NASA/TM-2008-215550

Guidance and Control Software Project Data

Volume 1: Planning Documents

Edited by
Kelly J. Hayhurst
Langley Research Center, Hampton, Virginia

December 2008

NASA/TM-2008-215550

Guidance and Control Software Project Data

Volume 1: Planning Documents

Edited by
Kelly J. Hayhurst
Langley Research Center, Hampton, Virginia

National Aeronautics and
Space Administration

Langley Research Center
Hampton, Virginia 23681-2199

December 2008

Available from:

NASA Center for AeroSpace Information (CASI)
7115 Standard Drive
Hanover, MD 21076-1320
(301) 621-0390

National Technical Information Service (NTIS)
5285 Port Royal Road
Springfield, VA 22161-2171
(703) 605-6000

Table of Contents

Abstract

The Guidance and Control Software (GCS) project was the last in a series of software reliability studies conducted at Langley Research Center between 1977 and 1994. The technical results of the GCS project were recorded after the experiment was completed. Some of the support documentation produced as part of the experiment, however, is serving an unexpected role far beyond its original project context. Some of the software used as part of the GCS project was developed to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. That standard requires extensive documentation throughout the software development life cycle, including plans, software requirements, design and source code, verification cases and results, and configuration management and quality control data. The project documentation that includes this information is open for public scrutiny without the legal or safety implications associated with comparable data from an avionics manufacturer. This public availability has afforded an opportunity to use the GCS project documents for DO-178B training. This report provides a brief overview of the GCS project, describes the 4-volume set of documents and the role they are playing in training, and includes the planning documents from the GCS project.

1 Introduction and Background on Software Error Studies

As the pervasiveness of computer systems has increased, so has the desire and obligation to establish the reliability of these systems. Reliability estimation and prediction are standard activities in many engineering projects. For the software aspects of computer systems, however, reliability estimation and prediction have been topics of dispute, especially for safety-critical systems. A primary challenge is how to accurately model the failure behavior of software such that numerical estimates of reliability have sufficient credibility for systems where the probability of failure needs to be quite small, such as in commercial avionics systems (ref. 1). A second challenge is how to gather sufficient data to make such estimates. Software reliability models are not used in the civil aviation industry, for example, because “currently available methods do not provide results in which confidence can be placed to the level required for this purpose.” (ref. 2)

In an effort to develop methods to credibly assess the reliability of software for safety-critical avionics applications, Langley Research Center initiated a Software Error Studies program in 1977 (ref. 3). A major focus of those studies was on generating significant quantities of software failure data through controlled experimentation to better understand software failure processes. The intent of the Software Error Studies program was to incrementally increase complexity and realism in a series of experiments so that the final study would have statistically valid results, representative of actual software development processes.

The Software Error Studies program started with initial investigations by the Aerospace Corporation to define software reliability measures and data collection requirements (ref. 4-6).

Next, Boeing Computer Services (BCS) and the Research Triangle Institute (RTI) conducted several simple software experiments with aerospace applications including missile tracking, launch interception, spline function interpolation, Earth satellite calculation, and pitch axis control (refs. 7-11). The experiment design used in these studies generally involved a number of programmers (denoted *n*) who independently generated computer code from a given specification of the problem to produce *n* versions of a program. In these experiments, no particular software development standards or life-cycle models were followed. Because the problems were relatively small and simple, the versions were compared to a known error-free version of the program to obtain information on software errors.

Although the initial experiments were small and simplistic compared with real-world avionics development, they yielded some interesting results that have influenced software reliability modeling. The BCS and RTI studies showed widely varying error rates for faults. This finding refuted a common assumption in early software reliability growth models that faults produced errors at equal rates. These studies also provided evidence of fault interaction where one fault could mask potentially erroneous behavior from another fault, or where two or more faults together cause errors when alone they would not. (ref. 12) Additional investigations with *n*-version programs (ref. 13) found that points in the input space that cause an error can cluster and form "error crystals". Extrapolating this finding to aerospace applications, where input signals tend to be continuous in nature, the error crystals may manifest themselves as clusters of successive faults that could have unintended consequences. (ref. 14)

The last project in the Software Error Studies program was the Guidance and Control Software (GCS) project. It built on the previous experiments in two ways: (1) by requiring that the software specimens for the experiment be developed in compliance with current software development standards, and (2) by increasing the complexity of the application problem (ref. 15). At the time of the GCS project, the RTCA/DO-178B guidelines, "Software Considerations in Airborne Systems and Equipment Certification," (ref. 2) were the primary standard sanctioned by the Federal Aviation Administration (FAA) for developing software to be approved for use in commercial aircraft equipment (ref. 16). The DO-178B document describes objectives and design considerations to be used for the development of software as well as verification, configuration management, and quality assurance activities to be performed throughout the development process. The DO-178B guidelines were selected as the software development standard to be used for the GCS specimens.

The software application selected for the GCS project, as the title indicates, is a guidance and control function for controlling the terminal descent trajectory of a planetary lander vehicle. This terminal descent trajectory is the same fundamental trajectory referred to as the "seven minutes of terror" in the entry, descent, and landing phase of a planetary mission, such as the recent Phoenix Mars Lander (ref. 17). For the GCS project, the software requirements were reverse engineered from a simulation program used to study the probability of success of the original NASA Viking Lander mission to Mars in the 1970s (ref. 18). It is important to emphasize that the software requirements documented for the GCS project, while realistic, are not the actual software requirements used for NASA's Viking Lander or any other planetary landers.

For the GCS experiment, two[1] teams of software engineers were each tasked to independently design, code, and verify a GCS program, following the software development guidance in DO-178B, as closely as possible. In addition to those teams, another GCS version was produced, without the constraint of compliance with DO-178B, to aid development and verification of the requirements and simulation environment. Once all versions were complete, data on residual

[1] The original plan for the GCS project called for three independent teams. Due to funding constraints, only two teams were able to complete the project.

errors was supposed to be collected by running all the versions simultaneously in a simulation environment, and using any discrepancies among the results of the versions as possible indications of errors.

Results of the operational simulations and data collection are described in (ref. 15). The purpose of this report is not to repeat those results, but to disseminate some of the project documentation that has an unanticipated utility beyond its original project context. The project documentation of interest is the documentation developed by the teams required to comply with the DO-178B standard. That standard requires extensive records of all of the software development life cycle activities. For the GCS project, those records included 18 documents consisting of life cycle plans, development products including requirements and source code, verification cases and results, and configuration management and quality control data. Comparable data from a commercial avionics system would not be available for public review because of proprietary and other legal considerations. The GCS project documentation is not subject to those considerations because it is not data from an actual operational, or even prototype, system. But, the data has sufficient realism to provide a window into the types of activities and data involved in the production of DO-178 compliant software, which makes the GCS documentation desirable from a training perspective.

The remainder of this report provides a brief overview of aspects of the GCS project relevant to using the documentation for training. This information includes a description of the GCS application, a synopsis of the software development processes used to follow the DO-178B guidance, and the data that was generated as a result. Because the complete set of compliance documents is large, the documents have been divided into four sets (planning, development, verification, and other integral process documents) contained in separate volumes of this report. Volume 1 includes in Appendices A-E all of the GCS documents generated as part of the planning process documentation.

2 Guidance and Control Software Application

The requirements for the GCS application focus on two primary functions: (1) to provide guidance and engine control of the lander vehicle during its terminal phase of descent onto the planet's surface, and (2) to communicate sensory information to an orbiting platform about the vehicle and its descent. Figure 1 shows a sketch of the lander vehicle, taken from (ref. 18), noting the location of the terminal descent propulsion systems.

The guidance package for the lander vehicle contains sensors that obtain information about the vehicle state and environment, a guidance and control computer, and actuators providing the thrust necessary for maintaining a safe descent. The vehicle has three accelerometers (one for each body axis), one Doppler radar with four beams, one altimeter radar, two temperature sensors, three strapped-down gyroscopes, three opposed pairs of roll engines, three axial thrust engines, one parachute release actuator, and a touch down sensor. The vehicle has a hexagonal, box-like shape; three legs and a surface sensing rod protrude from its undersurface.

In general, the requirements for the planetary lander only concern the final descent to the surface. Figure 2 shows a sketch of the phases of the terminal descent trajectory.

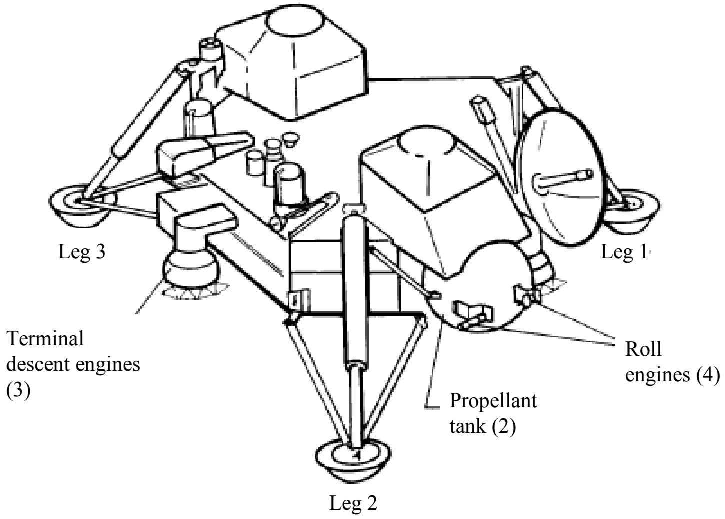


Figure 1. Lander with Terminal Descent Propulsion Systems

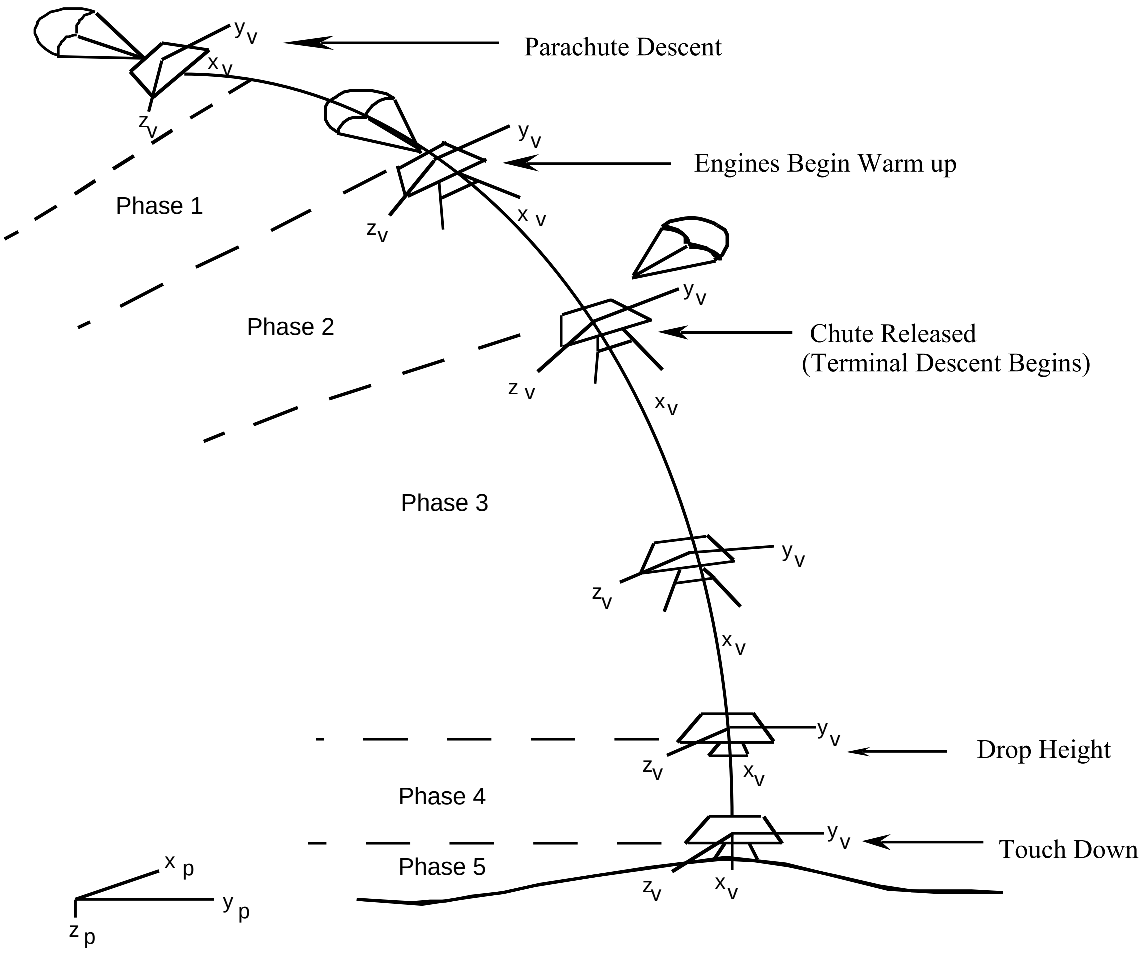


Figure 2. A Typical Terminal Descent Trajectory

After the lander has dropped from orbit, the software controls the engines of the vehicle to the surface of a planet. The initialization of the GCS starts the sensing of vehicle altitude. When a predefined engine ignition altitude is sensed by the altimeter radar, the GCS begins guidance and control of the lander. The axial and roll engines are ignited; while the axial engines are warming up, the parachute remains connected to the vehicle. During this engine warm-up phase, the aerodynamics of the parachute dictate the vehicle's trajectory. Vehicle attitude is maintained by firing the engines in a throttled-down condition. Once the main engines become hot, the parachute is released and the GCS performs an attitude correction maneuver and then follows a controlled acceleration descent until a predetermined velocity-altitude contour is crossed. The GCS then attempts to maintain the descent of the lander along this predetermined velocity-altitude contour. The lander descends along this contour until a predefined engine shut off altitude is reached or touchdown is sensed. After all engines are shut off, the lander free-falls to the surface.

The software requirements for this guidance and control application are contained in a document called the *Guidance and Control Development Specification* (in Volume 2). As mentioned earlier, the initial requirements for this application were reverse engineered from a simulation program used to study the probability of success of the original NASA Viking Lander mission to Mars. Prior to use in the experiment, the requirements were revised to make them suitable for use in an *n*-version software experiment. Each of the GCS programs for the experiment were developed from the same requirements document.

3 Software Life Cycle Processes and Documentation

Having some of the project teams adhere to the DO-178B guidelines as they created a software version for the experiment was a significant element of the GCS project, requiring the development and tracking of numerous software engineering artifacts not normally associated with a software engineering experiment. The purpose of DO-178B is to provide guidelines for the production of software such that the completed implementation performs its intended function with a level of confidence in safety satisfactory for airworthiness. Along with the production of software is the generation of an extensive set of documents recording the production activities.

DO-178B defines software development activities and objectives for the development life cycle of the software, and the evidence that is needed to show compliance. The life-cycle processes are divided into planning, development, and integral processes. The planning process defines and coordinates the software development processes and the integral processes. The software development processes involve identification of software requirements, software design and coding, and integration; that is, the development processes directly result in the software product. Finally, the integral processes function throughout the software development processes to ensure integrity of the software products. The integral processes include software verification, configuration management, and quality assurance processes. Section 11 of DO-178B describes data that should be produced as evidence of performing all of the life cycle process activities (see Table 1).

For the GCS project, some of this data was common for all of the teams, and other data was intended to be specific to each team. For example, each team worked with the same plans, standards, and requirements. Then, each individual team was responsible for independently developing their own design, code, and corresponding verification data. To distinguish the versions, each team was assigned a planetary name: Mercury, Venus, and Pluto[2].

[2] At the time the GCS experiment was conducted, Pluto had not yet been relegated to non-planet status.

Table 1. Life Cycle Data

Planning Process Documents	Development Process Documents	Integral Process Documents
• Plan for Software Aspects of Certification • Software Development Plan • Software Verification Plan • Software Configuration Management Plan • Software Quality Assurance Plan • Software Requirements Standards • Software Design Standards • Software Code Standards	• Software Requirements Data • Design Description • Source Code • Executable Object Code	• Software Verification Cases and Procedures • Software Verification Results • Software Life Cycle Environment Configuration Index • Software Configuration Index • Problem Reports • Software Configuration Management Records • Software Quality Assurance Records • Software Accomplishment Summary

The DO-178B data associated with the development of the Pluto version of the GCS was selected for publication. Most of the GCS documents correspond directly with the life cycle data listed in Table 1. All together, the documentation includes over 1000 pages. So, for dissemination purposes, the Pluto data was divided into the following 4 subsets:

Volume 1: Planning Documents

- *Plan for Software Aspects of Certification of the Guidance and Control Software Project*
- *Software Configuration Management Plan for the Guidance and Control Software Project*
- *Software Quality Assurance Plan for the Guidance and Control Software Project*
- *Software Verification Plan for the Guidance and Control Software Project*
- *Software Development Standards for the Guidance and Control Software Project*

Volume 2: Development Documents

- *Guidance and Control Software Development Specification*
- *Design Description for the Pluto Implementation of the Guidance and Control Software*
- *Source Code for the Pluto Implementation of the Guidance and Control Software*

Volume 3: Verification Documents

- *Software Verification Cases and Procedures for the Guidance and Control Software Project*
- *Software Verification Results for the Pluto Implementation of GCS*
- *Review Records for the Pluto Implementation of the Guidance and Control Software*
- *Test Results Logs for the Pluto Implementation of the Guidance and Control Software*

Volume 4: Other Integral Processes Documents

- *Software Accomplishment Summary for the Guidance and Control Software Project*
- *Software Configuration Index for the Guidance and Control Software Project*
- *Problem Reports for the Pluto Implementation of the Guidance and Control Software*
- *Support Documentation Change Reports for the Guidance and Control Software Project*
- *Configuration Management Records for the Guidance and Control Software Project*
- *Software Quality Assurance Records for the Guidance and Control Software Project*

Appendices A thru E in this volume contain all of the original planning documents for the GCS Project. The *Plan for Software Aspects of Certification*, in Appendix A, provides a comprehensive overview of the GCS Project including an overview of the guidance and control application, statement of certification considerations, discussion of the software development plan, and the project milestones and schedule. The *Configuration Management Plan*, *Software Quality Assurance Plan*, and *Verification Plan,* in Appendices B-D, provide details about the activities to be conducted to satisfy DO-178B objectives for those processes. Appendix E contains the S*oftware Development Standards* that specify constraints and rules on defining the software requirements, and designing and coding the software. These standards, along with the software requirements, set the basis for evaluating actual project results with expected results.

The content of the documents in the appendices has not been altered from the original versions produced during the project.

4 Role in Training

At the time of the GCS project, there was no publicly available information, such as templates, or examples, or training courses, to help a novice developer generate the type of evidence that a certificating authority would expect to see to demonstrate compliance with DO-178B. As mentioned earlier, compliance data from a real avionics system is not typically available for public review because of various legal and safety considerations. For example, an avionics manufacturer would likely consider the design and implementation of a system to be proprietary. Those considerations do not apply to the data from the GCS project, because neither the requirements nor the software versions represent an actual system with safety, liability, or other considerations.

In addition to the availability of data, the GCS requirements and DO-178B compliance data are sufficiently realistic to serve as an example of a DO-178B project: one that is small enough in scale to be studied in a training course. The GCS documentation provides a window into the activities and data produced throughout the development life cycle to comply with DO-178B. Because the Federal Aviation Administration (FAA) was aware of the GCS project, they recognized the potential value of the documentation for training. The FAA has designed software training to include a case study portion that addresses avionics software issues that arise from the application of the DO-178B guidelines. The case study gives students the opportunity to use auditing techniques to identify flaws in lifecycle data. Because the GCS data was produced by novices, there are plenty of flaws to find.

5 Summary

From 1977-1994, NASA Langley Research Center conducted a Software Error Studies program that generated data that provided insights into the software failure process and into conducting software engineering experiments as well. The GCS project was the final experiment in that program. A unique feature of the GCS project was the requirement for some of the software specimens used in the experiment to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. The project documentation produced to meet that requirement has had the unanticipated benefit of serving as case study material in software certification training long after the conclusion of the original experiment. Volume 1 of this report contains all of the planning documents from the GCS project. Other volumes of this report contain the rest of the GCS compliance data including development, verification, configuration management and quality assurance documents.

6 References

1. Littlewood, Bev, and Strigini, Lorenzo, Software Reliability and Dependability: a Roadmap, 22nd International Conference on Software Engineering, Future of Software Engineering Track, June 4-11, 2000, Limerick Ireland, pp. 175 – 188.

2. Software Considerations in Airborne Systems and Equipment Certification. Doc. No. RTCA/DO-178B, RTCA, Inc., Dec. 1, 1992.

3. Finelli, George B.: NASA Software Failure Characterization Experiments. Reliability Engineering & System Safety, vol. 32, pp. 155–169, 1991.

4. Hecht, H.; Sturm, W. A.; and Tratlner, S.: Reliability Measurement During Software Development. NASA CR-145205, 1977.

5. Hecht, H.: Measurement Estimation and Prediction of Software Reliability. NASA CR-145135, 1977.

6. Maxwell, F. D.: The Determination of Measures of Software Reliability. NASA CR-158960, 1978.

7. Nagel, Phyllis M.; and Skrivan, James A.: Software Reliability: Repetitive Run Experimentation and Modeling. NASA CR-165836, 1982.

8. Nagel, P. M.; Scholz, F. W.; and Skrivan, J. A.: Software Reliability: Additional Investigation Into Modeling With Replicated Experiments. NASA CR-172378, 1984.

9. Dunham, Janet R.: Experiments in Software Reliability: Life-Critical Applications. IEEE Transactions on Software Engineering, vol. SE-12, no. 1, Jan. 1986, pp. 110–123.

10. Dunham, J. R.; and Lauterbach, L. A.: An Experiment in Software Reliability Additional Analyses Using Data From Automated Replications. NASA CR-178395, 1987.

11. Dunham, Janet R.; and Pierce, John L.: An Empirical Study of Flight Control Software Reliability. NASA CR-178058, 1986.

12. Dunham, Janet R.; and Finelli, George B., Real-Time Software Failure Characterization, IEE Aerospace and Electronic Systems Magazine, pp. 38-44, November 1990.

13. Ammann, P. and Knight, J.: "Data Diversity: An Approach To Software Fault Tolerance", Digest of Papers FTCS-17: The 17th Annual International Symposium on Fault Tolerant Computing, Pittsburg, Pennsylvania, July 1987.

14. Finelli, George B, Results of Software Error-Data Experiments, AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, September 7-9, 1988, Atlanta, Georgia, AIAA-88-4436.

