, accepted, and maintained separately from other system components. In practice, it is a component that is convenient and sensible to document and control as an entity. (See *software configuration item*.)

Converted unit. Existing unit that changes slightly (up to 25 percent of its content is changed, added, or deleted). Its origin is usually external to the project. (Contrast with *new unit, adapted unit, transported unit*.)

COTS (or **GOTS) software.** Commercial (or government), off-the-shelf software. COTS and GOTS software includes (1) unique components that must be delivered with the product to execute the operational software and (2) development tools that must be delivered with the product to support maintenance of the software.

Customer. The organization that procures software products for itself or another organization.

Cycle time. Elapsed calendar time (not effort) required to complete a given piece of work. For software efforts, cycle time usually refers to either (1) the full product engineering period (from initial receipt of product requirements through acceptance by the customer of the fully functional delivered product) or (2) a release cycle (from agreement on the requirements for the release through acceptance by the customer of the delivered product). The latter case, the release cycle, can pertain to new development releases as well as maintenance and enhancement releases. (See also *software life cycle.*)

Design. Those characteristics of a system or software CI that are selected by the software team in response to the requirements (see *requirements*). Some will match the requirements; others will be elaborations of requirements, such as definitions of all error messages in response to a requirement to display error messages; still others will be implementation related, such as decisions about what software units and logic to use to satisfy the requirements.

Development. Production of a new product that leads to delivery of all initially planned functional capability. The new product may be built from newly created components, reused components (with or without adaptations), GOTS components, and COTS components. (Contrast with *enhancement, maintenance;* see also *software engineering*.)

Documentation. A collection of data, regardless of the medium (or media) on which it is recorded, that generally has permanence and can be read by humans or machines. The significance of this definition is that documents do not necessarily have to be separately bound entities; they may comprise data in several media (for example, plans, CASE tools, databases).

Enhancement. A major addition or change in the functionality of an operational system; often includes many of the same activities as new development. (Contrast with *new development, maintenance;* see also *software engineering*.)

Firmware. The combination of a hardware device, computer instructions, and computer data that reside as read-only software on the hardware device.

GOTS software. (See *COTS/GOTS software*.)

Inspection. A process whereby products are reviewed for correctness, completeness, quality, and compliance with requirements and standards. The process is carried out by one or more peers of the product's developer. (Contrast with *walkthrough*.)

Joint application design (JAD). A facilitated workshop technique that brings together principal stakeholders to solve a well-defined problem to produce a well-defined product or set of products.

Life cycle. (See *software life cycle*.)

Life-cycle model. A framework on which to map activities, methods, standards, procedures, tools, products, and services (for example, waterfall, spiral).

Life-cycle phase. A division of the software effort into non-overlapping time periods. Life-cycle phases are important reference points for the software manager. Multiple activities may be performed in a life-cycle phase; an activity may span multiple phases. (Contrast with *activity*.)

Maintenance. Implementation of problem fixes and minor enhancements to an operational product. (Contrast with *new development, enhancement;* see also *software engineering*.)

Method. A technique or approach, possibly supported by procedures and standards, that establishes a way of performing activities and arriving at a desired result.

Methodology. "A collection of methods, procedures, and standards that defines an integrated synthesis of engineering approaches to the development of a product." (Reference 22)

Milestone review. A process or meeting involving representatives of both the customer and the software team, during which project status, software products, and project issues are examined and discussed.

Module. A cohesive group of software units that performs a software function.

New development. (See *development*.)

New technology. A technology that has not yet been proven to be effective in practice in a particular NASA application area or domain. It may have been proven elsewhere, however. (See also *technology*.)

Non-developed item (NDI). A component that is not newly created by the software team. This includes reused components, COTS and GOTS components, and components developed by other NASA groups or contractor personnel.

Organization. "A unit within [NASA] within which many projects are managed as a whole. All projects within an organization share a common top-level manager and common policies." (Reference 22)

Phase. (See *life-cycle phase*.)

Policy. "A guiding principle, typically established by senior management, which is adopted by an organization or project to influence and determine decisions." (Reference 22) (Contrast with *procedure, standard*.)

Procedure. A written description of the roles, responsibilities, and steps required for performing an activity or a subset of an activity. (Contrast with *policy, standard*.)

Process. "A sequence of steps performed for a given purpose; for example, the software engineering process." (Reference 9)

Process asset library (PAL). A NASA library, including a life-cycle methodology description, standards and procedures, guidebooks and handbooks, style guides, software profiles, key lessons, study results, tailored processes, examples of acceptable products, etc.

