SOFTWARE FORMAL INSPECTIONS GUIDEBOOK

National Aeronautics and
Space Administration
Washington, DC 20546

Approved: August 1993

FOREWORD

The Software Formal Inspections Guidebook is designed to support the inspection process of software developed by and for NASA. This document provides information on how to implement a recommended and proven method for conducting formal inspections of NASA software.

This Guidebook is a companion document to NASA Standard 2202-93, Software Formal Inspections Standard, approved April 1993, which provides the rules, procedures, and specific requirements for conducting software formal inspections. Application of the Formal Inspections Standard is optional to NASA program or project management. In cases where program or project management decide to use the formal inspections method, this Guidebook provides additional information on how to establish and implement the process.

The goal of the formal inspections process as documented in the above-mentioned Standard and this Guidebook is to provide a framework and model for an inspection process that will enable the detection and elimination of defects as early as possible in the software life cycle. An ancillary aspect of the formal inspection process incorporates the collection and analysis of inspection data to effect continual improvement in the inspection process and the quality of the software subjected to the process.

The software formal inspection process supports NASA Management Instruction (NMI) 2410.10B, NASA Software Management, Assurance, and Engineering Policy, effective April 20, 1993.

This document has been prepared under the direction of the personnel shown on the following page. Specific questions concerning this publication should be referred to one of them, while general questions should be referred to the Office of Safety and Mission Assurance, NASA Headquarters, Washington, D.C. 20546.

Original signed by

Charles W. Mertz
Acting Associate Administrator for Safety and Mission Assurance

NASA-GB-A302

SOFTWARE FORMAL INSPECTIONS GUIDEBOOK

Approvals

(Original Signed by)

Lawrence E. Hyatt
Manager, Software Assurance Technology Center

(Original Signed by)

Alice Robinson
Manager, NASA Software Assurance Program

(Original Signed by)

Dr. Daniel Mulville
Director, Engineering Division

TABLE OF CONTENTS

Note: The following items are not available in this electronic version:

I. GENERAL

Software formal inspections are in-process technical reviews of a product of the software life cycle conducted for the purpose of finding and eliminating defects. Formal inspections may be applied to any product or partial product of the software development process, including requirements, design, and code. Formal inspections are embedded in the process of developing products and are done in the early stages of each product's development.

Formal inspections were developed by Michael Fagan at IBM to improve software quality and increase programmer productivity. As such, the formal inspection process involves the interaction of the following five elements:

- Well-defined inspection steps
- Well-defined roles for participants (moderator, recorder, reader author, inspector)
- Formal collection of process and product data
- The product being inspected
- A supporting infrastructure

The formal inspection process was designed to help the developing organization produce a better product. The process also provides other advantages. As defects are found and fixed the quality of the product increases. A more technically correct base is available for the each new phase of development. The overall software life cycle cost is lower since defects are found early and are easier and less expensive to fix. The effectiveness of the test activity is increased and less time may have to be devoted to testing the product. Another important benefit of formal inspections is the immediate evaluation and feedback to the author from his/her peers which will bring about improvements in the quality of future products.

This guidebook describes the formal inspection process, shows where formal inspections fit during each life cycle phase, and provides some suggestions on how a formal inspection program may be started and improved. It is intended as a tutorial introduction to the inspection process. The Software Formal Inspection Process Standard, NASA-STD-2202-93 should be cited and used where formal definition of the process is needed, such as in process requirements and contracts.

II. CONCEPTS AND DEFINITIONS

A software formal inspection is a defect detection, defect elimination, and correction verification process carried out by a small group during the development life cycle. A defect is any error, nonconformance, or failure to satisfy a requirement in a product. The goal of formal inspections is to ensure that defects are fixed early in the life cycle rather than late, when they are harder to find and more costly to fix. Formal inspections are held throughout the software development life cycle phases to correct the products and to provide a short feedback loop for authors.

Formal inspections are distinguished from other types of peer reviews in that they follow a well-defined, tried, and proven process for evaluating, in detail, the actual product or partial product with the intent of finding defects (see Section III). They are conducted by individuals from development, test, and assurance, and may include users. Formal inspections are more rigorous and well-defined than other peer review processes such as walkthroughs, and significantly more effective. Inspections do not take the place of milestone reviews, status reviews, or testing.

Table 1 provides a list of candidate work products for formal inspections. The first category, Typical Work Products, indicates the most likely candidates for inspections. The designations in parentheses are commonly used by Fagan and others and are included for reference. The second category in Table 1, Additional Candidate Work Products, shows that virtually any product that has requirements, constraints, and guidelines can be examined by using the formal inspection process.

Inspections should be used to judge the quality of the software work product, not the quality of the people who create the product. For this reason, managers should neither participate in nor attend the inspection meeting. In addition, the results of inspections should be presented to management either statistically or as results for groups of products. This grouping of results will show managers the value of the inspection process without singling out any author. Using the inspection process to judge the capability of authors may result in less than honest and thorough results; that is, co-workers may be reluctant to identify defects if finding them will result in a poor performance review for a colleague.

Formal inspections are performed by a team that may be comprised of a moderator, reader, recorder, author, and other inspectors. The actual team size should consist of four to seven persons, although a minimum of three is acceptable. Each team member has a specific, defined role. In addition, it is the responsibility of the entire inspection team to find and report defects; therefore, all members of the team are considered inspectors. The moderator's primary responsibility is to accomplish a good

inspection. In addition, the moderator selects team members, prepares the team for the inspection, conducts the inspection meeting, and reports on the results. The reader's responsibility is to guide the team through the work product. The recorder's responsibility is to accurately report each defect during the inspection meeting. The author's responsibility is to answer inspectors' questions and, after the inspection meeting, correct found defects. In addition, support may be provided to the inspection process by the project library, which may assist the moderator in keeping the status and statistics of formal inspections. Refer to Section IV for more extensive information on each role.

Entrance and exit criteria are used to determine if a product is ready to be inspected (entrance) and has successfully completed the process (exit). Entrance criteria to be satisfied depend on the product, but generally involve a determination that the product is mature enough to be used as is after defects are removed. For example, entrance criteria for a code inspection are usually that the code has been compiled successfully and run through any static analyzers used by the project, such as an automated standards checker. It should not, however, have been tested. Exit criteria are used mainly to assure that major defects found during the inspection process have been corrected. Correction of minor defects, those that will have no impact on the use of the product, may not be required by exit criteria. All work on the product to be inspected should cease once it has been submitted for inspection, otherwise the purpose of the inspection process will be defeated. Thus, the inspection must be scheduled in a timely manner.

The formal inspection process and its results are supported and documented by a set of forms. Some of the forms that may be used are: Inspection Announcement, completed by the moderator, that notifies participants of the inspection date, time, location, and other important information; Preparation Logs, completed by each inspector, that list defects found and time spent in preparation; and an Inspection Defect List, completed by the recorder, to provide information on each defect. In addition, the moderator should complete a Detailed Inspection Report at the end of each inspection. Other useful forms include checklists that provide guidance for inspectors in finding possible defects, and the Inspection Summary Report that summarizes data found during the inspection. Refer to Appendix A for sample checklists, and Appendix B for sample inspection forms.

III. THE FORMAL INSPECTION PROCESS

In order to use the formal inspection technique to its fullest extent, i.e., to increase product quality while maintaining cost effectiveness, it is important to follow the procedures that have been tested and refined. A formal inspection is accomplished through a series of steps or stages using a trained inspection team. Team members include the moderator, reader, recorder, author, and other inspectors. Figure 1 is a chart of the formal inspection process and team members.

The seven stages that comprise the inspection process are described briefly below. In addition, if reinspection is required, several stages may be repeated if a high number of defects were found during the inspection or if defects required complex corrections. Determination of the need for reinspection is done by the inspection team at the end of the inspection meeting.

Planning	The period of time used to determine whether the product to be inspected meets the entry criteria, plan the inspection, select the team, assign roles, schedule the inspection meeting, and prepare and distribute the inspection forms materials. In addition, a determination is made on whether to hold an overview.
Overview	The optional stage in which the inspection team is provided with background information for the inspection. This stage may not be necessary if the team is already familiar with the product being inspected.
Preparation	Key stage in which members of the inspection team prepare individually for the inspection by reviewing and finding potential defects in the product being inspected. Potential defects found by individuals are discussed during the actual inspection meeting.
Inspection Meeting	Meeting where team members as a group review the product to find, categorize, and record, but not resolve, defects.
Third Hour	Optional additional time, apart from the inspection meeting, that can be used for discussion, possible solutions, or closure of open issues raised in the inspection meeting.
Rework	Stage when the author corrects defects found during the inspection.

Follow-up	Short meeting between the moderator and author to determine if defects found during the inspection meeting have been corrected and to ensure that no additional defects have been introduced.

Each stage of the formal inspection process is addressed more completely in the following sections. The responsibilities of each member of the inspection team are described in Section IV.

A. Planning

Activities in the planning stage are performed primarily by the moderator, who assures that the product is ready for inspection, selects the inspection team and assigns roles, schedules the meeting place and time, and assures the distribution of inspection materials. Inspection materials include the product to be inspected, the inspection announcement, the preparation log, background information, and checklists.

Entrance criteria are used to screen out products that are not ready for inspection. For example, the entrance criteria for inspection of code should require that a clean compilation has been achieved. Entrance criteria should also specify that available automatic tool checking (using such tools as static analyzers, spell checkers, CASE tools, compilers, etc.) should be performed prior to distributing material to the inspection team.

The moderator determines whether the product to be inspected is of reasonable size so that the inspection can be completed in one meeting. If not, the moderator divides the product into manageable pieces. Sample size criteria are given in Table 2. The moderator then selects members for the inspection team and assigns them roles (See Section IV). An inspection package containing the product to be inspected, checklists, references to relevant higher level documents, and blank preparation logs should be distributed to all inspection team members. Sample checklists and inspection forms are provided in Appendix A and Appendix B, respectively.

The moderator also must decide whether the inspection team members are adequately familiar with the material to be inspected or whether an overview must be held. The team should know the background material from which the product was derived (e.g., design and requirements for source code). The team also should know how the material fits into the overall system being developed. In most cases, team members will be adequately familiar from their own work on the project. If they are not, an overview by the author should be scheduled.

If there is a project library, it may support the moderator during the planning stage by keeping records of the items to be inspected and those that have been completed. Once the moderator selects the inspection team, the library may provide each inspector with the inspection package. The project library can provide copies of reference and backup material to the inspection team. If there is no project library, or if the library cannot provide the needed support, the moderator must perform these functions.