15. Hayhurst, Kelly J., Framework for Small-Scale Experiments in Software Engineering, Guidance and Control Software Project: Software Engineering Case Study, NASA/TM-1998-207666, May 1998.

16. Federal Aviation Administration, Advisory Circular, 20-115B, January 11, 1993.

17. Tobin, Kate, NASA Preps for '7 Minutes of Terror' on Mars, May 23, 2008, http://www.cnn.com/2008/TECH/space/05/23/mars.lander/index.html.

18. Holmberg, Neil A.; Faust, Robert P.; and Holt, H. Milton: Viking '75 Spacecraft Design and Test Summary. Volume I—Lander Design. NASA RP-1027, 1980.

Appendix A: Plan for Software Aspects of Certification for the Guidance and Control Software Project

Author: Kelly J. Hayhurst, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

A. Contents

A.1 Introduction

As stated in section 11.1 of the Requirements and Technical Concepts for Aviation RTCA/DO-178B guidelines, "Software Considerations in Airborne Systems and Equipment Certification," (ref. A.1) the Plan for Software Aspects of Certification for a project is the primary means used by the certification authority, namely the Federal Aviation Administration (FAA), for determining whether an applicant is proposing a software life cycle that is commensurate with the rigor required for the level of software being developed. To this extent, this document contains an overview of the Guidance and Control Software (GCS) Project including:

- an overview of the guidance and control application,
- statement of certification considerations,
- discussion of the software development plan, including the software life cycle processes and corresponding data, and
- the project milestones and schedule.

In an effort to increase our understanding of software, NASA Langley Research Center has conducted a series of experiments over the past twenty years to generate data to help characterize the software development process (ref. A.2). With an increased understanding of the failure behavior of software, improved methods for producing reliable software and assessing reliability can be developed. The current experiment, the GCS project, was started originally in 1985 at the Research Triangle Institute (RTI) (ref. A.3) to: (1) collect data on the faults that occur during the software life cycle, (2) collect data on faults that occur in operational guidance and control software, and (3) make observations on the effectiveness of life cycle processes that complies with the DO-178B guidelines. To do this, the GCS project involves the development of two separate implementations of the GCS where the life cycle activities comply with the RTCA DO-178B guidelines.

This document presents an overview of the software life cycle activities for this project and discusses why various development decisions were made, especially with respect to the experimental nature of this project. Details concerning the integral development processes are contained in the Software Verification Plan, Software Configuration Management Plan, and Software Quality Assurance Plan. The following section gives a general overview of the GCS project.

A.1.1 Overview of the GCS Project

For the GCS project, a GCS implementation is defined to be source code which fulfills the requirements outlined in the Guidance and Control Software Development Specification (ref. A.4), commonly referred to as the GCS specification. The development of two implementations of the GCS will be start from a common specification of the software requirements and proceed independently through the design, code, and integration processes. A GCS implementation will run in conjunction with a software simulator that provides input to the implementation based on an expected usage distribution in the operational environment, provides response modeling for the guidance and control application, and receives data from the implementation. The GCS simulator is designed to allow an experimenter to run one or more implementations in a multitasking environment and collect data on the comparison of the results from multiple implementations. Certain constraints are incorporated in the software requirements and project standards (especially standards regarding communication protocol) due to the nature of the GCS project.

A.1.2 Background

The first task in the start of the GCS project in 1985 was to develop the software requirements document for the guidance and control application. The original software requirements for the guidance and control application were reverse-engineered from a software program written in the late 1960's to simulate the Viking lander vehicle approaching the surface of the planet Mars (ref. A.5). Engineers at RTI produced the original requirements document for the guidance and control software, called the Guidance and Control Software Development Specification.

Since the project started in 1985, the DO-178A guidelines (ref. A.6) were originally used on the project as a model for the software development process. At RTI, three different programmer/analyst teams were assigned to develop GCS implementations. Because the GCS specification had already been generated, the DO-178A guidelines were to be applied to the development process starting with the design of the software implementations from the existing specification. The development of three separate implementations of the GCS following the DO-178A guidelines was started at RTI, along with the documentation of the software development process required by the DO-178A guidelines. The software development processes for the GCS project included the following processes:

- software design,
- software coding, and
- integration.

All three RTI-developed implementations of the GCS went through the design and coding processes and were at various stages of the integration process when they were delivered to NASA in the spring of 1992. After consultation with the FAA, a decision was made to extensively review and revise the GCS specification and restart the software development process under the DO-178B guidelines, which were released in December 1992. Upon delivery to NASA, new programmer and verification analyst teams were assigned along with support from new System Analysis, Software Quality Assurance, and Configuration Management personnel. However, due to resource limitations, only two of the implementations are being developed at Langley Research Center.

Due to the transitioning of the project from RTI to NASA along with the new focus on the DO-178B guidelines, the decision was made to revisit some of the original development activities. The following are the software development processes for the in-house GCS project:

- transitional software requirements development (focusing on the review and modification of the existing software requirements document),
- transitional software design, (where the existing design for each GCS implementation developed at RTI will be modified to meet the revised software requirements)
- software coding,
- integration.

The following chapter provides an overview of the GCS application, including a brief description of the software functions. A full account of the software requirements can be found in the Guidance and Control Software Development Specification, which serves as the Software Requirements Data for the GCS project.

A.2 Overview of the Guidance and Control Application

According to DO-178B, the software requirements process uses the system requirements and system architecture to develop the high-level requirements for the desired software. Correspondingly, DO-178B states that the Plan for Software Aspects of Certification should provide an overview of the system. For the GCS project, however, there is no real system to be developed nor documentation of real system requirements. The GCS project is solely a research effort to investigate the faults that occur in the development and operation of software, avionics applications in particular. The GCS implementations will only be executed in a simulated operational environment to collect software failure data. Consequently, the GCS project started with the definition of software requirements for a specific component of a guidance and control system, namely the terminal descent phase. Without system requirements, certain assumptions must be made in the development of the software requirements. Without system requirements, there also is no system safety assessment which is an important aspect of any development process that needs to comply with the DO-178B guidelines. Lack of system requirements also impacts the extent to which the project will comply with the DO-178B guidelines since no traces can be made from the software requirements back to the system requirements and safety assessment.

A.2.1 Software Overview

The definition of the software requirements for the GCS project focuses on two primary needs for the software: (a) to provide guidance and engine control of the lander vehicle during its terminal phase of descent onto the planet's surface and (b) to communicate sensory information to an orbiting platform about the vehicle and its descent. The lander vehicle to be controlled includes a guidance package containing sensors which obtain information about the vehicle state, a guidance and control computer, and actuators providing the thrust necessary for maintaining a safe descent. The vehicle has three accelerometers (one for each body axis), one Doppler radar with four beams, one altimeter radar, two temperature sensors, three strapped-down gyroscopes, three opposed pairs of roll engines, three axial thrust engines, one parachute release actuator, and a touch down sensor. The vehicle has a hexagonal, box-like shape with three legs and a surface sensing rod protruding from its undersurface. Figure A.1 shows a sketch of the lander vehicle during the terminal phase of descent, and Figure A.2 shows an engineering drawing of the vehicle from three perspectives.

In general, the GCS is designed to control a planetary lander during its final descent to the planet’s surface. After the lander has dropped from orbit, the software will control the engines of the vehicle to the surface of a planet. The initialization of the GCS starts the sensing of vehicle altitude. When a predefined engine ignition altitude is sensed by the altimeter radar, the GCS begins guidance and control of the lander. The axial and roll engines are ignited; while the axial engines are warming up, the parachute remains connected to the vehicle. During this engine warm-up phase, the aerodynamics of the parachute dictate the trajectory followed by the vehicle. Vehicle attitude is maintained by firing the engines in a throttled-down condition. Once the main engines become hot, the parachute is released and the GCS performs an attitude correction maneuver and then follows a controlled acceleration descent until a predetermined velocity-altitude contour is crossed. The GCS then attempts to maintain the descent of the lander along this predetermined velocity-altitude contour. The lander descends along this contour until a predefined engine shut off altitude is reached or touchdown is sensed. After all engines are shut off, the lander free-falls to the surface.

Figure A.1. The Lander Vehicle During Descent

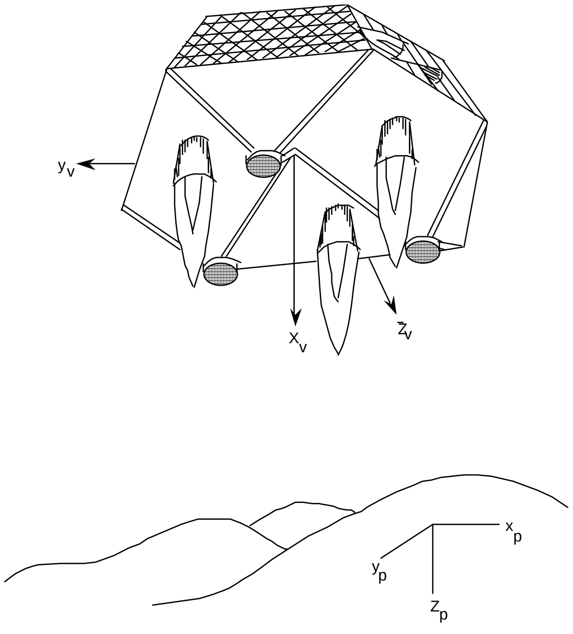
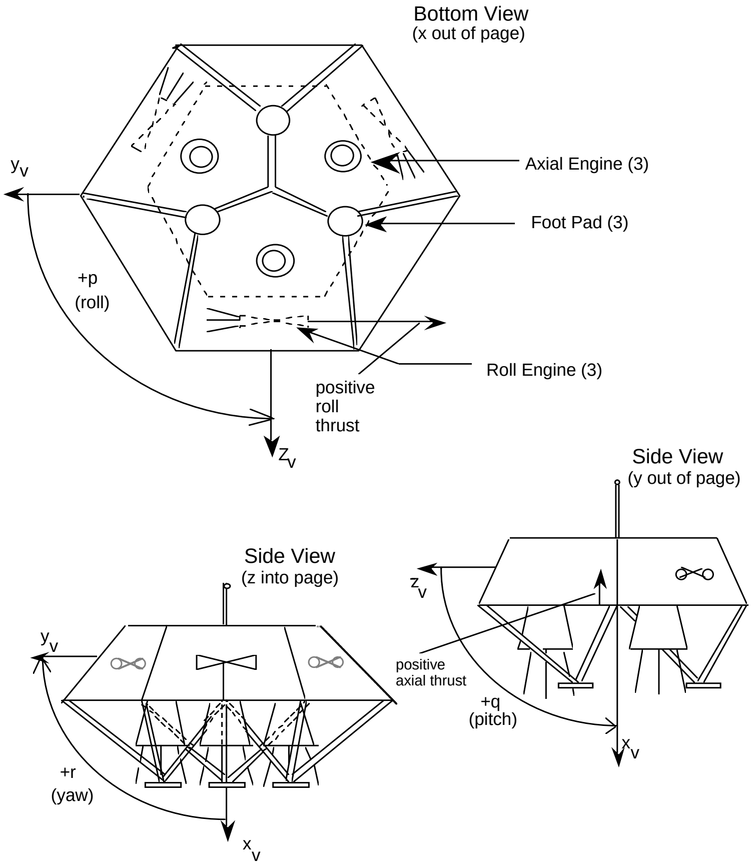
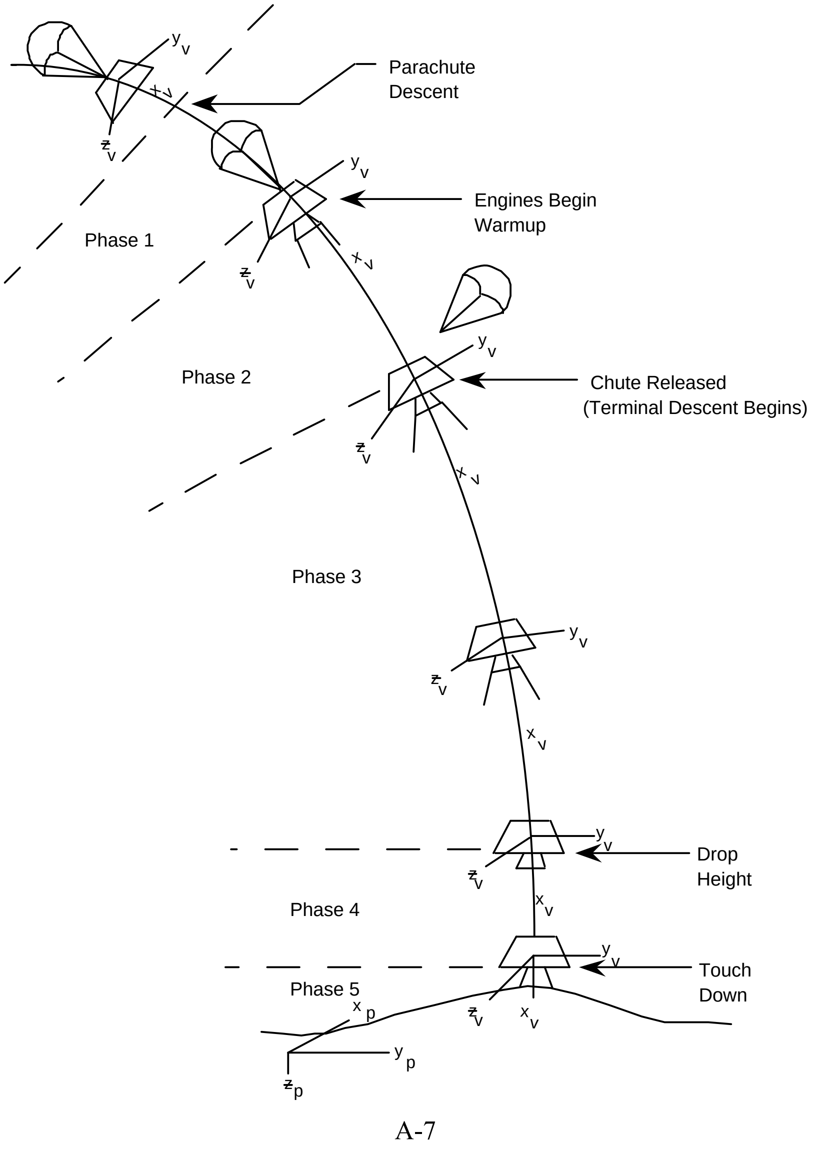


Figure A.2. Engineering Illustration of the Lander Vehicle



In general, the GCS is designed to control a planetary lander during its final descent to the planet's surface. After the lander has dropped from orbit, the software will control the engines of the vehicle to the surface of a planet. The initialization of the GCS starts the sensing of vehicle altitude. When a predefined engine ignition altitude is sensed by the altimeter radar, the GCS begins guidance and control of the lander. The axial and roll engines are ignited; while the axial engines are warming up, the parachute remains connected to the vehicle. During this engine warm-up phase, the aerodynamics of the parachute dictate the trajectory followed by the vehicle. Vehicle attitude is maintained by firing the engines in a throttled-down condition. Once the main engines become hot, the parachute is released and the GCS performs an attitude correction maneuver and then follows a controlled acceleration descent until a predetermined velocity-altitude contour is crossed. The GCS then attempts to maintain the descent of the lander along this predetermined velocity-altitude contour. The lander descends along this contour until a predefined engine shut off altitude is reached or touchdown is sensed. After all engines are shut off, the lander free-falls to the surface. Figure A.3 shows the phases of the terminal descent trajectory of the lander.

Figure A.3. A Typical Terminal Descent Trajectory



With the control laws specified in the software requirements, the probability that the lander will safely land on the planet’s surface should be at least 0.95; that is, given a large number of simulated trajectories, the lander should successfully land (as opposed to crashing) on the planet’s surface at least 95% of the time.

The following section concerns the certification aspects regarding this guidance and control application.

A.3 Certification Considerations

The two primary functions of the GCS are: (1) to provide guidance and engine control of the lander vehicle during its terminal phase of descent onto the planet's surface and (2) to communicate sensory information to an orbiting platform about the vehicle and its descent. Although there is not a system safety assessment for the GCS project, it is assumed that the loss of either of these functions could cause or contribute to a catastrophic failure condition for the vehicle. Consequently, the guidance and control application as defined in the GCS specification is considered to be Level A software, requiring the highest level of effort to show compliance with the certification requirements. Since the GCS is assumed to be Level A, (as opposed to a lower level requiring less effort to show compliance), no justification for this rating is provided.

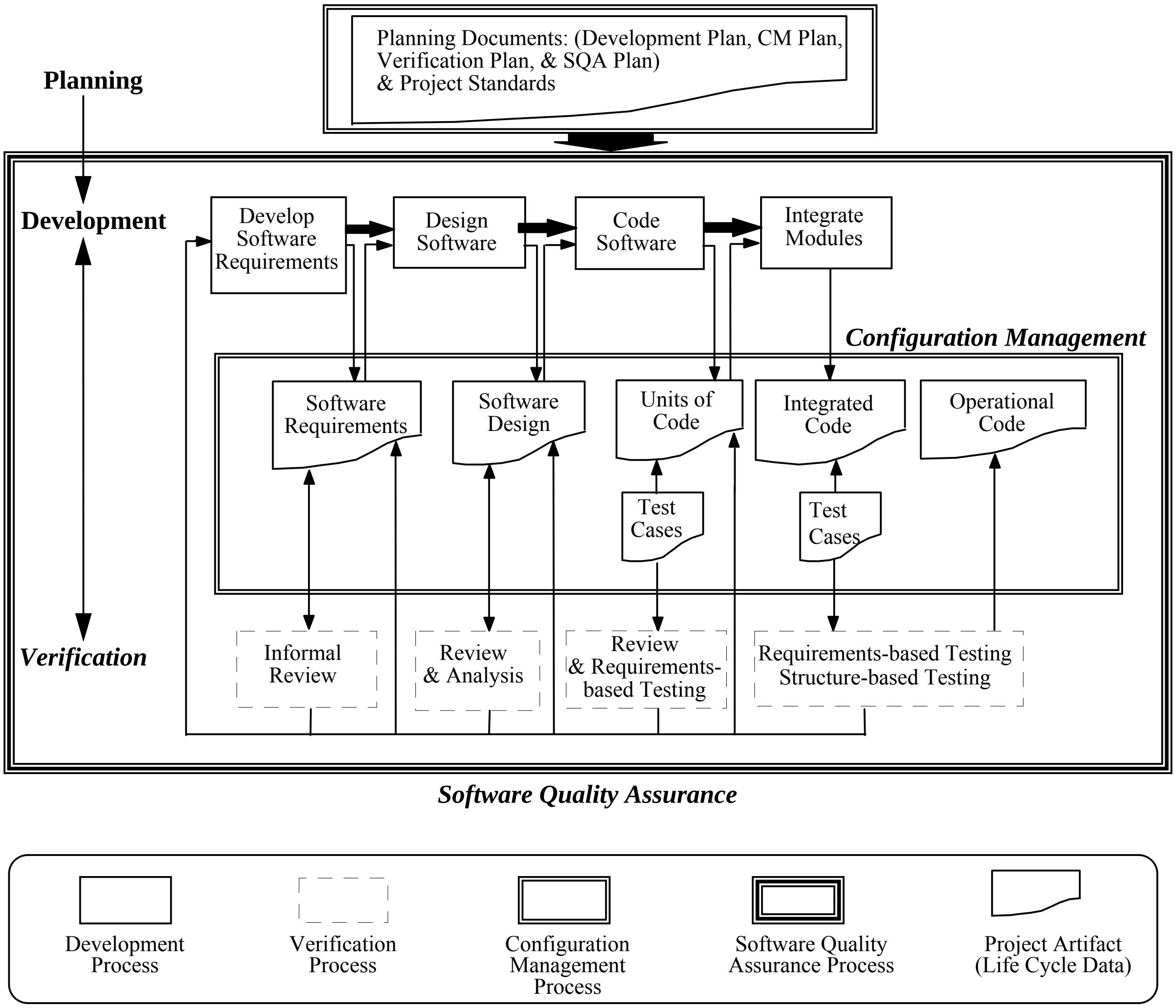
A.4 Software Development Plan

As discussed in chapter A.1, the software development processes for the GCS project consist of the requirements, design, code, and integration processes, where the project artifacts from the requirements and design processes are modifications of artifacts produced during the original effort at RTI. In general, the development processes follow a modified waterfall life cycle model as shown in Figure A.4.

In this figure, the planning process is shown at the top level, and this process feeds into the rest of the life cycle activities. Then, the software quality assurance (SQA) process monitors the rest of the life cycle processes, and the configuration management process controls the artifacts produced. For each of the four development processes, there is some level of verification activities. Note that the verification activity in the requirements process only consists of an informal review of the software requirements document, largely because there is no system requirements document or safety assessment for the project. After the requirements process, the remainder of the life cycle activities are intended to comply with DO-178B.

The following section describes the organizational responsibilities for all life cycle activities and provides more details on the life cycle processes and products.

Figure A.4. Life Cycle Activities Flow for the GCS Project



Planning Documents: (Development Plan, CM Plan, Verification Plan, & SQA Plan) & Project Standards
Planning
Development
Verification
Develop Software Requirements
Design Software
Code Software
Integrate Modules
Configuration Management
Software Requirements
Software Design
Units of Code
Integrated Code
Operational Code
Test Cases
Test Cases
Informal Review
Review & Analysis
Review & Requirements-based Testing
Requirements-based Testing Structure-based Testing
Software Quality Assurance
Development Process
Verification Process
Configuration Management Process
Software Quality Assurance Process
Project Artifact (Life Cycle Data)

A.4.1 Organizational Responsibility

The GCS project involves two independent teams, where each team, consisting of a programmer and verification analyst, is tasked to develop a single GCS implementation according to the DO-178B guidelines. The two GCS implementations have been assigned planetary names: Mercury and Pluto. In addition to the programmer and verification analyst teams, other project personnel are assigned the roles of Software Quality Assurance (SQA) representative, System Analyst (responsible for the software requirements), and Configuration Manager. Due to resource limitations, the software integral processes of Software Configuration Management and SQA will be administered independently across the implementations, but the systems and individuals used to carry out these processes will be the same. For example, one configuration management system will store all data items for all implementations, one person will do configuration management for all implementations, and one person will do SQA for all implementations. Further, there will not be a certification liaison process for the GCS project. Table A.1 lists the personnel assigned to the GCS project.

Note that in a real development project, the SQA representative would be different that the project leader and would report to a different management organization. However, due to personnel transfers and limitations on project resources, the same person ultimately was required to perform both roles.