Product. (See *software product*.)

Project. (See *software project*.)

Project process. A tailored version of the organization's standard process, defining and integrating specific life-cycle models, activities, methods, procedures, standards, and tools used to accomplish delivery of the project's required products and services. It is defined in the

Prototype. An early experimental model of a system, system component, or system function that contains enough capabilities for it to be used to establish or refine requirements, or to validate design concepts.

Qualification testing. Testing performed to verify and demonstrate (often to the customer) that a software CI or a system meets its specified requirements.

Record. To *record* information means *to set down in a manner that can be retrieved and viewed.* The result may take many forms, including but not limited to hand-written notes, hard-copy or electronic documents, and data recorded in CASE and project management tools. Information to be delivered to the customer must be in the form, format, and medium agreed to with the customer. For information not to be delivered to the customer, the software manager selects the appropriate form, format, and medium.

Release. A build that is delivered to the customer. (See *build*.)

Requirement. A characteristic that a system or software CI must possess in order to be acceptable to the customer.

Reused code. Code that has undergone no more than 25 percent change; that is, converted code and transported code. (See *converted unit, transported unit*.)

Software configuration item (CI). An aggregation of software that satisfies an end use function and is designated for separate configuration management by the customer or the maintenance team. Software CIs are selected on the basis of tradeoffs among software function, size, host or target computers, developer, support concept, plans for reuse, criticality, interface considerations, need to be separately documented and controlled, and other factors.

Software engineering. A set of activities that results in software products. Software engineering includes new development, modification, reuse, reengineering, maintenance, or any other activities that result in software products.

Software engineering environment. The facilities, hardware, software, firmware, procedures, and documentation needed to perform software engineering. Elements may include but are not limited to CASE tools, compilers, assemblers, linkers, loaders, operating systems, debuggers, simulators, emulators, documentation tools, and database management systems.

Software engineering process. An organized set of activities performed to translate user needs into software products.

Software life cycle. "The period of time that begins when a software product is conceived and ends when the software is no longer available for use." (Reference 9) A life cycle is typically divided into life-cycle phases. (See *life-cycle phase*.)

Software process. (See *software engineering process*.)

Software product. Software or associated information created, modified, or incorporated to satisfy the software requirements. Examples include plans, requirements, design, code, databases, test information, and manuals.

Software product evaluation. Activities performed by the software team to ensure that in-process and final software products meet criteria established for those products. (Contrast with *software quality assurance*.)

Software project. An undertaking requiring a concerted effort, which is focused on developing or maintaining a specific software product. Typically a software project has its own funding, cost accounting, and delivery schedule. That is, the project is the work to be done and the personnel assigned to perform the work; it is not the same as the product (for example, system) that they are to produce.

Software quality assurance (SQA). Activities performed by independent QA personnel to (1) ensure that each activity described in the software plan is performed in accordance with the software plan, and (2) ensure that each software product required by the organization or by software requirements exists and has undergone software product evaluations, testing, and corrective action as required. (Contrast with *software product evaluation*.)

Software system. A system consisting solely of software and possibly the computer equipment on which the software operates.

Software team. The group that engineers software products (including new development, modification, reuse, reengineering, maintenance, or any other activity that results in software products).

Software test environment. The facilities, hardware, software, firmware, procedures, and documentation needed to perform qualification, and possibly other, testing of software. Elements may include but are not limited to simulators, code analyzers, test plan generators, and path analyzers, and may also include elements used in the software engineering environment.

Software transition. The set of activities that enables responsibility for software engineering to pass from one organization, usually the organization that performs initial software development, to another, usually the organization that will perform sustaining engineering.

Software unit. Smallest physical element of software processed by source code translators, compilers, and assemblers.

Standard. Written criteria used to develop and evaluate a product or to provide and evaluate a service. (Contrast with *policy, procedure*.)

Sustaining engineering. Maintenance and enhancement of an operational product. (See *maintenance, enhancement*.)

System. Operational entity composed of a set of interrelated and cohesive configuration items (CIs). (See *configuration item (CI)*.)

Technology change management. "... [I]dentifying, selecting, and evaluating new technologies, and incorporating effective technologies into the organization. The objective is to improve software quality, increase productivity, and decrease cycle time for product engineering." (Reference 22)

Technology infusion. (Used synonymously with technology transfer; see *technology transfer*.)

Technology management. (See *technology change management*.)

Technology transfer. The successful importing or exporting of technology from lab to practice or from practice to practice.