B. Overview

The overview is an optional stage that is scheduled if the inspection team is not familiar with the material being inspected and its background. At the overview, the author presents the rationale for the product, its relationship to the rest of the products being developed, its function and intended use, and the approach used in developing it. This information is necessary to the inspection team so that it can perform a successful inspection. Interested persons aside from the inspection team may be invited to attend; however, all team members must attend.

An overview is scheduled if one or more of the following circumstances apply:

- The inspection team is not familiar with the product.
- The product is new or is being inspected for the first time.
- Inspections are new to the project.
- Novel techniques have been used in the product.

C. Preparation

Preparation is the key part of the inspection process. During this stage, inspection team members individually prepare for their roles in the inspection meeting. Each inspector reviews the product line by line. The inspector looks at the product for general problems as well as those related to his/her specific area of expertise. Checklists should be used during this stage for guidance on typical types of defects to be found in the type of product being inspected. In addition, the product being inspected is checked against higher level work products, standards, and interface documents to assure compliance and correctness. While familiarizing themselves with the product, inspectors record on the preparation log the defects found and the time spent during preparation. Completed preparation logs are submitted to the moderator prior to the inspection meeting. Sample checklists are shown in Appendix A; sample inspection forms are included in Appendix B.

Prior to the inspection meeting, the moderator reviews the logs submitted by each inspector to determine whether the team is adequately prepared. The moderator also checks for trouble spots that may need extra attention during the inspection, common defects that can be categorized quickly, and areas of major concern. If the logs indicate that the team is not adequately prepared, the moderator should reschedule the inspection meeting, as a team not fully prepared will waste time and will not be effective in finding defects. Preparation Log forms are returned to the inspectors at the inspection meeting for their use in pointing out defects.

D. Inspection Meeting

During the inspection meeting, the inspectors review the product as a team. Again, the focus of the meeting is finding defects. Briefly, the activities of the inspection meeting are: the reader provides a logical reading and interpretation of the product, the author provides clarifying information as needed, and the team identifies defects that are classified and recorded.

The moderator calls the meeting to order and notes the product being inspected. If the team is new or contains new members, the moderator may begin the inspection meeting by introducing team members, briefly describing their roles, and restating the purpose of the inspection and the product.

The reader then begins a logical and orderly interpretation of the product. This description should note the function of items (e.g., paragraphs, code blocks) and their relationship to the product and higher level documents. The inspection meeting is structured so that any inspector may interrupt the reader at any time when an item containing a possible defect is read. If a short discussion of the possible defect is needed, the reading is halted temporarily. The reading is resumed when the defect is noted and categorized by the recorder on the inspection defect list (see Appendix B). The moderator should try to limit discussions that appear to be consuming too much of the inspection meeting time. Imposing a time limit on discussions may be useful. If discussions are not completed within the time limit, the moderator will declare the issue unresolved and proceed with the meeting. The recorder should note unresolved issues as open items to be resolved during the third hour (see section E.).

The team should reach a consensus on whether each possible defect raised is an actual defect. Sometimes, what seems to be a potential defect may be a mistake on the part of the inspector or is only a misunderstanding that may be clarified by the author . If consensus cannot be reached, the potential defect should be recorded as an open issue to be dispositioned during the third hour. This ensures the right of every inspector to have each possible defect recorded and resolved. The recorder will itemize each agreed upon defect by recording its location, a brief description, its classification, and the inspector who found it.

The question of whether a potential defect is a real defect may be resolved by reference to parent documents, which should be available to the inspectors. If the discussion identifies the parent document as potentially in error, an open issue should be noted and the inspection should continue. Resolution of the issue (whether the defect is in the product being inspected or in the parent document) can be done during the third hour. Open issues are logged on the Inspection Report form.

To determine the priority for fixing the defects in the product, the inspection team or moderator should classify them by severity

(major or minor). For example, a defect that would cause the system to fail to satisfy a requirement would be classified as major; all others would be classified as minor (e.g., typographical errors, minor standards violation,). Additional data collected from each inspection is the classification of defects by type such as data error, requirements compliance, standards compliance, logic, interface, data, performance, and readability. If using the Inspection Defect List sample form in Appendix B, this information goes in the "Type" field for each defect.

At the end of the meeting, the number of defects are summarized, and the moderator and/or the team determines whether reinspection will be needed. At this time, the moderator also determines whether a third-hour activity is needed. If a third hour is needed, action items are assigned to individual inspectors at this time.

The team must focus on finding defects and should not be concerned with other activities such as problem solving. It is the responsibility of the moderator to control and focus the meeting. To avoid fatigue and thus missing defects, the inspection meeting should be limited to 2 hours. If the inspection is not completed in the original meeting, a second meeting must be scheduled.

After the meeting, the author and moderator meet briefly to estimate rework time and schedule the follow-up meeting. The author is given a copy of the defect list for reference during rework. Note that formal discrepancy reports (DRs) and change requests (CRs) are not written against defects found in the document or code being inspected; however, discrepancy reports should be written against any defects found in higher level documents. The reason that DRs and CRs are not required is that inspections take place before the product is under configuration control and that closure is assured by the defect list, the follow up stage, and by reinspections.

E. Third Hour

The third hour is time that is used for discussion or for closing open issues. A third hour should be scheduled when the author wishes to discuss corrections of defects found, or when open issues, such as a potential major defects in parent (higher level reference) documents, need to be dispositioned. The third hour may take the form of an additional meeting or of time for team members to perform and report on action items. It does not have to take place immediately after the inspection meeting and it does not have to be limited to 1 hour.

When used as meeting time, the third hour provides an opportunity to discuss solutions and resolve disagreements. Attendees may include any subset or superset of the inspection team including relevant managers (present for technical reasons only), outside

technical experts, and other people who could be affected by the way the defect is fixed or the issue resolved. In many cases, only those inspectors who have a specific interest need attend. During a third hour meeting, the author is provided with information that would make rework more effective, major defects found in parent documents are reported, and any open issues remaining from the inspection are resolved.

When used as an opportunity for individual inspectors to perform action items, the third hour usually is spent researching and dispositioning open issues, finding information to resolve a discrepancy, or writing discrepancy reports or change requests for major defects found in parent documents under configuration management.

F. Rework

The purpose of rework is to correct defects found during the inspection. The author is responsible for correcting all major defects noted in the inspection defect list. Minor defects should be resolved if cost and schedule permit. The moderator should make sure that information generated from any open issues or action items are communicated to and addressed by the author .

G. Reinspection

Reinspection may be required when there are a large number of defects in the product or when one or more defects require extensive or complicated corrections. Reinspection allows the changes to the product to be reviewed by the entire team instead of just the moderator. The moderator and the team decide the necessity for reinspection at the end of the inspection meeting.

H. Follow-Up

Follow-up is a short meeting between the moderator and the author to ensure that all major defects found during the inspection have been corrected and no secondary defects have been introduced. The author reviews the measures taken to correct each major defect with the moderator. The moderator ensures that all open issues have been resolved and that changes due to the resolution, if any, have been incorporated in the product. Corrections to minor defects also may be reviewed, but with less emphasis. The moderator ensures that the exit criteria for the type of inspection have been met. The moderator may include additional reviewers in the follow-up meeting if extra technical expertise is required.

If all of the major defects have been corrected, all open issues have been resolved, and the product has satisfied the exit criteria, then the moderator "Passes" the product by recording the completion of inspection on the Inspection Summary Report. If these conditions are not met, the author returns to the rework stage to make the necessary changes.

IV. THE INSPECTION TEAM

The inspection team is a small group of peer staff members with a vested interest in the product. The minimum team size is three persons, although a typical team varies from four to seven. Larger teams are generally used for high level documents, while smaller teams are used at detailed technical levels. Members are added when their point of view is needed. A good mix of inspectors representing various areas of expertise is important to the inspection process. The knowledge and experience of such a group, each looking at the product from his/her own perspective, helps bring many subtle defects to light. "Synergy," where an idea from one team member often leads to another idea from a different team member, is one indication of a healthy inspection process. The role of each individual is explained in the following sections.

A. Inspectors

Every member of the inspection team is considered an inspector in addition to any other assigned role. Inspectors are responsible for finding defects in the work during preparation and during the inspection meeting. In addition to functioning as an inspector, some members of the inspection team will carry out roles as moderator, author, reader, and recorder, as appropriate.

Primary candidates for inspectors are personnel involved with the product in immediately preceding, current, and following life cycle phases. For example, in a design inspection, good inspectors may be selected from the individuals who wrote the requirements, people who will code the design, and designers of interfacing segments of the system. An exception to this rule is that the author of a unit of code should not serve as an inspector for test procedures that are to be used to test that code, because the code author may want changes made to the test procedures so they will work with the code as it exists. Any group with an interest in the product should be considered for potential team members including Systems Engineering, Testing, Software Assurance, Systems Administrators and Operators, and Users. Sources for inspectors are not limited to the staff of the software development organization. Outside inspectors should be brought in when they have a particular expertise that would add to the effectiveness of the inspection.

B. Moderator

The moderator leads the inspection team and is responsible for ensuring that a good inspection is achieved. Because this role is critical to the formal inspection process, training for moderators is more important and extensive than that of other inspectors. The moderator is directly active in all stages of the inspection process except rework. Since acting as a moderator is time consuming and requires specific skills, moderators often are selected and trained by the development organization and then

assigned to a specific development project. Primary duties of the moderator include coordinating the selection of the inspection team, assigning team roles, and leading the team throughout the process. A major function of the moderator is to ensure that the team keeps its emotions in check and that the inspection meeting is not used to find faults with the author. The moderator is also responsible for assuring inspection data are collected on the inspection report forms.

C. Reader

The reader is responsible for guiding the inspection team through the product during the inspection meeting by reading or paraphrasing the product. An individual who will use the product during the next life cycle phase is an excellent candidate for reader as the process of reading and paraphrasing it will cause this potential user to become very familiar with the product before it is delivered. The reader also performs the duties of a regular inspector.

D. Recorder

The recorder is responsible for accurately recording information during the meeting about each defect found on the inspection defect list. The list should include the location of the defect, a brief description of it, its classification, and the identity of the inspector who found it. The recorder also must record information on any issues raised and not resolved, and any defects that are found in parent documents. The recorder also performs the duties of a regular inspector.

E. Author

The author is the originator of the product being inspected. As such, she/he is primarily responsible for providing information on the product being inspected and for answering inspectors' questions to ensure that simple misunderstandings are not classified as defects. In addition, the author should also perform the duties of a regular inspector. The author is required to correct all major defects found during the formal inspection, and minor defects as time and resources allow.