Table A.1 gives a general overview of the responsibilities of six major project roles.

Table A.1. GCS Project Personnel and Organization

Project Role	Responsible Personnel	Organization	Responsibility
Project Leader	Kelly Hayhurst	System Validation Methods Branch (NASA LaRC)	Managing all of the activities of the GCS project, including providing planning, technical direction, and coordination with respect to all life cycle processes, collecting and analyzing data, and scheduling the major milestones of the project to meet the goals of the project.
SQA representative	Kelly Hayhurst		Providing confidence that the software life cycle processes produce software that conforms to its requirements by assuring that project activities are performed in compliance with DO-178B and project standards, as defined in the planning documents.
Configuration Manager	Laura Smith	System Validation Methods Branch (NASA LaRC)	Providing configuration management of all life cycle data (documentation, design, code, test cases, and simulator) associated with the development of the GCS implementations. in accordance with the DO-178B guidelines and project standards.
System Analyst	Bernice Becher	System Validation Methods Branch (Lockheed)	Providing expertise regarding the software requirements for the guidance and control system (described in the GCS specification) to project participants, and maintaining the GCS specification in accordance with the DO-178B guidelines and project standards.
Programmers Mercury Programmer	Andy Boney	Computer Science Corp.	Independently developing one implementation of the guidance and control software according to the GCS specification, DO-178B guidelines, and the Software Development Standards. This includes the generation of the detailed design description, source code, and executable object code.
Pluto Programmer	Paul Carter	Computer Science Corp.	
Verification Analysts Mercury Analyst	Debbie Taylor	Computer Science Corp.	Defining and conducting all of the verification activities associated with the development of one GCS implementation according to the GCS specification, DO-178B guidelines, and the Software Development Standards.
Pluto Analyst	Rob Angellatta	System Validation Methods Branch (Lockheed)	
Simulator Operator	Bernice Becher		Developing, maintaining, and documenting the GCS simulator. Also, assists in running experiments.

Since the two GCS implementations are to proceed independently through the development process, special constraints have been placed on the level of communication allowed among the project participants. In particular, the programmers should not communicate with each other about their implementations, and the verification analysts are not permitted to discuss specific details about their implementations. The Software Development Standards contains more details on the communication protocol for all project participants.

A.4.2 Life Cycle Processes

At a high level, the software life cycle processes for the GCS project consist of: the software planning process, the software development processes, and the integral processes. The software planning process defines and coordinates the software development processes and the integral processes. The software development processes are made up of the software requirements, software design, software coding, and the integration processes; those processes that directly produce the software product. The integral processes surround the software development processes to ensure the correctness, control, and integrity of the software products. The integral processes are the software verification, configuration management, and quality assurance processes. Table A.2 shows the objectives for each of the life cycle processes based on the tables in Annex A of DO-178B.

Table A.2. Activities and Products of the Life Cycle Processes

Process Objectives	Major Activities	Products
Planning Process Define Development and Integral Processes - transition criteria - life cycle - project standards	Revise project planning documents from RTI to comply with DO-178B	Plan for Software Aspects of Certification Software Development Standards Software Verification Plan Software Configuration Management Plan Software Quality Assurance Plan
Development Process Define high-level requirements Define low-level requirements & software architecture Develop Source Code Generate Executable Object Code Identify derived requirements	Modify GCS specification (high-level requirements) Update (RTI-generated) detailed design descriptions (using Teamwork) Develop source code	Revised GCS specification (including any derived requirements) Detailed Design Description for Mercury and Pluto (before Design Review) Cleanly compiled version of Mercury and Pluto source code (before review & testing)
Software Quality Assurance Process Assure that development and integral processes comply with plans and standards Conduct Conformity Review	Review all processes and products for compliance Participate in design, code, and test case reviews Conduct software conformity review	SQA Records from all reviews for each implementation

Table A.2. (cont.) Activities and Products of the Life Cycle Processes

Process Objectives	Major Activities	Products
Verification Process Review High-level requirements Review low-level requirements & software architecture Review source code Test coverage of all software requirements (100% requirements coverage is achieved) Test coverage of software structure (multiple condition/decision coverage is achieved)	Conduct Team Design Inspection Conduct Team Source Code Inspection Develop and perform Requirements-based testing at four levels: unit, subframe, frame, and trajectory. Conduct analysis of source code (after requirements-based testing) to determine if MC/DC is achieved Perform Structure-based testing as necessary to achieve Modified Condition/Decision Coverage.	Traceability Matrix for software requirements Verification Procedures Post-Design Review Design Description for Mercury and Pluto Verification Results for Mercury and Pluto Design Reviews (including Design to Requirements Trace) Code-reviewed version of Mercury and Pluto source code Verification Results for Mercury and Pluto Reviews (including Code to Requirements Trace) Requirements-based Test Cases Structure-based Test Cases Mercury and Pluto versions that completed requirements-based testing Mercury and Pluto versions that completed structure-based testing Verification Results for Mercury and Pluto Testing (including Test case to Requirements Trace)
Configuration Management Process Provide identification for all configuration items Provide change control system Provide archive and retrieval services	Define labeling system for all configuration items Establish a change control system using the Code Management System (CMS) Develop a Problem and Action Reporting System for Development Products (CC1) and Support Documentation (CC2) Define and implement procedures for archive and retrieval Document and control software development environment	Configuration Management Index Life Cycle Environment Configuration Index Configuration Management Records Completed Problem and Action Reports Completed Support Documentation Change Reports

As with all life cycle models, there must be some criteria to indicate when to progress from one process to the other. The primary transition criteria for the development processes is based on the completion of the verification of the main products of those processes. Table A.3 gives the transition criteria for the GCS development processes.

Table A.3. Transition Criteria for the Software Development Processes

Development Process	Inputs	Transition Criteria to Next Process
Requirements	GCS specification from RTI	Informal review of version 2.2 of the GCS specification and approval by the project leader.
Design	version 2.2 of the GCS specification	Completion of all problem reports from the Design Review. (SQA approval is required for completion of problem reports.)
Code	Design Description	Completion of all problem reports from the Code Review.
Integration • Requirements-based Testing -------------------------------- • Structure-based Testing	Source Code Executable Object Code Requirements-based Test cases	Review, approval, and successful execution of all requirements-based test cases -- Review, approval, and successful execution of all structure-based test cases.

A.4.3 Software Life Cycle Data

The prime objective of the software development processes for the GCS project is to independently (within the constraints of the project) develop two implementations of the GCS and all corresponding life cycle data in compliance with the DO-178B guidelines. The detailed plans for achieving this objective are given in the following documents: Software Verification Plan, Software Configuration Management Plan, and Software Quality Assurance Plan. Each of these planning documents must comply with the DO-178B guidelines and will specify the following information:

- the inputs to that process, including feedback from other processes,
- the integral process activities,
- the availability of tools, plans, methods, and procedures.

The standards for the development products (requirements, design, and source code) and the other project documentation are given in the Software Development Standards. The Software Development Standards also contains a description of tools and methods to be used during development including requirements and design methods and programming language. Other fundamental information about project procedures (such as configuration management and problem reporting) are addressed in the Software Development Standards so that the document can serve as a single handbook for project participants.

Because both GCS implementations are to follow the same development and integral processes, only one set of planning documents (Plan for Software Aspects of Certification (which includes the Software Development Plan), Software Verification Plan, Software Configuration Management Plan, and Software Quality Assurance Plan) will be developed for the project along with a single Software Configuration Index. Most of the remaining life cycle data will be implementation specific. Table A.4 shows the responsible party for the life cycle data that corresponds with each process.

Table A.4. Organizational Responsibilities for the Software Life Cycle Activities

Software Life Cycle Process Activities	Software Life Cycle Data	Organizational Responsibility
Software Planning	Plan for Software Aspects of Certification	Project Leader
	Software Development Standards, including the Software Requirements Standards, Software Design Standards, and the Software Code Standards	
	Software Accomplishment Summary	
Software Development		
Transitional Software Requirements	GCS Specification (Software Requirements Data)	System Analyst
Transitional Software Design		
Designing the Mercury Implementation	Design Description for Mercury	Mercury Programmer
Designing the Pluto Implementation	Design Description for Pluto	Pluto Programmer
Software Coding		
Coding the Mercury Implementation	Source Code for Mercury	Mercury Programmer
Coding the Pluto Implementation	Source Code for Pluto	Pluto Programmer
Integration		
Generating Executable Object Code for Mercury	Executable Object Code for Mercury	Mercury Programmer
Generating Executable Object Code for Pluto	Executable Object Code for Pluto	Pluto Programmer
Integral		
Software Verification	Software Verification Plan	Mercury & Pluto Analyst
	Software Verification Procedures & Requirements-based Test Cases	
Verifying the Mercury Implementation	Structure-based Test Cases for Mercury	Mercury Analyst
	Software Verification Results for Mercury	
Verifying the Pluto Implementation	Structure-based Test Cases for Pluto	Pluto Analyst
	Software Verification Results for Pluto	
Configuration Management	Software Configuration Management Plan	Configuration Manager
	Software Configuration Index (including the Life Cycle Environment Configuration Index)	
	Problem Reports for Mercury and Pluto	
	Support Documentation Change Reports	
	Software Configuration Management Records	
Software Quality Assurance	Software Quality Assurance Plan	Software Quality Assurance Representative
	Software Quality Assurance Records	

A.5 Project Milestones and Schedule

Within a real software development project the certification authority would be involved in the development activities, at least to the extent of having visibility into the development processes as they progress. Because the GCS project is a research effort, the resources necessary to provide interaction between the project and the certification authority are not available. Further, because this project is not confined by constraints placed on a typical development project that must meet real production deadlines, a hard deadline schedule will not be produced for this project. However, the project does have milestones based on the development processes and a proposed schedule of the major project activities. Table A.5 gives the major project milestones and Table A.6 gives the project history and proposed schedule.

Because there is no certification liaison process for the GCS project, all project life cycle data as shown in Table A.4 will be made available to the certification authority at the completion of all development processes. The SQA representative will conduct a software conformity review upon project completion prior to submission for certification.

Table A.5. GCS Project Milestones

Project Phase	Milestones within each Phase
Requirements Phase	• Release version 2.2 of the GCS specification to the programmers
Design Phase	• Complete GCS designs to comply with version 2.2 of the GCS specification • Conduct Design Reviews • Complete all modifications to the design identified in Design Reviews • Initiate development of requirements-based test cases
Code Phase	• Develop source code • Conduct Code Review • Complete all modifications to the code identified in Code Review
Integration Phase	• Complete requirements-based testing • Complete analysis for Multiple Condition/Decision Coverage • Complete Structure-based testing as needed

Table A.6. GCS Project History and Schedule

Historical Events:	Date
Delivery of GCS life cycle data from Research Triangle Institute	5/92
Meeting with the FAA (DeWalt and Saraceni) to determine direction for project	9/20/92
Review of life cycle data (to determine extent of modifications necessary by LaRC)	9/92
Proposed Schedule of Events:	
Complete Modification of the GCS specification (release 2.2)	11/93
Complete Modification and Verification of GCS Designs	6/94
Complete Development and Verification of Source Code	10/94
Complete Development of Requirements-based Test Cases	8/94
Complete Requirements-based Testing	12/94
Complete Structure-based Testing	12/94

A.6 Conclusion

This document gives all project participants and the certification authority an overview of the Guidance and Control Software project and the corresponding software life cycle processes and products. This document is intended to be used in conjunction with the other major planning and standards document (Software Development Standards, Software Verification Plan, Software Configuration Management Plan, and Software Quality Assurance Plan) to provide the basis for all project activities in compliance with DO-178B.

A.7 References

A.1 RTCA Special Committee 152. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178B, Requirements and Technical Concepts for Aviation, December 1992.

A.2 George B. Finelli. Results of software error-data experiments. In AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, Atlanta, GA, September 1988.

A.3 Janet R. Dunham and George B. Finelli. Real-Time Software Failure Characterization, COMPASS'90: Proceedings of the Fifth Annual Conference on Computer Assurance, June 1990.

A.4 B. Edward Withers and Bernice Becher. Guidance and Control Software Development Specification, NASA Contractor Report (To be published)

A.5 Neil A. Holmberg, Robert P. Faust, and H. Milton Holt. Viking ‘75 Spacecraft Design and Test Summary, Volume I - Lander Design, NASA Reference Publication 1027, Langley Research Center, 1980.

A.6 RTCA Special Committee 152. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178A, Radio Technical Commission for Aeronautics, March 1985.

Appendix B: Software Development Standards for the Guidance and Control Software Project

Authors: Kelly J. Hayhurst, NASA Langley Research Center
Bernice Becher, Lockheed Martin Engineering and Sciences Corp.

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

B. Contents

B.1 Introduction

According to the Requirements and Technical Concepts for Aviation RTCA/DO-178B document entitled *Software Considerations in Airborne Systems and Equipment Certification* (ref. B.2), the purpose of the software development standards is to "define the rules and constraints for the software development process." To that extent, this document contains the Guidance and Control Software (GCS) project standards for the development of the software requirements, software design, and implemented code. These standards include constraints and rules on defining the software requirements, and designing and coding the software. These standards, along with the software requirements, will set the basis for evaluating actual project results with expected results.

This document also contains other project standards including communication protocol among the project participants and problem and action reporting procedures. It is hoped that this document will serve as a handbook for the project participants, especially those individuals responsible for the design and coding of the software. All project participants are expected to become familiar with and follow the standards set forth in this document. To provide a basis for understanding the various project standards and procedures, the following section gives an overview of the GCS project and the software development process.

B.1.1 The Software Development Process for the GCS Project

For the GCS project, a GCS implementation is defined to be code which fulfills the requirements outlined in the *Software Requirements Data*, commonly referred to in this project as the GCS specification. The current GCS project involves the development of separate implementations of the GCS where the development and verification activities comply with the RTCA/DO-178B guidelines which are required by the Federal Aviation Administration (FAA) for developing software to be certified for use in commercial aircraft equipment and with project standards (as defined in this document). Three of the major purposes of this project are to (1) collect data on the faults that occur during the software development process, (2) collect data on faults that occur in operational guidance and control software, and (3) make observations on the effectiveness of a development process that complies with the DO-178B guidelines. Special procedures and forms for tracking effort and error data have been developed to capture information in addition to that required by the DO-178B guidelines. These procedures are described later in this document.

A GCS implementation will run in conjunction with a software simulator that provides input based on an expected usage distribution in the operational environment, provides response modeling, and receives data from the implementation. The GCS simulator is designed to allow an experimenter to run one or more implementations in a multitasking environment and collect data on the comparison of the results from multiple implementations. Certain constraints have been incorporated in the software requirements and project standards (especially standards regarding communication protocol) due to the nature of the GCS project. Further information on goals of the GCS project is available in the *Plan for Software Aspects of Certification*.

The GCS project was started originally at Research Triangle Institute (RTI) (ref. B.1). The first task in the project was to develop the specification document for the guidance and control software application. Engineers at RTI produced the original requirements document for the guidance and control software, called the *Guidance and Control Software Development Specification.* The GCS specification contains more than just the software high-level requirements. The GCS specification embodies high level requirements and some level of software design. Thus, some of the necessary refinement of the software requirements has already been accomplished in the GCS specification. The chapter titled "Software Requirements Standards" describes the methods used to generate the original GCS requirements document and overviews the methods used in the original verification effort for the requirements.

Once the GCS specification was generated, a decision was made to have RTI use the DO-178**A** guidelines (ref. B.3) as a model for the software development process. Six people were divided into three different teams of 2 people each to develop three implementations. Each team, consisting of a programmer and verification analyst, was tasked to develop a single GCS implementation according to the DO-178A guidelines. The three GCS implementations were assigned planetary names: Mercury, Earth, and Pluto. The documentation for each implementation refers to the assigned planetary name. In addition to the programmer and verification analyst teams, other project personnel were assigned the roles of Software Quality Assurance (SQA) representative, system analyst (responsible for the software requirements), and configuration manager to work with the three implementation teams. The *Plan for Software Aspects of Certification* contains more details on the role of all project participants.

Because the GCS specification had already been generated, the DO-178A guidelines were to be applied to the development process starting with the design of the software implementations from the existing specification. The software development processes used by RTI included the following processes:

- software design,
- software coding, and
- integration.

All three RTI-developed implementations of the GCS went through the design and coding processes and were at various stages of the integration process when they were delivered to NASA. After consultation with the FAA, a decision was made to extensively review and revise the GCS specification and restart the software development process under the DO-178**B** guidelines, which were released very soon after the GCS implementations were delivered. Upon delivery to NASA, new programmer and verification analyst teams were assigned along with support from new System Analysis, SQA, and Configuration Management personnel.

Due to the transitioning of the project from RTI to NASA along with new focus on the DO-178B guidelines, the decision was made to revisit some of the original development activities and to develop only two implementations. In particular, the following activities are to be accomplished in addition to the regular life cycle development activities:

1. review and revision of the existing GCS specification, which will result in version 2.2 of the document,
2. definition of any additional information that needs to be specified to fulfill the requirements for the *Software Requirements Data* as described in Subsection 11.9 of DO-178B,
3. review and revision of the existing documentation describing the software development process to conform with the guidelines set forth in DO-178B (i.e. revising the RTI-generated *Plan for Software Aspects of Certification*, *Software Verification Plan*, *Software Configuration Management Plan*, *Software Quality Assurance Plan,* and the *Software Development Standards*), and
4. modification of each existing design (developed at RTI) by the newly designated programmer to bring the design up to version 2.2 of the GCS specification.

Thus, the software development processes for the in-house GCS project will include the following processes:

- transitional software requirements development (focusing on the review and modification of the existing software requirements),
- transitional software design,
- software coding, and
- integration.

The following chapter describes the methods used to develop the original GCS specification and the methods and standards for modifying the requirements. Standards for the design process are described in the chapter titled "Software Design Standards". The standards for the coding process are described in the chapter "Software Code Standards". Instructions to the programmers regarding their role in the various development processes and general purpose instructions to all project participants for data collection, communication, and configuration management are discussed in the remaining chapters. Note that there may be changes to various aspects of the development process (such as the software requirements or project standards) as the project progresses. New procedures and standards may be issued periodically and project documentation updated as appropriate.

B.2 Software Requirements Standards

According to DO-178B, the software requirements process uses the system requirements and system architecture to develop the high-level requirements for the desired software (ref. B.2). The objectives of this process are to ensure the clarity, consistency, and completeness of those requirements allocated to the software. For the GCS project, however, there is no real system to be developed nor documentation of real system requirements. Consequently, there also is no system safety assessment which is an important aspect of any development process that needs to comply with the DO-178B guidelines. The GCS project started with the definition of software requirements for a specific component of a guidance and control system. Without system requirements, certain assumptions must be made in the development of the software requirements. Lack of system requirements also impacts the extent to which the project will comply with the DO-178B guidelines since no traces can be made from the software requirements back to the system requirements and safety assessment.

The following section describes the development of the original specification for the software, including the methods, rules, and tools used in the development of the high-level requirements.

B.2.1 Development of the Requirements Documentation (Methods, Notations, and Constraints)

The original requirements for the guidance and control application were reverse-engineered during the mid 1980's by engineers at RTI from a software program written in the late 1960's to simulate the Viking lander vehicle approaching the surface of the planet Mars (ref. B.1). The definition of the software requirements focused on two primary needs for the software: (a) to provide guidance and engine control of the lander vehicle during its terminal phase of descent onto the planet's surface and (b) to communicate sensory information to an orbiting platform about the vehicle and its descent. As discussed above, the GCS specification embodies high-level requirements and some level of software design.

The RTI engineers used a version of the structured analysis for real-time system specification methodology by Hatley and Pirbhai (ref. B.4) to help create the original GCS specification. In general, the structured analysis method is based on a hierarchical approach to defining functional modules and the associated data and control flows. Structured analysis was chosen as the specification method as opposed to a formal specification language for two reasons: (1) to keep the specification development activity practical and, (2) to use a specification method which is currently used in industry (ref. B.5). The Computer Aided Software Engineering (CASE) tool, team*work* (ref. B.6), was used later in the project to refine some of the data and control flow diagrams in the GCS specification. Beyond the use of team*work* and the structured analysis approach to system specification, no constraints were placed on the use of requirements development tools.

The specification document includes data context and flow diagrams, control and context flow diagrams, and process and control descriptions. Figure B.1 defines the graphical symbols used in the specification's data flow and control flow diagrams, respectively. As stated in the GCS specification, the data flow diagrams describe the processes, data flows, and data and control stores. The data context diagram is the highest-level data flow diagram and represents the data flow for the entire software component.

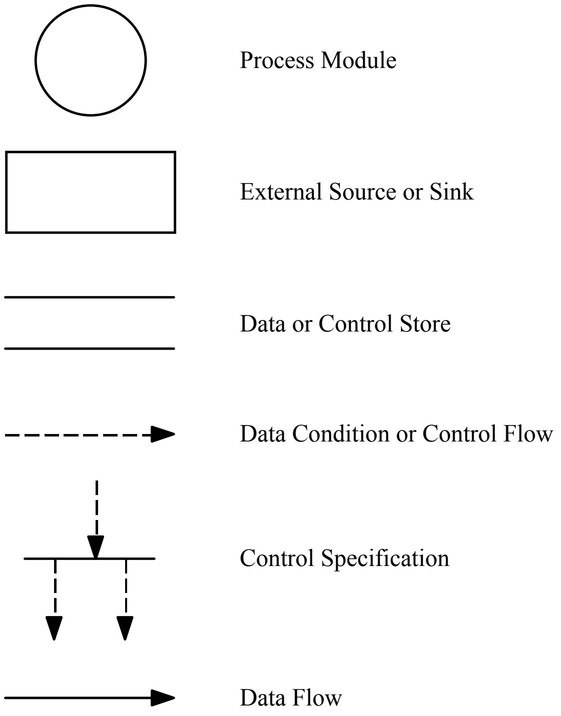


Figure B.1. Graphical Symbols Used in the GCS Specification's Flow Diagrams

The control flow diagrams describe processes, data condition and control signal flows, and data and control stores. The data condition and control signal flows are depicted using directed arcs with broken lines and simply show the logic involved in the system. Signal flows between the control flow diagram and the control specification have a short bar at the end of the directed arc. The control flow diagrams contain duplicate descriptions of the processes represented on the data flow diagram. The control context diagram representing the most abstract control flow is similar to the data context diagram.

The control specifications describe the control requirements of a system. These specifications contain the conditions when the processes detailed in the data and control flow diagrams are activated and de-activated. A Data Requirements Dictionary, containing definitions for both data and control signals, is also included as part of the GCS specification.