Technology utilization. The application and integration of appropriate technologies in software efforts.

Timebox. A subdivision of a release created to achieve a unit of production that is manageable in size, complexity, staffing, and duration. It is a planning construct that controls functionality delivered by fixing resources and time.

Transported unit. Existing unit that is used verbatim (except for possible changes to the development history for traceability). Its origin is usually external to the project. (Contrast with *new unit, adapted unit, converted unit*.)

Unit. (See *software unit*.)

Validation. "The process of evaluating software during or at the end of the software development process to determine whether it satisfies specified requirements." (Reference 9) (Contrast with *verification*.)

Verification. “The process of evaluating software to determine whether the products of a given development phase [or activity] satisfy the conditions imposed at the start of that phase [or activity].” (Reference 9) (Contrast with *validation*.)

Walkthrough. Detailed technical presentation of a limited aspect or portion of a product. These presentations are usually made by the product’s developer. (Contrast with *inspection*.)

Appendix B. Building for Reuse

Reuse can apply to people, code, processes, requirements, and design, as well as to plans, standards, and test scripts. Although a recent study of NASA software (Reference 1) indicated that most reuse at the Agency focuses solely on code, reusing any of these resources in the development of a new system is appropriate when the result will be improved reliability, productivity, and maintainability without a negative impact on performance or requirements satisfaction. Planning for reuse maximizes its benefits; therefore, design software projects for reuse from the outset. During the requirements definition and analysis and design phases, for example, developers need to identify potentially reusable architectures, designs, code, and approaches. To facilitate this process, provide developers with the structure and tools that will assist them in finding and reusing existing components and architectures rather than developing new software from scratch. Several activities can be performed throughout the life cycle to enable reuse:

- Perform *domain analysis* by examining the application domain to identify common requirements and functions. The result is a standard, general architecture or model that incorporates the common functions of a specific applications area and can be tailored to accommodate differences among individual projects.
- Domain analysis enables *requirements generalization*, which involves preparing requirements and specifications so that they cover a selected family of projects or missions.
- During the design phase, explicitly *design for reuse* by providing modularity, standard interfaces, and parameterization.
- Place reusable source code in a *reuse library*, along with the associated requirements, specifications, design documentation, and test data. Such a library should also contain a search facility that enables varied access to the software (for example, by keyword or by name).
- A final activity that enables reuse on future projects involves the application of *reuse preservation* techniques during the maintenance and operations phase. The maintenance team should avoid making quick fixes, which can negatively affect the reusability of the system. Rather, they should apply many of the same design practices that promote reuse during software development. Alternatively, a separate maintenance team might be established to maintain only the software in a reusable software library. The team would notify projects that incorporate software from the library of any changes to that software.

Systems employing a high percentage of reused software typically still undergo each phase of the full development life cycle, but certain reviews and documents may be consolidated and phase schedules collapsed or overlapped (see Figure Appendix B. –1). Reusing design, documentation, and code written for a previous project (and perhaps adapting it to some degree for use in the system to be developed) requires less effort than creating entirely new products.