V. FORMAL INSPECTIONS DURING THE SOFTWARE LIFE CYCLE

Formal inspections are in-process peer reviews conducted within the phase of the life cycle in which the product is developed. The following describes the life cycle phases of software development and suggests products that may be inspected during each phase. The software life cycle used is the NASA standard waterfall model; the material may be adapted to other life cycles if needed.

A. Software Concept and Initiation Phase

During this phase, the software concept is developed, the feasibility of the software system is evaluated, and the acquisition strategy is developed. Much of the documentation for the project is started or a draft provided within this phase. The most important document to inspect is the portion of the system requirements document that applies to software. This inspection has the shorthand designation of R0 (see table 1). Other candidates for inspection include system specifications and plans such as the software management and assurance plans. Potential inspectors are the users and assurers of this documentation and the system to be developed.

B. Software Requirements Phase

During this phase, the software concept and allocated system requirements are analyzed and documented as detailed software requirements. Test planning is started, with a preliminary method for verifying each requirement identified and included in a first version of a test plan. Risks are identified and planned for, and the size and scope of the remainder of the project is re-estimated. Methods, standards, and procedures are detailed and put in place.

The requirements document is the product that is most typically inspected in this phase. This is known as the R1 inspection. The requirements document should be inspected for completeness and accuracy, for traceability to higher level documents, and to assure that a sufficient base is provided for the software design. Other documents that are produced during this life cycle phase, such as the draft acceptance test plan, can also be inspected (Inspection IT1). Candidates for inspectors include the assurers and potential users of the documents, including designers, coders, and testers.

C. Software Architectural (Preliminary) Design Phase

During this phase, the overall design for the software is developed, allocating all of the requirements to software components. The architectural design inspection is designated I0.

The design should be inspected for satisfaction of and traceability to the requirements, correctness, clarity, codability, testability, and consistency.

D. Software Detailed Design Phase

During this phase, the architectural design is expanded to the unit level. Interface control documents are completed and test plans updated. Constraints and system resource limits are re-estimated and analyzed, and staffing and test resources are validated.

The detailed design should follow exactly the base-lined higher level design, and should be inspected for the same characteristics. As a secondary condition, the design should be inspected for satisfaction of software quality engineering standards such as information hiding, use of simple structures, etc. The detailed design inspection is designated I1.

Candidates for the inspection team may be selected from participants in the design, code, and test phases.

E. Software Implementation Phase

During this phase, the software is coded and unit tested. All documentation is produced in quasi-final form, including internal code documentation.

Code and all new documentation are the candidates for inspections during this phase. Code inspections (designated I2) should check for technical accuracy and completeness of the code, verify that it implements the planned design, and ensure good coding practices and standards are used. Code inspections should be done after the code has been compiled and all syntax errors removed, but before it has been unit tested. Other candidates are the integration and test plan and procedures, and other documents that have been produced. Documents should be inspected for accuracy, completeness, and traceability to higher level documents. The inspection team may be selected from participants in the detailed design, code, test, verification and validation, or from software quality assurance.

F. Software Integration and Test Phase

During this phase the software units are integrated into a completed system; nonconformances are detected, documented, and corrected; and it is demonstrated that the system meets its requirements. The integration and test plan is executed, the software documentation is updated and completed, and the products are finalized for delivery.

The final version of the Acceptance Test Plan should be inspected to detect defects in the definition of test cases and to verify that each test case will verify the requirements with which it is

associated. This is the IT1 inspection. Test case and test procedure inspections should verify that they are in accord with one another and with the Acceptance Test Plan. These inspections should verify that the test cases and procedures will execute properly and correctly, and that all needed data are available. Potential inspectors are representatives from any of the life cycle phases before or after this one.

While the products listed above are used in the Integration and Test Phase, they may have been developed in prior phases. Inspections should be integrated into the development process, and these products inspected when they are developed. If so, few or no inspections may actually be done during this phase; inspections are needed only if new test cases and procedures are developed.

G. Software Acceptance and Delivery Phase

The formal acceptance procedure is carried out during the acceptance and delivery phase. As a minimum, there is a requirements-driven demonstration of the software to show that it meets its requirements. The phase also may include acquirer tests, field usage, or other arrangements that are intended to assure that the software will function correctly in its intended environment.

There is little or no inspection activity during this phase.

H. Software Sustaining Engineering and Operations Phase

During this phase, the software is used to achieve the objectives for which it was developed and acquired. Corrections and modifications are made to the software to sustain its operational capabilities and to upgrade its capacity to support its users. Software changes may range in scope from simple corrective action up to major modifications that require a full life cycle process.

Formal inspections should be scheduled in response to the degree of new development activity involved. Significant new material to be incorporated into any product should be inspected. A useful technique is to have the need for inspections evaluated as part of the change control and configuration management process.

VI. STARTING A FORMAL INSPECTIONS PROGRAM

Formal inspections have been proven to be effective in detecting and removing defects, and to be cost effective when compared to the cost of finding defects by testing. However, they represent an up-front cost and a diversion from more traditional uses of software development resources. Since there may well be skepticism about inspections, beginning a program may find some resistance. Educating management in the advantages of formal inspections, particularly stressing how the up-front costs are likely to be made up by reduced testing costs may help to overcome the skepticism.

A critical first step in initiating an inspection program is to select the class of material to first be inspected. It is advisable to begin with material in which everyone will agree that defects have to be found and removed. Based on their own experiences in starting inspection programs, both JPL and LaRC recommend starting with requirements inspections, as the benefits of formal inspections are shown early in the software life cycle.

Defects in requirements also have more impact than those in other products. For example, for each defect found and corrected in the requirements, many "defects" would have been present in the design and code. If these resultant defects were not found until testing, they would cause a great deal of rework to many products. Such early results will be popular with management, and should raise enthusiasm for starting the program. Defects in requirements, especially, and in design, are more expensive to correct after the system has been implemented than are code defects.

NASA experience has shown that inspection of requirements and design will significantly reduce code "errors;" some projects conduct formal inspections of all of their requirements and design, but only inspect critical code. Although code inspections may be the easiest type of inspections to perform, they may not be the most productive.

Once a starting point for inspections has been selected, needed resources must be budgeted and roles and responsibilities decided upon. Resources will be needed for start-up costs, overhead costs, and operational costs. Start-up costs include training of moderators and other inspectors, development of forms and report formats, and acquisition of data processing resources for the recording and trending of inspection data. Overhead costs associated with the formal inspection program consist of moderator time for arranging and scheduling inspections and follow-up, and the moderator's or project librarian's role in making copies of materials available and keeping track of the status of items as they progress through the inspection process. In addition, while the collection and trending of data is important, it will consume some resources. Operational costs consist of the time spent by

project members preparing for and participating in inspection meetings.

The chief moderator is key to the whole formal inspection program and should be selected as part of the start-up. The chief moderator oversees and directs the inspection program; analyzes the effectiveness of the inspection process; and coordinates the evolution of the program, forms, and checklists. The chief moderator should be the moderator for the inspections on the initial project. When sufficient inspectors have been trained and become experienced, additional moderators may be selected from this pool and trained for new projects to which formal inspections are to be applied.

If it is possible to arrange, project libraries can make both the start-up and the inspection process run more smoothly; this support will allow the moderator to concentrate on the success of the inspection program. The project library helps the moderator to maintain lists of what is to be inspected and assists with some of the mechanics of the inspection process such as delivering materials, setting up meeting rooms, etc. The library should provide reference documentation for the inspectors.

Checklists to guide the inspectors may be developed from the samples provided in Appendix A, obtained from JPL, or developed specifically according to the needs of the program. In the long run, checklists will be needed for each type of material to be inspected and each major language used. For example, there should be checklists for requirements, design, Ada code, C code, user's guides, etc. At the start-up, only checklists for the limited items to be inspected are needed.

Forms to record results and collect inspection data should be defined. Example forms are shown in Appendix B. They should be tailored as needed to reflect working conditions and to capture the specific data desired. The standard forms will evolve over time.

Once the centralized resources are in place (e.g., chief moderator, librarian, forms, checklists, and data processing resources), project individuals who are to participate in the inspection process must be identified. Project technical people are the key to the program since they are the readers, inspectors, and the ones who have to use the inspection results to improve the product. Once participants have been identified, they should be trained. JPL has developed a formal inspection training class for NASA that is workshop-oriented and very effective. Alternatively, outside organizations have inspection training available.

Projects introducing inspections must plan to accommodate them. In addition to identifying inspectors for training, managers should plan time in schedules for inspections, analysis, and rework. Experience shows that the resources used for inspections are more than made up for in shortened test time and the costs of

finding and repairing defects that are imbedded in the system, but resources are needed at different points in the life cycle. Project or other management must also provide sufficient and appropriate space in which the inspection process can take place.

Once resources are available and the moderator and an adequate number of inspectors have been trained, the inspection program can begin. One important factor to have in operation from the beginning is a data collection program. Formal inspection data is easy to collect because the process is very structured. The data can also be used to improve the processes that produce the products that are inspected. The subject of data collection and evaluation and improvement of the inspection process is discussed in Section VII.

Once the inspection program is underway and there are several moderators, moderators should meet regularly to discuss problems and successes with the inspection program and suggest ways to improve it. This meeting should be chaired by the chief moderator who is responsible for carrying out or recommending improvements and evaluating whether the level of training and experience is being maintained.

VII. PROCESS EVALUATION AND IMPROVEMENT

Formal inspections have been demonstrated by many organizations to be an effective method for finding and removing defects in software products. However, just putting a formal inspections program in place does not guarantee that the program will operate at maximum efficiency. It is important to evaluate the implementation of the formal inspections process and to improve it by fine tuning the procedures that are followed. The items that most need to be tailored are the inspection rates and the checklists. If too large an amount of a software product is to be inspected at a meeting, the meeting will have to rush along too rapidly to be effective, or meetings will routinely have to be continued with resultant inefficiencies and schedule delays. If too little of the product is allocated to one inspection, the program will also not work at peak efficiency. The inspection rates must be tailored to the complexity of the product and the ability of the inspectors. Checklists should be tailored to ensure that inspectors pay attention to the types of errors that actually occur in the products being inspected. Fine tuning of the checklists will make more efficient use of preparation time and meeting time, and should help ensure that more of the defects are found.