The GCS project is targeted for VAX/VMS systems; that is, the GCS implementations and the simulator are designed to run on a VAX/VMS system. Consequently, all software requirements, standards, and instructions for the project assume a VAX/VMS system as the host system for the GCS implementations. A more detailed description of the software life cycle environment, including a description of the host operating system, can be found in the *Software Configuration Management Plan* .

B.2.2 Review of the Software Requirements

Although formal review, according to the DO-178B guidelines, of the GCS specification is beyond the scope of the project, steps were taken by RTI during the development of the original GCS specification to assure that the specification was as complete, precise, and verifiable as possible. Conducting peer reviews and informal walk-throughs and coding a prototype implementation were among those steps. During these activities, the changes made to the specification were recorded and categorized. More discussion on the methods used in the verification of the original specification is in the *GCS Development Specification Review Description* (ref. B.5).

Version 2.0 of the GCS specification, that resulted from those verification activities, was more than 122 pages of text, including appendices concerning the format of the specification, implementation notes, background on methods of integration, and a data requirements dictionary. The GCS specification was written for an experienced programmer with two or more years of full-time industrial programming experience. The GCS specification was intended to be implemented using a scientific programming language. In fact, the implementations to be developed for the GCS project are required to be coded in the FORTRAN language. A background in mathematics, physics, and numerical integration is considered beneficial in understanding the software requirements. A similar background is also considered beneficial for individuals required to verify a GCS implementation. Version 2.0 of the specification was released to the original programmers at RTI to start the development of their implementations. Version 2.1 of the specification was later released after a significant number of modifications were made.

During the transitional requirements development process of the project, version 2.1 of the software requirements was assessed in light of the DO-178B guidelines, especially with respect to the required contents of the *Software Requirements Data*. The *Software Requirements Data*, as described in Subsection 11.9 of DO-178B, contains the definition of the high-level requirements for the software component. After a review of and significant modification to the physics embodied in the software requirements are accomplished, version 2.2 of the GCS specification, which is the Software Requirements Data for the purposes of the GCS project, will be released to the new programmers, signaling the end of the transitional requirements process and the start of the transitional design process.

B.2.3 Derived Requirements and Modifications

According to DO-178B, the GCS specification is classified under control category 1 -- which means that the project must provide a formal system of problem reporting, change control, and change review for that data. All changes to the GCS specification, along with the other project support documentation, are made through a system of Support Documentation Change Reports. All questions raised by any member of the development team regarding the GCS specification are brought to the system analyst. The system analyst reviews all questions and determines if changes to the specification are required. When changes are deemed necessary, the system analyst submits a description of the necessary modification to the SQA representative and project leader for review. The chapter "Problem and Change Reporting" gives a more detailed description of the procedures and forms used for tracking, reviewing and approving changes to the GCS specification.

Once the modification is approved, a copy of the modification description is distributed to all project participants. The programmers are required to consider the impact of each modification to the software requirements on their implementation and make any appropriate changes to their software design and code. Similarly, the verification analysts should determine the impact of any modifications on the verification activities, especially test cases and requirements in the traceability data, and make any necessary corrections to the appropriate artifacts.

Derived requirements will be recorded as the verification analysts document the software requirements and their corresponding verification criteria for the traceability data. The programmers will also identify requirements derived during the design and coding processes in their software design descriptions and code, respectively. As derived requirements are identified, they will be added to the traceability data. Derived requirements will also be added to the traceability data as they are identified during the review and analysis of the software design and code, and these requirements will be verified through the remainder of the development processes. Since there is no system requirements or system safety assessment, there is no other mechanism other than the traceability data to account for the derived requirements.

The following chapter describes the software design standards defined for the GCS project.

B.3 Software Design Standards

The purpose of the software design process is to refine the software high-level requirements into a software architecture and the low-level requirements that can be used to implement the source code. The software design standards are provided to define the methods, rules, and tools to be used in the development of the software architecture and low-level requirements, as described in Subsection 11.7 of DO-178B. These standards should enable the software implementations to be uniformly designed.

During the transitional design process of the GCS project, the programmers are required to develop detailed software designs from existing GCS designs, as delivered from RTI. A detailed design should be a complete statement of the software low-level requirements that addresses exactly what needs to be accomplished in order to fulfill the objectives stated in the GCS specification; that is, the detailed design should contain an algorithmic solution. The low level requirements should be directly translatable into source code, with no further decomposition required.

B.3.1 Design Methods, Rules, and Tools

For the GCS project, the design of a GCS implementation should be developed using the structured analysis and design methods described by Hatley and Pirbhai (ref. B.4), DeMarco (ref. B.7) or Ward and Mellor (ref. B.8). Further, the designer is required to use the Computer Aided Software Engineering (CASE) tool, team*work* (ref. B.9), to develop the design. Team*work* is a product of (and registered trademark of) Cadre Technologies, Inc. The team*work* tool is used to aid in the structured design of the applications, and certain parts of the output from team*work* will be required for design and code reviews. Team*work* is composed of several tools that are available to the designer. The components of team*work* include, but are not limited to, the following components:

SA --- The base-line structured analysis tool,

RT --- An extension of SA that allows description of real-time systems, and

SD --- A parallel tool that follows the Ward and Mellor approach.

The designer may choose to use any of these tools. If the SA tool is chosen, the design will consist of Data Flow Diagrams (DFDs) that provide a representation of a system focusing on the data passed between processes and Process Specifications (P-Specs) that provide procedural descriptions of primitive processes (processes that cannot be further decomposed into more detailed DFDs). If the RT extension is used, the design will also contain Control Flow Diagrams (CFDs) and Control Specifications (C-Specs). The CFDs provide an additional representation of the system focusing on the control and data condition signals passed between processes, and the C-Specs relate input and output control flows, turn processes on or off, and trigger changes in the operating mode of the system. If the design is developed with the SD tool, the design will consist of Structure Charts that depict the partitioning of a system into modules, showing the hierarchy and organization of these modules and the communication interfaces among them, and Module Specifications (M-Specs) that describe the function of the modules represented in the design (ref. B.9). Although the P-Specs and M-Specs contain the detailed description of the algorithms for the code, these specifications should be limited in length to a) encourage a modular design and code and b) aid in review and verification. The constraints listed below should be followed when using team*work* to develop the GCS design.

- No P-Spec, C-Spec, or M-Spec should be greater than five pages in length when printed.
- The body section of the P-Specs and M-Specs may contain any combination of structured English and pseudo-code to provide a concise and unambiguous description of the process or module.
- The lists of input and output variables should be directly traceable to the specification. Any flows should be broken down to the elements as shown in the Data Requirements Dictionary in the GCS specification before entering the P-Spec, C-Spec, or M-Spec.
- Interrupts may not be used.
- Before printing the copy to be analyzed during the design review, a complete "balance" check should be conducted on the model. No changes should be made to the model between the last "balance" and the print.

In general each programmer is expected to follow good software engineering practices in the construction of the design; but, the design standards for this project do not extend beyond the constraints listed above. For example, no restrictions have been issued on the complexity of the design, such as limiting the number of nested calls or entry and exit points in the code components. However, each programmer should be mindful that this project involves the development of software that is considered to be Level A software (in the terminology of DO-178B), where anomalous behavior of the software could cause or contribute to a catastrophic failure condition for the vehicle. Excessive complexity of the design and code magnify the difficulty in verification of the software and, hence, could potentially increase the possibility of faults remaining in the software after verification.

In addition, no other design standards have been defined regarding naming conventions, scheduling, global data, or exception handling beyond those requirements set forth in the GCS specification. Although no constraints in terms of project standards have been placed on the use of event-driven architectures, dynamic tasking, and re-entry, the use of such methods in a GCS design should be discussed and the rationale for their use clearly explained in the design documentation. Further, no formal constraints have been placed on the use of recursion, dynamic objects, data aliases and compacted expressions. However, as stated above, the use of such techniques should be clearly discussed and justified in the design documentation.

As described in Paragraph 5.2.2 of DO-178B, a Design Description (Subsection 11.10) is a primary output of the software design process. The following section describes the outline of the information that should be contained in the design documentation.

B.3.2 Design Documentation

As discussed in Subsection 11.10 of DO-178B, the design description defines the software architecture and the low-level requirements that satisfy the software high-level requirements. The design document outline shown below describes the required contents of the detailed design description for each GCS implementation. This documentation includes introductory and overview commentary on the design generated with the team*work* tool. The document produced from this outline will be analyzed during the design review and will also be used to trace changes in the design to the code. As the software code is developed and modified, the design and the code will be modified to be kept consistent. Thus, it is important to have a carefully documented description of the software design.

The design document should follow a format loosely similar to that of the GCS specification or the Hatley book on real-time system specification (ref. B.4). Note that the outline given here is a suggested outline and may be rearranged or modified by the programmer as desired. However, the content of the design document should comply with the requirements stated in DO-178B.

I. **Introduction to** *Name of implementation*

a) **Top Level Description**

This subsection should give a brief overview of the context of the application (e.g., simulates the on-board navigational code for a planetary lander, etc.). This subsection should also provide a brief overview of the organization of the design.

b) **Comments on Method**

This subsection should contain any comments regarding the philosophy or methods used during the design of the software. The tools used to generate the design (e.g., team*work*/SA and team*work*/SD) should be specifically stated.

II. **Design Structure**

As described in Subsection 11.10 of DO-178B, this portion of the design description should contain a detailed account of how the software satisfies the specified software high-level requirements, including algorithms, data structures, and how software requirements are allocated to processors and tasks. The descriptions of any algorithms used (including those that were not supplied in the GCS specification) should be contained in the team*work* design. The following information should be included to provide an overview of the detailed design. The team*work* design should be included in an appendix.

a) **Data and Control Flow**

This section should describe the data flow and control flow of the design. References should be given to the appropriate team*work* diagrams. Note that the data and control diagrams may be combined into single diagrams for each level.

b) **Module Description**

This section should provide the software architecture and low-level requirements, developed using the team*work* tool, that satisfy the requirements given in the GCS specification.

If the design is developed using the team*work*/SA tool, this subsection should contain a brief overview of the P-Specs in the design. Each P-Spec and its primitive process should share the same inputs and outputs. The body section of each P-Spec should contain a clear description of how each process transforms its inputs and its outputs.

If the design is developed using the team*work*/RT tool, this subsection should contain a brief description of the C-Specs in the design. The C-Specs should describe how the input and output control flows relate, how processes are turned on or off, and how changes in the operating mode of a system are triggered.

If the design is developed using the team*work*/SD tool, this subsection should contain a brief overview of the M-Specs in the design. This overview should include information about design modules that may be combined into larger code modules. The M-Specs should provide a one-to-one mapping to the processes in the team*work* diagrams. The body of each M-Spec should clearly describe the function of the module.

c) **Scheduling**

This subsection should provide an overview of the scheduling procedures. This subsection should also describe any use of system support utilities, including GCS_SIM_RENDEZVOUS. References should be made to the appropriate portions of the team*work* design.

d) **Data Dictionary**

This subsection should contain the data dictionary for the team*work* design. This data dictionary should include all of the data dictionary entries in the GCS specification and any additional variables contained in the design that represent flows between processes. This subsection may also contain all the information pertaining to resource limitations, such as memory and timing constraints.

e) **Derived Requirements**

This subsection should identify any derived requirements that resulted from the software design process.

III. **References**

References used for the design and anticipated for the construction of the code should be listed here. This may take the form of a bibliography. The references should include one to the GCS specification.

With respect to the DO-178B guidelines for the design descriptions, discussions of partitioning methods, previously developed software components, and deactivated code are not applicable to the GCS project, and, consequently, are not contained in the design descriptions. Further, since the project does not have system requirements or a corresponding safety assessment, a discussion of design decisions that could be traceable to those requirements is not contained in the design documentation.

B.4 Instructions to Programmers Regarding the Transitional Design Phase

Subsection 5.2 of DO-178B describes the software design process. Each GCS programmer is responsible for complying with the guidelines in that section within the scope of the GCS project. This chapter describes the responsibilities of the programmers during the transitional design phase of the software development process for the GCS project. Within this transitional phase, special instructions for modifying the existing design have been included to provide guidance to the project programmers due to the special circumstances of this period.

During the transitional design phase, the new programmers are responsible for :

1. Modifying the original design of their implementation (developed at RTI) so that the new detailed design meets the requirements of the most current version of the GCS specification and the standards set forth in this document in the chapter "Software Design Standards". As described in the design standards, the CASE tool, team*work*, should be used to update the design. Only those modifications to the detailed design to correct functionality or eliminate unnecessary design detail should be made; that is, programmers should not make changes in the design simply because that is not the design they would have chosen or because they believe the design is inefficient. There should be a reasonable justification for each modification. All additional documentation as described in the section on design documentation also should be generated.

2. Submitting any questions they may have about the specification to the system analyst. The software package, VAX Notes (ref. B.10), should be used to ask questions about the specification (so there is a record of the questions and answers). See the section on the use of VAX Notes in the chapter "Communication Protocol."

3. Submitting the detailed design description for configuration management. When the design description is complete, each programmer should contact the configuration manager so that the design description can be placed into the appropriate VAX Code Management System (CMS) (ref. B.11) library. See the chapter "Instructions for Using CMS" for a description of some of the basic commands and procedures for using CMS on this project.

4. Providing a copy of the design description to the project leader after submitting the design description for configuration management. A copy of the design description

placed into a binder with sections clearly marked would be helpful. The project leader will contact the participants in the review to schedule the review sessions.

Each programmer is required to participate in the Design Reviews for his implementation. The procedures for conducting the design reviews and the description of the role that the programmer plays during the reviews are described in the *Software Verification Plan*. The procedures for the conduct of the Design Reviews will be distributed to all appropriate project personnel (including the programmers) prior to any reviews. Each programmer must respond to all Problem Reports issued during the design reviews using the action reporting procedures described in the chapter "Problem and Change Reporting". Questions about these procedures can be directed to the SQA representative or project management.

B.5 Software Code Standards

The purpose of the software coding process described in Subsection 5.3 of DO-178B is to develop source code that is traceable, verifiable, consistent, and that correctly implements the low-level requirements. As described in Subsection 11.8 of DO-178B, the software code standards define the programming languages, methods, rules and tools to be used to generate the GCS source code. The following standards describe the programming language to be used and constraints on the coding process. For the GCS project, the code standards are primarily focused on presentation and documentation (comments) requirements.

B.5.1 Programming Language

The GCS specification was written with the assumption that a GCS implementation would be coded in the FORTRAN language. Although the software could be implemented in a programming language other than FORTRAN, for this GCS project, the GCS implementations should be coded in VAX/VMS FORTRAN since the host system for the software is a VAX/VMS system. VAX FORTRAN (ref. B.12) is an implementation of the full FORTRAN-77 language that conforms to the American National Standard FORTRAN, ANSI X3.9-1978. All code must be written in VAX FORTRAN; no assembly language or other language is permitted. Programmers should use structured programming techniques whenever practical and should not use unconditional GOTO statements. No further limits have been placed on the use of the features of the VAX FORTRAN language, including the use of VAX FORTRAN extensions. The VAX/VMS FORTRAN compiler will be used to generate the object code which will then be linked into an executable image.

B.5.2 Code Presentation and Documentation

For this GCS project, the programmers are required to follow a few simple guidelines with respect to the presentation and documentation of the source code. With respect to presentation standards (line length, indentation, blank lines, etc.), programmers are only required to make the source code easily readable to aid in verification and future modification. Programmers are encouraged to make generous use of indentation and blank lines, but no specific constraints are imposed. With respect to documentation, each programmer should add descriptive comments to the source code wherever appropriate. The comments should provide sufficient information to allow changes to be made completely, consistently, and correctly while retaining the structure. The following items also are required for the documentation of the source code: module header blocks, a revision history (starting after the first Code Review), and a system for denoting modifications. Below is a brief description of these items.

Module Header Block -- Header blocks should be used at the beginning of each module to provide an overall summary of that module. Figure B.2 shows a general format for the module header. Each programmer may choose the exact style of the header block; that is, the style does not have to conform precisely to the style presented in Figure B.2, but all of the information should be included.

Revision History -- All modifications made to each module should be summarized in a section called revision history located directly under the header block for that module. Each modification to a module should be labeled with a version number, v#. For example, the first modification to a module would be labeled v1 and the second modification would be v2. The revision history also should contain the Action Report (AR) number associated with each change made to the module, the date the change is made, the name of the person implementing the change, and a description of the change.

Notation of Modifications -- Once the source code is submitted for code review, no code that is to be modified in response to a Problem Report may be deleted. The source code that is to be modified should be commented out (instead of deleted) and the new code added. The beginning of all areas of changes should be noted clearly with a comment line, as shown below, containing the following:

```
        !+
!  v#  Begin changes for AR#<action report number>.  <short description of change>
        !-
```

The end of change areas should be similarly marked by an "End Change" comment line.

```
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!
!
!      MODULE NAME:
!      PURPOSE:
!      ARGUMENTS:
!      NOTES:
!      AUTHOR:
!      IMPLEMENTATION NAME:
!      DATE FIRST SUBMITTED FOR CONFIGURATION MANAGEMENT:
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!
!
!                              REVISION HISTORY
!      v#  , <date>, <author name>, <description, including AR#>
!
!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
!!!!!!!!
```

Figure B.2. Module Header Block and Revision History

Naming conventions for subprograms, variables, and constants should be understandable (to aid traceability and verification) and conform to requirements in the GCS specification. The specification states specific requirements regarding the labeling of global data stores. The specification also places a constraint on the use of variables in addition to the global data store variables (see the GCS specification for further information). In addition to these constraints, no special coding tools should be used to generate the code. Beyond those stated here, no further constraints have been imposed on the coding process.

B.6 Instructions to Programmers Regarding the Coding Phase

This chapter describes the responsibilities of the programmers during the coding phase of the software development process. As stated previously, the source code should implement the low-level requirements and conform to the software architecture as defined in the software design as stated in Subsection 5.3 of DO-178B. The source code should also comply with the software code standards and be traceable to the design description.

During the coding process, each programmer should:

1. Generate source code that implements the detailed design description and conforms to the Software Coding Standards defined above.
2. Document, as described in Subsection 11.11 of DO-178B, the instructions for generating the object code from the source code and loading any data files that are necessary in addition to GCS_SIM_RENDEZVOUS. This documentation should also address any tools to be used to construct or manage the code. A template and specific instructions on using the VAX Module Management System (ref. B.14) to construct the code will be provided to the programmers along with specific instructions for generating the object code. The programmers are not required to provide instructions for linking the code.
3. Submit the source code for configuration management into the CMS library (by contacting the configuration manager) when development is complete and the code cleanly compiles. For the GCS project, the programmers are not permitted to link or execute their code.
4. Contact the project leader when the source code is ready for Code Review. The project leader will contact the participants in the review to schedule the review sessions.

Each programmer is required to participate in the Code Reviews for his implementation. The procedures for conducting the code reviews and the description of the role that the programmer plays during the reviews are described in the *Software Verification Plan*. The procedures for the conduct of the Code Reviews will be distributed to all appropriate project personnel (including the programmers) prior to any reviews. Each programmer must respond to all Problem Reports issued during the code reviews using the action reporting procedures described in the chapter "Problem and Change Reporting". Each programmer is also responsible for tracing any problems found in the code back to the design. The design description should be kept consistent with the source code.

In addition, it is critical that the programmers adhere to the constraints on communication among programmers and among programmers and verification analysts. Programmers should not discuss the GCS specification or their implementations, in general, with the other programmers or verification analysts. See the chapter concerning communication protocol for further direction. Questions about these procedures can be directed to the SQA representative or project management.

B.7 Instructions to Programmers Regarding the Integration Phase

The software integration process is discussed in Subsection 5.4 of DO-178B. The programmers do not have a large role to play during this phase of the development process. During this phase, the programmers should respond to all Problem Reports that are issued to them as a result of the verification activities that are conducted. The *Software Verification Plan* describes in detail the verification activities appropriate for this phase of the development process. As stated above, each programmer is responsible for tracing any problems found in the code back to the design, so that the design description is kept consistent with the source code.

B.8 Instructions for Using CMS

This chapter provides some basic information on the use of the VAX DEC/Code Management System (CMS) as a tool to aid in the configuration management activities for the GCS project. According to Subsection 7.2 of DO-178B, configuration management should be provided throughout the software development process for configuration identification, change control, baseline establishment, and archiving of the software life cycle data. For the GCS project, CMS will be used for the configuration management of the DO-178B life cycle data shown in Table B.1. All participants on the GCS project should become familiar with the basic concepts of CMS since most of the life cycle data will be managed using this tool. Details of the configuration management process for the GCS project can be found in the *Software Configuration Management Plan*.

An important element of configuration management is establishing the configuration identification for all of the elements that make up the life cycle data. A configuration item is defined in DO-178B as one or more components that are treated as a unit for configuration purposes. Paragraph 7.2.1 of DO-178B further states that each configuration item should be uniquely labeled. For the GCS project, a number of elements of the life cycle data may be combined into a single configuration item, while other elements of the life cycle data may be decomposed into separate configuration items. The management of the life cycle data will be based on the unique labels used for configuration identification. Table B.2 shows the labels for the configuration items that comprise the DO-178B life cycle data for the GCS project. Since many of the configuration items are implementation specific, the labels of the individual configuration items should refer to the specific implementation, as appropriate. For example, the source code for the Mercury implementation should be referred to as "Source Code for Mercury". All participants of the project should refer to the project's artifacts by the appropriate label for each configuration item. The labels given in Table B.2 for the configuration items will be used as the titles for the project documentation.

Table B.1. DO-178B Life Cycle Data Required for the GCS Project

Life Cycle Data	**Subsection Reference in DO-178B**	**Responsibility**
Plan for Software Aspects of Certification	11.1	Project Leader
Software Development Plan	11.2	Project Leader
Software Requirements Standards	11.6	Project Leader
Software Design Standards	11.7	Project Leader
Software Code Standards	11.8	Project Leader
Software Accomplishment Summary	11.20	Project Leader
Software Verification Plan	11.3	Verification Analysts
Software Verification Cases and Procedures*	11.13	Verification Analysts
Software Verification Results*	11.14	Verification Analysts
Software Quality Assurance Plan	11.5	SQA Representative
Software Quality Assurance Records*	11.19	SQA Representative
Problem Reports*	11.17	SQA Representative
Software Configuration Management Plan	11.4	Configuration Manager
Software Configuration Management Records*	11.18	Configuration Manager
Software Life Cycle Environment Configuration Index	11.15	Configuration Manager
Software Configuration Index*	11.16	Configuration Manager
Design Description*	11.10	Programmer
Source Code*	11.11	Programmer
Executable Object Code*	11.12	Programmer
Software Requirements Data	11.9	System Analyst

* These life cycle data will be implementation specific.

