JPL D-15378
Revision D

THE JPL SOFTWARE DEVELOPMENT PROCESS DESCRIPTION

Prepared by: Milton L. Lavin and Jeanne S. Makihara

JPL

Jet Propulsion Laboratory
California Institute of Technology

CHANGE HISTORY

Revision D
Effective date: November 15, 1999

- Addition of an abstract that contains bolded requirements and a brief explanation of how they are to be used. There are no changes to the requirements documented in Version 3.

- Addition of a document map, accessible from each page.

Revision C
Effective Date: October 5, 1998

- Separation of guidance from requirements.

- Expanded references to other JPL processes.

- Description of applicability of SDPD to R&D tasks.

- Integration of requirements levied by *NASA Policy Directive (NPD) 7120.4A* and *NASA Procedures and Guidelines (NPG) 7120.5A*, as well as guidelines from *NASA Technical Standard (NTS) 8719.13A* and *NASA Policy Directive (NPD) 2820.1* -- primarily in regard to risk assessment, safety, and metrics.

- Removal of Rules, Practices and Conventions section, with the guidance distributed elsewhere in the document, such as in the Implementation section.

- Revision of Appendix C to trace ISO 9001 requirements to specific SDPD requirements.

- Numerous changes throughout the document, but primarily in requirements and guidance in Sections 5 and 6, to ensure responsiveness to ISO 9001, to clarify ambiguities, and to improve readability.

Revision B
Effective Date: December 19, 1997

Addition of a development plan template, a cost estimation section, a process overview section, and minor changes throughout the document.

Revision A
Publication Date: April 29, 1997 (never issued as an official document)

First complete draft containing an elaboration of life cycle activities and support activities, and emphasizing tailoring individual efforts via a software development plan.

SOURCE: Design, Build, Assemble, and Test (DBAT) Policy

TABLE OF CONTENTS

Appendices

ABSTRACT OF REQUIREMENTS

This description of JPL's *Software Development Process Description* (SDPD) contains general requirements, guidelines, and suggestions for defining and managing a software-intensive project[1], project element, or task consistent with *ISO 9001*, as interpreted by *ISO 9000-3, "Quality Management and Quality Assurance Standards — Part 3: Guidelines for the Application of 9001 to the Development, Maintenance, and Supply of Software."* The SDPD is consistent with *NASA Policy Directive (NPD) 2820.1, NASA Software Policies*, which identifies compliance with *ISO 9001* (as described in *9000-3*) as acceptable evidence that this policy directive has been implemented.

A. Scope and Objectives - Applicability

The primary focus of the SDPD is the development of software supplied to both internal and external customers in conjunction with the design and implementation of missions, spacecraft, instruments, and ground systems for JPL's NASA sponsor. However, the methodology can be applied to work for reimbursable sponsors and to development of software used in JPL's institutional and business systems infrastructure.

The SDPD methodology also applies to firmware up to the point where testing in a simulated hardware environment is complete. At that point, the development of firmware is defined by the Electronic System Development sub-process of Design, Build, Assemble, and Test (DBAT).

The SDPD is intended to:

- Promote the use of comparable development practices across the Laboratory within broad classes of software,
- Establish a baseline for continual improvement of JPL's software development processes, and
- Identify requirements for compliance with JPL's implementation of *ISO 9001*.

B. Software Classes

Documentation, reviews, and critical development activities are identified for the first three classes of software identified below; non-deliverable software (Class D) is not addressed:

Class A: Mission-Critical: Flight or ground software that is necessary either to assure mission success, or if it does not function as specified, that could cause loss of spacecraft, seriously degrade the attainment of primary mission objectives, or cause injury to humans or flight hardware. Examples of serious degradation of mission objectives include loss of a mission-critical event, loss of science return from multiple instruments, or loss of a large fraction of the engineering telemetry data.

[1] Future uses of the term "project" are intended to encompass project elements and tasks, even if not explicitly stated.

Class B: Mission Support: Flight or ground software that is necessary for the science return from a single (non-critical) instrument, or supports the timely generation of mission sequences, or is used to process or analyze mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems for which potential work-arounds exist. Examples of Class B include software that supports pre-launch integration and test, mission data processing and analysis, analysis software used in trend analysis and calibration of flight engineering parameters, or software employed by the Network Operations and Control Center (which is redundant with systems used at the tracking complexes). Class B software must be developed carefully, but validation and verification effort is generally less intensive than for Class A.

Class C: Development Support: Software developed to explore a design concept; or support software development functions such as requirements management, design, test and integration, configuration management, documentation, etc.; or perform engineering data analysis. A defect in Class C software may cause rework but has no direct impact on mission objectives or system safety. Class C software is often used by several people in addition to the developer(s), and by its nature, can impact the quality of delivered products. Documentation and review of Class C software should be tailored to its intended use, with attention to long-term maintenance needs and to evolution of a design prototype into operational flight or ground software. Note: **Development tools that can introduce critical defects in Class A or B software must be regarded as belonging to the same class as the operational software.**

Class D: Non-deliverable software developed to meet a research objective or support individual engineering efforts. Generally, Class D software is intended for use only by the individual who developed it.

Although Class D software is excepted from the requirements documented in the SDPD, some of the SDPD requirements may be useful in defining and implementing this class of software. A project/task can encompass software that falls in more than one category. In such a case, it is the responsibility of the project/task manager to identify the software elements that fall in each class, and to tailor the development plan accordingly.

C. Applicability to R&D Tasks

Requirements on R&D software tasks are less stringent than those defined in this SDPD. While managers of R&D tasks can choose to employ all or part of the SDPD requirements to their areas, they are actually subject to the requirements specified either under the Generate Scientific Knowledge (GSK) domain, or the Develop Needed Technology (DNT) domain. At the time of this writing, a Technology Development process has been identified within the DNT domain. R&D task managers should refer to *http://dmie/* for the policies and procedures associated with this process.

In some cases, there are tasks identified as R&D that are not strictly so, but instead have the characteristics of a definition phase and essentially function as a precursor to an actual development project/task. Typically, prototypes or demonstration software is produced. In some cases, the prototype is ultimately used operationally rather than being abandoned. Thus, **where**

the software development effort is not strictly R&D, and there is strong potential that the product will be used operationally, the requirements in this SDPD apply.

D. Intended Use and Compliance

The SDPD is intended to provide guidance in planning and managing a software development effort up to the point where it is delivered to either an internal or external customer. Maintenance is also addressed to cover those cases where the contract with the customer calls for fixing defects and making enhancements post-delivery.

The SDPD is but one source of requirements on the management of a software development task, with other requirements being levied by program office directives, and the policies and procedures of the engineering organization responsible for the development. The resulting development plan documents how a particular project or task elects to respond to all such requirements. Thus, a development plan is effectively a tailored version of this process description. Note that since a project/task may encompass all four classes of software, it is the project/task manager's responsibility to 1) identify the class appropriate to each software element, and 2) tailor the development procedures, documentation, and reviews, accordingly. Although **no explicit waiver is required**, any deviations from the requirements specified in this SDPD must be adequately explained in the development plan, as required in Section 5.4.

Ultimately, the responsibility for implementing the requirements of the SDPD and related documentation rests with the JPL project or task manager. It is the manager's responsibility to tailor the requirements of the SDPD, plus relevant domain, program office, and line organization standards and policies, to the needs of the specific development job, the result being documented in project/task plans. Evidence that this responsibility has been met is contained in the suite of plans, product descriptions (e.g., requirements document, operations concept), and quality records produced during development. It must be emphasized that considerable flexibility is afforded in documentation, the essential idea being that all documentation should be directly useful to developing a high quality end product within the allocated budget.

E. Synopsis of SDPD Requirements

Detailed requirements on software classes A, B, and C are extracted from the main text and organized into Life Cycle Activities and Supporting Activities. Requirements are denoted by "shall" statements, and are distinguished by a ***different font (Arial) in bold italics***. Numbering of requirements has been preserved to facilitate reference to the full text. The essence of these detailed requirements may be summarized concisely:

- Written requirements and interface specifications, under configuration management.
- A development plan, tailored to the complexity of the project/task. This plan must have a description of the product, a task breakdown, a schedule, an estimate of development effort, and a staffing plan.
- A test plan together with documented test cases and procedures.

- Reviews of critical intermediate products — both code and documentation — as described in the development plan.

.

- A documented design; graphics are recommended.
- Configuration management of documents and code, as described in the development plan (or a separate configuration management plan).
- Independent system-level test with documented anomaly reports and change requests.
- Documentation to support the end user, system operator, and the maintainer. Embedded documentation is recommended.

5. Requirements — Life Cycle Activities

5.1 Software Methodology

A software development project/task shall be organized according to a life-cycle model that is described in the development plan in terms of: *[5.1]*

- ***Phases, along with milestones and activities to be performed during each phase;*** *[5.1a]*
- ***Phase outputs, including any documentation;*** *[5.1b]*
- ***Verification activities (e.g., reviews, demonstrations, tests) by phase.*** *[5.1c]*

5.2 Contract Review

The organization that has overall responsibility for a software development effort shall establish and maintain documented procedures for a commitment review of the development plan or proposal to ensure: *[5.2.1]*

- ***Scope of work for the current delivery is adequately defined and documented.*** *[5.2.1a]*
- ***Differences between the scope of work defined in the development plan, and that requested by the customer, are resolved.*** *[5.2.1b]*
- ***Responsibilities of the customer are identified.*** *[5.2.1c]*
- ***Mutually acceptable means have been defined for dealing with changes in requirements during development, as well as correction of post-delivery defects.*** *[5.2.1d]*
- ***The resources and schedule described in the development plan are adequate to accomplish the contractual deliverables.*** *[5.2.1e]*

A record of commitment review findings shall be maintained as part of the project/task quality record. *[5.2.2]*

5.3 Customer's Requirements Specification

The developer shall have a written set of software requirements that are sufficient to satisfy customer and/or user needs. *[5.3.1]*

Interfaces between the software product and external software or hardware items shall be specified, either directly or by reference. *[5.3.2]*

Software requirements, whether provided by the customer or formulated by the developer, shall be reviewed to ensure that: *[5.3.3]*

- ***the product is adequately defined,*** *[5.3.3a]*
- ***ambiguities and conflicting requirements have been resolved, and*** *[5.3.3b]*
- ***the requirements are stated so as to allow validation during product acceptance.*** *[5.3.3c]*

The software requirements specification shall be subject to change control procedures, once it is baselined (e.g., completion of document review, customer approval obtained). *[5.3.4]*

Approved changes in requirements shall be maintained as part of the project/task quality record. *[5.3.5]*

5.4 Development Planning

A development plan shall address the following: *[5.4.1]*

- ***Overall definition of the product, as in user needs addressed, deliverables, and critical functionality.*** *[5.4.1a]*
- ***Scope of development work to be performed, including management and supporting activities****. [5.4.1b]*
- ***Project life cycle, including:*** *[same as 5.1]*

  Phases, along with activities or milestones to be performed during each phase.

  Phase outputs, including any documentation

  Verification activities (e.g., reviews, demonstrations, tests) by phase.

- ***Project organization and technical interfaces: team structure; nature of project interfaces, both internal and external; roles and responsibilities, including customer responsibilities; use of subcontractors; and other crucial dependencies, such as critical equipment and facilities, and use of JPL support services.*** *[5.4.1c]*
- ***Project schedule.*** *[5.4.1d]*

- ***Risk assessment.*** *[5.4.1e]*

- ***Cost estimate/budget that summarizes the cost of the personnel and other resources required by the development.*** *[same as 6.10.1]*

- ***Staffing profile.*** *[5.4.1f]*

- ***Change control procedures for documenting, reviewing, approving, and communicating requirements changes to all affected parties.*** *[5.4.1g]*

- ***Change control procedures for documenting, reviewing, approving, and communicating design changes before their implementation.*** *[5.4.1h]*

- ***Review (or verification) policies and procedures that, at a minimum, address detailed technical reviews, and identify what is to be reviewed (including critical intermediate products) and when reviews are to be held.*** *[same as 6.9.1]*

- ***Procedures for verifying, storing, protecting, and maintaining items (e.g., software, data, hardware, specifications) supplied by the customer or designated third party.*** *[same as 6.7.1]*

- ***Procedures for verifying purchased or subcontracted products.*** *[same as 6.6.3]*

- ***Documentation plan and procedures.*** *[same as 6.2.1 and 6.2.2]*

- ***Scope and content of the training to be provided to project personnel.*** *[same as 6.8.1]*

- ***System administration plan, including approach to back-up/archiving, security, and virus protection.*** *[5.4.1i]*

- ***Definition of responsibility, and description of associated procedures, to identify and correct recurring problems in the development process.*** *[5.4.1j]*

- ***Metrics tailored to project needs, and the associated procedures for collecting, storing, and analyzing them.*** *[same as 6.4.1]*

- ***Planning of the following specific activities, including identification of any separate plans:*** *[5.4.1k]*

  Configuration management
  Integration and test
  Delivery and installation
  Maintenance

- ***Reuse strategy, if any, or identification of reusable elements — both those that can be adapted from previously implemented systems, and portions of the current application that will be designed for reuse.*** *[5.4.1l]*

- ***Identification of quality records, associated procedures, and retention times.*** *[same as 6.3.1 and 6.3.3]*

- ***Provisions for updating the plan as development proceeds.*** *[5.4.1m]*

- ***Explanations for any deviations made from SDPD requirements.*** *[5.4.1n]*

5.5 QUALITY PLANNING -- No Requirements.

5.6 DESIGN

Requirements and design activities shall be guided by a plan with milestones and detailed technical reviews tailored to the needs of each project/task. *[5.6.1]*

The design shall be documented and, prior to release, the resulting design documentation shall be reviewed to ensure that (a) the design meets the requirements and is responsive to acceptance criteria, (b) the design is verifiable, and (c) safety issues have been addressed. *[5.6.2]*

5.7 Implementation

Implementation activities shall be guided by one or more plans with milestones and detailed technical reviews tailored to the needs of each project/task. *[5.7.1]*

5.8 Testing and Validation

Software integration and testing shall be performed in accord with test planning and specification documentation that addresses: *[5.8.1]*

- ***Test requirements, which may be an elaboration of software requirements.*** *[5.8.1a]*

- ***Levels of testing required up to acceptance by the customer.*** *[5.8.1b]*

- ***Test cases, test procedures, test data and expected results.*** *[5.8.1c]*

- ***Method of documenting test status and results.*** *[5.8.1d]*

- ***Test environment, such as dedicated processors, test tools (purchased or developed), and user documentation.*** *[5.8.1e]*

- ***Approach for evaluating test tools, with respect to their ability to verify the product under test (e.g., through testing, published reviews).*** *[5.8.1f]*

- ***Procedures for correcting defects, including analyzing the cause of the defect, determining corrective action, and ensuring that the corrective action is taken.*** *[5.8.1g]*

Before delivery and acceptance by the customer, the developer shall validate the product under conditions similar to the user's application environment. *[5.8.2]*

Missing or deficient functionality (in light of customer/user expectations, based on a requirements document or other form of "contractual" document) shall be documented in a release description document or transfer agreement. *[5.8.3]*

Test records to be maintained as part of the project quality record shall include, at a minimum: *[5.8.4]*

- ***Anomaly reports (or problem/failure reports)*** *[5.8.4a]*
- ***Test tool checks, to evaluate whether the tools are capable of verifying the acceptability of the software product under development.*** *[5.8.4b]*
- ***Test results, with clear indications whether the product has passed or failed.*** *[5.8.4c]*

5.9 Delivery, Installation, and Acceptance

The activities comprising delivery, installation, and acceptance shall be defined in a plan or related documentation, that addresses: *[5.9.1]*

- ***Preparation of the acceptance test cases and acceptance criteria, with developer's responsibilities (if any) noted.*** *[5.9.1a]*
- ***Procedures to be used in documenting and resolving problems found following installation, whether during acceptance testing or delivery.*** *[5.9.1b]*
- ***Details of delivery and installation logistics, e.g., arranging for use of customer/user facilities and personnel in installation and test.*** *[5.9.1c]*
- ***Definition of developer's role (if any) in supporting transition to full operational use of the product.*** *[5.9.1d]*
- ***Identification of documentation to be delivered at installation, including installation and configuration procedures.*** *[5.9.1e]*
- ***Identification of training for the user and/or system administrator/operator.*** *[5.9.1f]*
- ***A schedule for key events pertaining to delivery, installation, and acceptance.*** *[5.9.1g]*
- ***Storage of archived software media to prevent deterioration and facilitate disaster recovery.*** *[5.9.1h]*
- ***Virus protection of software designated for delivery, during storage and electronic transmission.*** *[5.9.1i]*

After delivery, a baselined copy of the software and delivered documentation shall be archived. *[5.9.2]*

5.10 Maintenance

If the developer is tasked to perform maintenance, a maintenance plan shall be prepared, defining the scope of the activity and the developer's approach. *[5.10.1]*

If the developer is required to turn maintenance over to another organization, the development plan shall address the mechanism for transferring knowledge of the software to the maintainer. *[5.10.2]*

6. Requirements — Supporting Activities

6.1 Configuration Management

Configuration management procedures shall be documented and applied to deliverables: code, associated data files, and documentation. *[6.1.1]*

6.2 Documentation and Document Control

For each development effort, the development plan shall define: *[6.2.1]*

- ***Documents to be produced, e.g., document titles, form (web, file server, hard copy), content standards or guidelines to be followed.*** *[6.2.1a]*
- ***Procedures (including responsibilities) for producing, reviewing, approving, and controlling documents.*** *[6.2.1b]*

Documentation procedures shall address: *[6.2.2]*

- ***Which documents are subject to configuration management and at what point in the development cycle they are baselined.*** *[6.2.2a]*
- ***Preparation of a master document list, or equivalent control mechanism, to identify document status, and preclude the use of invalid or obsolete documents.*** *[6.2.2b]*
- ***Responsibility for approving and releasing documents, and promptly withdrawing obsolete documents from use.*** *[6.2.2c]*
- ***Identification of changes in released documents (to be done where practicable).*** *[6.2.2d]*
- ***Approach for ensuring that the master document list (or equivalent control mechanism), as well as pertinent versions of documents, are readily available.*** *[6.2.2e]*
- ***Directory/file permissions and back-up policies, where document control is achieved through electronic means.*** *[6.2.2f]*

6.3 QUALITY RECORDS

The development plan shall identify the pertinent quality records and describe procedures for collection, indexing, filing, storage, access, maintenance, and disposition of these records. *[6.3.1]*