In order to evaluate the effectiveness of the inspection program, the data collected from inspections should be routinely analyzed in order to reveal trends. The trends that should be evaluated are the amount of product inspected at a meeting, the time taken in preparation and in the inspection meeting, total defects found per inspection, the types of defects found, and the phases in the development life cycle where defects were found. The data for trending is normally collected by the moderator, using forms provided for this purpose (see appendix B).

In the case where a trend points to a decline in efficiency in, for example, the time spent preparing for and conducting inspections, action can be taken to analyze the procedures and correct the problem. The analysis might lead to changes in the amount of product scheduled for each meeting, or to the checklists provided to the participants, or to the training for the participants. If the trend data shows fewer defects are being found in inspections late in the life cycle, such as code inspections, an analysis might show that the inspectors were not adequately preparing, or it might show that the organization has becomes so effective in performing requirements and design inspections that there are few code defects to be found. In this case, steps may be taken to modify procedures to inspect only critical code.

The data from inspections may also be used in another manner. If, for example, the inspection data show that during code inspections a high percentage of code defects are due to defects introduced during the design phase of the life cycle, then steps could be taken to attempt to improve the effectiveness of both

the design process and the design inspections. This ability to point out the life cycle step in which defects were introduced is dependent on careful data gathering, but could pay high dividends. Any attempt to change the life cycle processes used in an organization should be done with great care, and information from other sources than just inspections should be used, but the inspection data could be of great assistance.

The evaluation process should be continuous, that is, the trend data should be kept up to date, it should be examined regularly, and the trends should be available to all participants in the inspections program. Only continuous monitoring can ensure the maximum cost effectiveness in the resources used for inspections.

The data gathered could be used to modify the inspection process itself. The third hour is one such modification, introduced by JPL to the process defined by Michael Fagan based on their analysis of their inspection program. Modification of the inspection process should be done only after very careful analysis and testing of the proposed changes. The formal inspection process is effective because it is well defined, well tested, and done in exactly the same manner time after time.

VIII. SUMMARY

The following summarizes the essentials of the formal inspection process and provides a quick reference:

1. Inspections are carried out on products that have been completed by their author but not yet tested, reviewed, or otherwise approved or baselined.

2. The objective of the inspection process is to detect and remove defects. Typical defects are errors of documentation, logic, and function.

3. Inspections are carried out by peers of the author. Participants in the inspection process should represent organizations that will use or will be affected by the material being inspected.

4. Inspections should not be used as a tool to evaluate workers. Management is not to be present during inspections. When a management official has technical expertise which is not available from other sources, that individual may be brought into the third hour.

5. A trained moderator leads inspections, and all participants should have training in the process.

6. Inspectors are assigned to and prepare for specific roles (e.g., reader, recorder, author).

7. Inspections are carried out in a prescribed series of steps from planning through follow-up.

8. Inspection meetings are limited to two hours.

9. Checklists of questions and typical defects are used to stimulate defect finding. Project-tailored entrance and exit criteria should be developed for each type of product to be inspected.

10. The product being inspected should be of an appropriate size that it can be inspected during a two hour meeting.

11. Correction of defects is the responsibility of the author, and is verified by the moderator. The inspection team must refrain from suggesting methods for correction during the inspection meeting.

12. Data and trends on the number of defects, the types of defects, and the time expended on inspections should be maintained. This information should be used to evaluate and improve the effectiveness of the inspection program.

APPENDIX A

SAMPLE CHECKLISTS

This appendix provides sample checklists for typical inspections: Architecture Design, Detailed Design, and Code Inspection. In addition, sample checklists used by JPL Software Product Assurance are provided including Architecture Design, Detailed Design, Code Inspection - "C" and Code Inspection - FORTRAN, Test Plan, Test Procedures and Function, Functional Design, Software Requirements, and Functional Requirements. These checklists are provided for illustration and for sample guidance for the moderator in preparing a tailored checklist for each type of inspection to be done for a project.

Checklists should be viewed as a starting point for investigation. An inspector should assure those items on the list are correct, but also should use judgment and experience to look for other possible faults. Also, some questions from an earlier phase of inspections may apply. As a Center or project gains experience in performing formal inspections, checklists should expand and change; revisions and additions should be created.

SAMPLE CHECKLISTS

Architecture Design Checklist

FUNCTIONALITY

1. Does the design implement the specifications and requirements?
2. Are the specifications and requirements complete?
3. Is the abstract algorithm specified for each sublevel unit/subunit?
4. Is the design functionally cohesive?

TRADE STUDIES

1. Have design trades been performed and documented?
2. Have the assumptions been documented?
   a. Are the goals defined?
   b. Are the trade criteria defined?
3. Will the selected design or algorithm meet all of its requirements?

PERFORMANCE

1. Are the primary performance parameters specified (real time, memory size, speed requirements, amount of disk I/O, etc.)?
2. Are synchronization requirements met (phasing, time-outs, etc.)?
3. Does the design embody the actual operating environment?
4. Is the impact of failure defined?

LOGIC

1. Is there missing or incomplete logic?
2. Are all possible states or cases defined?
3. Are actions taken correct in all cases?

DATA USAGE

1. Is the conceptual view documented for all objects, relationships, and parameters?
2. Is there any data structure needed that has not been defined or vice versa?

INTERFACES

1. Is the operator interface designed with the user in mind (i.e., vocabulary, useful messages)?
2. Will the interface facilitate troubleshooting?
3. Are all interfaces consistent with each other, other CSCs, and requirements?
4. Do all interfaces provide the required types and amounts of information?

TESTABILITY

1. Can the module that this design describes be tested or inspected to demonstrate that it satisfies requirements?
2. Can the program set be integrated and tested in an incremental manner?

Architecture Design Checklist (Continued)

RELIABILITY

1. Does the design provide for error detection and recovery (e.g., is input checking performed)?
2. Are abnormal conditions considered?
3. Does the design satisfy all systems integrity commitments for this product?
4. Are all error conditions/codes/messages specified completely and meaningfully?

MAINTAINABILITY

1. Is information hiding used? Is the design modular?
2. Do the CSCs have high cohesion and low coupling?
3. Does the documentation follow project or NASA standards?
4. Is there traceability between maintenance documents and the design?

CLARITY

1. Is the architecture, including the data flow and interfaces, clearly represented?
2. Are there multiple, consistent representations of the design?
3. Are all of the decisions, dependencies, and assumptions for this design documented?

TRACEABILITY

1. Are all parts of the design traced back to requirements?
2. Can all design decisions be traced backed to trade studies?

CONSISTENCY

1. Are data elements, procedures, and functions named and used consistently throughout the program set?
2. Are data elements, procedures, and functions consistent with external interfaces?

LEVEL OF DETAIL

1. Does the design have sufficient detail to proceed to the next phase?
2. Have all possible "To Be Determined (TBD)" statements been resolved?

Detailed Design Checklist

FUNCTIONALITY

1. Does the design implement the specified algorithm?
2. Will this design fulfill specified requirements?
3. Does it conform to the architecture?

LOGIC

1. Are all variables and constants defined and initialized? (Don't forget pointers.)
2. Is there logic missing?
3. Are literals used where a constant data name should be used?
4. Are greater-than, equal to, less-than-zero, or other (for switch case) conditions each handled?
5. Are branches correctly stated (the logic is not reversed)?
6. Are actions for each case correct?

DATA USAGE

1. Are all data blocks specified (for structure and usage) and used?
2. Are all routines that modify a data block aware of the data block's usage by any other routine?
3. Are all logical units, events, and synchronization flags defined?

PERFORMANCE

1. Are synchronization mechanisms correct and will they perform as required?

LINKAGES

1. Do argument lists match in number, type, and order?
2. Are all linkages input and output properly defined and checked?
3. Is the data area mapped as the receiving unit expects it to be?
4. Are messages issued for all error conditions?
5. Do return codes for particular situations match the global definition of the return code as documented?

TESTABILITY

1. Is the design described in a testable, measurable, or demonstrable form?
2. Does the design contain checkpoints to aid in testing?
3. Can all logic be tested?

Detailed Design Checklist (Continued)

RELIABILITY

1. Are defaults used and are they correct?
2. Are boundary checks performed on memory accesses (arrays, data structures, pointers, etc.) to ensure program memory is not being altered?
3. Have linkages been checked for inadvertent destruction of data (e.g., in Fortran, passing a constant to a subroutine and altering it within the subroutine)?
4. Is error checking on inputs, outputs, linkages, interfaces, and results performed?
5. Are undesired events considered (e.g., in spacecraft, single-event upset alters a key data location, are there backups, verification tests on the data, restart procedures)?

LEVEL OF DETAIL

1. Is the intent of all units or processes documented?
2. Is the expansion ratio of code to design documentation less than 10:1?
3. Are all required module attributes defined?
4. Are all assumptions made about this module (as seen by the remaining program set) documented?

MAINTAINABILITY

1. Does this unit have high internal cohesion and low external coupling (i.e., changes to this unit do not have any unforseen effects within the unit and have minimal effect on other units)?
2. Are any programming standards for the project in jeopardy of compromise because of the design (e.g., Does the header meet project standards? Does it include the purpose, author or developer, environment, nonstandard feature used, development history, input and output parameters, file used, data structures used, units invoking this one, and explanatory notes?)?

TRACEABILITY

1. Are all parts of the design traced back to the requirements?
2. Can all design decisions be tracked back to trade studies?

CONSISTENCY

1. Are data elements named and used consistently throughout the unit and unit interfaces?
2. Is the design of all unit interfaces consistent among themselves and with the system interface specification?

CLARITY

1. Is the unit design including the data flow, control flow, and interfaces, clearly represented?

Code Inspection Checklist
C

FUNCTIONALITY

1. Does each unit have a single purpose?
2. Is there code that should be in separate functions?
3. Is the code consistent with performance requirements?

TRACEABILITY

1. Does the code track the Detailed Design?

DATA USAGE

1. Data and Variables
   a. Are declarations of variables grouped into externals and internals?
   b. Do all but the most obvious declarations have comments?
   c. Is each name used for only a single purpose?

2. Constants
   a. Are all constant names uppercase?
   b. Are constants defined via "# define"?
   c. Are constants that are used in multiple files defined in an INCLUDE header file?

3. Pointers Typing
   a. Are variables declared as pointers used as pointers (not integers)?
   b. Are pointers initialized?

LINKAGE

1. Are "INCLUDE" files used according to project standards?
2. Are nested "INCLUDE" files avoided (even if permitted by the compiler)?
3. Are all data local in scope (internal static or external static) unless global linkage is specifically necessary and commented?
4. Are the names of macros all uppercase?