B.8.1 CMS Description

CMS is an on-line library system that helps track the software development process (ref. B.11). A CMS library is actually a VMS directory that contains specially formatted files. In general, CMS works by storing files called elements in a library, tracking changes made to these files, and monitoring access to the files. A file can contain text, source code, object code, test cases, etc. Each configuration item shown in Table B.2 will be placed in a unique CMS library. The configuration manager for the project will establish these libraries and has primary access to all CMS libraries. Access to the configuration items will be carefully controlled to help preserve the integrity of the life cycle data. Most project participants, including programmers and verification analysts, are not allowed direct access to the CMS libraries. The Software Configuration Management Plan contains more information on the change control procedures for the GCS project and the baselines for the life cycle data.

The basic structural unit of the CMS library is called an *element*. An element consists of a file and all of the versions of that file. A *generation* of an element is one specific version of that element. Elements can be combined into a *group*, consisting of the selected elements and all of their generations, that can be manipulated as a single unit. For example, an element can be a single test case developed to test a functional module and a group could be all of the test cases to test that module. Specific generations of elements can be clustered into a *class* and manipulated as a single unit. For example, the Post-Code Review class could represent the specific generations of elements that comprise the code resulting after the Code Reviews. The generation number for all of the elements of a class can be different, indicating that some elements have been changed more than others. Classes will be used to identify the life cycle data at specific phases in the development process.

Table B.2. Configuration Identification for the DO-178B Life Cycle Data

Life Cycle Data	Labels for the Configuration Items
Plan for Software Aspects of Certification Software Development Plan	Plan for Software Aspects of Certification
Software Requirements Standards Software Design Standards Software Code Standards	Software Development Standards
Software Accomplishment Summary	Software Accomplishment Summary
Software Verification Plan	Software Verification Plan Software Requirements Traceability Data
Software Verification Cases and Procedures*	Software Verification Cases* Software Verification Procedures
Software Verification Results*	Software Verification Results*
Software Quality Assurance Plan	Software Quality Assurance Plan
Software Quality Assurance Records*	Software Quality Assurance Records*
Problem Reports*	Problem and Action Reports* Support Documentation Change Forms
Software Configuration Management Plan	Software Configuration Management Plan
Software Configuration Management Records*	Software Configuration Management Records*
Software Life Cycle Environment Configuration Index Software Configuration Index*	Software Configuration Index *
Design Description*	Design Description*
Source Code*	Source Code*
Executable Object Code*	Executable Object Code*
Software Requirements Data	GCS Specification

* These configuration items will be implementation specific.

B.8.2 Basic CMS Commands

Once an item has been placed under configuration control, there must be a valid justification to change it. CMS uses a system of reservations and replacements to manage the elements of a library. Since the configuration manager has the primary responsibility for the configuration management activities, the rest of the project participants need to know only a few basic commands to manage their life cycle data. All project participants should use the labels given in Table B.2 when referring to specific configuration items. The following are basic CMS operations that project participants should learn. The *Guide to VAX/DEC Code Management System* (ref. B.11) provides more information about the commands available for CMS.

Fetch -- A copy of one or more specified element generations is placed in a directory for use by the participant. No changes to the file within the CMS library will be made. For example, a copy of the element generations that comprise the version of code to be reviewed at the Code Reviews (Pre-Code Review version of an implementation) may be fetched for all of the participants in the Code Review to examine in preparation for the Reviews.

Reserve -- A copy of one or more specified element generations is placed in a directory so that it can be modified by the participant. The element is marked within the CMS library that it is reserved so that no one else may make changes to it during this time. After the file has been modified, the file should be returned to the library (using the Replace command) and the changes will be made to the library copy. As an example of this command, a programmer should reserve a particular element of source code in order to make a change to it in response to a Problem Report.

Replace -- An element that has been reserved can be replaced and, in doing so, any changes to the reserved version (which may be completely different from the replacement file) are put into the library for later use. A new generation of that element is created. In the example where the programmer has reserved an element to make a change in response to a Problem Report, the programmer should replace that element when he has completed the necessary change.

If an element needs to be changed, it must be reserved, changed, and replaced. Every action which results in a change to the CMS library (including use of the RESERVE and REPLACE commands) is recorded in a history file, along with the name of the person requesting the action, the date, and a comment. The report number for each change should be noted in the comment for that reservation. The original version, or generation, of the element is generation 1. After an element is reserved and replaced, it becomes generation 2. All previous generations of any element are easily retrieved from CMS. A particular class of elements can also be reserved.

B.9 Problem and Change Reporting

According to Paragraph 7.2.3 of DO-178B, there should be a mechanism within the software development processes for problem reporting, tracking and corrective action in order to:

- record process non-compliance with software plans and standards,
- record deficiencies of the outputs of the life cycle processes,
- record anomalous behavior of the software products, and
- ensure resolutions of these problems.

An effective problem reporting and tracking system is also extremely important in terms of the project goals, because one of the major objectives of the GCS project is to collect software error data which can be used to help assess the reliability of the resultant software and also assess the effectiveness of different development and verification methods for generating reliable software. In the context of the GCS project, a problem is a question or issue raised for consideration, discussion, or solution regarding some artifact of the software development process. In the software development process, problems can be identified in practically all life cycle data, including the software requirements, software design and code, and test cases.

The tables in Annex A of DO-178B specify that certain life cycle data are classified under Control Category 1 (CC1), which means that the project must provide a formal system of problem reporting, change control, and change review for that data. Other life cycle data are classified under Control Category 2 (CC2), indicating that formal problem reporting and change control procedures are not required for certification. For the purposes of developing an efficient problem and change reporting system, the DO-178B life cycle data has been divided into three different categories: development products (shown in Table B.3); support documentation (shown in Table B.4); and records, results, and reports (shown in Table B.5). The life cycle data in the development products and support documentation categories are all under CC1. A unique problem and change reporting system has been established for each category under CC1.

B.9.1 Problem Reporting for Development Products

This section addresses the content and identification of problem reports for the development products, time frame for initiating problem reports, the method of closing problem reports, and the relationship to the change control activity in compliance with Subsection 11.4 of DO-178B. Note that the discussion of problem reporting procedures would typically appear in the *Software Configuration Management Plan*, according to DO-178B. However, since all project participants will be participating in the problem reporting, tracking and correction activities, repetition of the procedures in this document is appropriate.

The GCS Problem Report (PR) and Action Report (AR) forms, shown in Figures 3 and 4, respectively, will be used to document any problems and subsequent changes to the development products that arise during the development of the GCS implementations. The PR form is used to capture data concerning a possible problem that is identified during the software development process. The Problem Report contains

- information about when (in the development process) the problem was identified,
- the configuration identification of the artifact
- a description of the problem (such as non-compliance with project standards or output deficiency), and
- a history log for tracking the progress and resolution of the problem.

Table B.3. CC1 Development Products

Design Description
Source Code
Executable Object Code

Table B.4. CC1 Support Documentation

Plan for Software Aspects of Certification
Software Development Plan
Software Requirements Standards
Software Design Standards
Software Code Standards
Software Accomplishment Summary
Software Verification Plan
Software Verification Cases and Procedures
Software Quality Assurance Plan
Software Configuration Management Plan
Software Life Cycle Environment Configuration Index
Software Configuration Index
Software Requirements Data

Table B.5. CC2 Records, Results, and Reports

Software Verification Results
Software Quality Assurance Records
Problem Reports
Software Configuration Management Records

All problems are investigated to determine if indeed a fault has been detected, in which case corrective action is taken and properly documented. Each identified fault is traced to determine the source where the fault was introduced. The AR form is used to capture relevant information about the action that is taken in response to a Problem Report. The Action Report will contain the configuration identification of the artifact affected and a description of a change that is made to an artifact in response to the Problem Report. Change control procedures, as described in the *Software Configuration Management Plan*, should be followed when the actual change is made to a configuration item. In the case that no change is required in response to the PR, the AR form will contain the justification for not making any changes.

B.9.2 Instructions for Problem and Action Reports

In general, a project participant who identifies, in the course of his prescribed activities, something in a development product that may be regarded as a problem (such as a violation of a software requirement or project standard) is responsible for initiating a Problem Report. However, during those verification activities where a Moderator is present, the Moderator will have the authority to determine whether issuing a Problem Report is appropriate. Figure B.5 shows the flow of the problem reporting process, starting with the initiation of a PR to the final signature from the SQA representative indicating that the problem has been resolved. The following procedure, as shown in the flow chart, should be followed. During the development cycle,

1. The initiator of the PR form fills out the form from Section 2 through Section 8. The Continuation form should be used if additional space is required for further explanation.
2. The PR form is given to the SQA representative who assigns a PR number to it and logs this PR as an outstanding PR.
3. The SQA representative keeps the original PR form and gives a copy to the most appropriate member of the development project for examination.
4. The project member receiving a copy of the PR form should examine the appropriate artifact to determine if a change should be made. The response to the PR is made on an Action Report. If one or more changes are necessary, the change(s) are made and Action Reports describing the changes are written. When completing the Action Report, the respondent should contact the SQA representative to get the appropriate AR number. The respondent should refer to the AR number when requesting the appropriate configuration item from the configuration manager. This number should also be placed in the artifact comments when a change has been made. It is also important to make the change at this time.
5. The project member will return the PR form to the SQA representative with either one or more Action Reports. The SQA representative checks that the report(s) are properly filled out and contain an adequate description of the change or an adequate explanation for making no change. At this time the SQA representative may deem it necessary to give a copy of the PR form to a different member of the project. This process may repeat itself until the SQA representative decides no further changes are necessary without further review by the PR initiator. It is the responsibility of the SQA representative to make sure that each problem is properly traced back to its origin. The SQA representative notes the sequence of the PR distribution in the history section of the original PR form.
6. When all parties have responded to the PR, the SQA representative gives the original PR form and the Action Report(s) to the initiator. If the initiator feels that the problem is resolved, he signs off on the PR form and gives it to the SQA representative for final approval. If the initiator does not feel the problem is resolved, the initiator can seek further changes through the SQA representative. The SQA representative should make note of any problems in the History Log.
7. The SQA representative then reviews the Problem and Action Reports. If further modification is deemed necessary, the reports should be distributed for further action. Upon final approval of the reports, the SQA representative notes the total number of changes and the total number of no changes on the original PR form and signs and dates it signifying resolution of the problem. The SQA representative then indicates the resolution of this PR on the master list of PRs. The Action Report forms should be attached to the original PR form.
8. The SQA representative should notify the configuration manager that the configuration items that were modified have been approved and should be replaced in the CMS libraries.

B.9.3 Number System for the Problem and Action Reports

This section discusses the identification system for the Problem and Action Reports. Each GCS implementation will have its own set of Problem and Action Reports for the development products. The identification numbers for the Problem and Action Reports are of the form:

a.b where
a is the chronological number of the Problem Report
b is the chronological number of the action made in response to Problem Report "a"

The Problem Reports will be numbered: 1.0
2.0
3.0
...

The subsequent responses made (via Action Reports) to a Problem Report would be numbered:

<PR#>.1
<PR#>.2
<PR#>.3
...

For example, consider the third problem found with an implementation and suppose that 2 responses are made to the Problem Report. The Problem Report number would be 3.0 and the Action Report numbers would be 3.1 and 3.2

GCS Problem Report

page 1 of ____

1. PR #:	2. Planet:	3. Discovery Date:	4. Initiator & Role:

5. Activity at Discovery:

Development Phases \ Activity	Design Review	Code Review	Reading Code	Reading Specification	Test Readiness Review	Test Completion Review	Test Case Creation	Test Case Execution	˘ n	Other
Design		■	■		■	■	■	■		
Code	■				■	■	■	■		
Unit Testing										
Functional	■	■								
Structural	■	■								
Subframe Testing	■	■								
Frame Testing	■	■								
Top-Level Simulator Integration Testing	■	■								

6. Description of Problem:

7. Artifact Identification:

□ Design Description　□ Support Documentation ______________________

□ Source Code　□ Other ______________________

□ Executable Object Code ______________________

8. Test Case Identification:

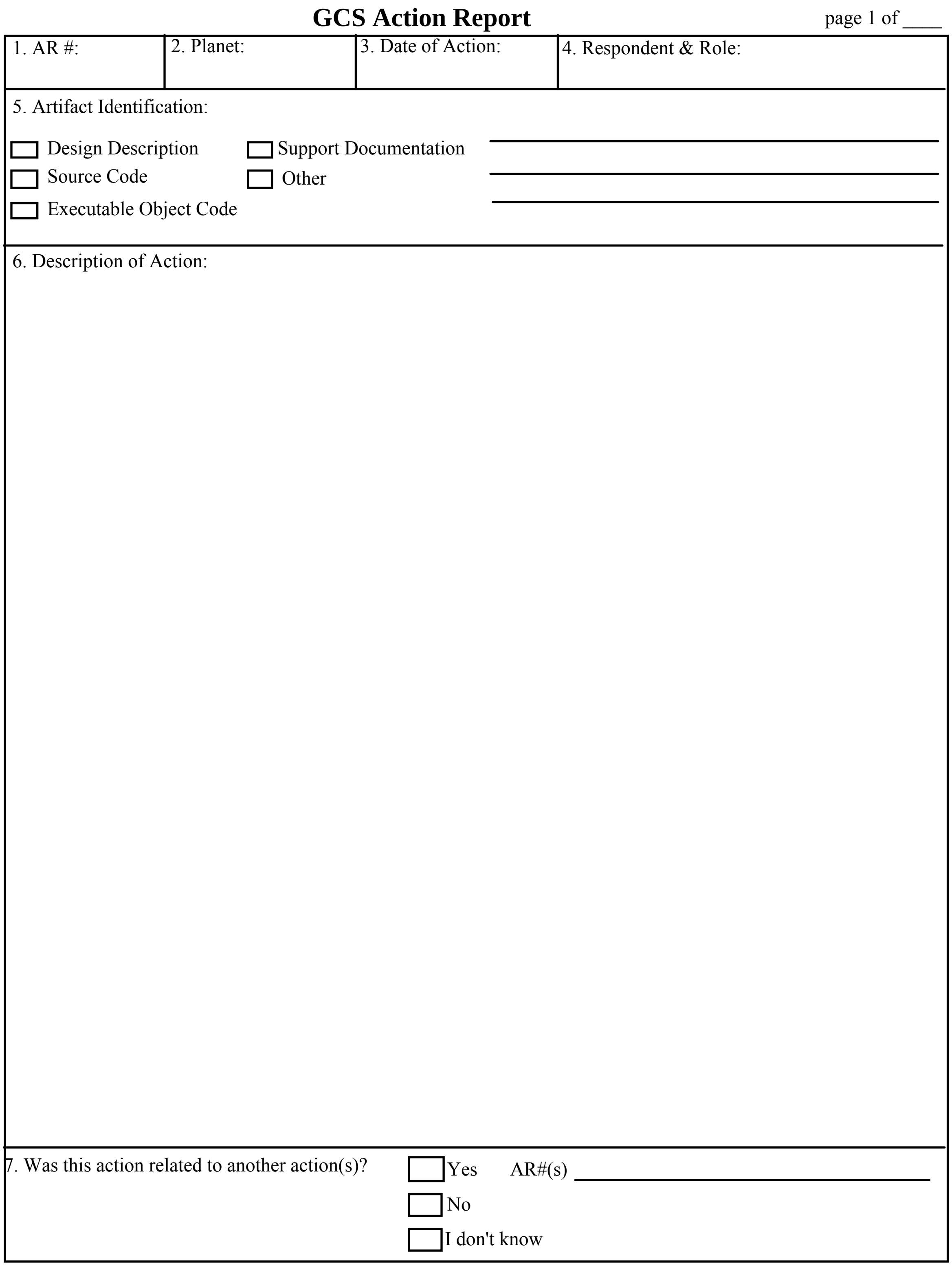
9. History Log:

Date To	Date From	Person	Comments	AR#

10. Total # of Changes: □　11. Total # of No Changes: □

12. Initiator Signature & Date	13. SQA Signature & Date

Figure B.3. GCS Problem Report Form



GCS Action Report

page 1 of ____

1. AR #:	2. Planet:	3. Date of Action:	4. Respondent & Role:

5. Artifact Identification:

□ Design Description □ Support Documentation ______________________________

□ Source Code □ Other ______________________________

□ Executable Object Code ______________________________

6. Description of Action:

7. Was this action related to another action(s)? □ Yes AR#(s) ______________________________

□ No

□ I don't know

Figure B.4. GCS Action Report Form

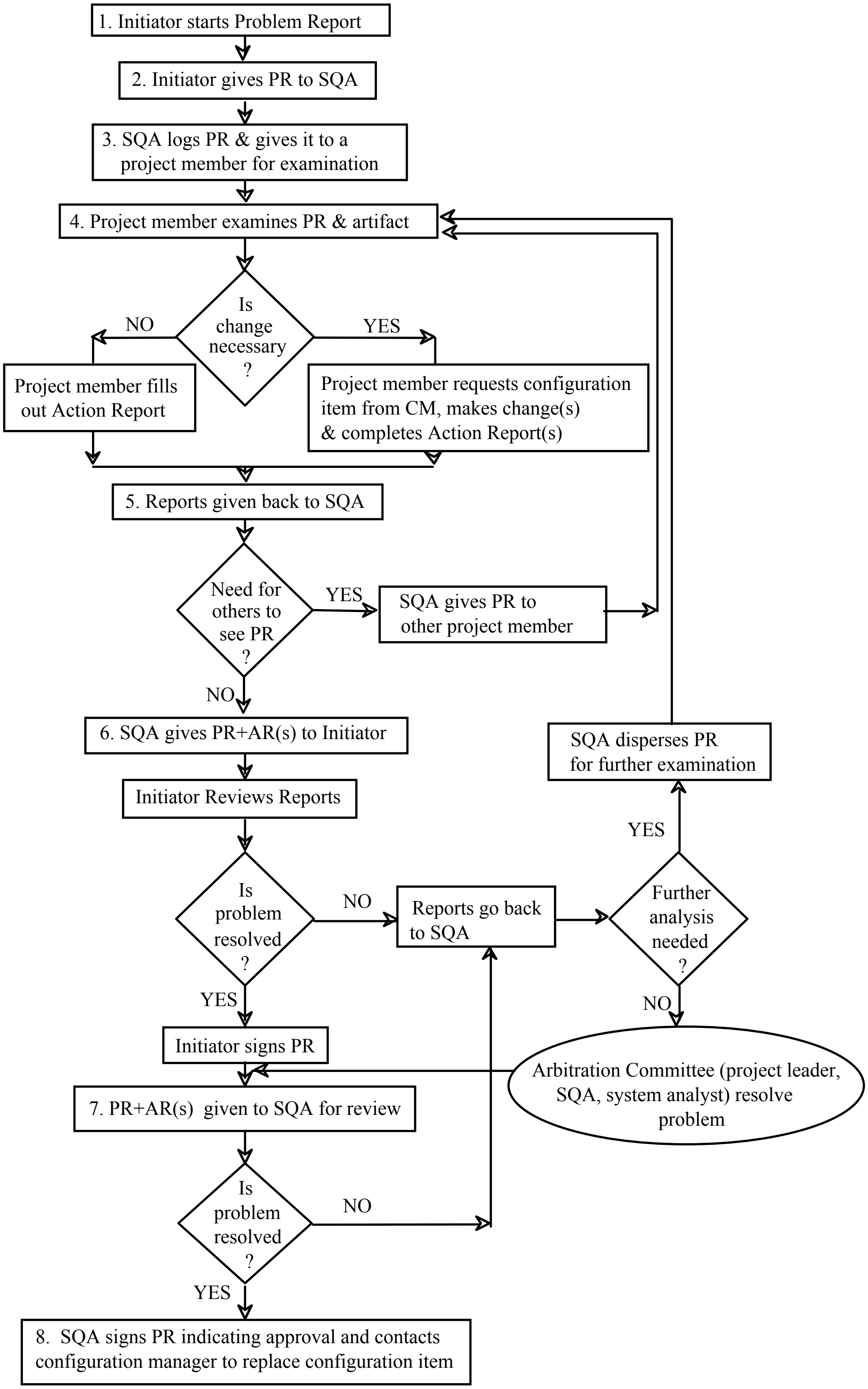

1. Initiator starts Problem Report
2. Initiator gives PR to SQA
3. SQA logs PR & gives it to a project member for examination
4. Project member examines PR & artifact
Is change necessary ?
NO
YES
Project member fills out Action Report
Project member requests configuration item from CM, makes change(s) & completes Action Report(s)
5. Reports given back to SQA
Need for others to see PR ?
YES
SQA gives PR to other project member
NO
6. SQA gives PR+AR(s) to Initiator
SQA disperses PR for further examination
Initiator Reviews Reports
YES
Is problem resolved ?
NO
Reports go back to SQA
Further analysis needed ?
YES
NO
Initiator signs PR
Arbitration Committee (project leader, SQA, system analyst) resolve problem
7. PR+AR(s) given to SQA for review
Is problem resolved ?
NO
YES
8. SQA signs PR indicating approval and contacts configuration manager to replace configuration item

Figure B.5. Flow of Problem Reporting Process for the Development Products

B.9.4 Completing the Problem Report Form

In this section, instructions for completing the fields of the PR form are stated. Specific instructions or further explanation for each section of the PR form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of attached pages. As many Continuation forms as necessary may be used.

1. **PR#**: to be assigned by the SQA representative
2. **Planet:** the name of the planet in whose development process this problem was identified
3. **Discovery Date**: date when this problem was identified. It is important to issue a PR form at the time a problem is identified.
4. **Initiator & Role**: name of the person who has identified the problem and the role (programmer, verification analyst, SQA representative, or system analyst) that person is fulfilling at the time of problem identification.
5. **Activity at Discovery**: The development cycle for each GCS implementation can be decomposed into 6 distinct phases. In this section, indicate the phase by placing an **X** in the appropriate box that corresponds to the development phase in which this problem was identified and the specific activity that was being performed at that time. If the Other category is appropriate, please put an explanation in Section b of the Continuation form.
6. **Description of Problem**: Provide an adequate description of the issue in question.
7. **Artifact Identification**: Check the box that corresponds to the artifact under consideration when the problem was identified. The label for the configuration item should be given along with the information in Table B.6 for each artifact. If a PR is being generated because the actual results from the execution of a test case did not agree with the expected results, the initial artifact under consideration would be the executable object code. The test case that surfaced the anomalous behavior would be identified in Section 8. If more space is needed, use Section b of the Continuation form.
8. **Test Case Identification**: If the failure of a test case is the reason for initiating this PR, fill in the appropriate test case number, including its configuration item label, element name(s), and generation #; otherwise, indicate Not Applicable (N/A).
9. **History Log**: to be filled in by the SQA representative. The SQA representative should log the sequence of dispersals of the PR, logging all ARs related to the PR and noting date of issuance, date of return, and the person receiving the PR form. The SQA representative should also note any anomalies in the resolution of the problem, such as disagreements in resolution between the initiator and the person making the change.
10. **Total # of Changes**: to be filled in by the SQA representative when all Action Reports are closed and the problem has been resolved. A total of 0 indicates that no change was made.