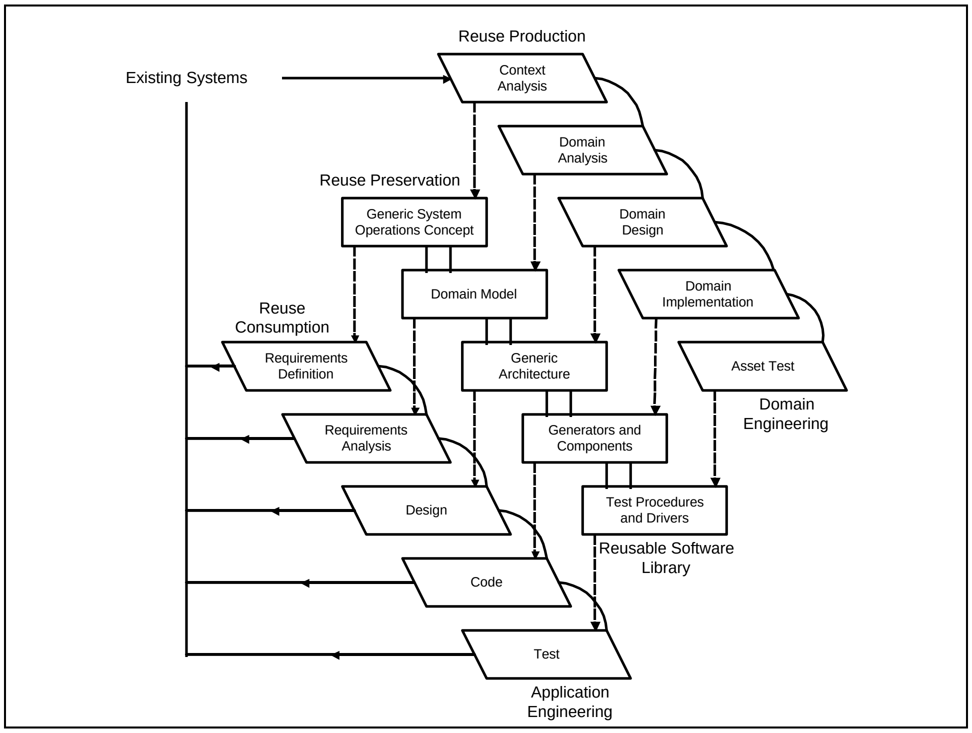

Figure Appendix B. –1. High-Reuse Life-Cycle Model

Appendix C. COTS, GOTS, Reused, and Other NDI Software Products

This appendix provides guidance to the software manager for satisfying the requirements of this guidebook when applied to incorporating COTS, GOTS, reused, and other NDI products.

Appendix C. .1 COTS Software Products

Using COTS products involves cultural implications. For example, in making a decision to incorporate COTS software, stakeholders must first be willing to accept the capabilities it provides. Although it is possible to negotiate with a vendor to obtain and modify the source code, in so doing vendor support will be lost, and the system will be inconsistent with future product releases.

When considering the use of a COTS product, conduct the following activities during the activities indicated:

- Prototype during requirements definition and analysis to evaluate the product's capabilities and performance. Whereas prototyping typically is conducted to clarify requirements or define a user interface, here it is performed to assess whether the product meets already defined functionality and performance requirements. Prototype to analyze the performance of the product, to assess its capabilities in relation to the requirements, and to evaluate its ease of use. Conduct this activity as early as possible in the life cycle, because the resulting decision on whether to use a COTS product may affect the overall system architecture. Defining the system architecture thus becomes an iterative step that evolves based on the results of the prototyping efforts, because only a certain requirement or set of requirements may be satisfied by COTS capabilities.
- As soon as prototyping results indicate that use of a COTS product is appropriate, plan for its interfaces to other COTS products or to the rest of the system. The input/output requirements of the COTS product must be carefully and completely defined and understood, so that its interfaces to other components, custom as well as COTS, can be created correctly. At this point, assess the following:
  - The effort required to develop the interface software versus the effort required to custom develop all the code
  - The maintainability of the COTS software
  - The reliability of the COTS software
  - The quality and reliability of vendor support (which will be crucial to successful implementation)

During design, address the connections to COTS products in the same way as other system interfaces. Because no design products exist for COTS software, devise artifacts that provide the traceability from COTS products to requirements.

During implementation, no code will be written for the COTS products themselves; however, testing becomes an issue because a COTS product cannot be tested at the unit level (that is, as in unit testing of custom software). The test plan must thus focus on the interfaces to COTS products. Apply additional rigor in analyzing the test output. Do not assume that less testing effort will be required for a COTS product; rather, the effort typically applied at the unit and module levels will simply be applied to testing at a higher level for COTS interfaces. If test results indicate that a COTS portion of the system is not performing as required, obtain vendor support in troubleshooting and correcting the problem.

Throughout the development cycle, reassess the selection of a COTS product when requirements changes affect a part of the architecture that is satisfied by that product.

Continually stay abreast of and reassess new versions of COTS products. Conduct additional prototyping to evaluate their use and enable the project to take advantage of new capabilities or to recognize possible changes in system architecture. However, at some point in the life cycle, it will be necessary to finalize the version selection and proceed with development, regardless of new capabilities that may become available. If using a multi-release approach, allow flexibility in the design to facilitate the upgrade of COTS products assigned to later releases.

During operations, weigh the need for capabilities provided by new versions of the COTS software against a possible lack of vendor support for older versions. For operational systems with multi-site installations, apply rigorous configuration management to ensure version control.

Managing a project that comprises predominantly COTS products is similar to managing a custom-development project. All of the same good management practices apply here as well. However, package vendors are a source of risk that needs special attention. When a project selects a package to be part of a system, it is buying a long-term relationship with a vendor. Understanding a vendor's financial stability, track record, and long-term strategy can be as important as understanding the vendor’s product. Not only must the project develop a partnership with its vendors, but it must also develop a partnership with the contracting and procurement side of its own organization. Software project managers must acquire a level of business understanding well beyond that needed in conventional development to successfully manage the project.