Required quality records include the following: *[6.3.2]*

- ***Approved changes in requirements*** *[same as 5.3.7]*

- ***Review (or verification) results*** *[same as 5.2.2]*
- ***Anomaly reports*** *[same as 5.8.4a]*
- ***Checks of test tools, to evaluate whether the tools are capable of verifying the acceptability of the software product under development.*** *[same as 5.8.4b]*
- ***Test results, with clear indications whether the product has passed or failed.*** *[same as 5.8.4c]*
- ***Change requests/orders generated during development and — if provided for in the contract — after delivery.*** *[6.3.2a]*

The retention times for project/task quality records shall be established in the development plan in accord with program office directives, with particular attention to needs of post-delivery maintenance. *[6.3.3]*

Quality records shall be stored in an environment conducive to the prevention of deterioration and loss, and in a manner so as to be readily retrievable. *[6.3.4]*

Pertinent subcontractor quality records shall be identified in the subcontract. *[same as 6.6.2b]*

6.4 Measurement

Metrics, and the associated procedures for collecting, storing, and analyzing them, shall be identified in a development plan, and shall be tailored to project needs. *[6.4.1]*

6.5 Tools and Techniques -- No requirements.

6.6 Purchasing and Subcontracts

Purchase orders shall clearly describe the product or service ordered, and shall be reviewed for adequacy by the developer prior to release. *[6.6.1]*

A development subcontract shall address: *[6.6.2]*

- ***In-process verification of subcontracted development, via reviews of intermediate products and/or other oversight activities as appropriate.*** *[6.6.2a]*
- ***Identification of subcontractor quality records to be maintained.*** *[6.6.2b]*
- ***Criteria and/or procedures for accepting subcontracted software.*** *[6.6.2c]*

Upon receipt, the developer shall ensure that a product or service that is purchased/subcontracted, or provided by a separate development organization, conforms to specified requirements, in accordance with procedures defined in the development plan. *[6.6.3]*

6.7 CUSTOMER-SUPPLIED PRODUCT/REUSED SOFTWARE

The developer shall establish and document procedures for verification, storage, protection, and maintenance of items (e.g., software, data, hardware, specifications) supplied by the customer or designated third party. *[6.7.1]*

6.8 TRAINING

The scope and content of the training to be provided to project personnel (e.g., development team, user, maintainer) shall be addressed in the development plan. *[6.8.1]*

6.9 REVIEWS

The development plan shall define review (or verification) policies and procedures that, at a minimum, address detailed technical reviews, and identify what is to be reviewed (including critical intermediate products) and when reviews are to be held. *[6.9.1]*

Review (or verification) results shall be maintained as quality records, and shall include a summary of requests for action and the responses thereto. *[6.9.2]*

6.10 COST ESTIMATION

For each new development, or incremental development of an existing system, the developer shall prepare a documented cost estimate/budget that summarizes the cost of the personnel and other resources required by the development. *[6.10.1]*

The JPL Software Development Process Description

1. SCOPE AND OBJECTIVES - APPLICABILITY

This description of JPL's *Software Development Process Description* (SDPD) contains general requirements, guidelines, and suggestions for defining and managing a software-intensive project[2], project element, or task consistent with *ISO 9001*, as interpreted by *ISO 9000-3, "Quality Management and Quality Assurance Standards — Part 3: Guidelines for the Application of 9001 to the Development, Maintenance, and Supply of Software."* The SDPD is consistent with *NASA Policy Directive (NPD) 2820.1, NASA Software Policies*, which identifies compliance with *ISO 9001* (as described in *9000-3*) as acceptable evidence that this policy directive has been implemented.

The primary focus of the SDPD is the development of software supplied to both internal and external customers in conjunction with the design and implementation of missions, spacecraft, instruments, and ground systems for JPL's NASA sponsor. However, the methodology can be applied to work for reimbursable sponsors and to development of software used in JPL's institutional and business systems infrastructure.

The SDPD methodology also applies to firmware up to the point where testing in a simulated hardware environment is complete. At that point, the development of firmware is defined by the Electronic System Development sub-process of Design, Build, Assemble, and Test (DBAT).

The SDPD is intended to:

- Promote the use of comparable development practices across the Laboratory within broad classes of software,
- Establish a baseline for continual improvement of JPL's software development processes, and
- Identify requirements for compliance with JPL's implementation of *ISO 9001*.

Documentation, reviews, and critical development activities are identified for the first three classes of software identified below; non-deliverable software (Class D) is not addressed:

Class A: Mission-Critical: Flight or ground software that is necessary either to assure mission success, or if it does not function as specified, that could cause loss of spacecraft, seriously degrade the attainment of primary mission objectives, or cause injury to humans or flight hardware. Examples of serious degradation of mission objectives include loss of a mission-critical event, loss of science return from multiple instruments, or loss of a large fraction of the engineering telemetry data.

[2] Future uses of the term "project" are intended to encompass project elements and tasks, even if not explicitly stated.

Class B: Mission Support: Flight or ground software that is necessary for the science return from a single (non-critical) instrument, or supports the timely generation of mission sequences, or is used to process or analyze mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems for which potential work-arounds exist. Examples of Class B include software that supports pre-launch integration and test, mission data processing and analysis, analysis software used in trend analysis and calibration of flight engineering parameters, or software employed by the Network Operations and Control Center (which is redundant with systems used at the tracking complexes). Class B software must be developed carefully, but validation and verification effort is generally less intensive than for Class A.

Class C: Development Support: Software developed to explore a design concept; or support software development functions such as requirements management, design, test and integration, configuration management, documentation, etc.; or perform engineering data analysis. A defect in Class C software may cause rework but has no direct impact on mission objectives or system safety. Class C software is often used by several people in addition to the developer(s), and by its nature, can impact the quality of delivered products. Documentation and review of Class C software should be tailored to its intended use, with attention to long-term maintenance needs and to evolution of a design prototype into operational flight or ground software. Note: **Development tools that can introduce critical defects in Class A or B software must be regarded as belonging to the same class as the operational software.**

Class D: Non-deliverable software developed to meet a research objective or support individual engineering efforts. Generally, Class D software is intended for use only by the individual who developed it.

Although Class D software is excepted from the requirements documented in the SDPD, some of the SDPD requirements may be useful in defining and implementing this class of software. A project/task can encompass software that falls in more than one category. In such a case, it is the responsibility of the project/task manager to identify the software elements that fall in each class, and to tailor the development plan accordingly.

Sections 1–4 of the SDPD discuss how this document is to be used, describe the role of software development within the context of JPL's process-based organization, cite reference documents, define essential terms, state a quality policy for software development, and identify acceptable software development standards. The remainder of the document contains requirements and guidelines pertinent to the life-cycle activities and supporting activities identified in ISO 9000-3. A trace to the requirements of *ISO 9001* may be found in Appendix C.

1.1 INTENDED USE AND COMPLIANCE

The SDPD is intended to provide guidance in planning and managing a software development effort up to the point where it is delivered to either an internal or external customer. Maintenance is also addressed to cover those cases where the contract with the customer calls for fixing defects and making enhancements post-delivery.

The SDPD establishes requirements that are needed to comply with JPL's implementation of *ISO 9001* in the software development arena. Requirements are denoted by "shall" statements, and are distinguished by a ***different font (Arial) in bold italics***. All other statements are provided strictly as guidance and are typically set apart under a "Guidance" heading. One should not infer that such guidance is unimportant; rather in most cases, it is highly recommended, and should be given serious consideration.

The SDPD is but one source of requirements on the management of a software development task, with other requirements being levied by program office directives, and the policies and procedures of the engineering organization responsible for the development. The resulting development plan documents how a particular project or task elects to respond to all such requirements. Thus, a development plan is effectively a tailored version of this process description. Note that since a project/task may encompass all four classes of software, it is the project/task manager's responsibility to 1) identify the class appropriate to each software element, and 2) tailor the development procedures, documentation, and reviews, accordingly. Although **no explicit waiver is required**, any deviations from the requirements specified in this SDPD must be adequately explained in the development plan, as required in Section 5.4.

Evidence of compliance with each "shall statement" is, in most cases, documentation. "Documentation" means a retrievable written record. It is the prerogative of the project/task to select the media and the format that best suits project/task needs. For example, recording requirements in an electronic memorandum and the use of a CASE tool to document design are both acceptable. Detailed software standards identified in Section 2 contain topic outlines that are intended to help organize more formal documents. Again, these outlines should be tailored to the needs of individual projects/tasks. Appendix A provides a template for a development plan, which addresses ISO requirements. Recommended documentation for software classes A, B, and C is summarized in Appendix B.

1.2 APPLICABILITY TO R&D TASKS

Requirements on R&D software tasks are less stringent than those defined in this SDPD. While managers of R&D tasks can choose to employ all or part of the SDPD requirements to their areas, they are actually subject to the requirements specified either under the Generate Scientific Knowledge (GSK) domain, or the Develop Needed Technology (DNT) domain. At the time of this writing, a Technology Development process has been identified within the DNT domain. R&D task managers should refer to *http://dmie/* for the policies and procedures associated with this process.

In some cases, there are tasks identified as R&D that are not strictly so, but instead have the characteristics of a definition phase and essentially function as a precursor to an actual development project/task. Typically, prototypes or demonstration software is produced. In some cases, the prototype is ultimately used operationally rather than being abandoned. Thus, **where the software development effort is not strictly R&D, and there is strong potential that the product will be used operationally, the requirements in this SDPD apply.**

1.3 SYNOPSIS OF SDPD REQUIREMENTS

Detailed requirements on software classes A, B, and C may be found in Sections 5 and 6. The essence of these detailed requirements may be summarized concisely:

- Written requirements and interface specifications, under configuration management
- A development plan, tailored to the complexity of the project/task. This plan must have a description of the product, a task breakdown, a schedule, an estimate of development effort, and a staffing plan
- A test plan together with documented test cases and procedures.
- Reviews of critical intermediate products — both code and documentation — as described in the development plan.

.

- A documented design; graphics are recommended
- Configuration management of documents and code, as described in the development plan (or a separate configuration management plan)
- Independent system-level test with documented anomaly reports and change requests
- Documentation to support the end user, system operator, and the maintainer. Embedded documentation is recommended.

1.4 OVERVIEW OF SOFTWARE DEVELOPMENT PROCESS ACTIVITIES

The software development process exists to meet the needs of the customer, user, maintainer, JPL management, vendors, subcontractors, and other third parties, as depicted in Exhibit 1.1. The software development process, consists of the activities identified in Exhibit 1.2. Each of these activities is detailed in Sections 5 and 6. The development activities portrayed in the exhibit are related using the metaphor of a state chart. Transactions between activities are denoted by labeled arrows. To reduce complexity, only key transactions are identified explicitly.

For expository convenience, Exhibit 1.2 groups two sets of elemental activities into meta-activities of Project Definition and Development Planning. *Project Definition* consists of Cost Estimation (6.10), Customer Requirements Specification (5.3), Software Methodology (5.1), and Contract Review (5.2). *Development Planning* encompasses preparation of the development plan (5.4) and related plans, such as Integration & Test (5.8), and Configuration Management (6.1). Quality Planning (5.5) is regarded as an integral part of development planning. Other plans (not separately identified in Exhibit 1.2) may include a plan for Installation and Acceptance (5.9), and a plan for Maintenance (5.10) if JPL has that responsibility.

A development effort begins with a discussion of customer requirements that are subsequently formalized in a proposal. The identified project definition activities are essential to preparation of both the proposal and the development plan (5.4) that details how the requirements will be satisfied. Once requirements (5.3) have been documented (6.2) to the satisfaction of all parties, they are baselined, and put under configuration management (6.1). After review by the customer and approval by JPL management, the development plan and any related plans are also baselined.

Once planning is complete, development commences. Development usually proceeds iteratively, and is organized into life-cycle (primary) activities:

- Design (5.6),
- Implementation (5.7),
- Testing and validation (5.8),
- Delivery/installation/acceptance (5.9), and
- Maintenance (5.10). Maintenance is fenced with a dashed line to indicate that JPL's responsibility may vary from project to project.

Supporting activities, which function throughout the development process, include:

- Identification of tools and techniques (6.5),
- Purchasing and subcontracts (6.6),
- Customer-supplied products /reused software (6.7),
- Training (6.8),
- Reviews (6.9), and
- Measurement (6.4) — the collection and use of metrics.

The development plan (5.4) details each of these activities. Once development is complete, the user is provided with the first release, together with the required training and documentation. Subsequent releases are produced by maintenance (5.10), a function possibly provided by a third party.

Design requirements and design details are subject to documentation and document control (6.2) as development proceeds. Code, development and test tools, and quality records (6.3) are subject to configuration management (6.1).

An important new requirement imposed by the *ISO 9001* standard is the maintenance of a quality records archive that contains evidence that the development process is functioning as intended. The quality records archive is used in both internal and third-party assessments, and is an essential requirement of *ISO 9001* certification. Additionally, this archive provides data useful in redesigning the development process to make it more efficient and responsive to project needs. In practice, the archive is usually distributed. The archive typically includes:

- Development plan, which tailors the process description for a particular project/task;
- Related plans for integration and test, configuration management, installation and acceptance, maintenance, etc.;
- Cost estimates;

- Records of reviews, including requests for action and the associated responses;
- Changes in requirements and design, including the rationale for these changes;
- Test records, including software anomaly reports or problem/failure reports, suggested design changes, summaries of test findings and status, and test tool verification records;
- Configuration management records and reports;
- Training records; and
- Metrics, as defined in the development plan.

1.5 RELATIONSHIP TO OTHER JPL PROCESSES AND DOMAINS

The Software Development Process is an element of the Design, Build, Assemble, and Test (DBAT) Process, and is used concurrently with other DBAT processes. It may also be used by other major processes within the Develop New Products (DNP) Domain, such as Project Leadership Processes (PLP), Mission System Design (MSD), Verify, Integrate, Validate, and Operate (VIVO), as well as processes within other domains, such as Provide Enabling Services (PES) Domain. The *Software Development* policy, which can be found at *http://dmie/*, describes the relationship between the Software Development Process and other JPL processes. It is incumbent upon the project/task manager to use DMIE (*http://dmie*) to become familiar with the JPL processes that are currently applicable to the project/task. A sample of the processes that interface with, or are used in conjunction with, the Software Development Process are provided below:

- Quality Assurance Engineering
- Engineering Standards
- Reliability Engineering
- System Safety
- Product Data Management, e.g., for cataloging and archiving quality records and product documentation, such as requirements, design, code, and user documentation.
- Project Planning
- Project Technical Management
- Risk Management
- Project Cost Estimation

EXHIBIT 1.1: Relationship of Software Development Process to External Entities

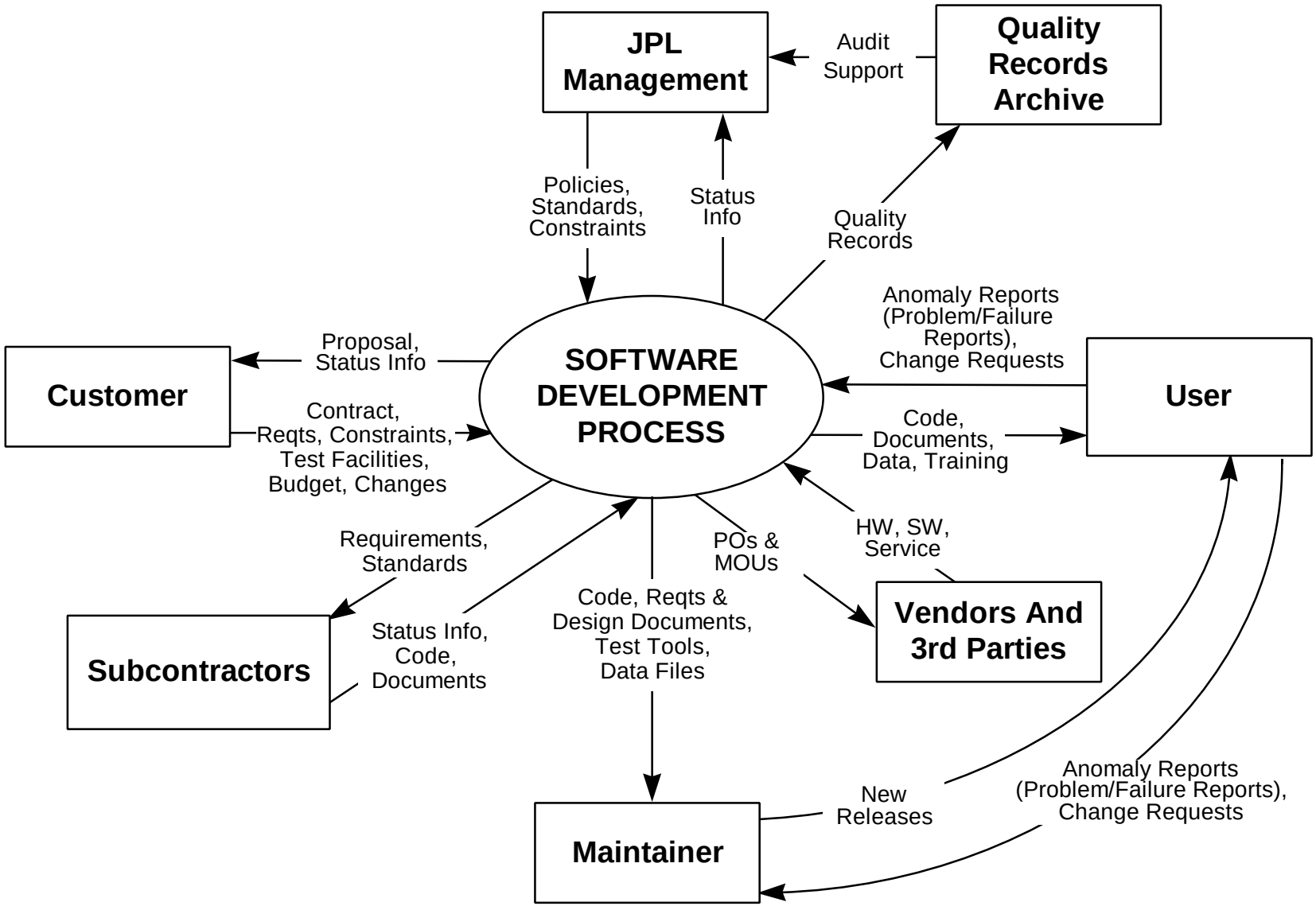
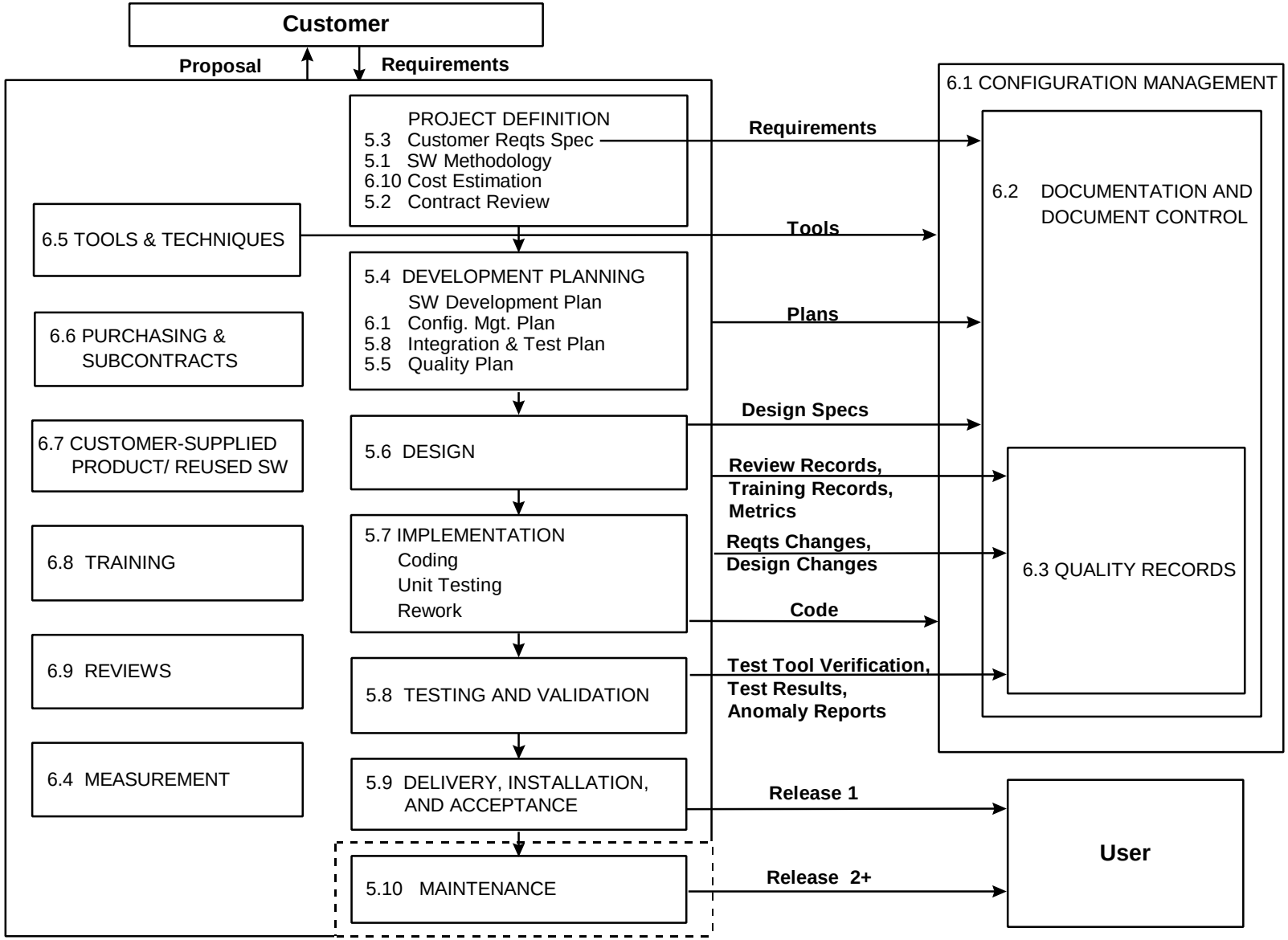