LOGIC

1. Lexical Rules for Operators
   a. Are unary operators adjacent to their operands?
   b. Are primary operators("->" "." "()" etc.) adjacent to their operands?
   c. Do assignment and conditional operators always have space around them?
   d. Are commas and semicolons followed by a space?
   e. Are key words followed by a blank?
   f. Is the use of " (" following function name adjacent to the identifier?
   g. Are spaces used to show precedence? If precedence is at all complicated, are parentheses used (especially with bitwise operations)?

Code Inspection Checklist (Continued)
C (Continued)

2. Evaluation Order
   a. Are parentheses used properly for precedence?
   b. Does the code depend on evaluation order, except in the following cases?
      1. expr1, expr2
      2. expr1 ? expr2 : exp2
      3. expr1 || expr2
   c. Are shifts used properly?
   d. Does the code depend on order of effects (e.g., i = i++; vs. i = i--)?

3. Control
   a. Are "if...else" trees and "switch" used clearly?
   b. Are "goto" and "labels" used only when justifiable, and always with well-commented code?
   c. Is the block of code that follows an "if" statement surrounded by { } brackets?

MAINTENANCE

1. Are nonstandard usages isolated in subroutines and well documented?
2. Does each unit have one exit point?
3. Is the unit easy to change?
4. Is the unit independent of specific devices where possible?
5. Is the system standard defined types header used if possible (otherwise, use project standard header, by "include")?
6. Is there traceability between maintenance documents and the code?

CLARITY

1. Comments
   a. Is the unit header informative and complete?
   b. Are there sufficient comments to understand the code?
   c. Are the comments in the units informative?
   d. Are comment lines used to group logically-related statements?
   e. Are the functions of arrays and variables described?
   f. Are changes made to a unit after its releases noted in the development history section of the header?

2. Layout
   a. Is the layout of the code such that the logic is apparent?
   b. Are loops indented and visually separated from the surrounding code?
   c. Are "ifs" indented?

Code Inspection Checklist (Continued)
C (Continued)

MISCELLANEOUS

1. If different "memory models" are supported by the compiler, does the code reflect the agreed-upon model with respect to declarations of pointers and functions? Typically, memory models go together with a notion of "near" and "far" pointers and function calls.

2. Are values returned from functions calls (that indicate success or failure of a request) checked before proceeding to the following code?

3. Are requests to dynamically allocate memory checked for success before attempting to write to that memory?

4. Are pointers immediately set to NULL (or 0) following the deallocation of memory? (Most compilers do not "zero" pointers on deallocation.)

5. Does code that writes to dynamically allocated memory via a pointer first check for a valid (non-zero) pointer?

6. Does code that passes a pointer to another function first check for a valid (non-zero) pointer?

7. Are functions declared to be "static" (hidden from other modules) if they are only to be called by other functions in the same module?

8. Are module level variables declared to be "static" (information hiding) if they are only used by functions within the same module?

9. Are recursive function calls used in mission critical software only if absolutely necessary and well documented? Recursive calls can produce unpredictable run time results due to the demands on the dynamically allocated "stack" space that is normally used to implement recursion.

10. Is dynamic allocation of memory minimized in mission critical software? If dynamic allocation is used, is there a means of estimating exhaustion of available memory and recovering from an inability to allocate additional memory?

JPL SOFTWARE PRODUCT ASSURANCE CHECKLISTS

I0 - Architecture Design Checklist (JPL)

CLARITY
1. Is the architecture, including the data flows, control flows, and interfaces, clearly represented?

COMPLETENESS
1. Are the goals defined?
2. Have all TBDs been resolved in requirements and specifications?
3. Can the design support any anticipated changes in the TBD requirements?
4. Have the impacts of the TBDs been assessed?
5. Has a risk plan been made for the parts of the design which may not be feasible?
6. Have design tradeoffs been documented? Does the documentation include the definition of the trade space and the criteria for choosing between tradeoffs?
7. Has design modeling been performed and documented?
8. Are all of the assumptions, constraints, decisions, and dependencies for this design documented?

COMPLIANCE
1. Does the documentation follow project and/or JPL standards?

CONSISTENCY
1. Are data elements, procedures, and functions named and used consistently throughout the program set and with external interfaces?
2. Does the design reflect the actual operating environment? Hardware? Software?
3. When appropriate, are there multiple, consistent, representations of the design (i.e., static vs. dynamic)?

CORRECTNESS
1. Is the design feasible from schedule, budget, and technology standpoints?
2. Is the logic correct and complete?

DATA USAGE
1. Is the conceptual view for all composite data elements, parameters, and objects documented?
2. Is there any data structure needed that has not been defined, and vice versa?
3. Have data elements been described to a sufficiently low level of detail? Have valid value ranges been specified?
4. Has the management and use of shared and stored data been clearly described?

I0 - Architecture Design Checklist (JPL) (Continued)

FUNCTIONALITY
1. Are the specifications for the modules consistent with the full functionality required for the module in the Software Requirements Document (SRD) and Software Interface Specifications (SIS-1)?
2. Is an abstract algorithm specified for each sublevel module?
3. Will the selected design or algorithm meet all of the requirements for the module?

INTERFACES
1. Are the functional characteristics of the interfaces described?
2. Will the interface facilitate troubleshooting?
3. Are all interfaces consistent with each other, other modules, and requirements in SRD, SIS-1/2?
4. Do all interfaces provide the required types, amounts, and quality of information?
5. Have the number and complexity of interfaces been effectively balanced against one another to result in a small number of total interfaces, each of which is of acceptable complexity?
6. Is the operator interface designed with the user in mind (i.e., precise and non-jargon vocabulary, useful messages)?

LEVEL OF DETAIL
1. Has the size of each sublevel module been estimated (lines of code)? Is it reasonable?
2. Is a reasonably large and representative set of the possible states or cases considered?
3. Is the design of sufficient detail to proceed to the detailed design phase?

MAINTAINABILITY
1. Is the design modular?
2. Do the modules have high cohesion and low coupling?

PERFORMANCE
1. Has performance modeling been performed when appropriate and has it been documented?
2. Are all performance parameters specified (e.g., real time constraints, memory size, speed requirements, amount of disk I/O)?
3. Do processes have time windows (e.g., flags may be needed to "lock" structures, semaphores, some code may need to be non-interruptible)?
4. Have all critical paths of execution been identified and analyzed?

I0 - Architecture Design Checklist (JPL) (Continued)

RELIABILITY

1. Does the design provide for error detection and recovery (e.g., input checking)?
2. Are abnormal conditions considered?
3. Are all error conditions specified completely and accurately?
4. Does the design satisfy all systems integrity commitments for this product?

TESTABILITY

1. Can the program set be tested, demonstrated, analyzed, or inspected to show that it satisfies requirements?
2. Can the program set be integrated with previously tested code and can it be tested incrementally?

TRACEABILITY

1. Are all parts of the design traced back to requirements in SRD, SIS-1, other project documents?
2. Can all design decisions be traced back to trade studies?
3. Has the impact of special or unusual features of inherited designs on the current design been addressed?
4. Are all known risks from inherited designs identified and analyzed?

I1 - Detailed Design Checklist (JPL)

CLARITY
1. Is the intent of all units or processes documented?
2. Is the unit design, including the data flow, control flow, and interfaces, clearly represented?
3. Has the overall function of the unit been described?

COMPLETENESS
1. Have the specifications for all units in the program set been provided?
2. Have all the acceptance criteria been described?
3. Have the algorithms (e.g., in PDL) used to implement this unit been specified?
4. Have all the calls made by this unit been listed?
5. Has the history of inherited designs been documented along with known risks?

COMPLIANCE
1. Does the documentation follow project and/or JPL standards?
2. Has the unit design been created using the required methodology and tools?

CONSISTENCY
1. Are data elements named and used consistently throughout the unit and unit interfaces?
2. Are the designs of all interfaces consistent with each other and with the SIS-2 and SSD-1?
3. Does the detailed design, together with the architectural design, fully describe the "as-built" system?

CORRECTNESS
1. Is there logic missing?
2. Are literals used where a constant data name should be used?
3. Are all conditions handled (greater-than, equal-to, less-than-zero, switch/case)?
4. Are branches correctly stated (the logic is not reversed)?

DATA USAGE
1. Are all the declared data blocks actually used?
2. Have all the data structures local to the unit been specified?
3. Do all routines that modify data (or files) shared with other routines access that shared data (or files) according to a correct data sharing protocol (e.g., mutual exclusion via semaphores)?
4. Are all logical units, event flags, and synchronization flags defined and initialized?
5. Are all variables, pointers, and constants defined and initialized?

I1 - Detailed Design Checklist (JPL) (Continued)

FUNCTIONALITY
1. Does this design implement the specified algorithm?
2. Will this design fulfill its specified requirement and purpose?

INTERFACE
1. Do argument lists match in number, type, and order?
2. Are all inputs and outputs properly defined and checked?
3. Has the order of passed parameters been clearly described?
4. Has the mechanism for passing parameters been identified?
5. Are constants and variables passed across an interface treated as such in the unit's design (e.g., a constant should not be altered within a subroutine)?
6. Have all the parameters and control flags passed to and returned by the unit been described?
7. Have the parameters been specified in terms of unit of measure, range of values, accuracy, and precision?
8. Is the shared data areas mapped consistently by all routines that access them?

LEVEL OF DETAIL
1. Is the expansion ratio of code to design documentation less than 10:1?
2. Are all required module attributes defined?
3. Has sufficient detail been included to develop and maintain the code?

MAINTAINABILITY
1. Does this unit have high internal cohesion and low external coupling (i.e., changes to this unit do not have any unforeseen effects within the unit and have minimal effect on other units)?
2. Has the complexity of this design been minimized?
3. Does the header meet project standards (e.g., purpose, author, environment, nonstandard features used, development history, input and output parameters, files used, data structures used, units invoking this one, units invoked by this one, and explanatory notes)?
4. Does the unit exhibit clarity, readability, and modifiability to meet maintenance requirements?

PERFORMANCE
1. Do processes have time windows?
2. Have all the constraints, such as processing time and size, for this unit been specified?

I1 - Detailed Design Checklist (JPL) (Continued)

RELIABILITY

1. Are default values used for initialization and are they correct?
2. Are boundary checks performed on memory accesses (i.e., arrays, data structures, pointers, etc.) to insure that only the intended memory locations are being altered?
3. Is error checking on inputs, outputs, interfaces, and results performed?
4. Are meaningful messages issued for all error conditions?
5. Do return codes for particular situations match the global definition of the return code as documented?
6. Are undesired events considered?