This life-cycle concept requires the formation of a well-integrated team consisting of end-users, domain experts, software engineers, independent testers, a system administrator, and a procurement official, all coordinated by a project leader. The team roles must remain intact for the entire system life cycle, and team members must be empowered to make decisions as to which system requirements are critical. Many of the traditional roles change slightly or require different skills. In addition, the project team includes two new roles, the system administrator and a procurement official, who play significant roles in comparison to the minimal support needed in these areas on traditional software development projects. A capable and interested procurement official and systems administrator will be great assets to the project. If possible, arrangements should be made to have a single procurement official and a system administrator assigned to the project. Every effort should be made to make them aware of project needs and to welcome them as members of the team.

The primary project roles are as follows:

End-User. The end-users are the people who will be using the system operationally. They have a clear understanding of the requirements and the operational environment and are empowered to negotiate requirements changes and represent the end-user organization.

Domain expert. Domain experts have extensive experience in the problem domain and are aware of existing packages that are available within the domain. They have experience with, or are at least aware of, other package-based systems within the domain from which architectures can be reused.

Software engineer. Software engineers or developers are responsible for developing glueware and integrating the packages. They are responsible for engineering a solution that meets the customer's and end-users' quality expectations. It is best if the software engineers have some experience with COTS integration or have specific experience with the packages being used.

Independent tester. Independent testers are responsible for verifying that the system meets its requirements. Experience with the application domain, incremental testing, and black-box testing is helpful.

Procurement officer. Procurement officers are responsible for obtaining demonstration copies for evaluation; purchasing selected products and negotiating for extensions of demonstration copies until official receipt of product; monitoring and extending license expiration dates. They are responsible for keeping the project point-of-contact informed of expected product arrival dates and the terms of the contracts.

System administrator. System administrators are responsible for installing COTS products as they are received and setting up accounts as they are needed. They can also help troubleshoot problems with hardware/package compatibility. It is critical that a system administrator be available to provide services immediately upon request, so it is best if one is dedicated to the project.

Appendix C. .2 Evaluating COTS, GOTS, Reused, and Other NDI Software Products

The software manager specifies in the software plan the criteria for evaluating COTS, GOTS, reused, and other NDI software products for use in fulfilling software requirements. General criteria are the software product's ability to meet specified requirements and cost-effectiveness over the life of the system. Examples of specific criteria include but are not limited to the following:

- Ability to provide required capabilities and meet required constraints
- Ability to provide required safety, security, and privacy
- Reliability and maturity, as evidenced by an established track record
- Testability
- Interoperability with other system and system-external elements

- Fielding issues, including:
  - Restrictions on copying and distributing the software or documentation
  - License or other fees applicable to each copy
- Maintainability
  - Likelihood the software product will need to be changed
  - Feasibility of accomplishing that change
  - Availability and quality of documentation and source files
  - Likelihood that the supplier will continue to support the current version
  - Impact on the system if the current version is not supported
  - The customer’s data rights to the software product

    Warranties available
- Short- and long-term cost impacts of using the software product
- Technical, cost, and schedule risks and tradeoffs in using the software product

Appendix C. .3 Guidelines for Performing Required Activities Involving COTS, GOTS, Reused, and Other NDI Software Products

The following guidelines are provided to help interpret and satisfy the requirements to perform life-cycle activities:

- Any software product required by this document may be a COTS, GOTS, reused, or other NDI software product as long as it meets the criteria established in the software plan. The software product may be used as is or modified.
- When COTS, GOTS, reused, or other NDI software has been selected to be incorporated into the delivered software product, some requirements in this document must have special interpretation. Table Appendix C. –1 provides this interpretation. Key issues are whether the software will be modified, whether the unmodified software constitutes an entire software CI or only one or more software units, and whether the unmodified software has a positive performance record. The table is presented in a conditional manner: If an activity in the left column is required for a given type of software, the table tells how to interpret the activity for COTS, GOTS, reused, or other NDI software of that type.