Exhibit 1.2: Inter-Relationships Among Development Activities, Emphasizing Quality System Transactions



In some cases, an instantiating process will use all elements of the SDPD described below in Sections 5 and 6. In other cases, the process will be primarily concerned with executing selected activities or sub-processes, such as:

- Defining a software product and subsequently, managing the subcontracted implementation;
- Performing supporting activities such as evaluation of third party software, configuration management, independent verification and validation, quality assurance, documentation;
- Maintaining the software after installation at the user's site(s).

Although its applicability is intended to be broad, the SDPD was written to support the development of mission critical (Class A) and mission support (Class B) software by the engineering design and development processes within DNP, especially:

- Mission and spacecraft modeling and trade-off studies done in Mission and System Design (MSD);
- On-board processors, controllers, and diagnostic software developed by elements of Design, Build, Assemble, and Test (DBAT), such as the Electronic System Development sub-process; and
- Navigation, sequencing, command, telemetry, and testbed software developed by elements of Verify, Integrate, Validate, and Operate (VIVO).

1.6 SDPD REVISIONS

Recommendations for changes can be submitted to the authors of this SDPD. This document is subject to the change control procedure of the Provide Engineering Standards process, i.e., "Change Control for JPL Level III Category A Engineering Standards." (Refer to *http://dmie/*.)

1.7 NOTATIONS

As mentioned previously, requirements are denoted by "shall" statements, and are distinguished by a ***different font (Arial) in bold italics***. Requirements are also numbered sequentially within each requirements section and bracketed. In addition, where requirements trace to clauses in the *ISO 9001* standard, the ISO clause is also identified and bracketed. Where requirements trace to *ISO 9000-3*, which is the software interpretation of *9001*, the bracketed ISO reference includes the word "guidance."

2. REFERENCES

CMU/SEI-93-TR-006, Taxonomy-Based Risk Identification, located at *http://www.sei.cmu.edu/topics/publications/documents/93.reports/93.tr.006.html*. (Software Engineering Institute)

EIA/IEEE J-STD-016-1995 (formerly P1498), Trial Use Standard for Information Technology — Software Life Cycle Processes — Software Development —Acquirer-Supplier Agreement, September 30, 1995.

IEEE 610.12: 1990, IEEE Standard Glossary of Software Engineering Terminology.

ISO 2382-1:1984, Data Processing — Vocabulary — Part 01 Fundamental Terms.

ISO 8402:1986, Quality — Vocabulary.

ISO 9001:1994, Quality Systems — Model for Quality Assurance in Design, Development, Production, Installation, and Servicing.

ISO 9000-3:1997(E), Guidelines for the Application of ISO 9001:1994 to the Development, Supply, Installation and Maintenance of Computer Software.

ISO 12207:1995, Information Technology — Software Life Cycle Processes.

NASA Policy Directive 2820.1:1998, NASA Software Policies.

NASA Policy Directive (NPD) 7120.4A: 1996, Program/Project Management, (available at *http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/Policies/Program_Formulation/ N_PD_7120_4A.html*).

NASA Procedures and Guidelines (NPG) 7120.5A: 1998, NASA Program and Project Management Processes and Requirements (available at *http://nodis.hq.nasa.gov/Library/ Directives/NASA-WIDE/Procedures/Program_Formulation/N_PG_7120_5A.html*).

NASA Technical Standard (NTS) . 8719.13A, Software Safety, (available at *http://www.hq.nasa.gov/office/codeq/ns871913.htm*).

In addition to the above standards, Table 1-1 provides a list of IEEE standards pertaining to software issues, which the reader may choose to refer to in planning the software life cycle and activities. Note that these standards may not be referred to explicitly in this SDPD, but are provided for informational purposes to facilitate the tailoring of one's software life cycle, activities, and documentation.

JPL Standards and Related Documents:

JPL D-560: 1993, JPL Standard for Systems Safety.

JPL D-4000:1988, JPL Software Management Standards Package.

JPL D-10401:1995, JPL Standard for Reviews.

JPL D-7090:1990, A Guide to Developing Requirements.

JPL D-8431:1991, D-4000 Standards Application Guide: Project Measures.

JPL D-8433:1991, D-4000 Standards Application Guide: D-4000 Activities and Products.

JPL D-9085:1991, D-4000 Applications Guide: Findings and Recommendations from Case Studies of Technical Reviews.

JPL D-10459:1993, D-4000 Standards Applications Guide: Using Quality Factors and Measures to Focus on Customer Satisfaction.

JPL D-12018:1994, D-4000 Standards Applications Guide: Life Cycles.

JPL D-12019:1994, D-4000 Standards Applications Guide: Prototyping.

JPL D-12020:1994, D-4000 Standards Applications Guide: Documentation.

JPL D-12021:1994, D-4000 Standards Applications Guide: Testing.

JPL D-12022:1995, D-4000 Standards Application Guide: Software Reuse.

JPL D-12023:1995, D-4000 Standards Applications Guide: Software Milestone Reviews.

JPL D-13922:1996, A Guide to Selecting and Applying Software Development Standards.

JPL D-15951:1998, Risk Management Handbook for JPL Projects

JPL D-16110: 1998, Engineering Economic Analysis Group, Systems Analysis Section. DSN Guidelines for Presenting Software Costs and Schedules at Major Milestone Reviews.

JPL Policy, JPL Cost Estimation, located at *http://dmie*.

JPL Policy, Risk Management, located at *http://dmie*.

JPL Policy, Software Development, located at *http://dmie*.

JPL Policy, System Safety, located at *http://dmie*.

JPL Policy, Work Breakdown Structure, located at *http://dmie*.

JPL System Procedure, “Change Control for JPL Level III Category A Engineering Standards,” (Provide Engineering Standards Process), located at *http://dmie*.

Table 1-1. IEEE Software Development Standards

Document Number	Title
IEEE 610.12	Standard Glossary of Software Engineering Terminology
IEEE 730	Standard for Software Quality Assurance Plans
IEEE 730.1	Guide for Software Quality Assurance Planning
IEEE 828	Standard for Software Configuration Management Plans.
IEEE 829	Standard for Software Test Documentation
IEEE 830	Recommended Practice for Software Requirements Specifications
IEEE 982.1	Standard Dictionary of Measures to Produce Reliable Software
IEEE 982.2	Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software
IEEE 1002	Standard Taxonomy for Software Engineering Standards
IEEE 1008	Standard for Software Unit Testing
IEEE 1012	Standard for Software Verification and Validation Plans
IEEE 1016	Recommended Practice for Software Design Descriptions
IEEE 1016.1	Guide to Software Design Descriptions
IEEE 1028	Standard for Software Reviews and Audits
IEEE 1042	Guide to Software Configuration Management
IEEE 1044	Standard Classification for Software Anomalies
IEEE 1044.1	Guide to Classification for Software Anomalies
IEEE 1045	Standard for Software Productivity Metrics
IEEE 1058.1	Standard for Software Project Management Plans
IEEE 1059	Guide for Software Verification and Validation Plans
IEEE 1061	Standard for a Software Quality Metrics Methodology
IEEE 1062	Recommended Practice for Software Acquisition
IEEE 1063	Standard for Software User Documentation
IEEE 1074	Standard for Developing Software Life Cycle Processes
IEEE 1074.1	Guide for Developing Software Life Cycle Processes
IEEE 1219	Standard for Software Maintenance
IEEE 1228	Standard for Software Safety Plans
IEEE 1298	Software Quality Management System, Part 1: Requirements
IEEE 1348	Recommended Practice for the Adoption of Computer-Aided Software Engineering (CASE) Tools
IEEE 1420.1	Standard for Information Technology—Software Reuse—Data Model for Reuse Library Interoperability: Basic Interoperability Data Model
IEEE 1420.1A	Supplement to IEEE Standard for Information Technology—Software Reuse—Data Model for Reuse Library Interoperability: Asset Certification Framework
IEEE 1430	Guide for Information Technology—Software Reuse—Concept of Operations for Interoperating Reuse Libraries
IEEE-J-STD-016	Standard for Information Technology — Software Life Cycle Processes
IEEE-12207	Standard for Information Technology — Software Life Cycle Processes

Full-text versions of these standards are available through *http://standards.jpl.nasa.gov*.

3. DEFINITIONS AND ACRONYMS

Terms in this document are defined in *ISO 2382-1*, *ISO 8402*, *IEEE 610.12* and *D-4000*. Supplemental definitions and acronyms follow:

Baselined: A document or software item that has been put under configuration management to control changes.

CASE: Computer-aided software engineering.

COTS: Commercial off-the-shelf, as applied to software and/or hardware purchased from a vendor.

Customer: The entity that has provided the funding for development and/or maintenance of a software product. Customers may be either external to JPL (e.g., NASA or another government agency) or internal (e.g., a JPL project, task or organization). "Purchaser" is the equivalent term in *ISO 9000-3*. It is common for the customer and the user to be separate and distinct entities. JPL program offices often serve as surrogates for external customers, representing the interests of a NASA Headquarters Code.

Developer: The entity that is responsible for designing and implementing a software-intensive product. Often the developer collaborates with the customer in defining requirements. Within JPL, the developer is typically a project spanning multiple technical divisions, or a task that is wholly contained within one division. "Supplier" is the equivalent term in *ISO 9000-3*.

Development: All activities carried out to create a software product.

Development Plan: The term refers to an independent document, a part of another document or several documents, all of which pertain to project policies, plans, procedures, schedules, etc. that govern a project. (Refer to Section 5.4.)

Implementation: The particular activities of detailed design, coding, and unit testing; a subset of development.

Life Cycle: The set of interrelated activities that stretch from the initial determination of need through management, definition, design, implementation, test, delivery, training, and post-delivery modification/ maintenance, terminating when the application is retired from use. A life cycle is divided into phases — defined segments of work — that are relevant to a specific software product, with each phase typically being terminated by an end-gate or milestone. This SDPD is intended to be broad enough to encompass a wide spectrum of life-cycle models, such as waterfall, incremental, evolutionary, and spiral/iterative development.

Maintainer: The entity that is responsible for making changes to the software after acceptance and installation at the user's site(s).

Project: Refers to the development effort (e.g., project, project element, task) addressed by the development plan.

Quality Record: Documented evidence that development is proceeding in accord with the plans, and that the JPL quality system (also referred to as the JPL Product Delivery System) is functioning as intended. Quality records include agreements about changes in requirements, reasons for key design decisions and modifications in the development plan, minutes of design reviews and response to Requests for Action, software anomaly reports (or problem/failure reports), test records, verification of test tools, software metrics, etc. Additionally, quality records are useful in identifying targets for process improvement.

Software: Intellectual creation comprising the programs, procedures, algorithms, rules, data, and any associated documentation pertaining to the operation of a data processing system. Software programs that are burned into a chip are termed firmware. This SDPD treats firmware up to the point of simulated operation in a hardware environment. The Electronic System Development sub-process of the Design, Build, Assemble, and Test Process addresses production of the chip and subsequent integration and test of the assembly and subsystem of which the firmware is a part.

Software Item: Any identifiable part of a software product at an intermediate step or at the final step of development; in Department of Defense nomenclature, a Computer Software Configuration Item (CSCI) would be a high-level software item.

Test Case: A set of test inputs, execution conditions, and expected results developed for a particular objective, such as to exercise a particular program path or to verify compliance with a specific requirement. (*IEEE 610.12, D-4000*)

Test Data: The input needed to establish initial conditions for a test case, force execution of a particular path in the software, or to simulate an external interface. (*IEEE 610.12*)

Test Driver: A software module used to invoke a module under test and, often, provide test inputs, control and monitor execution, and report test results. (*IEEE 610.12*)

Test Procedure: Detailed instructions for the set-up, execution, and evaluation of results for a given test case. (*IEEE 610.12, D-4000*)

User: The entity that employs the capabilities of the software product to perform a larger function, such as — control the systems on board a spacecraft, deliver raw data to a principal investigator, support mission design, prepare financial status reports, support software integration and test, etc. In some cases, the user is an inanimate object such as a spacecraft; in other cases, the user may be a human. Some software systems are complex enough to require a human operator who assigns user privileges, installs software, performs back-up and recovery, troubleshoots problems, and generally supports the end user. In this SDPD, operators are treated as a user subclass.

Validation: The process of evaluating the software product to ensure compliance with the customer/user's requirements.

Verification: The process of evaluating the products of a life-cycle phase to ensure correctness and consistency with respect to the products and standards provided as inputs to that phase.

4. QUALITY SYSTEM — FRAMEWORK

Section 4 addresses topics that pertain to the framework of a quality system:

- The software quality policy that states JPL's overall goal in producing software, along with the objectives for achieving it;
- General roles and responsibilities (e.g., of the Software Development Methodology Process owner, JPL program offices, and project/task managers) to achieve software quality; and
- Internal audits, which serve as a mechanism for ensuring compliance, as well as identifying opportunities for improving the software development process.

4.1 SOFTWARE QUALITY POLICY

It is the goal of all JPL software development projects/tasks, and the organizations of which they are a part, to define, develop, and deliver software products in a manner that results in total customer satisfaction. This goal will be achieved via:

- Explicit definition of the end product in a manner that permits the user to envision how it will be employed;
- Choice of a software architecture that facilitates growth in functionality and elaboration into a design that accommodates changes easily and promotes reuse;
- Development plans that give the customer, project/task management, and the individual implementer frequent milestones with which to measure progress;
- Identification of software elements with high potential for reuse and utilization of design practices that facilitate reuse;
- A cost-effective documentation strategy that communicates crucial design information to the developers, and facilitates use and maintenance of the product;
- Explicit provisions for training those who will have frequent contact with the product after delivery — the end user, the operator/system administrator, and the maintainer;
- Participation of the customer and user in relevant development activities — especially change management, design of the user interface, and in-process validation of the design;
- Continual improvements to the software development process tailored to the needs of each development organization.

4.2 ROLES AND RESPONSIBILITIES

The SDPD is written and maintained by the owner of the Software Development Process, in response to a requirement to articulate Lab-wide policy on software development and to set minimum requirements. The intent is to document best JPL practice in response to *ISO 9001* requirements, as interpreted by *ISO 9000-3*, while providing for tailoring of these requirements to meet the needs of individual projects, tasks and organizations. A closely related responsibility of the process owner is to support periodic internal audits of JPL's software development activities. The SDPD will be modified annually in response to these audits and other user feedback, with the intent of promoting continual process improvement.

Although the requirements and guidelines contained in the SDPD were developed for use by the Develop New Products (DNP) Domain, they are meant to be general enough to be relevant to software development under the cognizance of other domains and all JPL program offices. Each domain and program office has a suite of documents that define required business practices within its own area. Program office documents should be construed as a tailoring and elaboration of the requirements stated in the SDPD, with a few notable exceptions, such as corrective and preventive action, control of quality records, and statistical techniques (metrics) — all of which are *ISO 9001* requirements that are new to the Laboratory. Accordingly, it is the responsibility of each program office to determine how to comply with the *ISO 9001* requirements in each of these new areas, which the SDPD has interpreted in general terms for software development.

Ultimately, the responsibility for implementing the requirements of the SDPD and related documentation rests with the JPL project or task manager. It is the manager's responsibility to tailor the requirements of the SDPD, plus relevant domain, program office, and line organization standards and policies, to the needs of the specific development job, the result being documented in project/task plans. Evidence that this responsibility has been met is contained in the suite of plans, product descriptions (e.g., requirements document, operations concept), and quality records produced during development. It must be emphasized that considerable flexibility is afforded in documentation, the essential idea being that all documentation should be directly useful to developing a high quality end product within the allocated budget.