TESTABILITY

1. Can each unit be tested, demonstrated, analyzed, or inspected to show that they satisfy requirements?
2. Does the design contain checkpoints to aid in testing (e.g., conditionally compiled code, data assertion tests)?
3. Can all logic be tested?
4. Have test drivers, test data sets, and test results for this unit been described?

TRACEABILITY

1. Are all parts of the design traced back to the requirements?
2. Can all design decisions be traced back to trade studies?
3. Have all the detailed requirements for each unit been specified?
4. Have the unit requirements been traced to the Software Specification Document (SSD-1)? Have the SSD-1 specifications been traced to the unit requirements?
5. Has a reference to the code or the code itself been included?

I2 - Code Inspection Checklist (JPL)

C

FUNCTIONALITY

1. Does each module have a single function?
2. Is there code which should be in a separate function?
3. Is the code consistent with performance requirements?
4. Does the code match the Detailed Design? (The problem may be in either the code or the design.)

DATA USAGE

A. Data and Variables
   1. Are all variable names lower case?
   2. Are names of all internals distinct in 8 characters?
   3. Are names of all externals distinct in 6 characters?
   4. Do all initializers use "="? (v.7 and later; in all cases should be consistent).
   5. Are declarations grouped into externals and internals?
   6. Do all but the most obvious declarations have comments?
   7. Is each name used for only a single function (except single character variables "c", "i", "j", "k", "n", "p", "q", "s")?

B. Constants
   1. Are all constant names upper case?
   2. Are constants defined via "# define"?
   3. Are constants that are used in multiple files defined in an INCLUDE header file?

C. Pointers Typing
   1. Are pointers declared and used as pointers (not integers)?
   2. Are pointers not typecast (except assignment of NULL)?

CONTROL

1. Are "else__if" and "switch" used clearly? (generally "else__if" is clearer, but "switch" may be used for not-mutually-exclusive cases, and may also be faster).
2. Are "goto" and "labels" used only when absolutely necessary, and always with well-commented code?
3. Is "while" rather than "do-while" used wherever possible?

LINKAGE

1. ARE "INCLUDE" files used according to project standards?
2. Are nested "INCLUDE" files avoided?
3. Is all data local in scope (internal static or external static) unless global linkage is specifically necessary and commented?
4. Are the names of macros all upper case?

I2 - Code Inspection Checklist (JPL) (Continued)

C (Continued)

COMPUTATION

A. Lexical Rules for Operators
   1. Are unary operators adjacent to their operands?
   2. Do primary operators "->" "." "()" have a space around them? (should have none.)
   3. Do assignment and conditional operators always have space around them?
   4. Are commas and semicolons followed by a space?
   5. Are keywords followed by a blank?
   6. Is the use of "(" following function name adjacent to the identifier?
   7. Are spaces used to show precedence? If precedence is at all complicated, are parentheses used (especially with bitwise ops)?

B. Evaluation Order
   1. Are parentheses used properly for precedence?
   2. Does the code depend on evaluation order, except in the following cases?
      a. exprl, expr2
      b. exprl? expr2 : exp2
      c. exprl & & expr2
      d. exprl || expr2
   3. Are shifts used properly?
   4. Does the code depend on order of effects? (e.g., i = i++;)?

MAINTENANCE

1. Are library routines used?
2. Are non-standard usages isolated in subroutines and well documented?
3. Does each module have one exit point?
4. Is the module easy to change?
5. Is the module independent of specific devices where possible?
6. Is the system standard defined types header used if possible (otherwise use project standard header, by "include")?
7. Is use of "int" avoided (use standard defined type instead)?

CLARITY

A. Comments
   1. Is the module header informative and complete?
   2. Are there sufficient comments to understand the code?
   3. Are the comments in the modules informative?
   4. Are comment lines used to group logically-related statements?
   5. Are the functions of arrays and variables described?
   6. Are changes made to a module after its release noted in the development history section of the header?

I2 - Code Inspection Checklist (JPL) (Continued)
C (Continued)

B. Layout

1. Is the layout of the code such that the logic is apparent?
2. Are loops indented and visually separated from the surrounding code?

C. Lexical Control Structures

Is a standard project-wide (or at least consistent) lexical control structure pattern used:

e.g.

```
    while (expr)
            {
            stmts;
            }
or
            while (expr) {
                  stmts;
            }                                    ETC.
```

I2 - Code Inspection Checklist (JPL) (Continued)

FORTRAN

FUNCTIONALITY

1. Do the modules meet the design requirements?
2. Does each module have a single purpose?
3. Is there some code in the module which should be a function or a subroutine?
4. Are utility modules used correctly?
5. Does the code match the Detailed Design specifications? If not, the design specifications may be in error.
6. Does the code impair the performance of the module (or program) to any significant degree?

DATA USAGE

A. General
   1. Is the data defined?
   2. Are there undefined or unused variables?
   3. Are there typos, particularly "O" for zero, and "l" for one?
   4. Are there misspelled names which are compiled as function or subroutine references?
   5. Are declarations in the correct sequence? (DIMENSION, EQUIVALENCE, DATA).

B. Common/Equivalence
   1. Are there local variables which are in fact misspellings of a COMMON element?
   2. Are the elements in the COMMON in the right sequence?
   3. Do EQUIVALENCE statements force any unintended shared data storage?
   4. Is each EQUIVALENCE commented?

C. Arrays
   1. Are all arrays DIMENSIONed?
   2. Are array subscript references in column, row order? (Check all indices in multi-dimensioned arrays.)
   3. Are array subscript references within the bounds of the array?
   4. Are array subscript references checked in critical cases?
   5. Is each array used for only one purpose?

I2 - Code Inspection Checklist (JPL) (Continued)

FORTRAN (Continued)

D. Variables
   1. Are the variables initialized in DATA statements, BLOCK DATA, or previously defined by assignments or COMMON usage?
   2. Should variables initialized in DATA statements actually be initialized by an assignment statement; that is, should the variable be initialized each time the module is invoked?
   3. Are variables used for only one purpose?
   4. Are variables used for logical unit assignments?
   5. Are the correct types (REAL, INTEGER, LOGICAL, COMPLEX) used?

E. Input and Output
   1. Do FORMATs correspond with the READ and WRITE lists?
   2. Is the intended conversion of data specified in the FORMAT?
   3. Are there redundant or unused FORMAT statements?
   4. Should this module be doing any I/O? Should it be using a message facility?
   5. Are messages understandable?
   6. Are messages phrased with the correct grammar? Do messages read like a robot or person talking? Robot: "Mount tape on drive. Turn on." Person: "Mount the tape on the tape drive. Then turn the tape drive on."
   7. Does each line of a message fit on all of the expected output devices?

F. Data
   1. Are all logical unit numbers and flags assigned correctly?
   2. Is the DATA statement used and not the PARAMETER statement?
   3. Are constant values constant?

CONTROL

A. Loops
   1. Are the loop parameters expressed as variables?
   2. Is the initial parameter tested before the loop in those cases where the initial parameter may be greater than the terminal parameter?
   3. Is the loop index within the range of any array it is subscripting? Is there a check in critical cases such as COMMONs?
   4. Is the index variable only used within the DO loop?
   5. If the value of the index variable is required outside the loop, is it stored in another location?
   6. Does the loop handle all the conditions required?
   7. Does the loop handle error conditions?

8. Does the loop handle cases which may "fall through"?

I2 - Code Inspection Checklist (JPL) (Continued)

FORTRAN (Continued)

9. Is loop nesting in the correct order?
10. Can loops be combined?
11. If possible, do nested loops process arrays as they are stored, with the innermost loop processing the first index (column index) and outer loops processing the row index?

B. Branches

1. Are branches handled correctly?
2. Are branches commented?
3. When using computed GO TOs, is the fall-through case tested, checked, and handled correctly?
4. Are floating point comparisons done with tolerances and never made to an exact value?

LINKAGE

1. Does the CALLing program have the same number of parameters as each routine?
2. Are the passed parameters in the correct order?
3. Are the passed parameters the correct type? If not, are mismatches proper?
4. Are constant values passed via a symbol (variable) rather than being passed directly?
5. Is an unused parameter named DUMMY, or some name which reflects its inactive status?
6. Is an array passed to a subroutine only when an array is defined in the subroutine?
7. Are the input parameters listed before the output parameters?
8. Does the subroutine return an error status output parameter?
9. Do the return codes follow conventions?
10. Are arrays used as intended?
11. If array dimensions are passed (dynamic dimensioning) are they greater than 0?
12. If a subroutine modifies an array, are the indices checked, or are the dimensions passed as parameters?
13. Does a subroutine modify any input parameter? If so, is this fact clearly stated?
14. Do subroutines end with a RETURN statement and not a STOP or a CALL EXIT?
15. Does a FUNCTION routine have only one output value?

COMPUTATION

1. Are arithmetic expressions evaluated as specified?
2. Are parentheses used correctly?
3. Is the use of mixed-mode expressions avoided?
4. Are intermediate results stored instead of recomputed?
5. Is all integer arithmetic involving multiplication and division performed correctly?

I2 - Code Inspection Checklist (JPL) (Continued)

FORTRAN (Continued)

6. Do integer comparisons account for truncation?
7. Are complex numbers used correctly?
8. Is the precision length selected adequate?
9. Is arithmetic performed efficiently?
10. Can a multiplication be used instead of a division? If so, is it commented so as not to obscure the process?

MAINTENANCE

1. Are library routines used?
2. Is non-standard FORTRAN isolated in subroutines and well documented?
3. Is the use of EQUIVALENCE limited so that it does not impede understanding the module?
4. Is the use of GO TOs limited so that it does not impede understanding the module?
5. Does each module have one exit point?
6. Is there no self-modifying code? (No ASSIGN statements, or PARAMETER statements.)
7. Is the module easy to change?
8. Is the module independent of specific devices where possible?
9. Where possible, are the CALLing routine parameter names the same as the subroutine parameter names?
10. Are type declarations implicit rather than explicit when possible?

CLARITY

1. Is the module header informative and complete?
2. Are there sufficient comments to understand the code?
3. Are the comments in the modules informative?
4. Are comment lines used to group logically-related statements?
5. Are the functions of arrays and variables described?

IT1 - Test Plan Checklist (JPL)

CLARITY
1. Does the Test Plan clearly specify the order of the steps of all integration testing?