Table B.6. Information for Artifact Identification

Artifact	Information
Design Description	diagram, P-Spec #, C-Spec #, or M-Spec #
Source Code	element name & generation #
Executable Object Code	element name & generation #
Support Documentation	specific chapter, section, and table or figure reference, as appropriate
Other	be as specific as possible

11. **Total # of No Changes**: to be filled in by the SQA representative when all Action Reports are closed and the problem has been resolved.

12. **Initiator Signature & Date**: The person who initiates the PR should sign and date the original PR form here when the problem has been resolved.

13. **SQA Signature & Date**: After checking that the problem is satisfactorily resolved and all necessary changes have been properly made, the SQA representative should sign and date the original PR form indicating closure of this PR.

B.9.5 Completing the Action Report Form

In this section, instructions for completing the fields of the AR form are stated. Specific instructions or further explanation for each section of the AR form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

1. **AR#**: to be assigned by the SQA representative. The respondent should contact the SQA representative to get the appropriate AR number. When a change is indicated, the AR# can be incorporated in the comments which describe this change in the code or design.

2. **Planet:** the name of the planet associated with the person making this action.

3. **Date of Action**: date when this action was taken. In case of changes, it is important to complete the AR form at the time a change is being made.

4. **Respondent & Role**: name of the person who is making the response and his role (programmer, verification analyst, SQA representative, or system analyst).

5. **Artifact Identification**: Check the box that corresponds to the artifact in question. The information in Table B.6 should be specified for each artifact. In case of responses made to the support documentation, the label for the configuration item should be cited. If more space is needed, use Section b of the Continuation form.

6. **Description of Action**: provide a general description of the change that was made or an explanation of why no change is necessary. In case of responses made to the support documentation, the appropriate modification number from the Support Documentation Report Form should be cited.

7. **Was this action related to another action(s)?:** Check the appropriate box to indicate whether this action is related to another action. If yes, indicate the relevant AR#(s).

B.9.6 Problem Reporting for Support Documentation

The problem and change reporting for the support documentation will be conducted through the use of Support Documentation Change Reports. Although the Support Documentation Change Report form shown in Figure B.6 does not capture as much detailed information as the Problem Report, this form does capture the information necessary to comply with Paragraph 7.2.3 of DO-178B. Once a support document enters the configuration management system, all further changes to that document will be controlled through the Support Documentation Change Reports; that is, all changes to any support documentation must be accompanied by an approved Support Documentation Change Report. Each configuration item that is a part of the support documentation will have its own set of change reports. The SQA representative will keep a log of all change reports for each configuration item.

The following procedure, as shown in the flow chart in Figure B.7, should be followed for initiating and completing the Support Documentation Change Report for all support documentation.

1. The author of the support documentation fills out Sections 1, 2, 4, and 5 of the Support Documentation Change Report form. The Continuation form should be used if additional space is required for further explanation.
2. The form is given to the SQA representative who determines if the change request is reasonable and assigns a modification number to the report if the request is approved.
3. The SQA representative logs this as an outstanding change report for the particular configuration item and returns the form to the author to implement the change.
4. The author requests to reserve the affected configuration item and must refer to the modification number when making the request.
5. The author implements the requested change to the configuration item.
6. When the modification is completed, the author completes Section 6 of the form, places the configuration item in the appropriate place for the configuration manager to retrieve, and returns the form to the SQA representative for review.

Support Documentation Change Report page 1 of ____

1. Configuration Item:	2. Date:	3. Modification #:
4. Part of Configuration Item Affected:		
5. Reason for Modification:		
6. Modification		
7. SQA Signature & Date:		

Figure B.6. Support Documentation Change Report Form

7. The SQA then reviews the change for consistency and compliance with project plans and standards. If the change is not acceptable, the SQA representative can work with the author to implement the necessary modifications. The project leader will arbitrate if the author and SQA representative cannot reach consensus.

8. When the change has been completed and approved by the SQA representative, the SQA representative should notify the configuration manager that the configuration item that was modified has been approved and should be replaced in the appropriate CMS library.

B.9.7 Completing the Support Documentation Change Report Form

In this section, instructions for completing the fields of the Support Documentation Change Report form are stated. Specific instructions or further explanation for each section of the Support Documentation Change Report form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

1. **Configuration Item:** the label for the configuration item that needs to be changed.
2. **Date:** date that this change report is being initiated.
3. **Modification #:** to be provided by the SQA representative. The author should give the form to the SQA representative to get the number and corresponding authorization to implement the change.
4. **Part of the Configuration Item Affected:** describe the location of the proposed change. Chapter and section references should be included as appropriate.
5. **Reason for Modification:** explanation detailing why the configuration item should be changed.
6. **Modification:** description of the change including the following information as appropriate: original text (that is to be changed), action (such as deletion, addition, or modification), and modified text (the correct text to be inserted). If substantial changes are made, the affected pages should be attached to the form.
7. **SQA Signature and Date:** After checking that the change is acceptable and has been properly made, the SQA representative should sign and date the form indicating approval of this change.

B.9.8 Completing the Continuation Form

The Continuation Form provides extra space in addition to the PR, AR, and Support Documentation Change Report forms. Figure B.8 shows the Continuation Form. Specific instructions or further explanation for each section of the Continuation form are provided below.

______________ Report Continuation: Fill in the blank with the name of the form that is being continued.

page__ of __: Fill in the page number and total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

a. **Report #**: the number of the report that is being continued

b. **Notes/Explanation**: This section is to be used to continue comments or descriptions from any section of a report.

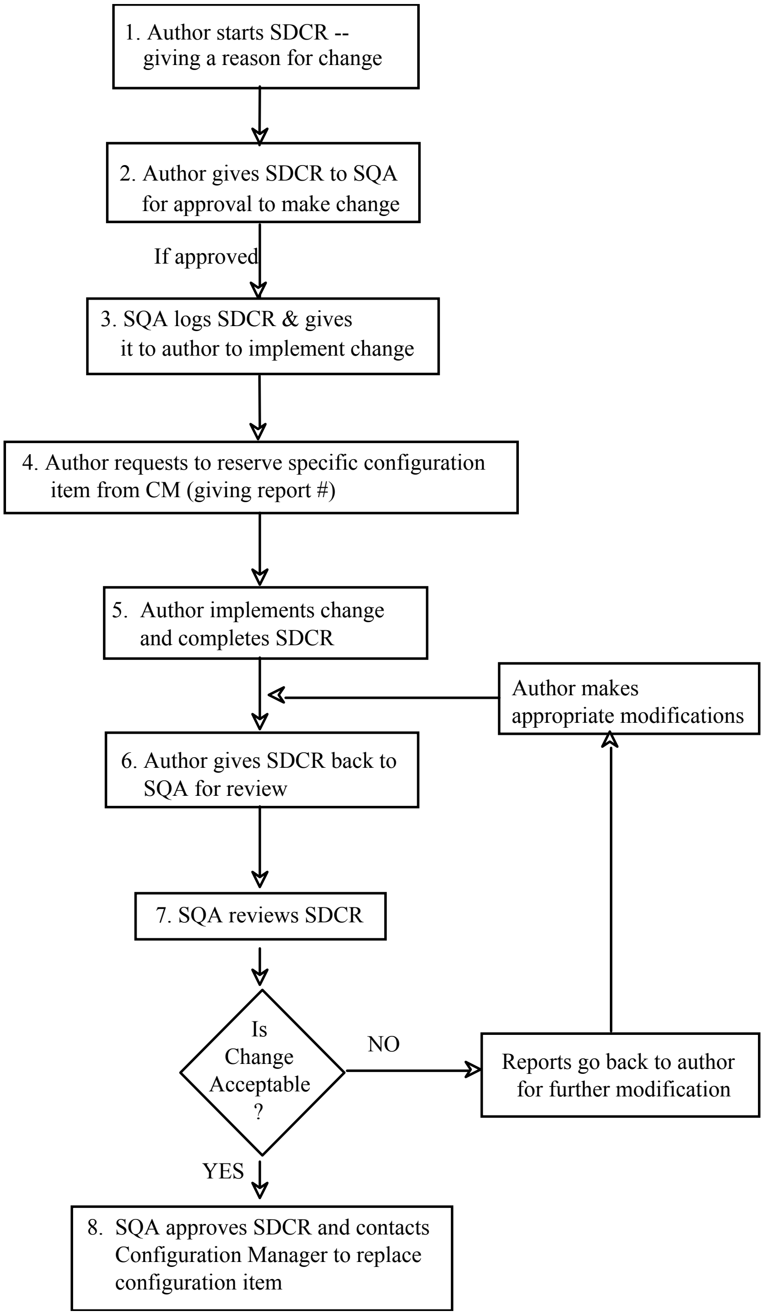
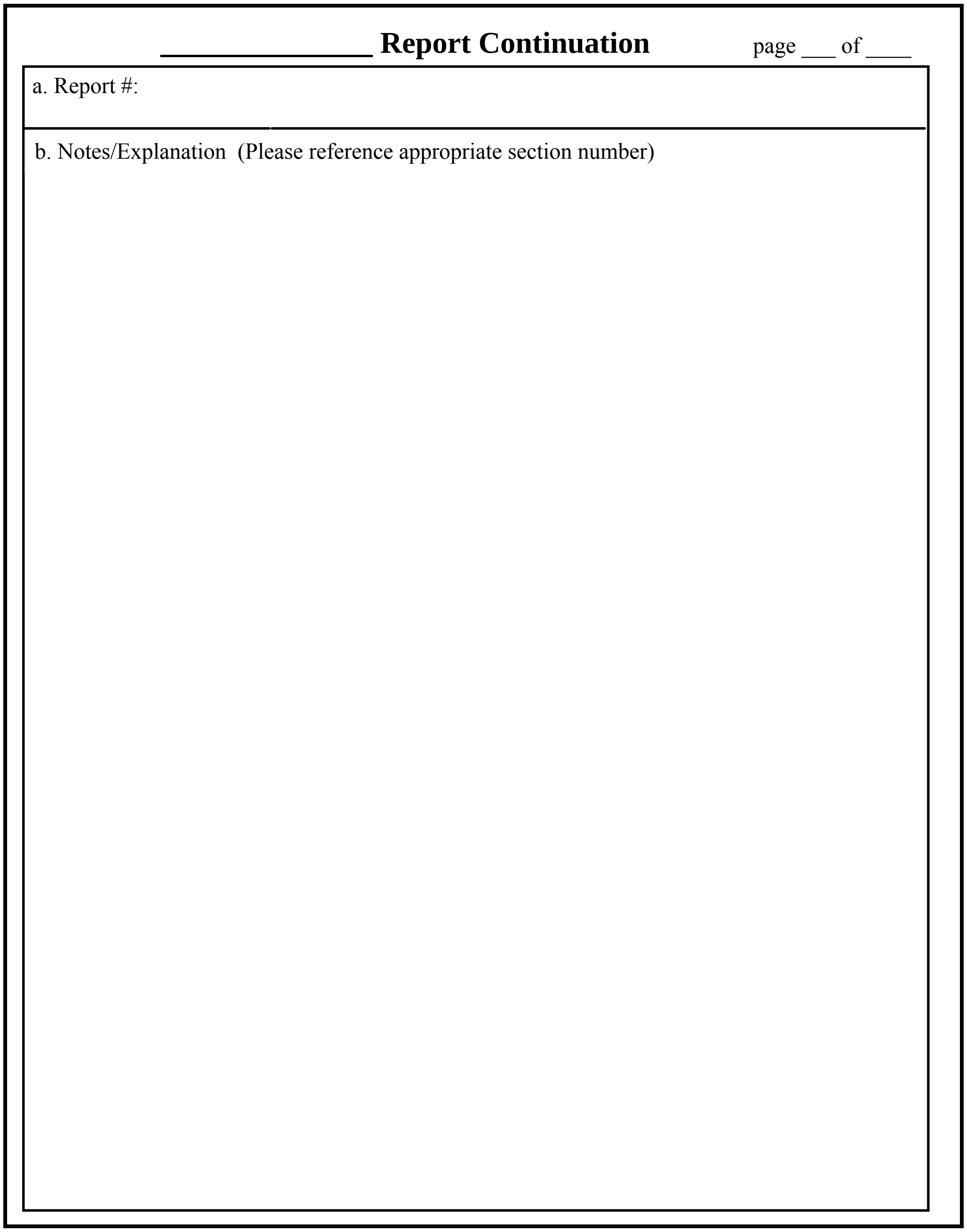

1. Author starts SDCR -- giving a reason for change
2. Author gives SDCR to SQA for approval to make change
If approved
3. SQA logs SDCR & gives it to author to implement change
4. Author requests to reserve specific configuration item from CM (giving report #)
5. Author implements change and completes SDCR
Author makes appropriate modifications
6. Author gives SDCR back to SQA for review
7. SQA reviews SDCR
Is Change Acceptable ?
NO
Reports go back to author for further modification
YES
8. SQA approves SDCR and contacts Configuration Manager to replace configuration item

Figure B.7. Flow of Change Reporting Process for the Support Documentation



_______________ **Report Continuation** page ___ of ____

a. Report #:

b. Notes/Explanation (Please reference appropriate section number)

Figure B.8. Report Continuation Form

B.10 Collecting Effort Data

The DO-178B guidelines do not address the collection of effort data for a software development process. However, one of the major objectives of the GCS project is to make observations on the effectiveness of a development process that complies with the DO-178B guidelines. Part of the effectiveness assessment includes a report on the effort hours expended to accomplish various development activities. For the GCS project, effort data will be collected throughout the DO-178B development process for the GCS implementations from all of the major project participants (programmers, verification analysts, SQA representative, configuration manager, and system analyst). There is a unique data collection form specific to each particular role in the development process, and each participant will be required to record effort on a daily basis. The list of activities on each form is not an exhaustive list of activities required by the participants in the project, but instead represents the primary activities where effort data is of interest. Consequently, the effort hours listed on the effort data forms may not reflect the total number of hours a participant has worked on the GCS project during the given time period. Each form will be used to collect information over a period of one week (Sunday through Saturday). These forms are given in the Section 13. The following are the general procedures for recording the effort data.

On the form, the participant will fill in his name, the name of the planet to which he is assigned if applicable, and the dates for the week that the effort is being recorded (for example, 9/20/92-9/26/92 for the week of September 20-26, 1992). Then, for each day of the week, the participant records the number of hours spent in each of the specified activities. Although the activities for which the effort is recorded are largely self-explanatory, additional instructions regarding these activities are given in the Section 13. Time should be recorded to the nearest tenth of the hour (rounding up) for each activity. For example, if a programmer spends 4 hours and 21 minutes making changes to his code due to a code review, he would record 4.4 hours in the appropriate place on the effort data form. There is no need to record a "0" when no effort has been expended in a particular activity. However, if no effort data has been recorded for any of the activities during a given week, the effort data form should still be filled out by placing a "0" in the first entry in the column labeled "Totals" and drawing a straight line through the remainder of the Totals column. The forms should be submitted to the project leader the following week. Any questions regarding the effort data should be directed to the project leader.

B.11 Communication Protocol

Because the GCS software development process is part of a larger experiment framework for studying the characteristics of the software failure process, maintaining a high degree of independence among the different GCS implementations is important. Hence, the control of communication among the project participants is very important. A software product called VAX Notes will be used as the principal means of formal communication among project participants to help maintain control of the communication among the various project participants and provide an automated system for recording the exchange of certain information. (See the *Software Life Cycle Environment Configuration Index* or further information on VAX Notes.) The relationships, for communication, among the project participants have been divided into two classes: primary communication and secondary communication. The following diagram shows those participants included in the primary communication class.

Primary Communication Flows

Programmers	<------------>	System Analyst
Programmers	<------------>	Configuration Manager
Verification Analysts	<------------>	Configuration Manager

In the primary communication class, it is important to capture the communication that takes place between each pairing of participants. All questions about the GCS specification should be addressed to the system analyst. It is especially important to capture the questions that the programmers ask the system analyst about the specification and the response from the system analyst. All questions to the system analyst should be specific to the GCS specification as opposed to questions about implementation specific issues. Additionally, the programmers and verification analysts should use VAX Notes when making requests for elements from the configuration manager, and the configuration manager should respond using VAX Notes.

The relationships in the secondary communication class are shown below.

Secondary Communication Flows

Verification Analysts	<------------>	System Analyst
Programmers	<------------>	SQA Representative
Verification Analysts	<------------>	SQA Representative

In this class, the need to capture all communication between each pairing is not critical. Verification analysts may use VAX Notes to ask the system analyst specific questions regarding the GCS specification, but they may not ask implementation specific questions. Questions regarding project policies, procedures and standards should be addressed to the SQA representative. VAX Notes may be used here as a convenient medium for communicating and capturing a record for future reference of that information; but communication using VAX Notes in these cases is not required.

Along with the VAX Notes conferences established for the communication flows in the primary and secondary classes, there will be a general Announcements conference available to all project participants. General information about the project such as schedule changes and meeting announcements or updates to policies and procedures affecting all project participants may be posted to this conference.

B.11.1 Conventions for Communication between Programmers and System Analyst

All communication between the system analyst and the programmers should be done using VAX Notes so that records can be kept of the questions asked about the GCS specification and the responses made to those questions. This section describes specific conventions that the programmers should follow when using VAX Notes to communicate with the system analyst about the specification.

Special VAX Notes conferences and classes have been established to aid communication. The VAX Notes class which contains the relevant conferences is called GCS. The relevant conferences in the class GCS are as follows:

Announcements: contains announcements from either the Project Leader or the SQA representative to all GCS project participants

SA-All-Programmers: contains announcements from the system analyst to all of the programmers

SA-Mercury-Programmer: contains all communications between the system analyst and the Mercury Programmer

SA-Pluto-Programmer: contains all communications between the system analyst and the Pluto Programmer

Within these conventions regarding communication, the words **topic**, **reply** and **note** will be used in the strict sense of a VAX Notes topic, reply, or note, respectively. The examples given here are not meant to be realistic in terms of any specific version of the GCS specification, but are given merely as examples of notes formatted according to the conventions outlined here.

B.11.2 General Rules Regarding Topics and Replies

If a programmer has a new question to discuss with the system analyst, then the programmer should send the question to the system analyst by writing a new VAX Notes **topic** into the relevant conference. When the system analyst responds to the question for the first time, a VAX Notes **reply** to the VAX Notes topic will be sent If the programmer wishes to respond to either the original topic or the first reply, then that person should send another **reply** to the same topic, and the system analyst will do the same. In other words, as long as the conversation is related to the original topic or any of the replies to that topic, then all communications will be in the form of sequential replies to that same topic; however once the programmer wishes to ask about or discuss a new issue, then writing a new VAX Notes topic is appropriate.

Normally, each topic should contain only a single question. A topic may contain more than one question only in the case where the questions are **very closely related** to each other. Each question in a topic should be very specifically stated.

Conventions and Formats for Notes

- **Note Title**

  Each title may contain up to 63 characters (see page 3-5 of *Guide to VAX Notes*). The title should be as informative as possible about the contents of the note because when one performs a directory of the notes in a conference, the title appears, but the text of the note does not.

  Topic Title

  The topic title should be written according to a strict format because parts of it will be used by the system analyst to organize the notes. The topic title should have the following format, where the "/" is an actual literal that must appear. The item inside the closed brackets is conditionally required (see below). The format is:

  Topic Title = </Topic Source[/Figure-Table]/Topic Description>

  - Topic Source (required)

    The Topic Source is either the name of the section(s) in the specification or the name of a modification to the specification, to which the question applies. The specification section names are predefined and appear in Table B.7 below. The programmer must use at least the first four characters of the section name if the section name has four or more characters, but may use more if so desired. If the actual section name has less than four characters, then the full section name should be used. In those cases where the first four characters are not unique, substitutions are given in the table below, and those substitutions must be used instead of the actual section name. In each case, the required part of the section name is bolded. If the source of the question is a Support Documentation Change Report, then the Topic Source should be "Modx.y-z", where x.y-z is the number of the modification. If, for some reason, none of the predefined section names nor a modification number is appropriate, then one should use the substitute name "other" and describe the source in the text part of the topic. In the case where the question applies to more than one source, list all the applicable sources separated by commas.

Examples of valid Sources are:

aecl
AECLP
cp
dad1,dad2,dad3
intr
INTRODUCTION
Terminal Desc
vehd
MOD2.2-1
other

Table B.7. Specification Section Names

Section Name as it Appears in Table of Contents	Required Substitutions
arsp	
asp	
appendix a	**appa**
appendix b	**appb**
appendix c	**abbc**
bibliography	
cp	
crcp	
contents	
conventions	
data dictionary 1	**dad1**
data dictionary 2	**dad2**
data dictionary 3	**dad3**
definitions	
engines	
exception handling	
foreword	
general	
gp	
gsp	
introduction	
level 0 spec	**lev0**
level 1 spec	**lev1**
level 2 spec	**lev2**
level 3 spec	**lev3**
list of figures	**lisf**
list of tables	**list**
notation	
preface	
purpose	
reclp	
requirements	
rotation	
tdlrsp	
tdsp	
terminal descent	
title page	
tsp	
use of tables	**tabl**
vehicle configuration	**vehc**
vehicle dynamics	**vehd**
vehicle guidance	**vehg**
(none)	**othe**r

- Figure-Table (required only if question involves a numbered Figure or Table)

  If the question or issue involves a Figure or Table in the specification, then the abbreviation "Fig" or "Tab" should appear followed by the actual figure or table number with no intervening spaces.

  Examples of valid Figure-Table are:
  Fig3.1
  figB.1
  Tab5.11
  TABC.1

- Topic Description (required)

  Topic Description is a description of the question or issue in the topic text. This description may contain any characters acceptable to VAX Notes. It is suggested that the description begin with "Q:" for a question, "A:" for an answer or "S:" for a statement.