4.3 INTERNAL AUDIT

Software development organizations will participate in periodic internal audits (led by the Enterprise Process and Standards Program Office) to assure continued compliance with *ISO 9001* and to identify additional opportunities for improving JPL's software development methodology. For software-intensive efforts, these audits are scheduled on the basis of the importance and status of the activity and examine, at the minimum:

- Tailoring of the project/task plans,
- Documentation of software requirements,
- Status of interface definitions,

- Adequacy of configuration management procedures,

- Closure of critical action items from reviews, as defined by the project/task.

Results of these audits, including recommendations for improved practices, will be provided to both the manager whose area is audited, and the JPL Product Delivery System Management Representative. Approved recommendations will be implemented via changes in Laboratory procedures — including this process description and related program office directives. The documentation of project/task plans, designs, and reviews of development progress (i.e., quality records — see Section 6.3) will be important inputs to an audit.

5. QUALITY SYSTEM — LIFE-CYCLE ACTIVITIES

This section, in conjunction with Section 6 that follows, outlines a general methodology for software development at JPL, and describes the activities employed to implement the methodology. This section is devoted to those activities that define the software life cycle.

Requirements are denoted by "shall" statements, and are distinguished by a ***different font (Arial) in bold italics***. Requirements are also numbered sequentially within each requirements section and bracketed. In addition, where requirements trace to clauses in the *ISO 9001* standard, the ISO clause is also identified and bracketed. Where requirements trace to *ISO 9000-3*, which is the software interpretation of *9001*, the bracketed ISO reference includes the word "guidance."

5.1 SOFTWARE METHODOLOGY

A software development project/task shall be organized according to a life-cycle model that is described in the development plan in terms of: *[5.1] [ISO 4.4.1 guidance]*

- ***Phases, along with milestones and activities to be performed during each phase;*** *[5.1a] [ISO 4.4.2 guidance]*

- ***Phase outputs, including any documentation;*** *[5.1b] [ISO 4.4.2 guidance]*

- ***Verification activities (e.g., reviews, demonstrations, tests) by phase.*** *[5.1c] [ISO 4.4.2 guidance]*

Guidance:

This SDPD should not be interpreted to dictate the use of any particular life-cycle model. Rather, flexibility is permitted in defining the life cycle, each project/task being encouraged to adapt the general methodology described here in a way that best suits its needs. Given that many projects are incorporating COTS products in their delivered products, the reader should be aware of life-cycle issues unique to such an environment. The COTS discussion on the Software Engineering Institute's (SEI) web site at *http://www.sei.cmu.edu/cbs/* may provide a good starting point.

In describing one's life-cycle model or development process, it is not sufficient to merely assert that a particular standard (e.g., *D-4000*) or methodology (e.g., spiral development, rapid development) will be followed, since projects rarely conform strictly to a textbook description, due to different sponsor relationships, team size and composition, budget, schedule, etc. Thus, in virtually every instance, tailoring of any given standard or methodology will be required. In

addition, it is legitimate to find that during the initial planning activities, early phases may be better defined than later phases. In all cases, it is expected that the plan will be updated as the project progresses.

Ideally, it is useful to identify inputs to, and/or criteria for entering, each phase. Similarly, identifying criteria for exiting a phase should be considered.

In practice, phases overlap, and the activities needed to complete a phase often extend beyond the milestone that marks a critical review of phase activities. Nonetheless, it is accepted practice to schedule a milestone review at a point in time when the activities of a phase are substantially complete. When such a milestone is reached, the phase products identified in the development plan are reviewed, and review results recorded as part of the project/task quality record. (See Section 5.4–Development Planning, Section 6.9–Reviews, and Section 6.3–Quality Records.)

Below is a set of activities and milestones that are useful for planning the development of Class A and B software:

- Identification of user needs, development objectives, and constraints;
- Definition of a system architecture responsive to the development objectives;
- Preparation of a development plan;
- Elaboration of the user needs and development objectives into software requirements;
- Completion of the design, and verification that it satisfies the requirements. In a complex system there may be multiple levels of design with checkpoints at each level.
- Completion of integration and test. The complexity of the system will dictate the levels of test and integration needed. At the minimum, test is needed at the level of an atomic software element and the system level.
- Validation that the product satisfies the user's requirements, and is acceptable to the customer;
- Delivery to the customer and installation in the user's environment.

It is strongly encouraged that the nomenclature used in *JPL D-4000, JPL Software Management Standards Package*, be used to tailor a life cycle to project/task needs and define documentation and reviews tailored to those needs. This use of *D-4000* does not compel the developer to satisfy any of the "shall's" in *D-4000*. Tailoring guidance may be found in the *D-4000* Application Guides (see Section 2.0–Applicable Documents), which help adapt the *D-4000* methodology to fast-moving, concurrent development. Thus, this process description reaffirms the *D-4000* nomenclature for documentation as the standard nomenclature to be used at JPL. Additional guidance on the content of software documentation may be found in *EIA/IEEE J-STD-016, Standard for Information Technology — Software Life Cycle Processes: Software Development. JPL D-13922, A Guide to Selecting and Applying Software Development Standards*, provides a high-level mapping between *D-4000* documents and the product documentation identified in

EIA/IEEE J-STD-016. A related standard, *ISO 12207, Information Technology — Software Life Cycle Processes*, provides guidance in defining software life cycle processes, activities, and tasks. It has been used by some ISO auditors to evaluate a software organization's compliance with *ISO 9001*.

5.2 CONTRACT REVIEW

The contract between the customer and the developer can take a variety of forms. If the customer is NASA Code S, this contract may be expressed in terms of an Approval Letter, Task Order, Program Operating Plan, a Work Authorization Document, or an RTOP. If the customer is another government agency, this contract is normally documented in a Task Plan. If the customer is internal — a project or a program office — the contract takes the form of an implementation or development plan, or a work package agreement. The interests of all stakeholders should be noted during the contract review process, including the user and the organization that will maintain the software after delivery.

The organization that has overall responsibility for a software development effort shall establish and maintain documented procedures for a commitment review of the development plan or proposal to ensure: *[5.2.1] [ISO 4.3.1]*

- ***Scope of work for the current delivery is adequately defined and documented.*** *[5.2.1a] [ISO 4.3.2a]*
- ***Differences between the scope of work defined in the development plan, and that requested by the customer, are resolved.*** *[5.2.1b] [ISO 4.3.2b]*
- ***Responsibilities of the customer are identified.*** *[5.2.1c] [ISO 4.3.2 guidance]*
- ***Mutually acceptable means have been defined for dealing with changes in requirements during development, as well as correction of post-delivery defects.*** *[5.2.1d] [ISO 4.3.2 guidance, 4.3.3]*
- ***The resources and schedule described in the development plan are adequate to accomplish the contractual deliverables.*** *[5.2.1e] [ISO 4.3.2c]*

A record of commitment review findings shall be maintained as part of the project/task quality record. *[5.2.2] [ISO 4.3.4]*

Guidance:

Issues to consider when defining customer responsibilities include:

- Facilities, tools, and software to be provided by the customer or third parties
- Requirements definition if additional work is needed
- Design validation.

It is strongly encouraged that the commitment review address contingencies and risks, including crucial interdependencies, as well as schedule margin and budget reserve.

5.3 CUSTOMER'S REQUIREMENTS SPECIFICATION

The developer shall have a written set of software requirements that are sufficient to satisfy customer and/or user needs. *[5.3.1] [ISO 4.4.4 and guidance]*

Interfaces between the software product and external software or hardware items shall be specified, either directly or by reference. *[5.3.2] [ISO 4.4.4 guidance]*

Software requirements, whether provided by the customer or formulated by the developer, shall be reviewed to ensure that: *[5.3.3]*

- ***the product is adequately defined,*** *[5.3.3a] [ISO 4.4.4]*
- ***ambiguities and conflicting requirements have been resolved, and*** *[5.3.3b] [ISO 4.4.4]*
- ***the requirements are stated so as to allow validation during product acceptance.*** *[5.3.3c] [ISO 4.4.4 guidance, ISO 4.4.8]*

The software requirements specification shall be subject to change control procedures, once it is baselined (e.g., completion of document review, customer approval obtained) . *[5.3.4] [ISO 4.3.3, 4.4.4 guidance]*

Approved changes in requirements shall be maintained as part of the project/task quality record. *[5.3.5]*

Guidance:

Occasionally, it is possible to identify only high-level requirements at the beginning of development. In such a case, the initial development effort is often devoted to a definition phase that produces a high-level design sufficient to create firm cost and schedule estimates.

To facilitate the developer's understanding of requirements and their subsequent elaboration into a design, the development organization should seek to obtain a designated user representative who can elaborate written requirements, clarify ambiguities, help fill gaps, etc.

A review of software requirements typically determines whether:

- The customer and/or user feels that their needs have been satisfied
- The developer has adequate information to proceed with design
- All external interfaces have been defined
- Operational safety has been satisfactorily addressed
- Requirements are stated in a manner that facilitates validation.

Changes to the baselined requirements specification should reference pertinent correspondence with the customer. The mechanism for handling and communicating requirements changes is to be addressed in the development plan, as described in Sections 5.2 and 5.4.

5.4 DEVELOPMENT PLANNING

Preparation of a development plan compels the developer to identify and think through the multitude of issues involved in designing and implementing the product. It is preferable that this plan be written as one coherent document. However, it may be expedient for the developer to document the plan in a set of related documents that separately address issues such as product definition, scope of work, task organization, schedule, build plan, integration and test plan, etc. Thus, **the term "development plan" is used in this document in a generic manner** to refer to various forms of planning documentation. In instances where the plan is fragmented, it is strongly encouraged that there be some documentation that unifies the segments of the plan by providing pointers to all portions of the plan.

A development plan shall address the following: *[5.4.1] [ISO 4.4.2]*

- ***Overall definition of the product, as in user needs addressed, deliverables, and critical functionality.*** *[5.4.1a] [ISO 4.4.2 guidance]*

- ***Scope of development work to be performed, including management and supporting activities***. *[5.4.1b] [ISO 4.4.2 guidance]* Scope of work is usually documented in a work breakdown structure, with an accompanying narrative.

- ***Project life cycle, including:*** *[same requirement as 5.1] [ISO 4.4.1 guidance]*

  Phases, along with activities or milestones to be performed during each phase. *[same requirement as 5.1a] [ISO 4.4.2 guidance]*

  Phase outputs, including any documentation *[same requirement as 5.1b] [ISO 4.4.2 guidance]*

  Verification activities (e.g., reviews, demonstrations, tests) by phase *[same requirement as 5.1c]*

- ***Project organization and technical interfaces: team structure; nature of project interfaces, both internal and external; roles and responsibilities, including customer responsibilities; use of subcontractors; and other crucial dependencies, such as critical equipment and facilities, and use of JPL support services.*** *[5.4.1c] [ISO 4.4.3, ISO 4.4.2 guidance]*

- ***Project schedule.*** *[5.4.1d] [ISO 4.4.2 guidance]*

- ***Risk assessment.*** *[5.4.1e] [ISO 4.4.2 guidance]*

- ***Cost estimate/budget that summarizes the cost of the pers onnel and other resources required by the development.*** *[same requirement as 6.10.1] [ISO 4.4.2 guidance]*

- ***Staffing profile.*** *[5.4.1f] [ISO 4.4.2 guidance]*

- ***Change control procedures for documenting, reviewing, approving, and communicating requirements changes to all affected parties.*** *[5.4.1g] [ISO 4.3.3, 4.4.9, and 4.5]*

- ***Change control procedures for documenting, reviewing, approving, and communicating design changes before their implementation.*** *[5.4.1h] [ISO 4.4.9, 4.5]*

- ***Review (or verification) policies and procedures that, at a minimum, address detailed technical reviews, and identify what is to be reviewed (including critical intermediate products) and when reviews are to be held.*** *[same requirement as 6.9.1] [ISO 4.4.5, 4.4.6, 4.4.7]*

- ***Procedures for verifying, storing, protecting, and maintaining items (e.g., software, data, hardware, specifications) supplied by the customer or designated third party.*** *[same requirement as 6.7.1] [ISO 4.7]*

- ***Procedures for verifying purchased or subcontracted products.*** *[same requirement as 6.6.3] [ISO 4.6.4, 4.10.2]*

- ***Documentation plan and procedures.*** *[same requirement as 6.2.1and 6.2.2; refer to actual requirements for elaboration] [ISO 4.5]*

- ***Scope and content of the training to be provided to project personnel.*** *[same requirement as 6.8.1] [ISO 4.18, 4.1.2.2]*

- ***System administration plan, including approach to back-up/archiving, security, and virus protection.*** *[5.4.1i] [guidance in ISO 4.4.2, 4.9, 4.15.2, 4.15.3, 4.15.5]* (See related requirements specific to deliveries, Section 5.9.)

- ***Definition of responsibility, and description of associated procedures, to identify and correct recurring problems in the development process.*** *[5.4.1j] [ISO 4.14.1 and 4.14.3]*

- ***Metrics tailored to project needs, and the associated procedures for collecting, storing, and analyzing them.*** *[same requirement as 6.4.1]*

- ***Planning of the following specific activities, including identification of any separate plans:*** *[5.4.1k]*

  Configuration management *[same requirement as 6.1.1; refer to actual requirement for elaboration] [guidance in ISO 4.8 and 4.5]*
  Integration and test *[same requirement as 5.8.1; refer to actual requirement for elaboration] [ISO 4.10.1]*
  Delivery and installation *[same requirement as 5.9.1; refer to actual requirement for elaboration] [ISO 4.9, 4.15.1]*
  Maintenance *[same requirements as 5.10.1 and 5.10.2; refer to actual requirements for elaboration] [ISO 4.19, 4.4.2 guidance, 4.4]*

- ***Reuse strategy, if any, or identification of reusable elements — both those th at can be adapted from previously implemented systems, and portions of the current application that will be designed for reuse.*** *[5.4.1l]*

- ***Identification of quality records, associated procedures, and retention times.*** *[same requirements as 6.3.1 and 6.3.3; refer to actual requirements for elaboration]* (See Section 6.3–Quality Records.) *[ISO 4.16]*

- ***Provisions for updating the plan as development proceeds.*** *[5.4.1m] [ISO 4.4.2]*

- ***Explanations for any deviations made from SDPD requirements.*** *[5.4.1n]*

Guidance:

The ***review of the development plan*** is to be done before development begins, with the customer and all organizations involved in its implementation participating in that review. When the plan is revised, e.g., in spiral or iterative developments, or when replanning activities are needed to address scope changes, the revisions need to be reviewed as well.

Revisions to the development plan will not necessarily cover all of the topics described in the requirements of this section. For example, there are projects/tasks that are engaged in ongoing, evolutionary developments, where incremental versions of the application are released on a regular basis, e.g., every six months. In such established and stable environments, a revised plan might only cover those topics which have undergone change since the previous plan was issued.

In defining ***project organization***, the *D-4000* standards for each of the development phases and related activities provide useful descriptions of roles and responsibilities.

The reader should become familiar with the policies and associated documentation pertaining to the JPL policy identified as "***Work Breakdown Structure***" (refer to *http://dmie/)*. This policy arises from the JPL process called "Project Technical Management."

In planning the ***schedule***, its granularity must be small enough to ensure effective control. In addition, the incremental build plan should be addressed.

In planning for ***defect correction and change control*** for large projects, a change control board with customer representation may be appropriate.

In developing a plan for addressing ***security*** concerns, the developer should refer to Lab-wide, program office, or line organization standards. Applicable policies, procedures and standards associated with the Provide Computer Security Services process are available at *http://dmie/*. One might also refer to the following JPL site: *http://security/*.

The need to identify and correct ***recurring problems in the development process*** falls within the realm of continuous software process improvement. Illustrative recurring problems are incomplete or ambiguous requirements, inaccurate interface definition, inadequate design and coding standards, ineffective walkthroughs and design reviews, inadequate unit testing, incomplete test planning, ineffective re-testing of corrected code, etc.

Risk assessment is the process of identifying, analyzing, mitigating, and monitoring risk through the life of the project. The reader should refer to:

(a) *NASA Policy Directive (NPD) 7120.4A*[3], *Program/Project Management*, which is part of the NASA Prime Contract, states that risk assessment should address technical, cost and schedule issues.

(b) *NASA Procedures and Guidelines (NPG) 7120.5A*[2], *NASA Program and Project Management Processes and Requirements*, elaborates on the requirements in *NPD 7120.4A*. It also identifies examples of risks and constraints which are useful for project managers to use in examining their own projects. The reader should also refer to the Project Plan template in *NPG 7120.5A* which includes a description of the risk management topics to be addressed in a project plan.

(c) JPL policy on *Risk Management*, located at *http://dmie*. This policy references *NPG 7120.5A*, as well as *JPL D-15951, Risk Management Handbook for JPL Projects*.

NPG 7120.5A requires that for each primary risk (i.e., those having both high probability and high impact/severity), the following be provided:

- description of risk, including primary causes and contributors, actions embedded in the project to date to reduce or control it, and information collected for tracking purposes.
- estimate of the probability (qualitative or quantitative) of occurrence together with the uncertainty of the estimate.
- significant cost impacts, given its occurrence.
- significant schedule impacts, given its occurrence.
- potential mitigation measures
- characterization of the risk as “acceptable” or “unacceptable” with supporting rationale.

The reader may also want to refer to *CMU/SEI-93-TR-006, Taxonomy-Based Risk Identification*, located at *http://www.sei.cmu.edu/topics/publications/documents/93.reports/93.tr.006.html*. This Software Engineering Institute (SEI) technical report includes a self-evaluation questionnaire which managers can use as a tool to spot potential risk areas. Risk areas are organized into three broad categories that address (a) the product, (b) processes (development, management), development environment, work environment, and (c) program constraints with respect to resources, contract, and program interfaces.

[3] According to the NASA Prime Contract, effective FY99, JPL is to comply with *NASA Policy Directive (NPD) 7120.4A* and *NASA Procedures and Guideslines (NPG) 7120.5A*. *NPD 7120.4A* is available at *http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/Policies/Program_Formulation/N_PD_7120_4A.html*. *NPG 7120.5A* is available at *http://nodis.hq.nasa.gov/Library/Directives/NASA-WIDE/Procedures/Program_Formulation/N_PG_7120_5A.html*.

Safety issues are addressed in the *ISO 9001* standard, and also constitute an element of risk management, according to *NPG 7120.5A*, section 4.5–Safety and Mission Success, and Environmental Management. The reader should refer to *http://dmie/* for the policy and any procedures and standards arising from the System Safety process at JPL, including *JPL D-560, JPL Standard for Systems Safety*.