COMPLETENESS
1. Does the Test Plan specify the overall approach and policy for acceptance test?
2. Does the Test Plan include a description of the type of hardware and software system environment to be used?
3. Does the Test Plan define success criteria for all tests?
4. Does the Test Plan adequately describe the functions being tested?
5. Does the Test Plan explicitly describe those functions that will not be tested during integration test?
6. Does the Test Plan describe conditions under which testing will be halted and resumed during integration test?
7. Does the test case set adequately exercise all significant code changes, particularly interface modifications?
8. Does the Test Plan adequately describe integration test baselines?
9. For a phased delivery, does the Test Plan establish test baselines in each phase for use in the next phase?
10. Does the Test Plan define sufficient and proper regression testing?

COMPLIANCE
1. Does the Test Plan list all the specifications, standards, and documents necessary for its development?

CONSISTENCY
1. Has the order of integration tests been defined to match the order of integration specified in higher level documents?
2. Is the Test Plan consistent with higher level test plan documents?

CORRECTNESS
1. Are the Test Plan entrance and exit criteria realistic?
2. Are all necessary drivers and stubs identified and available to test the function as specified?
3. Are all dependencies between the input simulator and the hardware addressed?

DATA USAGE
1. Does the test case set include adequate coverage of illegal and conflicting input combinations?
2. Does the test case set include adequate usage of default input values?
3. Does the test case set exercise an adequate number of program error paths?

FUNCTIONALITY

1. Is the Test Plan adequate to meet acceptance criteria?

IT1 - Test Plan Checklist (JPL) (Continued)

INTERFACES

1. Does the test case set adequately exercise the handling of information flow across external interfaces?

LEVEL OF DETAIL

1. Is the coverage of the test case set sufficiently complete to provide confidence that the functions being tested operate correctly within their intended environment?

MAINTAINABILITY

1. Are control and incorporation of changes to the specifications, design, or coding that may occur during test contained in the Test Plan?

PERFORMANCE

1. Are performance goals for the test procedures explicitly stated?

RELIABILITY

1. Is sufficient test data collected and documented to support estimation of the software's reliability?

TESTABILITY

1. Is the testing approach feasible?
2. Are all those requirements considered untestable and unable to be tested identified, and is it explained why they are untestable or unable to be tested?
3. Have development and procurement of test facilities (input simulators and output analyzers), methods, and tools been scheduled with adequate lead time?
4. Are the testing schedules described to a sufficient level of detail (testing schedules are described for each individual function to be tested)?
5. Is the method of estimating resource usage required for testing identified?
6. For multiple builds, have all requirements been identified on a per-build basis?
7. Have the roles and responsibilities for all personnel involved in the test activity been identified?
8. Is the specification of test facilities consistent with the test success criteria?
9. Are there any scheduling conflicts among the testing personnel schedules?
10. Does the Test Plan call for the participation of independent quality assurance personnel to verify test activity?
11. Does the Test Plan call for independent testing?

IT1 - Test Plan Checklist (JPL) (Continued)

TRACEABILITY

1. Do the acceptance tests exercise each requirement specified in higher level documents such as Functional Requirements Documents (FRD), Functional Design Document (FDD), and SRD?
2. Are the test acceptance criteria traceable to higher level requirements documents such as the SIS, User's Guide/Software Operator's Manual (UG/SOM), FRD, SRD, and FDD?
3. Does the test case set for integration test exercise each interface described in higher level documents (SIS and SSD)?

IT2 - Test Procedure and Function Checklist (JPL)

CLARITY

1. Are the operator instructions explicit and clear for ease of execution of the test procedure?
2. Are the steps of the set-up and test procedures precise, unambiguous, and listed as individual items?
3. Are there "progress" messages that will notify the operator when significant parts of the test are being executed?
4. Are the criteria for success and failure clear and unambiguous?

COMPLETENESS

1. Is the expected response to each step of the test procedure described with the operator instructions for that step?
2. Does the test procedure list the precedence of tests?
3. Does the test procedure indicate the significance of proper evaluation of test results?
4. Do the test procedures lead to the determination of success or failure?

COMPLIANCE

1. Does the Test Plan list all the specifications, standards, and documents necessary for its development?

CONSISTENCY

1. Are all dependencies of the test procedure identified?

CORRECTNESS

1. Do the observed results of performing the procedure agree with the expected program behavior?
2. Are the interfaces between the code being tested and the test equipment and software correct?
3. Are the formats of the input data correct?
4. Are the operator instructions presented step-by-step and in the order in which they must be performed?
5. Is the function being tested accurately described?
6. Is the function being tested the latest revision?
7. Is the description of the purpose of this test procedure complete and accurate?
8. Are there criteria for test success and failure?

DATA USAGE

1. Are an adequate number of control paths in the tested function exercised?
2. Are an adequate number of logical condition expressions in the tested function exercised?

IT2 - Test Procedure and Function Checklist (JPL) (Continued)

3. Do the test cases demonstrate the program's response to illegal and conflicting input data?

FUNCTIONALITY
1. Is each requirement associated with this function exercised by this test procedure?
2. Does the procedure state whether or not it is possible to continue in the event of a program stop or indicated error? If so, does it indicate the method for restarting or other recovery action?

INTERFACES
1. Does the test case set adequately exercise the handling of information flow across external interfaces?

LEVEL OF DETAIL
1. Are all normal and abnormal completion messages identified?

MAINTAINABILITY
1. Are control and incorporation of changes to the specifications, design, or coding that may occur during test contained in the Test Plan?

PERFORMANCE
1. If a performance criterion is associated with any step of the test procedure, is that criterion explicitly stated along with the operator instructions for that step?

RELIABILITY
1. Has the test equipment been validated and calibrated?
2. Has the test software been validated?
3. Have all input data been verified?
4. Is sufficient test data collected and documented to support estimation of the software's reliability?

TESTABILITY
1. Does the test procedure identify all of the equipment, software, and personnel required for testing?
2. Can the test procedure be performed with minimal support from the development team?
3. Is the test procedure consistent with the capabilities of the test facilities?
4. Is the testing schedule described to a sufficient level of detail?
5. Does the test procedure call for the participation of independent quality assurance personnel to very testing activity?
6. Does the test procedure call for independent testing?

IT2 - Test Procedure and Function Checklist (JPL) (Continued)

TRACEABILITY

1. Does the test procedure list all specifications, procedures, handbooks, or manuals required for operation?
2. Is the traceability shown between the requirements and the acceptance test combinations?
3. Are the criteria for success traced to requirements?
4. Is the creator of each test case dataset identified?

R0 - Functional Design Checklist (JPL)

CLARITY

1. Have the hardware and software environments been described? Have all external systems been included?
2. Has the high level architecture been described, illustrated and made consistent with the lower level descriptions?
3. Has the primary purpose for the software been defined?
4. Has the overall functional design been described?

COMPLETENESS

1. Have feasibility analyses been performed and documented (e.g., prototyping, simulations, analogies to current system)?
2. Have all design and implementation goals and constraints been defined?
3. Have the capabilities of each component for each stage or phased delivery been identified?
4. If assumptions have been made due to missing information, have they been documented?
5. Have all TBD requirements from the FRD been analyzed?
6. Have trade studies been performed and documented?
7. Have all tradeoffs and decisions been described and justified? Are selection criteria and alternatives included?
8. Has the subsystem been sized (using lines of code or an alternate method)?
9. Have initialization, synchronization, and control requirements been described?

COMPLIANCE

1. Does the documentation follow project and/or JPL standards?

CONSISTENCY

1. Are the requirements in this document consistent with each other?
2. Are the requirements consistent with the FRD, external interfaces, and any other related documents?

CORRECTNESS

1. Does the design seem feasible with respect to cost, schedule, and technology?
2. Do state diagrams clearly represent the timing?
3. Have assumptions about intended sequences of functions been stated? Are these sequences required?
4. Is the design consistent with the actual operating environment (e.g., hardware timing, precision, event sequencing, data rates, bandwidth)?

R0 - Functional Design Checklist (JPL) (Continued)

DATA USAGE
1. Are data elements named and used consistently?
2. Has all shared data between subsystems been identified?
3. Have the means for shared data management been described? Are the subsystems which set and/or use the shared data indicated?
4. Has the dataflow among hardware, software, personnel, and procedures been described?

FUNCTIONALITY
1. Are all described functions necessary and sufficient to meet the mission/system objectives?
2. Are all inputs to a function necessary and sufficient to perform the required operation?
3. Are all the outputs produced by a function used by another function or transferred across an external interface?
4. Do all functions clearly state how the output is derived from input or shared data?
5. Are all functional states defined?

INTERFACE
1. Are the internal and external interfaces clearly defined?
2. Have all interfaces between systems, hardware, software, personnel, and procedures been functionally described?
3. Have the requirements for data transfer across each interface been stated?
4. Have the number and complexity of the interfaces been minimized and are they consistent?
5. Are the inputs and outputs for all the interfaces sufficient and necessary?

LEVEL OF DETAIL
1. Are the requirements free of unwarranted design?
2. Does each requirement in the FRD trace to one or more requirements in the FDD?
3. Is there enough detail to proceed to the next phase of the life cycle?
4. Have all "TBDs" been resolved?

MAINTAINABILITY
1. Have the requirements for software maintainability been specified?
2. Have risk areas of the design been identified and isolated? Does the design complexity agree with development risk, cost, and schedule?

R0 - Functional Design Checklist (JPL) (Continued)

3. Have all inherited or procured subsystems been documented? Has a cost/benefit analysis been identified?
4. Are reusable parts of other designs being used? Has their effect on design and integration been stated?
5. Are the requirements weakly coupled? Have the number of requirements that are affected when one requirement is changed been minimized?
6. Have analyses been done for cohesion, coupling, traffic statistics, etc.?
7. Do the design features enable the system to meet maintainability requirements?

PERFORMANCE

1. Are all performance attributes, assumptions, and constraints clearly defined?
2. Do all explicit and implicit performance requirements have metrics expressed (e.g., timing, throughput, memory size, accuracy, precision)?
3. For each performance requirement identified (explicit or implicit):
   a. Have the performance estimates been documented?
   b. Do rough estimates indicate that they can be met? Is the impact of failure defined?
   c. Do experiments, prototypes, or analyses verify that the requirements can be met?

RELIABILITY

1. Has an explicit reliability goal been stated?
2. Do the design features enable the system to meet reliability requirements?
3. Are normal operating conditions/errors taken into account? Are special states considered (e.g., cold starts, abnormal termination, recovery)?
4. Have fault tolerance features been identified or analyzed?
5. Have the subsystem level error detection, reporting, and recovery features for internal and external errors been described?