Table Appendix C. –1. Guidelines for Using COTS, GOTS, Reused, and Other NDI Software Products

(1 of 2)

Required Activity	**Interpretation**				
	Software CIs To Be Used Unmodified		**Software Components To Be Used Unmodified**		**Software Components Being Modified for or During the Project**
	Positive Performance Record	**No or Poor Performance Record**	**Positive Performance Record**	**No or Poor Performance Record**	
Software project planning	Include the activities in this table in project plans.				
Software CI requirements definition and analysis	Specify the project-specific requirements the software CI must meet; verify through records or retest that the software CI can meet them.		Consider the component’s capabilities and characteristics in specifying the requirements for the software CI of which it is a part.		
Software CI-wide design	No requirement: the software CI-wide design decisions have already been made.		Consider the component’s capabilities and characteristics in designing software CI behavior and making other software CI-wide design decisions.		
Software CI architectural design	No requirement: the software CI’s architecture is already defined.		Include the component in the software CI architecture and allocate software CI requirements to it.		
Software CI detailed design	No requirement: the software CI’s detailed design is already defined.		No requirement: the component is already designed.		Modify the component’s design as needed.
Software implementation	No requirement: the software for the software CI’s components is already implemented.		No requirement: the software for the component is already implemented.		Modify the software for the component.
Unit testing	No requirement: the software CI’s units are already tested.	Perform selectively if in question and units are accessible.	No requirement: the unit is already tested.	Perform this testing.	
Integration and testing	No requirement: the software CI’s components are already integrated.	Perform selectively if in question and components are accessible.	Perform except where integration is already tested or proven.	Perform this testing.	
Software CI qualification testing	No requirement: software CI is already tested and proven.	Perform this testing.	Include the component in software CI qualification testing.		

Table Appendix C. –1. Guidelines for Using COTS, GOTS, Reused, and Other NDI Software Products

(2 of 2)

Required Activity	**Interpretation**				
	Software CIs To Be Used Unmodified		**Software Components To Be Used Unmodified**		**Software Components Being Modified for or During the Project**
	Positive Performance Record	**No or Poor Performance Record**	**Positive Performance Record**	**No or Poor Performance Record**	
Preparation for Software Delivery	Include the software for the software CI or component in the executable software; prepare source files for the software CI or component, if available; include version descriptions; handle any license issues; prepare or provide as-built design descriptions for software whose design is known; cover use of the software CI or component, as appropriate, through existing, new, or revised user or operator manuals; install the software CI or component at the support site; demonstrate regenerability if source is available; include the training offered.				
Software project close-out	Apply to activities performed and software products prepared, modified, or used in incorporating this software.				
Software product V&V	Apply to software products prepared or modified in incorporating this software; for software products used unchanged, apply unless a positive performance record or evidence of past evaluations indicates that such an V&V would be duplicative.				
Software configuration management	Apply to all software products prepared, modified, or used in incorporating this software.				
Software quality assurance	Apply to all activities performed and all software products prepared, modified, or used in incorporating this software.				
Milestone reviews	Cover the software products prepared or modified in incorporating this software; explicitly discuss COTS, GOTS, reused, and other NDI products.				
Software process improvement	Apply to all activities performed in engineering this software.				
System requirements analysis	Consider software's capabilities in defining the system and operations concept and system requirements.				
	Use test or performance records to confirm ability to meet needs.	Test to confirm ability to meet needs.	Use test or performance records to confirm ability to meet needs.	Test to confirm ability to meet needs.	Use tests or records to determine potential to meet needs.
System-wide design	Consider the software's capabilities and characteristics in designing system behavior and in making other system-wide design decisions.				
System architectural design	Include the software CI in the system architecture; allocate system requirements to it.		Consider the component's capabilities and characteristics in designating software CIs and allocating system requirements to them.		
Software CI and hardware CI integration and testing	Perform, except where integration is already tested or proven.	Include the software CI in software CI and hardware CI integration and testing.	Include the component in software CI and hardware CI integration and testing.		
System qualification testing	Include the software CI in system qualification testing.		Include the component in system qualification testing.		

Appendix D. System-Level Considerations

When the software CI is part of a larger hardware-software system for which the organization has system-level responsibilities, a number of additional considerations must be taken into account. In the following paragraphs regarding system-level activities, if the software covered by this document is part of a hardware-software system for which this document covers only the software portion, *participate* means take part in, as described in the software plan. If the software (and the computers on which it executes) is considered to constitute a system, *participate* means be responsible for.

Appendix D. .1 System Requirements Analysis

The software requirements analysts participate in system requirements analysis in accordance with the requirements discussed in the subsections that follow.

Analysis of User Input

The software requirements analysts participate in analyzing user input provided by the customer to gain an understanding of user needs. This input may take the form of need statements, surveys, problem reports and change requests, feedback on prototypes, interviews, or other user input or feedback. This input is used to formulate the system and operations concept and the system requirements.