JPL D-560 levies safety requirements on flight systems, ground-based systems, and research and development programs — including systems developed in-house, contracted, or purchased. It requires that project/task managers develop a safety plan (which is to include software safety) — an example of which is provided in the appendix of *D-560*. Section 2.11 (Computer Systems) of *D-560* includes requirements pertaining to fault tolerance, hardware/software inhibits, software design (e.g., modularization of safety-critical functions), and software hazard analysis. Section 3.8 (Software) of the same standard includes requirements on operator displays.

In addition, the reader may find it useful to refer to *NASA Technical Standard (NTS) 8719.13A, Software Safety*[4]. *NTS 8719.13A* applies to software and firmware, including government furnished equipment, purchased software (including COTS software), and other reused software in the system. The standard addresses safety issues by life-cycle phase, and describes the software safety analyses to be performed.

Other issues to consider when formulating a development plan include:

- ***User's role*** in defining, designing, and testing the system.
- ***Rules, practices, and conventions*** to be used in design and development. (See Section 5.7–Implementation.)
- ***Tools and techniques*** to be used, e.g., for project planning and scheduling, requirements management, etc. (See Section 6.5–Tools and Techniques.)
- Identification of ***procurements*** — development platforms, development tools, commercial software packages to be integrated into the end product, etc. (See Sections 6.5–Tools and Techniques, and 6.6–Purchasing and Subcontracts.)
- Design and implementation of the ***development environment***.
- ***Disaster recovery***.
- ***Conversion of*** a design ***prototype*** to a delivered product, to the extent that this can be anticipated at the time the development plan is written.

A development plan template is provided in Appendix A. In ***tailoring planning documentation***, the developer should refer to Lab-wide, program office, or line organization standards. The JPL policy on *Project Planning* refers to the project plan requirements in *NPG 7120.5A*. Other standards, such as *D-4000*, ISO, and IEEE are available as guidance. See Section 2 (References) and Table 1-1.

[4] *NTS 8719.13A* is available at *http://www.hq.nasa.gov/office/codeq/ns871913.htm*.

5.5 QUALITY PLANNING

Guidance:

The philosophy of the SDPD and *ISO 9001* is to build quality into the product as development progresses. Thus, quality planning is viewed as being inherent to development planning, and as such, is addressed throughout a development plan. However, it is discussed here to emphasize its importance, and to draw attention to development activities that can have large impacts on quality. Quality planning activities include such things as:

- Verification and validation activities (e.g., design reviews, code walkthroughs, and tests) of:
  - developed products
  - externally-received products, such as purchased COTS, customer-supplied products, or products supplied by other development organizations within JPL.
- Design and code analysis, including comparison of design alternatives, trade-offs between coupling and cohesion, verification of completeness vis-a-vis software requirements, evaluation against design patterns, comparison of code with style guide, etc.
- Configuration management and change control of software, data, and documentation.
- Reporting and monitoring of defect correction.
- Identification and correction of the root causes of design and coding defects.

It is also useful to establish quality objectives for the project/task, e.g., fraction of development effort expended on rework, the percent of the code that was comprehensively exercised in system testing, the proportion of reused software in the final product, etc.

5.6 DESIGN

Requirements and design activities shall be guided by a plan with milestones and detailed technical reviews tailored to the needs of each project/task. *[5.6.1] [ISO 4.4.2, 4.4.6, 4.4.7]*

The design shall be documented and, prior to release, the resulting design documentation shall be reviewed to ensure that (a) the design meets the requirements and is responsive to acceptance criteria, (b) the design is verifiable, and (c) safety issues have been addressed. *[5.6.2][ISO 4.4.5]*

Guidance:

The design and implementation activities transform the sponsor's requirements specification into a software product. Design activities encompass the translation of the user's requirements into functional requirements, a functional design, software requirements, a software architecture, and more detailed specifications for individual software elements. Because software is complex, it is imperative that these activities be carried out in a manner that builds quality into the product,

rather than depending on test and validation to assure quality. To that end, milestones and detailed technical reviews must be defined in a development plan, with the level of detail commensurate with the complexity of the project/task.

A critical element of the design process is the definition of external and internal interfaces, which may be documented separately from performance requirements. These design products should be documented in a manner appropriate to each project/task. Failing to document them risks serious communications problems which can result in costly rework and/or a dissatisfied customer. Section 6.2 and Appendix B recommend documentation appropriate for Class A, B, and C software.

The design process should be structured to:

- Facilitate debugging, testing, and subsequent maintenance;
- Monitor closely the design of both internal and external interfaces;
- Hold working-level detailed technical reviews of designs as they are deve loped; these reviews should employ working documentation.

Evidence that design documentation meets requirements is often provided in the form of a traceability matrix. ***Traceability*** can be performed between requirements, design specifications, source files, software procedures or functions, and test cases, or some subset of those. Exactly which products will be traced to which other products, and whether the traceability is performed in a forwards manner, or a backwards manner, will differ from project to project.

5.7 IMPLEMENTATION

Implementation activities shall be guided by one or more plans with milestones and detailed technical reviews tailored to the needs of each project/task. *[5.7.1] [ISO 4.4.2, 4.4.6, 4.4.7, 4.10.3]*

Guidance:

Implementation involves taking detailed specifications and translating them into working code. Often a design is prototyped in code prior to documenting the design of the finished software element, thus making it difficult to separate the process of implementation from design. If the ultimate target of a prototyping effort is Class A or B software, the code must eventually satisfy the pertinent documentation and review requirements. In general, implementation should be structured to:

- Create a running version of the product as ea rly as practicable;
- Adhere to documented coding standards, including naming conventions, code format, and in-code documentation requirements; a history of code changes can facilitate failure analysis.
- Encourage conformance to design rules, including restrictions on the use of language constructs and the complexity of code aggregates.

- Employ peer review of the detailed design and the code itself.
- Emphasize thorough unit testing by each implementer; this facilitates both subsystem and system integration, thereby speeding the implementation process.
- Produce design documentation and acceptance criteria traceable to functional requirements.

5.8 TESTING AND VALIDATION

Software integration and testing shall be performed in accord with test planning and specification documentation that addresses: *[5.8.1] [ISO 4.4.2 guidance, 4.10.1]*

- ***Test requirements, which may be an elaboration of software requirements.*** *[5.8.1a] [ISO 4.10.1 guidance]*
- ***Levels of testing required up to acceptance by the customer.*** *[5.8.1b] [ISO 4.10.1 guidance]*
- ***Test cases, test procedures, test data and expected results.*** *[5.8.1c] [ISO 4.10.1 guidance]*
- ***Method of documenting test status and results.*** *[5.8.1d] [ISO 4.10.1 guidance, 4.12]*
- ***Test environment, such as dedicated processors, test tools (purchased or developed), and user documentation.*** *[5.8.1e] [ISO 4.10.1 guidance]* The actual test environment should be defined precisely enough to ensure repeatability.
- ***Approach for evaluating test tools, with respect to their ability to verify the product under test (e.g., through testing, published reviews).*** *[5.8.1f] [ISO 4.11.1]*
- ***Procedures for correcting defects, including analyzing the cause of the defect, determining corrective action, and ensuring that the corrective action is taken.*** *[5.8.1g] [ISO 4.13, 4.14.1, 4.14.2]*

Before delivery and acceptance by the customer, the developer shall validate the product under conditions similar to the user's application environment. *[5.8.2] [ISO 4.4.8, 4.10.4]*

Missing or deficient functionality (in light of customer/user expectations, based on a requirements document or other form of "contractual" document) shall be documented in a release description document or transfer agreement. *[5.8.3] [ISO 4.13.2]*

Test records to be maintained as part of the project quality record shall include, at a minimum: *[5.8.4]*

- ***Anomaly reports (or problem/failure reports)*** *[5.8.4a][ISO 4.13.2]*
- ***Test tool checks, to evaluate whether the tools are capable of verifying the acceptability of the software product under development.*** *[5.8.4b] [ISO 4.11.1]*
- ***Test results, with clear indications whether the product has passed or failed.*** *[5.8.4c] [ISO 4.10.5, 4.4.7]*

Guidance:

Testing may be required at several levels, from the individual software item to the complete software product. Testing of a software unit (typically by the programmer who implemented the unit) is called unit testing, an activity subsumed under implementation. The subsequent testing addressed by Section 5.8 focuses on the integration and test of collections of software elements (program sets), plus subsystem/system-level integration and test.

Note that the ***test plan*** may be incorporated into the development plan, although typically, it will be a separate document. Additional issues that should be considered in developing the test plan include:

- ***Types of testing*** required, e.g., functional tests, boundary tests, performance tests, usability tests, regression tests, etc. Regression testing is especially valuable in incremental or iterative development and in verifying the correction of critical anomalies.
- Procedure to be followed in ***categorizing and prioritizing anomalies***. Anomaly reports should be categorized to show impact of the defect on product usability and capability to meet the customer's requirements.
- ***Customer/user involvement*** in the testing process — including additional personnel, hardware, and travel expense if travel to the user site is required. User participation is encouraged in defining tests that closely simulate the operational environment.
- ***Training*** required for test personnel, if not addressed in the development plan.
- ***Schedule and staffing*** plan, if not addressed in the development plan.

Test procedures should be developed with attention to demonstrating the repeatability of tests, particularly for real-time systems. Once the test procedures have been developed, it is strongly recommended that automated execution of the subsequent tests be employed.

In ***tailoring test documentation***, the developer should refer to Lab-wide, program office, or line organization standards. Other standards, such as *D-4000*, ISO, and IEEE are also available as guidance. See Section 2 (References) and Table 1-1.

5.9 DELIVERY, INSTALLATION, AND ACCEPTANCE

The activities comprising delivery, installation, and acceptance shall be defined in a plan or related documentation, that addresses: *[5.9.1] [ISO 4.4.2 guidance, 4.9, 4.15.1]*

- ***Preparation of the acceptance test cases and acceptance criteria, with developer's responsibilities (if any) noted.*** *[5.9.1a] [ISO 4.10.4 guidance; based on ISO 4.4.8 and 4.10.5]*

- ***Procedures to be used in documenting and resolving problems found following installation, whether during acceptance testing or delivery.*** *[5.9.1b] [ISO 4.14.1, 4.14.2]*

- ***Details of delivery and installation logistics, e.g., arranging for use of customer/user facilities and personnel in installation and test.*** *[5.9.1c] [ISO 4.9 guidance]*

- ***Definition of developer's role (if any) in supporting transition to full operational use of the product.*** *[5.9.1d] [ISO 4.9 guidance]*

- ***Identification of documentation to be delivered at installation, including installation and configuration procedures.*** *[5.9.1e] [ISO 4.9 guidance]* Section 6.2 and Appendix B identify the documentation recommended for each class of software.

- ***Identification of training for the user and/or system administrator/operator.*** *[5.9.1f] [ISO 4.9 guidance]*

- ***A schedule for key events pertaining to delivery, installation, and acceptance.*** *[5.9.1g] [ISO 4.9 guidance]*

- ***Storage of archived software media to prevent deterioration and facilitate disaster recovery.*** *[5.9.1h] [guidance in ISO 4.15.3 and 4.15.5]*

- ***Virus protection of software designated for delivery, during storage and electronic transmission.*** *[5.9.1i] [guidance in ISO 4.15.2, 4.15.3, 4.15.6]*

After delivery, a baselined copy of the software and delivered documentation shall be archived. *[5.9.2]*

Guidance:

Because a JPL developer usually has a very close working relationship with the customer, the customer's decision to accept the product often comes after a lengthy test process. Typically, there is acceptance testing both before and after delivery, with particular attention to resolution of post-delivery problems. Delivery to an internal customer is typically formalized by a transfer agreement.

Acceptance testing and planning for integration of the product into operational use are the responsibility of the customer. In practice, the customer often shares these responsibilities with the developer. Accordingly, the development plan should identify what role the customer expects the

developer to play in acceptance and delivery. For example, pre-acceptance tests performed at JPL prior to delivery are commonly repeated after installation at the user site, with developer assistance.

As part of determining readiness for delivery, ***acceptance criteria*** are typically based on the satisfactory completion of testing, and the number and nature of the defects remaining in each category.

Section 5.10 identifies additional topics that should be addressed in the delivery and installation plan if an external organization is responsible for maintenance.

5.10 MAINTENANCE

If the developer is tasked to perform maintenance, a maintenance plan shall be prepared, defining the scope of the activity and the developer's approach. *[5.10.1] [ISO 4.4.2 guidance, 4.4, 4.19]*

If the developer is required to turn maintenance over to another organization, the development plan shall address the mechanism for transferring knowledge of the software to the maintainer. *[5.10.2]*

Guidance:

Maintenance of the software product is required to deal with both defects and modifications arising from use. Maintenance activities for software products are typically classified as:

- Problem resolution
- Functional expansion or perfo rmance improvement
- Interface modification
- Adaptation to new operational environment

Maintenance may be done by the user, by a separate organization under contract to the user, or by the developer. Occasionally, the developer arranges with the customer to integrate the designated maintenance contractor into the development team, thereby transferring a level of product knowledge that is impractical to impart via documentation.

Fixing defects and enhancing delivered functionality should both be addressed in a maintenance plan, which takes the form of a tailored development plan since maintenance is subject to the same discipline that is applied to development. Thus, typical topics in a maintenance plan are change control and the handling of change orders, configuration management of code and documents, testing, correction of defects introduced by maintenance activities, updating user and design documentation, installing new releases, and informing all users of recently discovered problems. In addition, the plan should address the support of third-party software included in the end product. Note that in cases where major enhancements are to be delivered, a revised development plan is needed.

In the case where maintenance is turned over to another organization, the developer should consider the following issues in the development plan:

- Detailed definition of the status of the software at the time maintainer assumes responsibility, including prioritization of pending changes desired by the customer/user.
- Transfer of the software tools and associated databases used in design, coding, test and integration, configuration management, installation, and the tracking of changes; training in the use of these tools should be provided by the developer.
- Identification of reports, studies, and data, previously prepared by the developer and relevant to improving product performance.
- Mechanism for familiarizing the maintainer with the architecture of the software, rationale for key design decisions, and all of the product documentation, including in-code documentation; design documentation should include design rules and coding standards.
- Identification of other developer responsibilities to support transition to maintenance.

6. REQUIREMENTS — SUPPORTING ACTIVITIES

This section, in conjunction with Section 5 that precedes it, describes the activities employed to implement the general methodology (see Section 5.1) for developing software at JPL. This section is devoted to the supporting activities that are performed throughout a life cycle.

6.1 CONFIGURATION MANAGEMENT

Configuration management procedures shall be documented and applied to deliverables: code, associated data files, and documentation. *[6.1.1] [guidance in ISO 4.8, 4.5, and 4.13.1]*

Guidance:

Configuration management provides a mechanism for identifying, controlling, and tracking the versions of each software item and associated documentation. It encompasses the process for handling change requests and change orders. Configuration management applies at all levels of a development effort and includes work in progress by individual developers as well as code that has been formally submitted to the project/task configuration management function.Versions delivered earlier and still in use must also be maintained and controlled.

Besides pertaining to deliverable code, it is highly recommended that configuration management procedures also be applied to:

- Operating systems, design and implementation tools, and development environments (especially compilers).
- Test hardware and software, including test tools (whether developed at JPL or purchased), test drivers, automated test scripts; as well as test cases, test procedures, and test data.

- Reusable software libraries, including behavioral models and mo deling tools (whether purchased or developed at JPL).

Configuration management procedures may be documented in a development plan, or a separate plan. The configuration management plan should document 1) the required activities, 2) responsibilities for c arrying out these activities, 3) tools and techniques to be used, and 4) the stage at which items will be brought under configuration control. Configuration management responsibilities typically include the following:

- Identify uniquely the versions of e ach software item, both software under development and commercial software used as development tools or integrated into the product; identify the versions of each software item which together constitute a specific build or a delivered product.

- Provide coordination for updating multiple products in one or more locations; differences in site-unique versions should be identified and tracked.

- Identify differences between controlled versions — both source code differences and differences in functionality between versions.

- Document the software and hardware used in the development environment, including version and known problems; trace software items to the operating system and development tools, so that the development environment may be accurately recreated.

- Build production software items into a linked set of modules ready for integration and test; rebuild previous development or delivered versions on request.

- Control simultaneous updating of a given software item by more than one person.

- Identify and track all actions resulting from anomaly reports and change requests, from initiation through release.

- Collect and summarize metrics to help assess the state of product development.

- Monitor and report on the status of software items, anomaly reports and change requests, and the implementation of approved changes.

- Archive the software for each delivered product, together with its associated documentation and quality records.

- Identify test status of software items under configuration mana gement. Examples of test status include untested (under development), unit test, integration test, acceptance test, defect fixing, released.

- Handle change requests and change orders.

6.2 DOCUMENTATION AND DOCUMENT CONTROL

For each development effort, the development plan shall define: *[6.2.1]*

- ***Documents to be produced, e.g., document titles, form (web, file server, hard copy), content standards or guidelines to be followed.*** *[6.2.1a]*

- ***Procedures (including responsibilities) for producing, reviewing, ap proving, and controlling documents.*** *[6.2.1b] [ISO 4.5.1]*

Documentation procedures shall address: *[6.2.2]*

- ***Which documents are subject to configuration management and at what point in the development cycle they are baselined.*** *[6.2.2a] [ISO 4.5.1 guidance]*

- ***Preparation of a master document list, or equivalent control mechanism, to identify document status, and preclude the use of invalid or obsolete documents.*** *[6.2.2b] [ISO 4.5.2]*

- ***Responsibility for approving and releasing documents, and promptly withdrawing obsolete documents from use.*** *[6.2.2c] [ISO 4.5.3, 4.5.2]*

- ***Identification of changes in released documents (to be done where practicable).*** *[6.2.2d] [ISO 4.5.3]*

- ***Approach for ensuring that the master document list (or equivalent control mechanism), as well as pertinent versions of documents, are readily available.*** *[6.2.2e] [ISO 4.5.2]*

- ***Directory/file permissions and back-up policies, where document control is achieved through electronic means.*** *[6.2.2f] [ISO 4.5.2 guidance]*

Guidance:

It is most important that the product's capabilities, its design, and the plans governing its development be communicated to all interested parties — the customer, the user, the development team, and the maintainer. The scope of the documentation activity is described in a documentation plan, which may be either a section of the development plan or a separate document.

In developing the documentation plan, it is rarely sufficient to merely assert that a particular standard will be followed, since projects typically do not conform strictly to any given standard. In virtually every instance, tailoring of documentation—whether with respect to the document set, responsibilities, content, timing for producing the documents—will be required. Tailoring should be approved by the responsible program manager. Evidence of this approval might take the form

of the program manager's signature on the plan, or a statement of approval in the form of an e-mail message or hard copy memorandum, or transfer of a document from a "Draft Documents" directory to an "Approved Documents" directory.