TESTABILITY

1. Can the program sets be tested, demonstrated, analyzed, or inspected to show that they satisfy requirements?
2. Can the subsystem components be developed and tested independently? Incrementally?
3. Have any special integration or integration testing constraints been levied?

TRACEABILITY

1. Are the priorities of the requirements documented? Is the impact of not achieving the requirements defined?
2. Are requirement traceability exceptions justified?

3. Have all of the requirements been allocated to hardware, software, personnel, or procedures?

R0 - Functional Design Checklist (JPL) (Continued)

4. Are all functions, structures, and constraints traced to requirements and vice versa?
5. Are requirements stated in a manner so that they can be uniquely referenced in subordinate documents?
6. Are the architectural components for each stage of implementation identified for reference in subordinate documents?

R1 - Software Requirements Checklist (JPL)

CLARITY
1. Are the goals of the subsystem defined?
2. Is the terminology consistent with the users' and/or sponsors' terminology?
3. Are the requirements clear and unambiguous?
4. Is a functional overview of the program set provided?
5. Is an overview of the operational modes, states, and concept described?
6. Have the software environment (co-resident program sets) and hardware environment (specific configurations) been specified?
7. If assumptions that affect implementation have been made, are they stated?
8. Have the requirements been stated in terms of inputs, outputs, and processing for each function?

COMPLETENESS
1. Are required attributes, assumptions, and constraints of the program set completely listed?
2. Have all requirements and constraints been assigned a priority?
3. Have the criteria for assigning requirement priority levels been defined?
4. Have the requirements been stated for each delivery or staged implementation?
5. Have requirements for installation (packaging, site preparation, operator training) been specified?
6. Have the target language, development environment, and run-time environment been chosen?

COMPLIANCE
1. Does the documentation follow project and/or JPL standards?

CONSISTENCY
1. Are the requirements mutually consistent?
2. Are the requirements in this document consistent with the requirements in related documents?
3. Are the requirements consistent with the actual operating environment (e.g., check hardware timing, precision, event sequencing, data rates, bandwidth)?
4. Do the requirements stay within the capability of the requirements allocated by the FDD?

CORRECTNESS
1. Do the requirements seem feasible with respect to cost, schedule, and technology?
2. Are the requirements consistent with the actual operating environment (e.g., hardware timing, precision, event sequencing, data rates, bandwidth)?

R1 - Software Requirements Checklist (JPL) (Continued)

DATA USAGE

1. Have the data type, rate, units, accuracy, resolution, limits, range, and critical values for all internal data items been specified?
2. Have the data objects and their component parts been specified?
3. Has the mapping between local views of data and global data been shown?
4. Has the management of stored and shared data been described?
5. Has a list of functions that set and/or use stored and shared data been provided?
6. Are there any special integrity requirements on the stored data?
7. Have the types and frequency of occurrence of operations on stored data (e.g., retrieve, store, modify, delete) been specified?
8. Have the modes of access (e.g., random, sequential) for the shared data been specified?

FUNCTIONALITY

1. Are all described functions necessary and sufficient to meet the mission/system objectives?
2. Are all inputs to a function necessary and sufficient to perform the required operation?
3. Does each function clearly describe how outputs (and shared data) are generated from inputs (and shared data)?
4. Are all function states defined?

INTERFACE

1. Are the inputs and outputs for all the interfaces sufficient and necessary?
2. Are all the outputs produced by a function used by another function or transferred across an external interface?
3. Are the interface requirements between hardware, software, personnel, and procedures included?
4. Have the contents, formats, and constraints of all the displays been described in the SRD or Software Operator's Manual (SOM-1)?
5. Are all data elements crossing program set boundaries identified?
6. Are all data elements described here or in the SIS-1?
7. Has the data flow between internal software functions been represented?

R1 - Software Requirements Checklist (JPL) (Continued)

LEVEL OF DETAIL

1. Are the requirements free of design?
2. Have all "TBDs" been resolved?
3. Have the interfaces been described to enough detail for design work to begin?
4. Have the accuracy, precision, range, type, rate, units, frequency, and volume of inputs and outputs been specified for each function?
5. Have the functional requirements been described to enough detail for design work to begin?
6. Have the performance requirements been described to enough detail for design work to begin?

MAINTAINABILITY

1. Are the requirements weakly coupled (i.e., changing a function will not have adverse and unexpected effects throughout the subsystem)?
2. Will the requirements minimize the complexity of the design?
3. Have FRD and FDD maintainability requirements been levied to functions?
4. Have FRD and FDD portability requirements been levied to functions?
5. Has the use of inherited design or code or pre-selected tools been specified?

PERFORMANCE

1. Have the FRD and FDD performance requirements been allocated to each function?
2. Have the resource and performance margin requirements been stated along with the means for managing them?

RELIABILITY

1. Have quality factors been specified as measurable requirements or prioritized design goals?
2. Have FRD and FDD reliability requirements been levied to functions?
3. Have FRD and FDD availability requirements been levied to functions?
4. Have FRD and FDD security/safety requirements been levied to functions?
5. Are error checking and recovery required?
6. Are undesired events considered and their required responses specified?
7. Are initial or special states considered (e.g., cold starts, abnormal termination)?
8. Have assumptions about intended sequences of functions been stated? Are these sequences required?

R1 - Software Requirements Checklist (JPL) (Continued)

TESTABILITY

1. Can the program set be tested, demonstrated, analyzed, or inspected to show that it satisfies the requirements?
2. Are the individual requirements stated so that they are discrete, unambiguous, and testable?
3. Have the overall program set acceptance criteria been established?
4. Have clear pass/fail criteria for the acceptance tests been established?
5. Have the test methods (test, demonstration, analysis, or inspection) been stated for each requirement?

TRACEABILITY

1. Are all functions, structures, and constraints traced to requirements, and vice versa?
2. Have the FDD and Integrated Software Functional Diagram (ISFD) requirements been allocated to functions of the program set?
3. Do the requirements (or traceability matrix) indicate whether they are imposed by the FDD or whether they are derived to support specific FDD requirements?
4. Have the FRD, FDD, and any derived design goals and implementation constraints been specified and prioritized?
5. Is each requirement stated in a manner that it can be uniquely referenced in subordinate documents?

SY - Functional Requirements Checklist (JPL)

CLARITY
1. Are requirements specified in an implementation free way so as not to obscure the original requirements?
2. Are implementation and method and technique requirements kept separate from functional requirements?
3. Are the requirements clear and unambiguous (i.e, are there aspects of the requirements that you do not understand; can they be misinterpreted)?

COMPLETENESS
1. Are requirements stated as completely as possible? Have all incomplete requirements been captured as TBDs?
2. Has a feasibility analysis been performed and documented?
3. Is the impact of not achieving the requirements documented?
4. Have trade studies been performed and documented?
5. Have the security issues of hardware, software, operations personnel and procedures been addressed?
6. Has the impact of the project on users, other systems, and the environment been assessed?
7. Are the required functions, external interfaces and performance specifications prioritized by need date? Are they prioritized by their significance to the system?

COMPLIANCE
1. Does this document follow the project's system documentation standards? Does it follow JPL's standards? Does the appropriate standard prevail in the event of inconsistencies?

CONSISTENCY
1. Are the requirements stated consistently without contradicting themselves or the requirements of related systems?
2. Is the terminology consistent with the user and/or sponsor's terminology?

CORRECTNESS
1. Are the goals of the system defined?

DATA USAGE
1. Are "don't care" condition values truly "don't care"? ("Don't care" values identify cases when the value of a condition or flag is irrelevant, even though the value may be important for other cases.) Are "don't care" condition values explicitly stated? (Correct identification of "don't care" values may improve a design's portability.)

SY - Functional Requirements Checklist (JPL) (Continued)

FUNCTIONALITY
1. Are all functions clearly and unambiguously described?
2. Are all described functions necessary and together sufficient to meet mission and system objectives?

INTERFACES
1. Are all external interfaces clearly defined?
2. Are all internal interfaces clearly defined?
3. Are all interfaces necessary, together sufficient, and consistent with each other?

MAINTAINABILITY
1. Have the requirements for system maintainability been specified in a measurable, verifiable manner?
2. Are requirements written to be as weakly coupled as possible so that rippling effects from changes are minimized?

PERFORMANCE
1. Are all required performance specifications and the amount of performance degradation that can be tolerated explicitly stated (e.g., consider timing, throughput, memory size, accuracy and precision)?
2. For each performance requirement defined:
   a. Do rough estimates indicate that they can be met?
   b. Is the impact of failure to meet the requirement defined?

RELIABILITY
1. Are clearly defined, measurable, and verifiable reliability requirements specified?
2. Are there error detection, reporting, and recovery requirements?
3. Are undesired events (e.g., single event upset, data loss or scrambling, operator error) considered and their required responses specified?
4. Have assumptions about the intended sequence of functions been stated? Are these sequences required?
5. Do these requirements adequately address the survivability after a software or hardware fault of the system from the point of view of hardware, software, operations personnel and procedures?

TESTABILITY
1. Can the system be tested, demonstrated, inspected or analyzed to show that it satisfies requirements?
2. Are requirements stated precisely to facilitate specification of system test success criteria and requirements?

SY - Functional Requirements Checklist (JPL) (Continued)

TRACEABILITY

1. Are all functions, structures and constraints traced to mission/system objectives?
2. Is each requirement stated in such a manner that it can be uniquely referenced in subordinate documents?

APPENDIX B

Appendix B consists of graphics that were inserted into the text. Machine readable copies are not available, thus Appendix B has been deleted from this version

APPENDIX C

REFERENCES

Ackerman et. al., "Software Inspections: An Effective Verification Process," IEEE Software, May 1989.

Fagan, Michael E., "Design and Code Inspections to Reduce Errors in Program Development," IBM Systems Journal, Vol. 15, No. 3, 1976.

Fagan, Michael E., "Advances in Software Inspections," IEEE Transactions on Software Engineering, Vol. SE-12, No. 7, July 1986.

Freedman, Daniel P. and Weinberg, Gerald M., "Handbook of Walkthroughs, Inspections, and Technical Reviews, Evaluating Programs and Projects," 1982.

Jet Propulsion Laboratory, Formal Inspections for Software Development, Rev. E (Training Course), 1990.

Kelly, John C. and Hops, Jonathan, "Software Inspections: An Experience Report" (Draft), January 26, 1990.

Kelly, John C., "Original Working Draft of Inspections and Walkthroughs Guidebook," January 1990.