System and Operations Concept

The software requirements analysts participate in defining and recording the operational concept for the system. The result includes all applicable items in the system and operations concept documentation standard, including the preparation of any required operational scenarios.

System Requirements

The software requirements analysts participate in defining and recording the requirements to be met by the system and the methods to be used to ensure that each requirement is met. The result includes all applicable items in the system requirements specification (SRS) documentation standard.

If a system consists of subsystems (or CIs), the activity in this subsection is intended to be performed iteratively with the system design activities to define system requirements, design the system and identify its subsystems, define the requirements for and interfaces among those subsystems, design the subsystems, identify their components, and so on.

Appendix D. .2 System Design

The software requirements analysts and software design architects participate in system design in accordance with the requirements discussed in the subsections that follow.

System-Wide Design Decisions

The software requirements analysts and software design architects participate in defining and recording system-wide design decisions (that is, decisions about the system's behavioral design and other decisions that affect the selection and design of system components). The result includes all applicable items in the system-wide design section of the system design specification (SDS) documentation standard.

Design decisions remain at the discretion of the software requirements analysts and software design architects unless formally converted to requirements. The software team is responsible for fulfilling all requirements and demonstrating this fulfillment through qualification testing. Design decisions act as software team-internal "requirements," to be implemented, imposed on contractors (if applicable), and confirmed by software team-internal testing; but their fulfillment need not be demonstrated to the customer.

System Architectural Design

The software requirements analysts and software design architects participate in defining and recording the architectural design of the system (identifying the components of the system, their interfaces, and a concept of execution among them) and the traceability between the system components and system requirements. The result includes all applicable items in the architectural design and traceability sections of the SDS documentation standard.

Appendix D. .3 Software CI and Hardware CI Integration and Testing

Software CI and hardware CI integration and testing means integrating software CIs with interfacing hardware CIs and software CIs, testing the resulting groupings to determine whether they work together as intended, and continuing this process until all software CIs and hardware CIs in the system are integrated and tested. The software qualification testers participate in developing and recording test plans (in terms of inputs, expected results, and V&V criteria), test procedures, and test data for conducting software CI and hardware CI integration and testing. The test plans cover all aspects of the system-wide and system architectural design. The software qualification testers participate in software CI and hardware CI integration and testing in accordance with the software CI and hardware CI integration test plans and procedures. The software team participates in analyzing the results of software CI and hardware CI integration and testing. Software-related analysis and test results are recorded in appropriate product V&V records files. The software team makes necessary revisions to the software, participates in retesting, and updates other software products as needed, based on the results of software CI and hardware CI integration and testing.

Appendix D. .4 System Qualification Testing

System qualification testing is performed to demonstrate (often to the customer) that system requirements have been met. It covers the SRS. This testing contrasts with software team-internal system testing, performed as the final stage of software CI and hardware CI integration and testing.

The persons responsible for fulfilling the requirements in this section are not the persons who performed detailed design or implementation of software in the system, although those persons may participate, for example, by contributing test plans that rely on knowledge of the system's internal implementation.

The software qualification testers participate in developing and recording the test preparations, test plans, and test procedures to be used for system qualification testing and the traceability between the test plans and the system requirements. For software systems, the results include all applicable items in the software CI qualification test plan documentation standard. The software qualification testers participate in preparing the test data needed to carry out the test plans and in providing the customer advance notice of the time and location of system qualification testing. They participate in system qualification testing in accordance with the system test plans and procedures. The software team participates in analyzing and recording the results of system qualification testing. For software systems, the result includes all applicable items in the software CI qualification test report documentation standard. The software team makes necessary revisions to the software, provides the customer advance notice of retesting, participates in retesting, and updates other software products as needed, based on the results of system qualification testing.

Abbreviations and Acronyms

AT	acceptance test *or* testing
ATRR	acceptance test readiness review
BDR	build design review
BQT	build qualification testing
CASE	computer-aided software engineering
CDR	critical design review
CI	configuration item
CM	configuration management
COCOMO	Constructive Cost Model
COTS	commercial-off-the-shelf
CCB	configuration control board
DFD	data flow diagram
DR	discrepancy report
FCA	functional configuration audit
FQT	formal qualification testing
GOTS	government-off-the-shelf
GSFC	Goddard Space Flight Center
HQ	headquarters
IDR	internal DR
IRM	Information Resources Management
IV&V	independent validation and verification
JAD	joint application development
JPL	Jet Propulsion Laboratory
MSFC	Marshall Space Flight Center
NASA	National Aeronautics and Space Administration
NDI	non-developed item
NMI	NASA Management Instruction
O&M	operations and maintenance
ORR	operational readiness review