  Example of a valid topic description:
  Q: Why is THETA initialized to zero?
  Examples of valid Topic titles:
  /RECLP/ Q: Why is THETA initialized to zero?
  /Aecl/ Q: What does "to the nearest integer" mean?
  /TDLRSP/Tab5.11/ Q: What is the meaning of "Σ"?

<u>Reply Title</u>

The format of the reply title is as follows:

Reply Title = <Reply Description>

The Reply Description is any text which concisely describes the contents of the text of the note. It does not have to subscribe to any particular format. It may contain any characters acceptable to VAX Notes.

Example of valid Reply Title:
A: Text in spec is incorrect. SA will Issue Formal Mod.

- **Note Text**

Each programmer should read Section 3.1.1 of *Guide to VAX Notes*, "Making Your Notes More Readable" and should exercise personal judgment in using these suggestions as guidelines in writing the text part of the note.

<u>Topic Text</u>

The following items may be included in the topic. The bolded items are literals that should appear in the text. The "Page:" and "Location:" items should appear first and second, respectively. If the page has a modification, one should include that information in the page number. The statements and questions may appear in any order, but each should start on a new line.

PAGE: <the starting page number of the problem or issue> (required)
LOCATION: <description of the starting location> (required)
Q: <the question or problem> (optional)
S: <a statement or comment> (optional)

Example of Topic Text:
Page: 65
Location: Section labeled "DETERMINE PULSE INTENSITY AND DIRECTION", third sentence from the end of the paragraph.
S: The text states: "The variable THETA will be initialized to the value zero by INIT_GCS."
Q: Since THETA is the roll angle, it does not seem logical that it would always be initialized to zero. Is this sentence correct, and if so, why?

Example of Topic Text:
Page: 38 (with Mod 2.2-1)
Location:"DETERMINE ENGINE TEMPERATURE"
Q: Why should the engine temperature be set in AECLP?

Reply Text

The following items may be included in the reply text. The bolded items are literals that should appear in the text. The "RE:" entry should appear first. The statements, questions, and answers may appear in any order, but each should start a new line.

RE:<Note-range> (required) (Note-range is the range of note(s) to which this reply is a response. If the note numbers are not contiguous, then list several ranges separated by commas.)

S: <statement> (optional) (to be used if this part of the text is merely a comment or statement)

Q: <question> (optional) (to be used if this part of the text is a question)

A: <answer> (optional) (to be used if this part of the text is an answer to a previous question)

Example of Reply Text:
RE: 12.1
S: The answer in note 12.1 is logical as far as it goes.
Q: It leaves unanswered the following question: Why is temperature calculated before calculating limiting errors?

- **Keywords**

Topic Keywords

The keyword will be the literal "v" followed immediately by the specification version number, which is the actual version number appearing on the title page of the specification to which this question (or modification) applies.

Examples of valid topic keywords:
v2.2
V2.2

At the present time, keywords will not be needed on replies.

B.11.3 Optional Notification From Within VAX Notes Using MAIL Utility

It should be noted that VAX Notes does not immediately notify a member of a conference when a note has been written into the conference. If the programmer wishes the system analyst to be notified immediately, the "FORWARD" command from within VAX Notes can be used to send a copy of the new note, with an optional preface, to the system analyst, or alternatively the "SEND" command from within VAX Notes can be used to send a notification to the system analyst that a new note has been written to the conference (See *Guide to VAX Notes*, Sections 3.5 and 3.6)

B.11.4 Using Text Files for Note Creation

There does not appear to be a way to change the text of a note once the note has been entered into the conference. For that reason, it may be helpful when writing notes to first write the text of the note into a text file using an editor in order to verify that it is correct. Then from within VAX Notes, the note can be written into the conference from the text file. For example:

```
From VMS:
    $edit note.txt  (create the text for the note)

Then, from VAX Notes:
    Notes>write note.txt ,or
    Notes>reply note.txt
```

Figure B.9 shows an example conversation that might take place between a programmer and the system analyst using VAX Notes. Then, Figure B.10 shows the directory that would result from the conversation shown in Figure B.9.

```
=============================================================================
Note 1.0      /AECLP/Q: What does "to the nearest integer" mean?     4 replies
AIR19::PG "Programmer"                        5 lines  31-DEC-1992 13:02
-------------------------------------------------------------------

Page: 41

Location: "COMPUTE AXIAL ENGINE VALVE SETTINGS", last sentence.

S: The specification uses the phrase "to the nearest integer".

Q: What exactly is meant by the phrase "to the nearest integer" ?
=============================================================================
Note 1.1      /AECLP/Q: What does "to the nearest integer" mean?        1 of 4
AIR19::SA "system analyst"                       7 lines  31-DEC-1992 15:43
              -< A: Definition of "to the nearest integer" >-
-------------------------------------------------------------------------------

RE: 1.0
A: The phrase "to the nearest integer" means the following:
    If the fractional part of the real number is  less than .5, then
    set AE_CMD to the closest integer which is less than the real number;
    if the fractional part of the real number is greater than .5, then
    set AE_CMD to the closest integer which is greater than the real number.
=============================================================================
Note 1.2      /AECLP/Q: What does "to the nearest integer" mean?        2 of 4
AIR19::"PG "Programmer"                           4 lines  31-DEC-1992 16:24
                   -< Q: What if real number = .5? >-
-------------------------------------------------------------------------------

RE: 1.1

Q: What should be done in the case where the fractional part of the real number is exactly equal
to .5?

=============================================================================
Note 1.3      /AECLP/Q: What does "to the nearest integer" mean?        3 of 4
AIR19::SA "system analyst"                       6 lines  31-DEC-1992 16:27
           -< A:Method for treating case where real number = .5 >-
-------------------------------------------------------------------------------

RE: 1.2
A:  If the fractional part of the real number is exactly equal to .5,
    then treat it as equivalent to the case where the fractional part
    of the real number is greater than .5.
=============================================================================
Note 1.4      /AECLP/Q: What does "to the nearest integer" mean?        4 of 4
AIR19::SA "system analyst"                       4 lines  31-DEC-1992 16:30
                -< S: Support Documentation Change Report will be issued >-
-------------------------------------------------------------------------------

RE: 1.0 - 1.3
S:  A Support Documentation Change Report for the Specification will be issued to fully resolve
this issue.
```

Figure B.9. Example of a Conversation Between the Programmer (PG) and System Analyst(SA)

```
                              conversation-examples
Created: 31-DEC-1992 12:58           1 topic        Updated: 31-DEC-1992 16:30
Topic     Author          Date            Reply Title
          AIR19::PG       31-DEC-1992     4  /AECLP/Q: What does "to the nearest integer"
                                          mean?
          AIR19::SA       31-DEC-1992     1.1  A: Definition of "to the nearest integer"
          AIR19::PG       31-DEC-1992     1.2  Q: What if real number = .5?
          AIR19::SA       31-DEC-1992     1.3  A: Method for treating case where real number =
                                          .5
          AIR19::SA       31-DEC-1992     1.4  S: Support Documentation Change Report will be
                                          issued
```

Figure B.10. Directory of All Notes in the Conversation Example

B.12 Documentation Guidelines

As shown in the list, found in Table B.1, of DO-178B life cycle data that will be produced as part of this project, each participant in the project is responsible for some portion of the data. This chapter gives some minimal guidance in the preparation of documentation associated with the GCS project. Since many of the configuration items that make up the support documentation will refer to each other and will be in an evolutionary process at the beginning of the project, it is important to use a common set of labels for all of the configuration items. The appropriate labels for the configuration items are given in Table B.2. For those items that are implementation specific, the labels should refer to the appropriate implementation when appropriate. The configuration item labels given in Table B.2 will serve as the titles for the project documentation.

In general, the contents of all support documentation must follow the descriptions given in Section 11 of the DO-178B guidelines, where applicable to the GCS project. The support documents should be formatted in accordance with the standards for NASA technical publications (ref. B.15). All of the support documentation for this project should also contain the same preface, as given in the beginning of this document, to provide a common background statement for the documents. Furthermore, the electronic versions of the project's support documentation that are stored in the CMS libraries will be produced using Microsoft Word (ref. B.16). See the *Software Life Cycle Environment Configuration Index* for more information on the word processing tools used on the GCS project.

The support documentation and development products will evolve as the project progresses. As discussed in Subsection 4.2 of DO-178B, all support documentation will be completed prior to that point in time in the software life cycle necessary to provide timely direction to the personnel performing the software development and integral processes; e.g. all support materials for conducting a design review (including the design standards, description of the design review, review procedures, checklists, traceability matrix, problem and action reporting procedures and forms, and the configuration management and SQA guidelines) must be in place prior to conducting a design review. The SQA representative is responsible for assuring that all plans and necessary materials are developed and reviewed for consistency at the appropriate phases of the development process, as per Subsection 8.2 of DO-178B. The project leader must also review and approve all support documentation.

B.13 Effort Data

The following are the effort data forms and the specific instructions for completing the forms for each of the significant participation roles in the GCS project. The programmers, verification analysts, SQA representative, configuration manager, and system analyst are required to record their effort. The general project policy for collecting effort data is given in the chapter titled "Collecting Effort Data". Copies of the effort data forms will be given to the project participants at the start of the project.

B.13.1 Instructions to the Programmers for Recording Effort

This section provides specific instructions to the programmers for recording the amount of effort exerted for each of the activities listed on the effort data form for the programmers. The effort data form for the programmers is shown in Figure B.11. The general programmer activities as listed on the form are given below, followed by a statement that details the specific activities for which effort should be accounted.

1. Changing Design during Transitional Design Phase: record time spent reading and understanding version 2.2 of the GCS specification, learning about teamwork, making modifications to the teamwork design (generated at RTI) to bring it up to version 2.2, and preparing the design description. This will include most of the time spent on the GCS project prior to the first Design Review.
2. Developing Source Code: record time spent developing source code to meet the detailed design description. This will include all time spent generating the source code until the time of the first Code Review.
3. Participating in Design Reviews and Code Reviews: record all time spent preparing for the reviews and attending the reviews. Preparation time includes time spent at the Overview meeting for the Design Review and any time spent inspecting the design or code in anticipation of a review. If a Design or Code Review is conducted in response to a modification to the specification, place an * by the hours indicated.
4. Changing Design due to: record time spent making modifications to the detailed design description in response to a Problem Report issued during one of the particular development phases listed. Problem Reports for the design will not be issued until the first Design Review.
5. Changing Code due to: record time spent for making modifications to the software code in response to a Problem Report issued during one of the particular development phases listed. Problem Reports for the code will not be issued until the first Code Review.
6. Responding to Modifications to the Requirements: record time spent reading and understanding the Support Documentation Change Reports for the GCS specification and making changes to the design or code due to modifications to the GCS specification. Effort should be recorded in this category only after the first Design Review.

NAME:____________________				WEEK:_____________________			
Effort Hours for Programmer Activities							
	WEEKDAY						
Programmer Activities	Su	M	T	W	H	F	Total
1. Changing Design during Transitional Design Phase							
2. Developing Source Code							
3. Participating in Design Reviews							
Code Reviews							
4. Changing Design due to: Design Review							
Code Review							
Unit Test (functional)							
Unit Test (structural)							
Subframe Test							
Frame Test							
Top-Level Simulator Integration Test							
5. Changing Code due to: Code Review							
Unit Test (functional)							
Unit Test (structural)							
Subframe Test							
Frame Test							
Top-Level Simulator Integration Test							
6. Responding to Modifications to the Requirements Change to Design							
Change to Code							

Figure B.11. Form for Recording Effort Data from Programmers

B.13.2 Instructions to the Verification Analysts for Recording Effort

This section provides specific instructions to the verification analysts for recording the amount of effort exerted for each of the verification activities listed on the effort data form. Figure B.12 shows the form that the verification analysts will use to record their effort data. The general verification activities as listed on the form are given below, followed by a statement that details the specific activities for which effort should be accounted.

1. **Developing Verification Plans, Procedures, and Tools**: record time spent developing and documenting the verification plans and procedures and tools (such as checklists, traceability data, test cases, test drivers, etc.) during each of the development phases. Note that effort recorded under the category Transitional Design Phase should include time spent understanding version 2.2 of the GCS specification, learning about aspects of software verification, and establishing procedures and tools for the initial verification activities. In addition, effort in the Transitional Design Phase category will include time spent establishing and documenting the traceability data and matrix, and the Design Review procedures and checklists.
2. **Participating in Verification Activities**: record all time spent doing the verification activities defined for each of the development phases. This time should include time spent preparing for the reviews (including attendance to the Overview meeting for the Design Review and inspecting a design or code in anticipation of a review), attending the reviews, running test cases, writing Problem Reports when necessary, and re-executing test cases to determine if a problem is resolved during each of the development phases.
3. **Responding to Modifications to the Requirements**: record time spent making changes to any verification plans, procedures or tools, or conducting a verification activity (such as re-executing test cases or attending a new Design Review) due to Support Documentation Change Reports for the GCS specification. Effort should be recorded for this activity only after the first Design Review. Effort should be recorded in the development phase where the changes are made. For example, if a test case used in the functional part of the unit testing needs to be changed in response to a modification, the effort hours should be recorded in the category "Unit Test Phase Functional." If the change relates more to a general verification procedure or tool (such as the traceability matrix), record the effort hours in the current development phase.

NAME:____________________ WEEK:___________________
___

Effort Hours for Verification Analyst Activities

	WEEKDAY						
Verification Analyst Activities	Su	M	T	W	H	F	Total
1. Developing Plans, Procedures, and Tools Transitional Design Phase							
Coding Phase							
Unit Test Phase: Functional							
Structural							
Subframe Test Phase							
Frame Test Phase							
Top-Level Simulator Integration Test Phase							
2. Configuring Life Cycle Data for: Transitional Design Phase							
Coding Phase							
Unit Test Phase: Functional							
Structural							
Subframe Test Phase							
Frame Test Phase							
Top-Level Simulator Integration Test Phase							
3. Responding to Modifications to the Requirements Transitional Design Phase							
Coding Phase							
Unit Test Phase: Functional							
Structural							
Subframe Test Phase							
Frame Test Phase							
Top-Level Simulator Integration Test Phase							

Figure B.12. Form for Recording Effort Data from Verification Analysts

B.13.3 Instructions to the SQA Representative for Recording Effort

This section provides specific instructions to the SQA representative for recording the amount of effort exerted for each of the SQA activities listed on the effort data form. The effort form is shown in Figure B.13. Since there is only one person assigned to provide the SQA services for the GCS project, the primary SQA activities (conducting reviews and tracking Problem Reports) are separated on the form for each of the GCS implementations, Mercury and Pluto. Since the SQA procedures and Support Documentation Change Reports for the GCS specification are common among the implementations, those categories for recording that effort are not separated according to implementation. The general SQA activities as listed on the form are given below, followed by a statement that details the specific activities for which effort should be accounted.

1. **Developing Plans, Procedures, and Tools**: record time spent developing and documenting the SQA plans and procedures and tools (such as the master logs for tracking the Problem Reports) in accordance with the DO-178B guidelines for the GCS project.

2. **Participating in Reviews:** record time spent preparing for, attending, and generating the SQA report for reviews conducted in each of the development phases for each of the GCS implementations. Preparation time includes time spent preparing for and conducting the Overview meeting for the Design Reviews. If a review is conducted in response to a Support Documentation Change Reports for the GCS specification, place an * by the hours indicated.

3. **Reviewing Problem Reports**: record time spent reviewing, assigning identification numbers to, distributing and tracking, and logging the Problem Reports during each of the development phases for each of the GCS implementations. For time spent reviewing Problem Reports that resulted from Support Documentation Change Reports for the GCS specification, place an * by the hours.

4. **Conducting Audits**: record time spent preparing for, conducting, and recording the results of audits for each of the GCS implementations.

5. **Reviewing Modifications to the Requirements**: record time spent reviewing Support Documentation Change Reports for the GCS specification. Effort should be recorded in this category only after the first Design Review.

NAME:____________________ WEEK:____________________

Effort Hours for Software Quality Assurance Activities

		WEEKDAY						
Software Quality Assurance Activities		Su	M	T	W	H	F	Total
1. Developing Plans, Procedures, and Tools								
<u>Mercury</u> Design:	2. Review							
	3. Problem Reports							
Code:	2. Review							
	3. Problem Reports							
Unit:	2. Review							
	3. Problem Reports							
Subframe:	2. Review							
	3. Problem Reports							
Frame:	2. Review							
	3. Problem Reports							
Top-Level	2. Review							
Simulator Integration	3. Problem Reports							
4. Audits								
<u>Pluto</u> Design:	2. Review							
	3. Problem Reports							
Code:	2. Review							
	3. Problem Reports							
Unit:	2. Review							
	3. Problem Reports							
Subframe:	2. Review							
	3. Problem Reports							
Frame:	2. Review							
	3. Problem Reports							
Top-Level	2. Review							
Simulator Integration	3. Problem Reports							
4. Audits								
5. Reviewing Modifications to the Requirements								

Figure B.13. Form for Recording Effort Data from the SQA Representative

B.13.4 Instructions to the Configuration Manager for Recording Effort

This section provides specific instructions to the configuration manager for recording the amount of effort exerted for each of the configuration management activities listed on the effort data form. Figure B.14 shows the effort form for the configuration manager. Since there is only one person assigned to provide the configuration management services for the GCS project, some of the configuration management activities have been separated on the form for each of the GCS implementations, Mercury and Pluto. The general configuration management activities as listed on the form are given below, followed by a statement that details the specific activities for which effort should be accounted.

1. **Developing Plans, Procedures, and Tools**: record time spent developing and documenting the configuration management plans and procedures and tools (such as creating the CMS libraries for the project's life cycle data) in accordance with the DO-178B guidelines for the GCS project. Effort involved in learning about configuration management practices and CMS should also be included here.

2. **Configuring life cycle data for**: record time spent performing the configuration management activities, such as reserving, replacing, and fetching GCS elements or baselining, for each of the GCS implementations, differentiating between the programmer and verification analyst for each implementation. Also record time, in the category "General Project", for time spent providing configuration management for those aspects of the project that are common to all implementations, including the primary planning documents (*Plan for Software Aspects of Certification*, *Software Verification Plan*, *Software Configuration Management Plan*, and *Software Quality Assurance Plan*).

NAME:____________________				WEEK:____________________			
Effort Hours for Configuration Management Activities							
	WEEKDAY						
Configuration Management Activities	Su	M	T	W	H	F	Total
1. Developing Plans, Procedures, and Tools							
2. Configuring Life Cycle Data for: Mercury Programmer							
Mercury Verification Analyst							
Pluto Programmer							
Pluto Verification Analyst							
General Project							

Figure B.14. Form for Recording Effort Data from the Configuration Manager

B.13.5 Instructions to the System Analyst for Recording Effort

This section provides specific instructions to the system analyst for recording the amount of effort exerted for each of the system analyst activities listed on the effort data form. Figure B.15 shows the effort form for the system analyst. In general, the system analyst is responsible for the definition and maintenance of the software requirements for the project. The activities for the system analyst as listed on the form are given below, followed by a statement that details the specific activities for which effort should be accounted.

1. **Maintaining the GCS Specification**: record time spent reviewing the GCS specification for correctness and completeness and issuing any Support Documentation Change Reports that are deemed necessary.
2. **Consulting for**: record time spent responding to questions about the GCS specification from the programmers and verification analysts for each of the GCS implementations.
3. **Participating in Reviews for**: record time spent preparing for and attending the Design and Code reviews for each of the GCS implementations. Preparation time includes time spent at the Overview meeting for the Design Review and any time spent inspecting the design or code in anticipation of a review. If the Design or Code Reviews are held in response to a Support Documentation Change Report, place an * by the hours indicated.

NAME:____________________			WEEK:_______________________				
Effort Hours for System Analyst Activities							
	WEEKDAY						
System Analyst Activities	Su	M	T	W	H	F	Total
1. Maintaining the GCS Specification							
2. Consulting for: Mercury							
Pluto							
3. Participating in Reviews for: Mercury							
Pluto							

Figure B.15. Form for Recording Effort from the System Analyst

B.14 References

B.1. Finelli, George B.: Results of Software Error-Data Experiments. In *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference*, Atlanta, GA, September 1988.

B.2. RTCA Special Committee 167. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178B, Requirements and Technical Concepts for Aviation, Dec. 1992.

B.3. RTCA Special Committee 152. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178A, Radio Technical Commission for Aeronautics, March 1985.

B.4. Hatley, Derek J.; and Pirbhai, Imtiaz A.: *Strategies for Real-Time System Specification.* Dorset House Publishing Company, New York, New York, 1987.

B.5. Shagnea, Anita M.; and Dunham, Janet R.: GCS Development Specification Review Description. Technical Report, Research Triangle Institute, Research Triangle Park, NC 27709, 1989. Prepared under NASA Contract NAS1-17964; Task Assignment No. 8.

B.6. Team*work/*SA Team*work/*RT User's Guide. Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1991.

B.7. DeMarco, Tom: *Structured Analysis and System Specification.* YOURDON Inc., 1133 Avenue of the Americas, New York, NY 10036, 1978.

B.8. Ward, Paul; and Mellor, Steven: *Structured Development for Real-Time Systems.* Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1985.

B.9. Team*work/*SD User's Guide. Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1991.

B.10. Guide to VAX Notes. Digital Equipment Corporation, Maynard, Massachusetts, March 1986.

B.11. *Guide to VAX DEC/Code Management System*. Digital Equipment Corporation, Maynard, Massachusetts, April 1987.

B.12. Programming in VAX FORTRAN. Digital Equipment Corporation, Maynard, Massachusetts, September 1984.

B.13. Roberts, Alan; Rich, Don; and Pierce, John: *Internal Document : VMS FORTRAN Code Generation Guidelines.* Software R & D Department, Center for Digital Systems Research, Research Triangle Institute, Research Triangle Park, NC, June 1986.

B.14. *Guide to VAX DEC/Module Management System*. Digital Equipment Corporation, Maynard, Massachusetts, April 1987.

B.15. NASA Publications Manual. NASA SP-7013, 1974.

B.16. Microsoft Word User's Guide. Microsoft Corporation. 1991.

Appendix C: Software Verification Plan for the Guidance and Control Software Project

Authors: Patrick Quach, Lockheed Martin Engineering and Sciences Corp.
Debbie Taylor, Computer Sciences Corp.

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

C. Contents

C.1 Introduction

This document is the *Software Verification Plan* for the Guidance and Control Software (GCS) project. The plan details verification activities to be conducted to satisfy DO-178B criteria for software verification and discusses issues important to the verification process. The procedures and processes will be applied to all implementations of GCS.