This *Software Development Process Description* groups software product documentation into the following five general categories:

1. Development plans, including possibly detailed sub-plans for specific activities, such as build planning, test and integration, configuration management, quality assurance, reviews, etc.

2. Requirements, which are usually derived from a statement of user needs or a concept of operations, with attention to external interfaces and system performance. Changes in requirements must be documented. Written requirements define the technical content of the agreement between developer and customer and are used as the basis for acceptance criteria.

3. Design documentation, including functional or architectural design, subsystem specification and detailed design, in-code comments, and implementation guidelines, such as design rules, naming conventions, and coding standards.

4. Test documentation, including test plans, test requirements, design of the test environment, test cases, test data descriptions, test procedures, anomaly reports, and test summary status reports.

5. User documentation, including a user's guide, an installation and operations manual, and a release description document, or transfer agreement, that documents as-delivered capabilities, unresolved problems, and recommended work-arounds.

Recommended documentation for each class of software is summarized in Appendix B. (Particular attention is paid to documentation useful for design prototyping.) In tailoring documentation, the developer should refer to Lab-wide, program office, or line organization standards. Other standards, such as *D-4000*, *EIA/IEEE J-STD-016 —Software Life Cycle Processes* are also available as guidance, and contain illustrative outlines of software documentation. [See Section 2 (References) and Table 1-1.] .

Process documentation, which includes records of both technical and management reviews, quality assurance evaluations of products and process, records of independent verification and validation, etc., and other quality records are described in Section 6.3. *JPL D-12020, D-4000 Standards Applications Guide: Documentation* gives guidance on tailoring a documentation strategy to specific project/task needs.

Following each software delivery, the project/task manager should ensure that any design documentation necessary for documenting the evolution of the design, or for satisfying contractual commitments, is archived.

A protected on-line documentation system is highly recommended to ensure that developers have the latest version of requirements and design documentation and to facilitate preparation of both review materials and deliverable documents. In practice, document control is often a responsibility of project/task configuration management.

6.3 QUALITY RECORDS

Quality records are generated to ensure that the development process is producing a reliable, robust, safe system that meets the customer's requirements. Additionally, quality records are maintained to support project/task management and demonstrate the effective operation of the quality system, as determined by periodic internal audits and *ISO-9001* re-certification.

The development plan shall identify the pertinent quality records and describe procedures for collection, indexing, filing, storage, access, maintenance, and disposition of these records. *[6.3.1] [ISO 4.16]*

Required quality records include the following *[6.3.2]*

- ***Approved changes in requirements*** *[same requirement as 5.3.7]*
- ***Review (or verification) results*** *[same requirements as 5.2.2 –Commitment Review findings, and 6.9.2]*
- ***Anomaly reports*** *[same requirement as 5.8.4a]*
- ***Checks of test tools, to evaluate whether the tools are capable of verifying the acceptability of the software product under development.*** *[same requirement as 5.8.4b]*
- ***Test results, with clear indications whether the product has passed or failed.*** *[same requirement as 5.8.4c]*
- ***Change requests/orders generated during development and provided for in the contract – after delivery*** *[6.3.2a] [based on ISO 4.9]*

The retention times for project/task quality records shall be established in the development plan in accord with program office directives, with particular attention to needs of post-delivery maintenance. *[6.3.3] [ISO 4.16]*

Quality records shall be stored in an environment conducive to the prevention of deterioration and loss, and in a manner so as to be readily retrievable *[6.3.4] [ISO 4.16]*

Pertinent subcontractor quality records shall be identified in the subcontract. *[same requirement as 6.6.2b [ISO 4.16]*

Guidance:

It is highly recommended that critical design decisions, replanning rationales, and other watershed events also be captured as quality records.

In practice, quality records for JPL will be archived in a distributed manner. To facilitate retrieval, each project/task should establish and maintain a master list of records, identifying the location and custodian of each set of records. An on-line records catalog is recommended.

6.4 MEASUREMENT

Metrics, and the associated procedures for collecting, storing, and analyzing them, shall be identified in a development plan, and shall be tailored to project needs. *[6.4.1] [ISO 4.16]*

Guidance:

It is the project manager's responsibility to identify the metrics to be collected, and to comply with any policies levied by JPL or the project sponsor(s). Metrics are chosen to track product quality, provide visibility into the development process for tracking development progress, assess readiness to deliver, and to identify problems in the development process with the objective of improving that process. *NPD 7120.4A, Program/Project Management* which is part of the Prime Contract, states that metrics should reflect technical, schedule and cost status. *NPG 7120.5A, NASA Program and Project Management Processes and Requirements* provides requirements on project management metrics. *NPD 2820.1, NASA Software Policies*[5], suggests collecting metrics pertaining to software cost and schedule baseline deviations, safety, quality, and reliability.

Recommended metrics are:

- Deviations from the staffing plan, development milestones, and budget (planned vs. actual); deviations from projected earned value should be considered.
- Comparison of planned and actual output, typically done in terms of thousands of lines of source code (KSLOC), function points, or module count; projection of output required to complete the product. An automated tool should be used for KSLOC counts.
- Analysis of anomalies or defects during development; plots of anomalies identified and corrected are customarily used during development to assess build stability, and readiness for acceptance testing and delivery. Analysis of defect data may also be used to find and correct problems in the development process.
- Number of changes in required product capabilities after the requirements were baselined.
- Elapsed time between the identification and resolution of a problem, by development phase.

In addition to taking remedial action to correct development problems, metric data should be used to establish improvement goals for both product and process. Process improvement is the responsibility of the larger organization in which a project/task resides.

6.5 TOOLS AND TECHNIQUES

Guidance:

[5] At the time of this writing, *NPD 2820.1, NASA Software Policies*, is not part of the NASA Prime Contract, effective FY99, and hence, is not actually binding on JPL. Reference is made to it in this document for guidance purposes only.

The developer employs tools, facilities, and techniques as needed to support both product development and maintenance of the quality system. Whether purchased or created by the development team, these tools should be identified in the development plan. Each program office may wish to designate recommended sets of tools for use throughout a program area. Consideration should be given to the following tools and techniques:

- Tools to support project planning and resource (e.g., budget, schedule) tracking
- Requirements management tools
- Tools supporting the selected design and implementation methodology, including CASE tools, tools to identify syntax errors, tools to monitor the use of coding standards, automated documentation tools, and design analysis tools
- Reusable software libraries
- Configuration management packages
- Automated testing tools
- Groupware, which supports electronic collaboration.

6.6 PURCHASING AND SUBCONTRACTS

Purchase orders shall clearly describe the product or service ordered, and shall be reviewed for adequacy by the developer prior to release. *[6.6.1][ISO 4.6.3]*

A development subcontract shall address *[6.6.2]*

- ***In-process verification of subcontracted development, via reviews of intermediate products and/or other oversight activities as appropriate.*** *[6.6.2a] [ISO 4.6.2b]*
- ***Identification of subcontractor quality records to be maintained.*** *[6.6.2b] [ISO 4.16]*
- ***Criteria and/or procedures for accepting subcontracted software.*** *[6.6.2c][IS O4.10.2 guidance]*

Upon receipt, the developer shall ensure that a product or service that is purchased/subcontracted, or provided by a separate development organization, conforms to specified requirements, in accordance with procedures defined in the development plan. *[6.6.3] [IS O4.6.1. 4.6.4, 4.10.2]*

Guidance:

A purchased product may be a development tool, a software or hardware item intended for inclusion in the end product (e.g., COTS), a subcontracted portion of the end product, or a service (e.g., independent validation and verification). The reader may want to refer to *http://www.sei.cmu.edu/cbs* for issues that must be addressed when using a COTS product.

Note that if the developer or customer wants to verify a subcontracted product at the subcontractor's premises, such arrangements should be specified in the contract (per *ISO 900 1* clauses 4.6.4.1 and 4.6.4.2).

Subcontractor selection should be done on the basis of the organization's ability to meet subcontract requirements, including quality requirements. JPL Procurement is responsible for selection of subcontractors/vendors and for establishing and maintaining records of subcontractor performance.

6.7 CUSTOMER-SUPPLIED PRODUCT/ REUSED SOFTWARE

The developer shall establish and document procedures for verification, storage, protection, and maintenance of items (e.g., software, data, hardware, specifications) supplied by the customer or designated third party. *[6.7.1] [ISO 4.7]*

Guidance:

In many areas, previously developed applications are routinely adapted for use in new missions. Mission and spacecraft design, on-board command and data processing, navigation, sequencing, telemetry, and operational science analysis are all examples of applications where common customer requirements may be distinguished from predictably unique needs in order to permit reuse within a relatively stable software architecture. Moreover, it is now common practice to integrate commercial-off-the-shelf (COTS) software into the end product. Cost-effective development requires that a developer continually evaluate the applicability of existing software — both for inclusion in the end product and for use as development tools. Software developers in areas exhibiting high reuse potential should consider periodic reassessments pointed at expanding the scope of common modules and identifying tools to expedite requirements definition and implementation.

Occasionally, the developer is required to use items provided by the customer or by a designated third party. Such items may include COTS software or software developed by a third party, development tools, test and operational data, interface and other specifications, and hardware. The development plan should address the support of included third-party software in planning for maintenance of the end product.

6.8 TRAINING

The scope and content of the training to be provided to project personnel (e.g., development team, user, maintainer) shall be addressed in the development plan. *[6.8.1] [ISO 4.18 and 4.1.2.2]*

Guidance:

Training may be required for three distinct groups: the development team, the user, and the maintainer (if that organization is separate from the other two). Developer training encompasses use of design methodologies, development tools, languages, etc., as well as acquiring knowledge of the problem domain in which the software product will be employed. The line organization in which the development team resides is responsible for identifying future training needs and maintaining individual records of training and experience in order to support assignment of qualified personnel to development tasks.

Training the user and product maintainer should be a consideration in creating the statement of work that documents the agreement between the customer and developer. Provision should be made for the creation of the necessary product documentation and any supplementary training materials, as well as making the development team available for both stand-up instruction and hands-on training, as required.

6.9 REVIEWS

The development plan shall define review (or verification) policies and procedures that, at a minimum, address detailed technical reviews, and identify what is to be reviewed (including critical intermediate products) and when reviews are to be held. *[6.9.1] [ISO 4.4.5, 4.4.6, 4.4.7]*

Review (or verification) results shall be maintained as quality records, and shall include a summary of requests for action and the responses thereto *[6.9.2] [ISO 4.4.6]*

Guidance:

In ***tailoring the review process***, the developer should refer to Lab-wide, program office, or line organization standards. The Lab-wide standard for reviews is defined in *JPL D-1040 , JPL Standard for Reviews*, and describes major milestone reviews (referred to as "design reviews" in *D-1040*), detailed technical reviews (i.e., working-level peer reviews), management reviews addressing status of work and resources, and other reviews. In addition, extensions and exceptions to the standard are defined and organized by program office. Other standards, such as *D-400*, ISO, and IEEE are also available as guidance. See Section 2 (References) and Table 1-1.

Reviews constitute an important, overarching activity that occurs at key points in the development life cycle. Major milestone reviews are held in conjunction with completing a development phase or sub-phase, prototyping a solution to a crucial subset of requirements, demonstrating that the product is ready for delivery, etc. Detailed technical reviews are held to determine whether a work product (e.g., requirements, subsystem and module design, code, test cases, plans, etc.) meets its completion criteria. Typically, detailed technical reviews are accomplished via walkthroughs or inspections, with the degree of rigor being tailored to the criticality of the software under development. In addition to their obvious quality control function, reviews are excellent opportunities to obtain customer input validating the planned functionality of the system. However, reviews consume scarce resources. Thus, like the philosophy applied to documentation, the effort expended in preparing for and conducting a review must be balanced against the probable rework avoided by catching errors and omissions early in the development process.

While reviews are required for Class A, B, and C software, the review policy for a project/task must be tailored to the class of software under development and to particular needs of the customer, user, the program office, and the management of the organization in which the developer resides. A development plan details how reviews will be carried out. To make the most effective use of the resources devoted to reviews, major milestone reviews should summarize, not reexamine, the topics addressed in detailed technical reviews. Major milestone reviews and detailed technical reviews should be conducted in accord with *JPL D-1040 1* and relevant program office directives. In the event of a serious deficiency, development should not proceed until the consequences are satisfactorily resolved, or the risk of proceeding has been assessed. Additional guidance on the conduct of detailed technical software reviews may be found in *D-1202 3*, *D-4000 Standards Applications Guide: Software Milestone Review* , and *D-9085, D-4000 Applications Guide: Findings and Recommendations from Case Studies of Technical Reviews.*

The reader should refer to *JPL D-1040 1* which describes many types of major milestone reviews. In general, reviews of a software task should address the following types of activities and milestones:

- Proposal or commitment review.
- Completion of user requirements.
- Completion of development plan.
- Completion of system/subsystem requirements and design (as appropriate).
- Completion of software requirements and design.
- Test readiness.
- Functional validation or pre-acceptance test.
- Pre-ship or delivery.
- Post-delivery evaluation of product and process.

Note that in accordance with *JPL D-560, JPL Standard for Safet* , it is required that safety be "included on the agenda of all formal and informal reviews associated with a project at all levels (design, delivery, test readiness, pre-ship, etc.) ", with the material to "be tailored to the nature of the review and the maturity of the project. In the case where there are no safety implications involved, a statement to that effect shall be included …" *NTS 8719.13A, NASA Technical Standard for Software Safety* also provides guidance for presenting results of safety analyses at various milestone reviews.

6.10 COST ESTIMATION

For each new development, or incremental development of an existing system, the developer shall prepare a documented cost estimate/budget that summarizes the cost of the personnel and other resources required by the development. *[6.10.1] [IS O4.4.2 guidance]*

Guidance:

Careful cost estimation and subsequent monitoring of expenditures are crucial to the success of any development effort. Preparation of a cost estimate has several purposes including:

- Verification in detail that the design concept and development approach are consistent with the customer's budget.
- Quantification of risk factors and their probable impact on development cost and schedule.
- Recording assumptions, analyses, and data that may be useful in future estimates — both cost updates and new work.

The sensitivity of the estimate should be evaluated against assumptions and risk factors. Illustrative assumptions include:

- Personnel skill and availability.
- Cost of required training in new development tools and techniques.
- Purchase of development hardware and tools, including lead times and adaptation t imes; note that tool selection can make or break a development.
- Reusability of legacy code, considering the effort required to understand and document the code, correction of defects, wrappers, redesign, etc.
- Availability of user personnel and faci lities for validation/acceptance.
- Identification of major risks facing the project/task and assessment of the potential impact of each risk on cost, schedule, and performance.
- Allocation of reserves (both budget and schedule) to absorb these risks, should they occur.

Analysis of data collected in the DSN Software Cost Analysis Database [6] shows that on DSN tasks employing phased deliveries, first deliveries experienced effort (i.e., cost of personnel) overruns up to 40%. In virtually every case, these overruns were accompanied by significant functionality

[6] Appendix C, Lessons Learned in Software Cost Estimation, *JPL D-16110, DSN Guidelines for Presenting Software Costs and Schedules at Major Milestone Reviews*

slips to later deliveries. The most common cause for inaccurate estimates (based on effort, rather than product size—see below) is omissions. *JPL D-16110, DSN Guidelines for Presenting Software Costs and Schedules at Major Milestone Reviews* provides a software development checklist that can be used in developing estimates. Activities that have been consistently reported by JPL software managers as having been underestimated include preparing for reviews, preparing documentation, completing testing, correcting anomalies, and specifying interfaces clearly. Replanning and rework should also be considered.

The better articulated the design, the better the estimate is likely to be. Thus, it is worthwhile to expend effort to define the application up front, in spite of pressure from the customer to get on with development. Cost estimates based on requirements alone can be off by 50–100% or more. A more satisfactory approach incorporates a formal definition phase in which a small, experienced team spends a limited time, say 4-6 months, developing core functional requirements, an architecture, an implementation plan, and a detailed cost estimate. The definition phase may also include prototyping of key elements, or in some cases a running prototype of the system kernel. In sum, the more detailed the design foundation, the more accurate will be the cost estimate and associated development schedule.

There are various methods for performing cost estimates, but two common approaches are basing the estimate on size, or basing it on effort estimates:

- Size: based on the size of the application (in source lines of code, function/feature points, object identification, screen counts, etc.), estimated productivity, and fully burdened salary cost. Sizing estimates often utilize rules of thumb and engineering judgment informed by analogy with prior development. Formal costing models, such as COCOMO, may be used to adjust for team skill, complexity, development tools, process maturity, learning curve, etc.

- Effort: based on a product-oriented work breakdown structure (WBS) and effort estimates for each product (e.g., documents, program sets, code units) and activities (e.g., configuration management, testing, product assurance, task management).

If time and resources permit, it is advisable to perform independent estimates — whether by two individuals, or using two different methods, such as one based on size, the other on effort. For additional information on preparing cost estimates, see *JPL D-1611 0* The reader should also become familiar with the policies and standards associated with the JPL process identified as Project Cost Estimation.

Any cost estimation method can typically be employed at various points in the life cycle; although again, the better articulated the design, the more information one has to produce a more accurate cost estimate. Whatever costing method is chosen, as development proceeds, periodic estimates of the cost-to-complete should be prepared in conjunction with normal progress tracking procedures. These estimates should be archived as supplements to the initial cost estimat e.

APPENDIX A

DEVELOPMENT PLAN TEMPLATE

1. Introduction

This development plan template is intended to serve as a checklist in identifying the issues that are useful to consider by a project, project element, or task, when developing a quality software product. As such, it is expected that this template will be used by such entities to tailor their particular development plans, in a manner that is appropriate to the characteristics of each project. (Throughout this template, **the term "project" is intended to include project elements and tasks**.) In instances where a planning topic is addressed separate from the development plan, the plan should contain a pointer to where the information can be found, such that the plan serves as a road map.

This template has been created based on *D-4000*, *EIA/IEEE J-STD-016*, and the *JPL Software Development Process Description* (the latter capturing *ISO 9001* requirements). The reader should also refer to *NPG 7120.5A, NASA Program and Project Management Processes and Requirements*, which contains an appendix describing the document contents of a project plan, as well as any other planning requirements that may be levied by the program office or line organization.

Note that development plan topics required by the SDPD are followed by a bracketed SDPD requirement number.

1.1 Identification

Full identification of the system and the software to which this document applies, including, as applicable, project name, [sub]system name, program set name, identification number(s), abbreviation(s), version number(s), and release number(s).

1.2 Project Definition Overview

a) Overview of this project, such as goals and objectives, user needs addressed, the general nature and critical functionality of the software, the system context and operational environment for the software, and deliverables. In some instances, it might be useful to specify what functionality is outside the scope of the software. *[5.4.1a]*

b) Identification of the project customer(s), e.g., sponsor(s), user(s).

c) Identification of software classification of all software being developed by the project/task, e.g., Class A, B, C, D as defined in the SDPD.