OSMA	Office of Safety and Mission Assurance
PAL	process asset library
PCA	physical configuration audit
PDR	preliminary design review
QA	quality assurance
QTRR	qualification test readiness review
RCR	release contents review
RDR	release design review
RQTRR	release qualification test readiness review
RRR	release requirements review
SCM	software configuration management
SCR	system concept review
SDR	system design review
SDS	system design specification
SEI	Software Engineering Institute
SEL	Software Engineering Laboratory
SPR	system *or* software problem report
SQA	software quality assurance
SRR	system requirements review
SRS	system requirements specification
SSR	software specification review
STR	system *or* software trouble report
SWDS	software design specification
SWG	Software Working Group
SWRS	software requirements specification
TBD	to be determined
TPM	Technical Performance Measurement
V&V	validation and verification (*also* validate and verify)

References

1. *Profile of Software at the National Aeronautics and Space Administration (NASA),* Software Engineering Program, NASA-RPT-004-95, March 1995.

2. Jeletic, K, R. Pajerski, C. Brown, *Software Process Improvement Guidebook,* Software Engineering Program, NASA-GB-001-95, January 1996.

3. *NASA Software Strategic Plan,* NASA Software Program, Fairmont, West Virginia, July 1995.

4. MIL-STD-498, *Software Development and Documentation*, Department of Defense, December 5, 1994.

5. Bassman, M., F. McGarry, R. Pajerski, *Software Measurement Guidebook,* Software Engineering Program, NASA-GB-001-94, August 1995. Also published as SEL-94-102, Software Engineering Laboratory, NASA/GSFC, June 1995.

6. Reusable Software Management Plan (SMP) and On-line Help Tool, Software Assurance Technology Center, NASA/GSFC, http://satc.gsfc.nasa.gov/Documents/smp/ smppage.html.

7. Landis, L., F. McGarry, S. Waligora, et al., *Manager's Handbook for Software Development (Revision 1),* SEL-84-101, Software Engineering Laboratory, NASA/GSFC, November 1990, http://fdd.gsfc.nasa.gov/mgr_hand/mnghnbk.html.

8. Alberts, C. J., et al., *Continuous Risk Management Guidebook* (DRAFT version 0.3), Software Engineering Institute, Carnegie Mellon University, January 1996.

9. ANSI/IEEE-STD-610.12-1990, "IEEE Standard Glossary of Software Engineering

10. Landis, L., S. Waligora, F. McGarry, et al., *Recommended Approach to Software Development (Revision 3),* SEL-81-305, Software Engineering Laboratory, NASA/GSFC, June 1992.

11. Boehm, B., "A Spiral Model of Software Development and Enhancement," *IEEE Computer*, May 1988.

12. Waligora,. S., *SEL Package-Based System Development Process*, Software Engineering Laboratory, NASA/GSFC, February 1996, http://fdd.gsfc.nasa.gov/cotsweb.pdf.

13. *NASA Software Formal Inspections Guidebook*, NASA-GB-A302, August 1993, accessible from http://www.ivv.nasa.gov/SWG/.

14. Weller, E., "Lessons from Three Years of Inspection Data," *IEEE Software*, September 1993.

15. Currit, P. A., M. Dyer, and H. D. Mills, "Certifying the Reliability of Software," *IEEE Transactions on Software Engineering*, Vol. SE-12, No. 1, January 1986, pp. 3–11.

16. Basili, V, and S. Green, "Software Process Evolution at the SEL," *IEEE Software*, Vol. 11, No. 4, July 1994, pp. 58–66.

17. Software Engineering Evaluation System Technical Assessment Procedures and Workshops, NASA Headquarters, Office of Safety and Mission Assurance, U.S. Army Missile Command, Redstone Arsenal, Alabama, 1994.

18. NASA Management Instruction 7120.4, "Management of Major System Programs and

19. NASA Handbook 7120.5, "Management of Major System Programs and Projects," November 1993.

20. Boehm, B. *Software Engineering Economics*, Prentice-Hall, Inc., Englewood Cliffs, NJ, 1981.

21. Condon, S., M. Regardie, M. Stark, and S. Waligora, *Cost and Schedule Estimation Report*, SEL-93-002, NASA/GSFC, November 1993.

22. Paulk, M, et al., *Key Practices of the Capability Maturity Model, Version 1.1*, Software Engineering Institute, Carnegie Mellon University, CMU/SEI-93-TR-25, February 1993.