1.3 Document Overview

Scope of this document, and relationship to other plans and documents *[5.4.1k]*, including controlling documents, such as Lab-level policies and standards, or those levied by the program office or line management.

1.4 Referenced Documents

References for documents mentioned in the plan, including such information as source of document, title, document number, revision, date.

1.5 Notation

Meaning of notation(s) used (if any), e.g., font differences, brackets.

2. Project Organization

a) If part of a larger project, provide brief description of organizational context for the project.

b) Description of how personnel are organized on the project, including : *[5.4.1c]*

1) team structure,

2) roles and responsibilities (e.g., of sub-teams and their leads), including customer responsibilities,

3) technical interfaces between team members or between sub-teams. If detailed procedures are provided elsewhere within project documentation, then one could provide a high-level description here, such as for how requirements and design are communicated. Should address who has final authority for technical decisions, particularly when such decisions affect more than one sub-team.

4) technical interfaces with outside organizations, e.g., system/software engineering personnel, test personnel, configuration management personnel,

5) use of subcontractors,

6) any crucial interdependencies, e.g., critical equipment, facilities, services.

3. Work Breakdown Structure, Project Resources and Schedule

The objective of this section is to demonstrate that there are adequate resources to accomplish the project objectives. (Often, this information is provided in appendices.) Typical topics include the following:

1) work breakdown structure (WBS) describing scope of work. *[5.4.1b]* The WBS should be product-oriented (e.g., program sets, home-grown software tools, documents), but also include activities not associated with a product, such as configuration management, product assurance, management and scheduling, testing, procurement support. The WBS should provide the basis for the cost estimates, and map to the schedule.

2) cost estimate/budget that summarizes the cost of the personnel and other resources, such as hardware/software acquisitions, training, etc .*[6.10.1]*

3) staffing profile , *[5.4.1f]*

4) schedule, which should include incremental builds, deliverables, and milestones for monitoring progress and reviewing intermediate products. (Functionality provided in each build should be documented in accompanying text.) If only a high-level schedule

is provided in the plan, pointers should be provided to working-level schedules that contain frequent milestones (e.g., two-week increments). *[5.4.1d, 5.1c]*

4. Project Inputs from External Sources

a) Items required from external sources, e.g., requirements from customer or from other development efforts.

b) Procedures for validating, storing, protecting, and maintaining items (e.g., software, data, hardware, specifications) provided by the customer or designated third party .*[6.7.1]*

5. Assumptions, Constraints and Risks

a) Assumptions, e.g., user representatives to provide in-house support, any subsystem interface changes will be funded by other projects, development to proceed based on requirements as of (date).

b) Constraints, such as those levied by customer, project, institution (JPL), or one ’s own line organization, e.g., schedule, workforce, funding, use of COTS, use of inherited software, security.

c) Risk assessment, including safety issues. Some risks may be tied to assumptions and constraints, and might include availability of funding, ability to hire in a timely fashion, availability of critical facilities or equipment, availability of customer-supplied input, unfamiliar development tools, poorly defined scope. Refer to references provided under the Guidance portion of Section 5.4 for other potential risk areas, as well as approaches for identifying and characterizing risks. *[5.4.1e]*

6. Software Development Process Overview

Description of software development process from requirements to delivery, including:

1) software development approach (e.g., waterfall, spiral/iterative, rapid application development),

2) identification of life-cycle phases, along with milestones and activities to be performed during each phase , *[5.1a]*

3) phase outputs, including any documentation, *[5.1b]*

4) verification activities (e.g., reviews, demonstrations, tests) , *[5.1c]*

5) system engineering activities, including roles and responsibilities,

6) user/customer involvement in the development process,

7) reuse strategy, if any, or identification of reusable elements – both those that can be adapted from previously implemented systems, and portions of the current application that will be designed for reuse . *[5.4l]*

7. Phase-specific Activities

a) Plans and procedures for how the various activities in each life-cycle phase will be accomplished, e.g., requirements analysis, design (such as evaluation of design alternatives, tradeoff analyses, evaluation against design patterns), implementation/unit testing, integration and test [to the extent they are not described fully in the Software Development Process Overview].

b) Description of verification activities (e.g., reviews, demonstrations, tests) [to the extent they are not described fully in the Software Development Process Overview or Reviews sections]. *[5.1c]*

8. Documentation

a) In general, the documentation approach should address such issues as:

   1) how requirements, design (including key design decisions and tradeoff analyses), code, and test will be communicated, such as through formal documentation, sponsor memoranda, electronic design notes, CASE tools, in-code documentation,

   2) how as-built design will be documented,

   3) how often documentation will be updated,

   4) what documentation will be archived.

Note that on-line documentation is strongly encouraged.

b) Identification of what documents are to be produced, e.g., titles, form (web, file server, hard copy), content standards or guidelines to be followed. *[6.2.1a]* How documentation standards will be tailored should be described. (Identification of documents might alternatively be described under Software Development Process Overview or Phase-Specific Activities.)

c) Procedures and responsibilities for producing/updating, reviewing, approving, and controlling documentation *[6.2.1b]*, including:

   1) which documents are subject to configuration management, and at what point in the development cycle they are baselined , *[6.2.2a]*

   2) preparation of a master document list, or equivalent control mechanism, to identify document status, and preclude the use of invalid or obsolete documents . *[6.2.2b]*

   3) responsibility for approving and releasing documents, and promptly withdrawing obsolete documents from use, *[6.2.2c]*

   4) identification of changes in released documents (to be done where practicable), *[6.2.2d]*

   5) approach for ensuring that the master document list (or equivalent control mechanism), as well as pertinent versions of documents, are readily available . *[6.2.2e]*

   6) directory/file permissions and back-up policies, where document control is achieved through electronic means . *[6.2.2f]*

9. Reviews

a) Types of reviews (or other verification methods) to be employed (i.e., major milestone reviews, detailed technical reviews, commitment review) *[6.9.1, 5.2.1]*

b) What is to be reviewed, including critical intermediate products —such as design, code, and documentation, *[6.9.1]*

c) When reviews (or other verification methods) are to be employed *[6.9.1]*

d) Policies and procedures associated with each type of review (or verification method) *[6.9.1, 5.2.1]*, e.g.,

1) Objectives (or refer to any applicable standards, such as *JPL D-1040 1*and program office directives). Also refer to required review criteria identified in SDPD requirement 5.6.2.

2) Approach (e.g., responsibilities, timing, agenda) [or refer to any applicable standards, such as *JPL D-1040 1*and program office directives],

3) How review results will be documented, and what quality records will be maintained from the reviews. Note that it is required that a summary of requests for action, and the responses to those requests be maintained . *[6.9.2]*

10. Development Testing

If the project has separate test planning and specification documents, then this section can contain pointers to those documents. Topics that must be addressed include: *[5.8.1]*

1) Test requirements, which may be an elaboration of software requirements . *[5.8.1a]*

2) Levels of testing required up to acceptance by the customer. *[5.8.1b]*

3) Test cases, test procedures, test data and expected results. *[5.8.1c]*

4) Method of documenting test status and results. *[5.8.1d]*

5) Test environment, such as dedicated processors, test tools (purchased or developed), and user documentation. *[5.8.1e]* The actual test environment should be defined precisely enough to ensure repeatability.

6) Approach for evaluating test tools, with respect to their ability to verify the product under test (e.g., through testing, published reviews). *[5.8.1f]*

7) Procedures for correcting defects, including analyzing the cause of the defect, determining corrective action, and ensuring that the corrective action is taken . *[5.8.1g]*

Other topics that should be addressed include:

8) Types of testing required, e.g., functional tests, boundary tests, performance tests, usability tests, regression tests, etc. Regression testing is especially valuable in

incremental or iterative development and in verifying the correction of critical anomalies.

9) Procedure to be followed in categorizing and prioritizing anomalies.

10) Customer/user involvement in the testing process — including additional personnel, hardware, and travel expense if travel to the user site is required . User participation is encouraged in defining tests that closely simulate the operational environment.

11) Training required for test personnel, if not fully addressed in the development plan.

12) Schedule and staffing plan, if not fully addressed in the development plan.

11. Acceptance Testing

If not addressed as part of a delivery and installation plan or separate test plan, then the following topics must be addressed:

a) preparation of the acceptance test cases and acceptance criteria, with developer 's responsibilities (if any) noted , *[5.9.1a]*

b) description of procedures to be used in documenting and resolving problems found during acceptance testing . *[5.9.1b]*

12. Quality Planning

a) Responsibilities and procedures for identifying and correcting recurring problems in the development process (e.g., ambiguous requirements, inaccurate interface definition, ineffective walkthroughs). *[5.4.1j]*

b) Use of quality assurance personnel and their responsibilities.

c) Identification of the project/task 's quality records, associated procedures, and retention times, with quality records to include, at a minimum : *[6.3.1, 6.3.3, 6.3.4]*

1) approved changes in requirement s *[5.3.5],*

2) review (or verification) results, including a summary of requests for action and the responses thereto *[6.9.2, 5.2.2]*

3) anomaly reports (problem/failure reports)*[5.8.4a],*

4) checks of test tools, to evaluate whether the tools are capable of verifying the acceptability of the software product under development *[5.8.4b],*

5) test results, with clear indications whether the product has passed or faile d *[5.8.4c]*

6) change requests/orders generated during development and — if provided for in the contract — after delivery . *[6.3.2a]*

In addition, it is highly recommended that critical design decisions, replanning rationales, and other watershed events be captured as quality records.

13. Configuration Management

If the project has a separate configuration management plan, then this section can contain a pointer to that document. Topics to be addressed are:

a) Detailed configuration management procedures applicable to deliverables: code, associated data files, and documentation . *[6.1.1]*

In addition, it is recommended that configuration management procedures also be applied to:

1) operating systems, design and implementation tools, development environments (especially compilers),

2) test hardware and software, including test tools (whether developed at JPL or purchased), test drivers, automated test scripts, as well as test cases, test procedures, and test data

3) reusable software libraries, including behavioral models and modeling tools (whether purchased or developed at JPL),

Configuration management responsibilities typically include the following:

1) Identify uniquely the versions of each software item, both software under development and commercial software used as development tools or integrated into the product; identify the versions of each software item which together constitute a specific build or a delivered product.

2) Provide coordination for updating multiple products in one or more locations; differences in site-unique versions should be identified and tracked.

3) Identify differences between controlled versions — both source code differences and differences in functionality between versions.

4) Document the software and hardware used in the development environment, including version and known problems; trace software items to the operating system and development tools, so that the development environment may be accurately recreated.

5) Build production software items into a linked set of modules ready for integration and test; rebuild previous development or delivered versions on request.

6) Control simultaneous updating of a given software item by more than one person.

7) Identify and track all actions resulting from anomaly reports and change requests, from initiation through release.

8) Collect and summarize metrics to help assess the state of product development.

9) Monitor and report on the status of software items, anomaly reports and change requests, and the implementation of approved changes.

10) Archive the software for each delivered product, together with its associated documentation and quality records.

11) Identify test status of software items under configuration management. Examples of test status could include development, unit test, integration test, acceptance test, defect fixing, released.

12) Handle change requests and change orders.

b) Responsibilities for carrying out the configuration management activities.

c) Tools and techniques to be used.

d) The stage at which items will be brought under configuration control.

e) Change control procedures for documenting, reviewing, approving, and communicating requirements changes to all affected parties .*[5.4.1g]*

f) Change control procedures for documenting, reviewing, approving, and communicating design changes before their implementation . *[5.4.1h]*

14. Development Standards

a) Identification of any standards that apply to the project, whether defined by the project, program office, customer, or JPL.

b) Rules, practices and conventions to be used in development, including coding standards and design rules.

c) Standards and procedures pertaining to traceability, requirements management, and other configuration management activities. Such standards and procedures could be addressed in the Configuration Management section of the development plan or in a separate configuration management plan. Similarly, documentation, review, and test standards and procedures could be addressed in the same manner.

15. Metrics

Identification of metrics, and the associated procedures (and responsibilities) for collecting, storing, and analyzing them . *[6.4.1]*

Metrics are chosen to track product quality, improve software development processes, and to manage project resources. See Section 6.4 of the SDPD for examples of different types of metrics.

16. Tools, Methods, and Environments

a) Identification of any tools and methods used by the project/task, such as for:

1) project planning and resource (e.g., budget, schedule) tracking,

2) requirements management,

3) design and implementation, including CASE tools, tools to identify syntax errors, tools to monitor the use of coding standards, automated documentation tools, and design analysis tools,

4) configuration management,

5) automated testing,

6) groupware, which supports electronic collaboration,

7) reusable software libraries,

8) maintenance of the quality system (e.g., managing and tracking quality records).

Note that although testing and configuration management tools are included in this list, they would likely be addressed in their respective sections of the development plan. In such a case, this Tools, Methods, and Environments section should contain a pointer to those sections.

b) Description of development, test, and target environments, and if pertinent, any issues that need to be addressed arising from differences between the environments. Include a discussion of how the development and test environments will be established, and administered. Also describe the environment in which other engineering activities are conducted, e.g., project/task planning and scheduling, requirements analysis, design.

c) System administration plan, including approach to back-up/archiving, security, and virus protection. *[5.4.1i]*

17. Procurements

a) Identification of procurements, e.g., subcontracted development, development platforms, development tools, COTS to be integrated into the end product.

b) Policy/procedures governing products or services that are subcontracted, purchaed, or provided by a separate development organization, including procedures for ensuring the product/service conforms to requirements. *[6.6.3]*

18. Training

Training needs unique to this task, including scope and content of training (e.g., problem domain, development methods, development tools) to be provided to project personnel, such as development team, users, and maintainers . *[6.8.1]*

19. Delivery and Installation

a) If the project has a separate delivery and installation plan, then this section can contain a pointer to that document. Topics to address include:

1) description of procedures to be used in documenting and resolving problems found after delivery, *[5.9.1b]*

2) details of delivery and installation logistics, e.g., arranging for use of customer/user facilities and personnel in installation and test , *[5.9.1c]*

3) definition of developer's role (if any) in supporting transition to full operational use of the product, *[5.9.1d]*

4) identification of documentation to be delivered at installation, including installation and configuration procedures , *[5.9.1e]*

5) identification of training for the user and/or system administrator/ operator ,*[5.9.1f]*

6) a schedule for key events pertaining to delivery and installation , *[5.9.1g]*

7) storage of archived software media to prevent deterioration and facilitate disaster recovery *[5.9.1h]*

8) virus protection of software designated for delivery, during storage and electronic transmission . *[5.9.1i]*

b) If not addressed elsewhere in this development plan or a separate test plan, then the following topics should be addressed in the delivery and installation plan:

1) preparation of the acceptance test cases and acceptance criteria, with developer ’s responsibilities (if any) noted , *[5.9.1a]*

2) description of procedures to be used in documenting and resolving problems found during acceptance testing . *[5.9.1b]*

c) If the developer is required to turn maintenance over to another organization, the following topics should be considered:

1) detailed definition of the status of the software at the time the maintainer assumes responsibility, including prioritization of pending changes desired by the customer/user,

2) transfer of the following to the maintainer: software tools and associated databases used in design, coding, test and integration, configuration management, installation, and the tracking of changes; training in the use of these tools,

3) identification of reports, studies, and data previously prepared by the developer, and relevant to improving product performance,

4) mechanisms for familiarizing the maintainer with the architecture of the software, rationale for key design decisions, and all of the product documentation, including in-code documentation, design rules and coding standards,

5) identification of other developer responsibilities to support transition to maintenance.

20. Maintenance

Definition of the scope of the maintenance activity, and the developer ’s approach to maintenance. *[5.10.1]* If the project/task has a separate maintenance plan, then this section can contain a pointer to that document. Topics to consider include:

1) change control and the handling of change orders for enhancing delivered functionality and fixing defects,

2) configuration management of code and documentation,

3) testing,

4) correction of defects introduced by maintenance activities,

5) updating user and design documentation,

6) installation of new releases,

7) mechanism for informing users of recently discovered problems.

Note that if maintenance is to be handled by an external organization, then the maintenance topics identified in the delivery and installation section should be addressed.

21. Plan Updates

Provisions for updating the development plan and any associated plans, as development proceeds. *[5.4.1m]*

22. Variances

Explanations for any deviations made from SDPD requirements . *[5.4.1n]*

Appendices

A. Acronyms (used in the development plan)

B. Glossary of Terms (used in the development plan)

APPENDIX B

RECOMMENDED PRODUCT DOCUMENTATION

This SDPD groups software product documentation into five general categories:

1. Development plans, including possibly detailed sub-plans for specific activities, such as build planning, test and integration, configuration management, quality assurance, reviews, etc. A plan permits concurrent efforts by breaking the job into pieces and defining interfaces; it also helps management monitor progress. Inadequate planning risks the ineffective use of scarce development resources.

2. Requirements, which are usually derived from a statement of user needs or a concept of operations, with attention to external interfaces and system performance. Changes in requirements must be documented. Written requirements define the technical content of the agreement between developer and customer and are used as the basis for acceptance criteria. If requirements are poorly done, the developer risks building the wrong thing, or permits differing interpretations of requirements by the development team. A clear statement of prioritized user needs that will be addressed is especially helpful if the customer 's technical representative changes while development is in progress.

3. Design documentation, including functional or architectural design, subsystem specification and detailed design, in-code comments, and implementation guidelines, such as design rules, naming conventions, and coding standards. Functional design allocates functions to hardware, software, and human operators with a view to long-term evolution of the system. If functional design is done poorly, future enhancements may be difficult to accommodate without expensive redesign/re-implementation. Additionally, an accurate top-level view of the system can help both new developers and maintainers understand the design quickly, thus reducing train-up time.

   Design documentation breaks the implementation into cohesive pieces that can be coded and tested concurrently. Interface definition is especially important, for experience shows that it is here where many integration problems arise. Good design documentation, including the design rationale and in-code comments, is vital to both the maintainer and to a development team member who must complete or modify a module designed by someone else. Poor design documentation makes maintenance expensive and can lead to re-implementation of a poorly understood piece of code.

   If the maintainer is different from the developer, additional effort may be devoted to design documentation. In addition to documentation of the as-built design, maintenance documentation should include design rules, naming conventions, coding standards, test cases, use of test tools, etc. Inadequate maintenance documentation can make this activity quite costly.

4. Test documentation, including test plans, test requirements, design of the test environment, test cases, test data descriptions, test procedures, anomaly reports, and test summary status reports. Development testing is based on documented design requirements. A test plan identifies test preparation work that can proceed in parallel with development. It also identifies

the required testing environment, permitting timely acquisition of hardware and software, and stimulates a reexamination of requirements to ensure they are verifiable. Inadequate test planning risks delayed initiation of system test, leading to schedule compression, late delivery, or delivery of a product that does not meet customer expectations.

Software test reports are necessary to ensure that discovered anomalies/defects can be reproduced independently by the assigned developer. Analysis of test reports can also baseline performance and help spot defects in the design and implementation process. Documentation of individual anomalies is essential if someone else has to fix them; summary reports of test status help management gauge build stability and thus, readiness to deliver or to proceed to the next development phase.

5. User documentation, including a user's guide, an installation and operations manual, and a release description document, or transfer agreement that documents as-delivered capabilities, unresolved problems, and recommended work-arounds. A user's guide and operator's manual are essential if someone other than the developer will use the system. Inadequate user documentation constrains the productive employment of the system, leading to costly train-up effort and ultimately to ineffective use of delivered capabilities. An inadequate release description hampers the user in understanding what the delivered system can and cannot do, and how to cope with known problems.

Exhibit B-1 identifies recommended documentation for classes A, B, and C of the four software classes defined in Section 1.0 and repeated below. Class D software is excepted from the requirements documented in the SDPD. However, some of the Class C requirements may be useful in defining and implementing this fourth class of software. Note that a system or subsystem can encompass software that falls in more than one category. In such a case, it is the responsibility of the project/task manager to identify the software elements that fall in each class and to tailor the development plan accordingly.

Class A: Mission-Critical: Flight or ground software that is necessary either to assure mission success, or if it does not function as specified, that could cause loss of spacecraft, seriously degrade the attainment of primary mission objectives, or cause injury to humans or flight hardware. Examples of serious degradation of mission objectives include loss of a mission-critical event, loss of science return from multiple instruments, or loss of a large fraction of the engineering telemetry data.

Class B: Mission Support: Flight or ground software that is necessary for the science return from a single (non-critical) instrument, or supports the timely generation of mission sequences, or is used to process or analyze mission data, or other software for which a defect could adversely impact attainment of some secondary mission objectives or cause operational problems for which potential work-arounds exist. Examples of Class B include software that supports pre-launch integration and test, mission data processing and analysis, analysis software used in trend analysis and calibration of flight engineering parameters, or software employed by the Network Operations and Control Center (which is redundant with systems used at the tracking complexes). Class B software must be developed carefully, but validation and verification effort is generally less intensive than for Class A.

Class C: Development Support: Software developed to explore a design concept; or support software development functions, such as requirements management, design, test and integration, configuration management, documentation, etc.; or perform engineering data analysis. A defect in Class C software may cause rework but has no direct impact on mission objectives or system safety. Class C software is often used by several people in addition to the developer(s), and by its nature, can impact the quality of delivered products. Documentation and review of Class C software should be tailored to its intended use, with attention to long-term maintenance needs and to evolution of a design prototype into operational flight or ground software. Note: **Development tools that can introduce critical defects in Class A or B software must be regarded as belonging to the same class as the operational software.**

Class D: Non-deliverable software developed to meet a research objective or support individual engineering efforts. Generally, Class D software is intended for use only by the individual who developed it.

Recommendations for Class A documentation refer to document outlines published in section 4.0 of the Level II standards from the *JPL Software Management Standards Packag* (*D-400 0*). Equally good outlines may be found in *EIA/IEE J-STD-01 6 Software Life Cycle Processe* . *s* Identification of separate documents has no implications for packaging; this is up to the project/task, as well as decisions on format and media. On-line documentation is encouraged, taking maximum advantage of intermediate engineering products.

Documentation must be tailored to the needs of each development, keeping in mind that the ultimate objective is communication of a body of information to a particular audience. Typically, the readers of a document are not those who wrote it. Thus, the amount of detail provided, and the effort spent on organizing the information to make it easily assimilated, are judgments to be carefully made. Those responsible for designing project/task documentation must balance the effort expended against the risk of incomplete communication and the probable cost impacts. *JPL D-1202 0*, a *D-400 0* planning guide for documentation, describes how selected JPL projects have designed software documentation to strike this balance.

EXHIBIT B-1: Recommended Product Documentation, by Software Class

Note: Documents from *D-400 0* are cited to provide a common nomenclature and identify content. *EIA/IEE E J-STD-016, Software Life Cycle Processes* has excellent examples of document content also. Identification of separate documents has no implications for packaging; this is up to the project/task, as well as decisions on format and media. On-line documentation is encouraged, taking maximum advantage of intermediate engineering products.

Document Type	Class A — Mission Critical	Class B — Mission Support	Class C — Development Support
Development Plan	All *D-400 0* topics; emphasizes build planning, test & integration, configuration mgt., reviews, independent QA	Similar to Class A, tailored to task complexity and needs for reviews and quality assurance	Product definition (objectives.), task breakdown, schedule with review milestones
Requirements	Testable Software Requirements (SRD) traceable to user needs; Interface Specification (SIS-1); change orders documented	Testable requirements based on user needs; interface spec.; design concept or operational concept may replace user need	Written requirements detailed enough to document agreement with customer and guide developers; changes documented
Design	Functional Design (FDD) Detailed Design (SSD-1,2) Interface Design (SIS-2) In-code documentation, adhering to design/coding standards	Similar to Class A, tailored to needs of development and post-delivery support; build natural work products into delivered documentation	Functional design/architecture plus design details required by a new developer; commented code remains essential
Test	Test Plan and Requirements (STP-1), Procedures (STP-2), and Reports (STP-3); procedures must define configuration to ensure repeatability	Similar to Class A, tailored to software complexity and task needs; consider testing needs of maintainer	Integration & test plan may be a part of build plan for complex applications; retrievable test procedure; optional anomaly reports
User	User's Guide/Software Operator's Manual (UG/SOM) Transfer Agreement/Release Description Document (RDD)	User documentation (may be embedded); installation and operations guide; RDD includes liens and work-arounds	User and maintenance documentation as required

APPENDIX C

TRACE OF *ISO 900 1* REQUIREMENTS TO THE *SOFTWARE DEVELOPMENT PROCESS DESCRIPTION*

ISO 900 1 Requirement	Response of *Software Development Process Description* (SDPD)
4.1.1 Quality policy	Software interpretation of JPL Quality Policy (4.1)
4.1.2.1 Responsibility and authority of those who manage, perform, and verify work affecting quality to be documented (4.1.2.1)	Project organization and technical interfaces to be documented (5.4.1c)
4.1.2.2 Resource requirements	Cost estimate/budget required (6.10.1)
4.1.2.3 Management representative	Not applicable at this level
4.1.3 Management review	Software development organizations to participate in Lab-wide internal audits (4.3)
4.2.1, 4.2.2 Documentation and implementation of quality procedures	The SDPD documents quality procedures for software, sets requirements for development planning, and describes good development practice. Tailoring of the SDPD requirements to individual projects or tasks is the responsibility of project/task managers and those providing oversight of development.
4.2.3 Quality Planning	Incorporated throughout SDPD requirements, but with focus on development plan (5.4). Section 5.5 emphasizes need for quality planning.
4.3.1 Procedures for contract review 4.3.2 Review of contract for adequacy and clarity of requirements; adequacy of development resources	Commitment review to ensure that scope of work is adequately defined, and adequate resources are provided (5.2.1a-e)

4.3.3 Contract amendment. Mechanism to alter scope of work, and changes to be transferred to those concerned	Development plan to address requirements changes and correction of post-delivery defects (5.2.1b). Software requirements specification subject to change control (5.3.4). Requirements changes to be communicated to affected parties (5.4.1g)
4.3.4 Records of contract reviews	Commitment review findings to be documented (5.2.2)
4.4.1 Documented procedures to control and verify design	Design and development procedures are documented throughout SDPD requirements.
4.4.2 Design and development plans that: - describe activities - assign responsibilities - identify adequate personnel/resources - provide for updating	Required development plan: - identifies project life cycle (5.1a-c) - defines scope of work and activities (5.4.1b) - includes a budget (6.10.1) and staffing plan (5.4.1f) - provides for updating as development progresses (5.4.1l) Requires plan for requirements, design, and implementation activities (5.6.1, 5.7.1) Requires plan for integration and test activities (5.8.1) Requires plan for delivery, installation, and acceptance activities (5.9.1) Requires plan for maintenance activities, if within project scope (5.10.1)
4.4.3 Organizational and technical interfaces to be defined	Project organization and technical interfaces to be documented in development plan (5.4.1c)
4.4.4 All applicable design input requirements: - identified and documented - reviewed to resolve incomplete, conflicting, ambiguous statements - revised to reflect contract review results	Customer's requirements to: - be written (5.3.1) - be reviewed to ensure adequate definition, resolution of ambiguous or conflicting requirements, , testability (5.3.3a-c) - address commitment review concerns (5.3.4)

4.4.5 Design output to: - be documented to permit verification and validation against input requirements - reference acceptance criteria - identify characteristics crucial to safe and proper function of product Design documents to be reviewed before release	Design to be documented and reviewed with respect to requirements, acceptance criteria, verifiability, and safety issues (5.6.2) Requirements to be stated so as to permit validation during acceptance tests (5.3.3c) Design and implementation activities to be associated with reviews tailored to project/task needs (5.6.1, 5.7.1) Development plan to document policy for milestone and detailed technical reviews (6.9.1) Design to be reviewed prior to release (5.6.2)
4.4.6 Design review to be done at appropriate points Participants to include pertinent designers and outside specialists, as required Review records to be maintained	Design and implementation activities to be associated with reviews tailored to project/task needs (5.6.1, 5.7.1) Review policies and procedures to be documented (6.9.1) Review results and response to Requests for Action to be maintained as quality records (6.9.2)
4.4.7 Design verification to be performed at appropriate points and results recorded	Reviews (or verification) policies and procedures to be described in development plan (6.9.1) Test results to be maintained as quality records (5.8.4c)
4.4.8 Design validation to ensure product conforms to defined user needs and/or requirements	Developer to validate product under conditions similar to user's application environment (5.8.2) Requirements to be stated so as to allow validation during product acceptance (5.3.3c) Acceptance test cases and criteria to be prepared (5.9.1a)

4.4.9 Design changes to be documented, reviewed and approved before implementation	Documented change control procedures required to document, review and approve requirements and design changes (5.4.1g,h) Quality records required for change requests/orders (6.3.2a)
4.5.1 Documented procedures for document and data control	Procedures for producing, reviewing, approving and controlling documents (6.2.1b) Procedures to identify documents to be subject to configuration management (6.2.2a) Documented change control procedures required to document, review and approve requirements and design changes (5.4.1g,h) Documented configuration management procedures required for deliverable data files and documentation (6.1.1)
4.5.2 Documents reviewed and approved Master list (or equivalent) used to preclude use of invalid and/or obsolete documents; must identify current revision status, and be readily available Controls to ensure: - pertinent documentation available where needed - obsolete documents removed promptly and archived as required 4.5.3 Document changes reviewed and approved by those who did it initially Nature of the change identified in the document or attachment	Preparation of master document list (or equivalent) to identify documentation status, and preclude use of invalid or obsolete documents (6.2.2b) Responsibility for approving and releasing documents, and promptly withdrawing obsolete documents from use, to be identified (6.2.2c) Defined approach for ensuring master document list (or equivalent) and documents are readily available (6.2.2e) Procedures to address directory/file permissions and back-up policies where document control is achieved through electronic means (6.2.2f)
4.6.1 Documented procedures to ensure that purchased product meets requirements	Developer to ensure purchased/subcontracted product or service conforms to specified requirements (6.6.3)

4.6.2 Evaluation of subcontractors. Developer (or agent) to: - select subcontractors on basis of capability to meet requirements - define controls on subcontracted work - maintain records of acceptable subcontractors	JPL Procurement to make subcontractor selection and maintain subcontractor records Subcontract to address in-process verification of subcontracted development
4.6.3 Purchasing documents to describe clearly work to be done or article(s) to be purchased Developer to review and approve purchasing documents before release	Purchase orders to clearly describe product or service ordered; developer to review them for adequacy prior to release (6.6.1)
4.6.4 Purchased product verification at subcontractor's site to be documented in purchase order Developer's customer also entitled to on-site subcontractor verification, without relieving developer of contractual responsibilities	Not required in SDPD, as location of product verification is contingent on subcontract. Guidance on Purchasing and Subcontracts (6.6) directs developers to include such language in a subcontract when applicable.
4.7 Documented procedures for control, verification, maintenance, and storage of customer-supplied product Customer to be informed of unsuitable items	Documented procedures required for verification, storage, protection, and maintenance of items (e.g., software, hardware, data, specifications) supplied by customer or designated third party (6.7.1)
4.8 Product identification and traceability. Documented procedures for identifying product during all stages of design, production, delivery, and installation Traceability of individual product or batches required where appropriate; documentation becomes part of quality record.	Documented configuration management procedures required for deliverables (code, data files, documents) (6.1.1)

4.9 Process control. Production, installation, and servicing processes to be planned to ensure execution under controlled conditions, to include: - documented procedures - use of suitable equipment and tools - compliance with standa rds - monitoring of process parameters and product characteristics - approval of processes and equipment - criteria for workmanship - equipment maintenance When it is not feasible to test process output, qualified operators are required Personnel qualifications documented, and records maintained for special processes	*ISO 900 1* paragraph 4.9 largely refers to replication of software products, but according to *ISO 9000- 3* also encompasses product release issues (as follows) Approach to back-up/archiving, security, and virus protection to be documented (5.4.1i) Plan for delivery and installation to address: - delivery and installation logistics - definition of developer's role in supporting transition to full operational use - documentation to be delivered at installation
4.10.1 Documented inspection and test procedures to verify requirements are met. Inspection and test records identified in quality plans or procedures	Integration and test activities to be performed in accord with test planning and specification documentation (5.8.1) Test records required by SDPD (5.8.4); quality records to be identified in development plan (6.3.1)
4.10.2 Receiving inspection and test . Incoming product to be verified prior to use, taking into account control exercised by vendor. Traceability required if released for production use prior to verification	Criteria and/or procedures for accepting subcontracted software to be addressed in development subcontract (6.6.2c) Documented procedures required to ensure that product or service that is purchased/subcontracted, or provided by a separate development organization, conforms to specified requirements (6.6.3)
4.10.3 In-process inspection and test to be performed in accord with documented procedures and completed prior to product release; all inspections and tests must be completed even if product is prematurely released under positive recall	Implementation activities to be guided by plans and reviews (5.7.1) Integration and test activities to be performed in accordance with test planning and specification documentation (5.8.1)

4.10.4 Final inspection and test to conform to documented procedures and be completed before product release to verify that product meets requirements	Before delivery and acceptance, developer to validate product (5.8.2) Acceptance test cases and criteria to be prepared (5.9.1a)
4.10.5 Inspection and test records required as evidence; records to clearly show whether product has passed or failed, according to acceptance criteria; inspection authority for product release to be identified Non-conforming product to be handled in accord with 4.13.	Test results to be maintained as quality records; test results to clearly indicate that product has passed or failed. (5.8.4c) Acceptance test cases and criteria to be prepared (5.9.1a)
4.11.1 Documented procedures to control, calibrate, and maintain inspection and test equipment, including software. Test hardware and software checked to ensure they can verify product acceptability Technical data on test equipment to be made available to customer, if required 4.11.2 Required control procedures (primarily applicable to calibrating test hardware)	All test tools verified prior to use (5.8.1f) *ISO 9000-3* states that calibration is not directly applicable to the developed software.
4.12 Product to be labeled to identify clearly the inspection and test status throughout production in order to ensure that delivered product has been adequately tested	Method of documenting test status and results to be defined in plan (5.8.1d)
4.13.1 Documented procedures to ensure that non-conforming product is prevented from unintended use	Documented procedures required for correcting defects (5.8.1g) Documented configuration management procedures required (6.1.1)

4.13.2 Responsibility for disposition of non-conforming product defined Non-conforming product to be reworked, scrapped, or accepted by concession with no repair; actual condition to be noted at delivery Repaired/reworked product to be re-inspected	Documented procedures required for correcting defects (5.8.1g) Missing functionality in delivered software to be documented (5.8.3) Anomaly reports (or problem/failure reports) to be maintained as quality record (5.8.4a)
4.14.1 Documented procedures for taking corrective and preventive action, including consequent changes in procedures.	See SDPD responses to 4.14.2 and 4.14.3.
4.14.2 Scope of corrective action procedures (for individual product or project): - handling customer complaints - investigation of defect root causes - determination of corrective action, including controls to ensure effectiveness	Documented procedures required for correcting defects (5.8.1g) Documented procedures required for documenting and resolving problems found following installation — whether during acceptance testing or delivery (5.9.1b)
4.14.3 Scope of preventive action procedures (for development process) - analysis of quality records across functions and products - determination of preventive actions, including controls to ensure effectiveness - inform management representative	Development plan to define responsibilities and associated procedures for correcting recurring problems in the development process for a project/task (5.4.1j)

4.15 Documented procedures for handling, storage, packaging, preservation, and delivery	Plan required for delivery and installation (5.9.1) Approach for handling virus protection of software designated for delivery, during storage and electronic transmission (5.9.1i) System administration plan to include approach to back-up/archiving, security, and virus protection (5.4.1i) Plan to address storage of archived software media to prevent deterioration and facilitate disaster recovery (5.9.1h)
4.16 Documented procedures required to identify, collect, index, and store quality records that demonstrate the effectiveness of the quality system Pertinent subcontractor records to be included Records to be legible and stored in a fashion to be readily retrievable, protected against loss, and provided to a customer, as agreed Record retention times to be specified	Development plan to identify project/task quality records and describe procedures for collection, indexing, filing, storage, maintenance, and disposition (6.3.1) Pertinent subcontractor records to be included (6.6.2b) Quality records to be stored in environment conducive to prevention of deterioration and loss, and in a manner so as to be readily retrievable (6.3.4) Retention times for quality records to be established in accord with program office directives, considering needs for post-delivery maintenance. (6.3.3) Minimum set of quality records identified (6.3.2)
4.17 Documented procedures required for planning and implementing internal quality audits to assess effectiveness of the quality system	500 organization's responsibility to establish procedures. Software Development Process Owner to participate.

4.18 Documented procedures required for identifying and satisfying training needs Task assignments to be based on education, training, and experience Training records to be maintained	Scope and content of training to be provided to project personnel (e.g., development team, user, maintainer) to be addressed in plan (6.8.1) JPL line organization responsible for task assignments, for maintaining training records, and identifying future training needs
4.19 Documented procedures required for servicing when this is a requirement	Maintenance plan required if JPL is tasked to perform this function. (5.10.1)
4.20.1 Identification of the need for statistical techniques for controlling and verifying both process capability and product characteristics 4.20.2 Documented procedures required to implement and control application of statistical techniques	Metrics, and associated procedures, to be identified in the development plan, and tailored to project/task needs (6.4.1)