Software verification activities serve as "filters" for the development process. These activities are independent from the development process, but effect the development process because each step of the development processes that produces an artifact will initiate a verification activity. As defined in Paragraph 6 of DO-178B (ref. C.2), software verification is a technical assessment of the results of the software development process as well as an assessment of the verification process. Two methods for verification addressed in this document are reviews/analysis, and testing. Other activities that support these methods will also be addressed as well as issues and considerations involved.

The following is a brief description of topics covered in this document. The Overview of Verification Activities section describes the composition and organization of each implementation team as well as the independence established for verification purposes. This is followed by the Verification Methods section which outlines the review procedures for the design and code artifacts. The following section describes the test coverage and testing strategy required by DO-178B and to be used to verify executables against the GCS Specification. The GCS Specification will serve as the *Software Requirements Data* for this project. The Verification Environment section briefly describes the tools to be used during the verification process and the programs developed to support verification activities. The hardware platform on which those activities will be carried out will also be described. The Transition Criteria section gives objectives to be met for each step in the verification process before proceeding to the next step. Lastly, the Reverification Guidelines describe procedures to be followed to verify an artifact after a correction has been made to the artifact.

C.2 Overview of Verification Activities

As part of the GCS project, there are two teams working independently on two implementations of the GCS. Each team consists of a Programmer/Designer and a Verification Analyst. Each Programmer/Designer has the duty of deriving a design from the GCS Specification and translating the design into source code. Each Verification Analyst will check both the design and the source code to ensure that both meet all the requirements in the GCS Specification as well as all the requirements for verifiability set out by DO-178B. As illustrated in Figure C.1, the Programmer/Designer is charged with producing the artifacts (with the exception of the GCS specification) that initiates the verification activities; while the Verification Analyst is charged with applying the verification activity to those artifacts until the transition criteria has been meet.

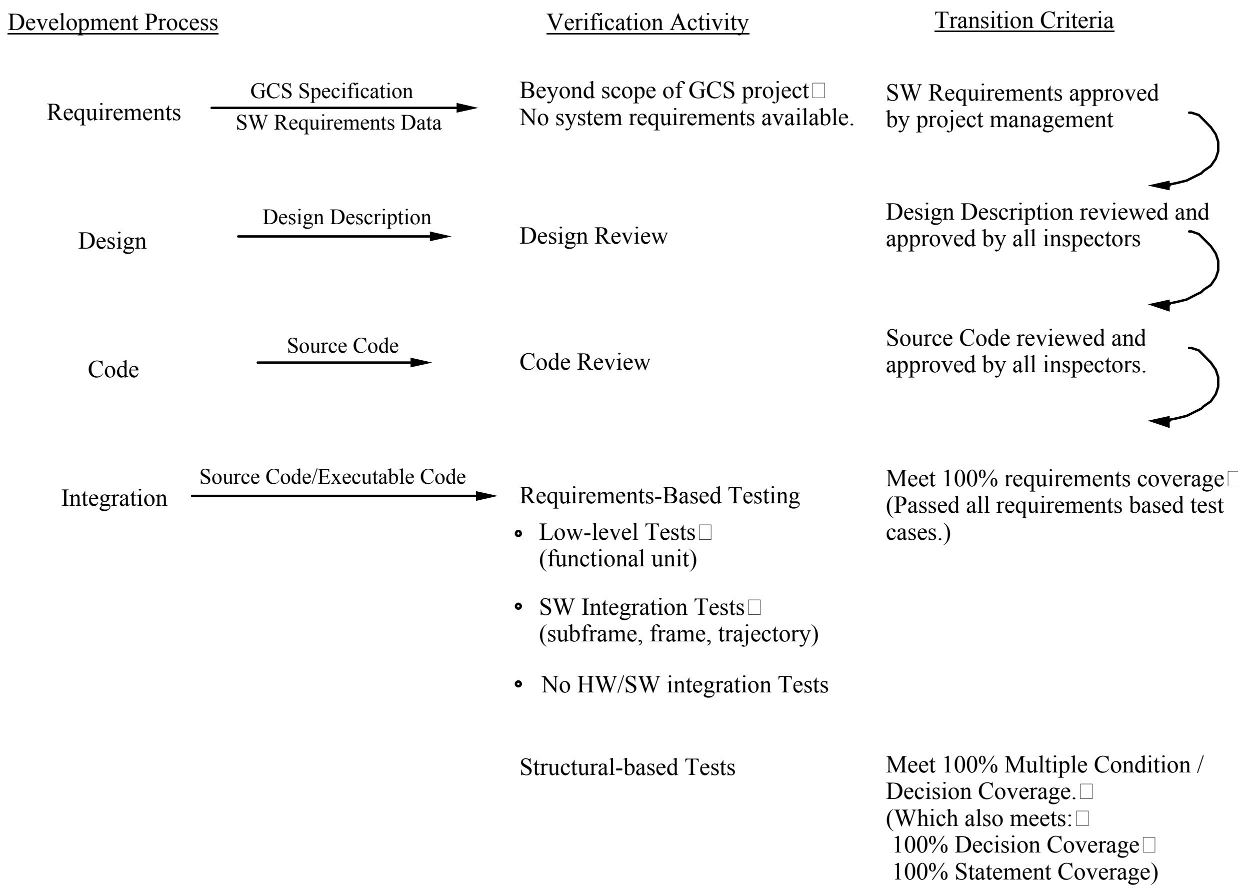

Development Process
Verification Activity
Transition Criteria
Requirements
GCS Specification
SW Requirements Data
Beyond scope of GCS project
No system requirements available.
SW Requirements approved by project management
Design
Design Description
Design Review
Design Description reviewed and approved by all inspectors
Code
Source Code
Code Review
Source Code reviewed and approved by all inspectors.
Integration
Source Code/Executable Code
Requirements-Based Testing
Low-level Tests (functional unit)
SW Integration Tests (subframe, frame, trajectory)
No HW/SW integration Tests
Meet 100% requirements coverage (Passed all requirements based test cases.)
Structural-based Tests
Meet 100% Multiple Condition / Decision Coverage.
(Which also meets:
100% Decision Coverage
100% Statement Coverage)

Figure C.1: Overview of verification activities.

For each implementation in the GCS project, verification activities begin after the initial design is completed as indicated in Figure C.1. When a design is completed initially, a review will be held to verify that it satisfies the requirements. Source code will only be written after completion of the design review. Note that the process of generating the source code is part of the software development processes and not part of the verification process. Once the source code is completed it is reviewed for consistency with the design and the GCS Specification. When this is complete, testing can begin on the executables generated by the source code. Testing is deemed complete when the test completion criteria, as described below, are met.

As described above, the verification activities will be coordinated with the software development processes. For each GCS implementation, however, verification activities are performed by a verification analyst not involved with the software development processes. This establishes independence between the software development and verification activities, which according to Myers (ref. C.3) are activities with conflicting objectives.

C.3 Verification Methods

This section describes the verification activities to be conducted during each phase of the development life cycle as well as the compliance documents produced from those activities. Some documents will be used, as specified in the DO-178B, to establish the traceability of requirements from the design process to actual test output. Verification activities for the GCS project include:

- Review and analysis of artifacts from the Design and Code processes
- Test case development and execution

The ultimate purpose of the software verification process is to detect and report errors that may have been introduced in the development processes. Although the removal of these errors is an activity of the software development processes, it is within the scope of the verification process to ensure that corrections are made and that no additional errors are introduced during the correction process. Software verification has several general objectives. These are:

- To ensure that the high-level requirements, hereafter referred to as the GCS Specification (ref. C.2), have been correctly developed into software architecture and low-level requirements.
- To ensure that the software architecture and low-level requirements have been correctly developed into Source Code that satisfies the low-level requirements and software architecture.
- To ensure that the Executable Object Code satisfies the software requirements.
- To ensure that the methods and approaches used to satisfy these objectives are technically correct and complete.

The review and analysis and testing activities will be carried out to satisfy the four verification goals. First, formal reviews will be conducted on the artifacts of the design and coding processes to ensure that they reflect the requirements of the respective development processes. The review procedure for the GCS project is described in the Verification Cases and Procedures document and is developed based on guidelines in DO-178B. Both the Design and Code will undergo this review procedure. The emphasis of each review will be discussed in the Review and Analysis Activities section.

The second major activity of verification process is test case development and execution. Testing according to DO-178B Paragraph 6.4 should demonstrate that the software satisfies its requirements and that all errors which could lead to unacceptable failure conditions have been removed. Additionally, enough test cases must be created so that the coverage criteria given in DO-178B is satisfied. A test strategy for GCS must accommodate low-level tests which verify that code segments are correctly implemented and high-level tests which verify that major functions of the requirements are met. The software for each GCS implementation will undergo testing to satisfy the criteria outlined in the Testing Activity section. This ensures that the executable modules satisfy the GCS Specification for functionality. Test case development and test execution strategy will be described in the Test Activities section in this document. The procedures for test case execution are in *Software Verification Cases and Procedures* .

C.4 Review and Analysis Activities

Because the GCS Specification is the highest level of system specification available and, for project purposes, assumed to be correct, the verification process will begin with verification of the design. This section gives an overview of the review and analysis to be conducted for each implementation. The same procedures will be used for the review and analysis of both the Design and Code, the difference being the artifact of the review. The actual procedures and responsibilities of participants are given in the Review Procedure in *Software Verification Cases and Procedures* .

C.4.1 Design Review Overview

According to Paragraph 5.2 of DO-178B, the purpose of the software design process is to refine the software high-level requirements into a software architecture and the low-level requirements that can be used to implement the source code. The software verification process demands that the artifact produced by the software design process, the detailed design, be reviewed to confirm that it fulfills the purpose stated above. The GCS Specification embodies the software high-level requirements as well as some level of software design. Thus, some of the necessary refinements of the software requirements have already been accomplished in the GCS Specification.

Paragraph 6.3.2 of DO-178B specifically states that the detailed software design should be reviewed and analyzed to identify and report errors that may be introduced during the software design process. The detailed design for each implementation will be subjected to an independent Design Review. In each case, the review will be conducted to satisfy the review and analysis objectives in DO-178B Paragraph 6.3. These objectives are further delineated below.

The first objective is to ensure that the software low-level requirements satisfy the high-level requirements and any derived requirements are correctly defined. The reviewers should ensure that the detailed design conforms to the requirements in the GCS Specification. That is, the detailed design should complete the design that has been expressed in the GCS Specification and reflect all of the high-level requirements and derived requirements. The design should address exactly what needs to be accomplished in order to fulfill the requirements stated in the GCS Specification. If each of the high- level requirements has been decomposed into its respective low-level requirements, then the low-level requirements should be directly translatable into source code.

Another objective expressed in DO-178B is that the low-level requirements and the design be accurate, unambiguous and non-conflicting. To satisfy this goal, reviewers should ensure that all annotations used in the design are clearly explained, the explanation of algorithms are concise, and all processing steps are listed in the order that they should occur.

In addition, reviewers should be alert for features of the design that presuppose hardware functionality that is actually not present in the target hardware. This should not be a major concern for the GCS project because the GCS software is targeted to run in coordination with a simulator that runs on the same computer as the implementation. Reviewers do, however, have to lookout for design features that cannot be implemented on the target computer or features that the compiler lacks.

The Design Review should also verify that the detailed design for each implementation complies with the Design Standards. The Design Standards require that the structured analysis and design methods described by Hatley and Pirbhai (ref. C.4), DeMarco (ref. C.5), or Ward and Mellor (ref. C.6) be used, and that the Computer Aided Software Engineering (CASE) tool, team*work* (ref. C.7), be used to develop the detailed design. See the *Software Development Standards* for more information on the Design Standards.

The Design Review should ensure that features in the design can be verified and that they are traceable back to the requirements document. This objective can be satisfied by ensuring that all features in the design are testable. Traceability of the required features from the GCS Specification to each GCS design can be established by using a Traceability Matrix for each implementation. The matrix will map each requirement in the specification to the corresponding features in the design.

Lastly, according to DO-178B, the reviewers must ensure that the algorithms described in the design perform what is required in the specification. Details about numerical accuracy in the design should be verified against those in the GCS Specification.

C.4.2 Code Review Overview

According to DO-178B, the purpose of the software code process is to produce source code that is traceable, verifiable and consistent. The source code generated during this process should comply with the Software Code Standards, generate an error free compilation and be traceable to the Design. The software verification process demands that the artifact produced by the software code process, the source code, be reviewed to confirm that it fulfills its purpose.

Paragraph 6.3.4 of DO-178B specifically states that the source code should be reviewed and analyzed to identify and report errors that may be introduced during the software coding process. For each implementation, there will be a formal Code Review to ensure that the objectives set forth in DO-178B are met by the source code.

One of the primary objectives of each GCS Software Code Review is to ensure that the source code is accurate and complete with respect to the low-level requirements identified in the Detailed Design. Reviewers should ensure that all required features in the design are implemented. This includes all of the original requirements as well as all of the derived requirements identified in the design review. Care must be taken to ensure that the source code contains no extraneous functions that are undocumented in the GCS Specification or the specific design.

The second objective for code reviewers, according to DO-178B, is to ensure that all the code matches the data and control flow defined by the software design. This includes extra or missing flows into functional units. The reviewers should also verify the execution sequence of all the modules to see if the control flow specified in the design is adhered to.

DO-178B also calls for verification of testability of source code. Reviewers should ensure that all of the source code in each implementation is testable without modification. For GCS implementations, this can be satisfied by verifying that modules can be tested individually or collectively. There should also be no statements that need to be altered in order to test the code.

The Code Review should also verify that the source code complies with the Software Code Standards. The Software Code Standards state that the source code should be written in VAX/VMS FORTRAN, using structured programming techniques. A formal header system to document changes is also required. See the *Software Development Standards* for more information on the Code Standards.

The Code Review will ensure that the software architecture and low-level requirements defined by the detailed design and the GCS Specification are traceable to the source code. This will be done by completing the Traceability Matrix for the source code. The Traceability Matrix should identify the part of the code that implements each requirement. Completing the Traceability Matrix also establishes that all requirements have been implemented.

The correctness and consistency of the source code will also be verified during the Code Review. This includes such things as proper handling of exceptions, identification of unused, un-initialized or undefined variables or constants, and other potential problems.

C.5 Testing Activities

This section will give a brief overview of the GCS testing activities. Specific test cases and procedures will be detailed in *Software Verification Cases and Procedures.* Testing activities will be modeled similar to Figure 6.1 of DO-178B -- with some project specific adjustments. DO-178B recommends that a multi-level testing plan be implemented to achieve complete requirement and structural coverage objectives. For this reason, GCS test cases are developed to cover the GCS Specification and the implementation code. Both test case development and execution strategy will be summarized here with detailed application discussion in their respective sections.

Before actual testing can commence, test cases must first be created. DO-178B stresses that test cases should be requirements-based because such test cases are the most effective for error detection. Additionally, DO-178B stresses in Paragraph 6.4.2 that the requirements-based test cases should have two categories; namely, normal range and robustness cases. Hence, requirements-based test cases will ensure that all requirements in the GCS Specification will be tested. For completeness, DO-178B also stresses the need for test cases based on structural coverage. These cases are derived from a specific code structure and test parts of the code flow that are not exercised by the requirements-based test cases.

When a GCS implementation is ready for testing, the Verification Analyst will test it at two levels deviating from the three levels recommended by DO-178B. The two levels, at which GCS implementations will be tested are

- Low-level Testing
- Integration Testing

Low-level testing entails execution of test cases derived from the software low-level requirements and structural coverage. Testing at this level will focus on executing the source code corresponding to the eleven functional units in the GCS Specification:

Axial Engine Control Law Processing	AECLP
Altimeter Radar Sensor Processing	ARSP
Accelerometer Sensor Processing	ASP
Communications Processing	CP
Chute Release Control Processing	CRCP
Guidance Processing	GP
Gyroscope Sensor Processing	GSP
Roll Engine Control Law Processing	RECLP
Touch Down Landing Radar Sensor Processing	TDLRSP
Touch Down Sensor Processing	TDSP
Temperature Sensor Processing	TSP

There will be a unique set of requirements-based test cases for each functional unit. Equivalence class partitioning and boundary value analysis will be used to determine the test cases. These cases test the valid and invalid equivalence classes as described below. These tests verify that the low-level requirements have been correctly implemented in the code.

Integration testing for the GCS implementations will be carried out in several phases. The first integration phase will occur at the subframe level. At this phase, test cases will treat each of the three subframes defined in the GCS Specification (Sensor Processing, Guidance Processing, and Control Law Processing) as aggregate units. The next phase of integration testing combines the subframes as an integral frame and tests only flows in and out of the frame. The final phase of integration testing involves integrating an implementation with the GCS Simulator. For this phase, test cases will exercise the implementation through complete trajectories. More details will be provided in the respective sections.

DO-178B additionally calls for testing at the hardware/software integration level. Because no hardware is involved in the GCS project except for the computer that will host the GCS Simulator and the GCS implementations, this type of integration testing will not be conducted. The GCS Specification only requires each implementation to be compatible with the GCS Simulator. The final phase of testing will satisfy this requirements by executing each implementation with the simulator.

Once the integration testing is completed, each implementation will be analyzed for structural coverage. During the analysis, a graph of the structure of each source code module will be generated. Then the paths covered by the requirements-based test cases will be compared to the graphs to identify any remaining structure that needs to be exercised to meet the structural coverage requirement. Test cases can then be created for those paths not yet exercised by the requirements-based test cases.

C.5.1 Test Case Selection and Coverage

As previously stated, test cases will be created based on software requirements and software structure. This section describes how the requirements and structural coverage criteria for test case development will be achieved.

C.5.1.1 Requirements-Based Test Coverage

The overall objective of requirements-based testing is to show that the software will perform the specified requirements. It should be noted that since all GCS implementations are developed using the same requirements specification, there will only be one set of requirements-based test cases. That is, the efforts of the two Verification Analysts working on the GCS project will be pooled to develop a single suite of requirements-based test cases.

Paragraph 6.4.2 of DO-178B gives guidelines for accomplishing this test coverage by dividing the test suite into two groups - normal range and robustness test cases. Normal range test cases are used to demonstrate that the software responds correctly to normal input conditions. Robustness test cases are used to demonstrate that the software responds gracefully to abnormal or unspecified input conditions, where gracefully here is taken to mean that the software does not respond in a way that is detrimental to the survival of the spacecraft. The GCS Specification only requires the developers to flag these conditions and continue. Paragraph 6.4.2.1 of DO-178B gives 4 criteria for selecting *normal range* test cases. These include:

- cases exercising valid equivalence class and boundary values for real and integer variables
- cases that exercise multiple iterations of time-related functions
- cases that exercise valid state transitions
- cases that test logical branching

Paragraph 6.4.2.2 of DO-178B gives 7 criteria for selecting *robustness* test cases. These include:

- cases exercising invalid equivalence classes for real and integer variables
- cases that test protection mechanisms for exceeded frame times
- cases that invoke invalid state transitions
- cases which initialize the software under abnormal circumstances
- failure modes of incoming data
- cases which test out-of-range loop counters for loops with calculated loop counters
- cases that test overflow or underflow conditions in calculations

C.5.1.1.1 Equivalence Class Testing

The first criteria mentioned for selecting both normal range and robustness test cases is based on Myer's technique of Equivalence Class Partitioning (ref. C.3). Equivalence class partitioning is based on dividing the input domain of a programming unit into equivalent groups (or classes) so that the results from testing one value from a class will be representative of other values in that equivalence class. Myers asserts that any error revealed by one input value in the group will also be repeated for other members of the equivalent input class. Hence testing one value in the equivalent group will reveal all the errors for that group. This reduces the number of test cases necessary to test the entire input domain.

Equivalence partitioning is applied to variables in the GCS Specification that are used in a non-discrete fashion. These are variables for which the RANGE field in the data dictionary of the GCS Specification is given by a continuous range of values instead of single-point values. Note that for each data element in the data dictionary a RANGE is given as well as a DATA TYPE. The DATA TYPE defines all possible values that can be stored by the data element while the RANGE identifies the a subset of allowable values for the GCS implementations.

For each variable to be tested, all possible values of that variable are divided into logical ranges depending on the usage and allowable range of the variable as defined in the GCS Specification. The ranges contained in the allowable values as given in the data dictionary are considered to be the "valid" equivalence class. The remaining ranges make up the "invalid" equivalence for that variable. The equivalence classes identified for testing are listed in an equivalence class table in *Software Verification Cases and Procedures*. Note that the list does not include all variables in the four data stores defined in the GCS Specification. This is because not all data elements defined in the GCS Specification are used as non-discrete enumeration.

For example, the variable ATMOSPHERIC_TEMP is an input to the functional units ASP and GSP. It is listed in the data dictionary in the GCS Specification as an 8-byte real variable with valid (or useful) values ranging from -200.0 to 25.0. This range of useful values gives rise to three equivalence classes since the full range of any 8-byte real variable on the platform the code is to run on is from -1.7×10^{38} to 1.7×10^{38}. Namely, the equivalence classes for ATMOSPHERIC_TEMP are:

1)	-1.7×10^{38}	to	-200.0
2)	-200.0	to	25.0
3)	25.0	to	1.7×10^{38}

The second range of numbers is a class because any value between (-200, 25) for ATMOSPHERIC_TEMP are "treated" the same in both functional units. Further, it is the only "Valid" equivalence class because the behavior of the functional unit has been defined for

ATMOSPHERIC_TEMP values in that range. The other two ranges are considered "Invalid" equivalence classes because the behavior of the functional units for values in those ranges are not defined for this variable in the GCS software.

As categorized in Paragraph 6.4.2 of DO-178B, test cases using the valid equivalence classes correspond to the normal range test cases while the test cases using the invalid equivalence classes are the robustness test cases. For each functional unit, the equivalence classes of all variables in the input space are identified. Then enough test cases are needed to test all the equivalence classes in order to satisfy this test coverage. Myers suggests that only one invalid equivalence class should be tested per test case. However, valid equivalence classes of several variables can be combined in a single test case to expedite testing. Hence, in the previous example, a minimum of three test cases would be necessary to adequately test ATMOSPHERIC_TEMP in each functional unit (ASP and GSP), one for each invalid equivalence class (1 & 3) and one for the valid equivalence class (2). Of course, the latter one can be combined with the valid equivalence classes of several other variables in the same functional unit to expedite testing. To help account for all the equivalence classes tests, a table is created to match the equivalence class with all the test cases that test that equivalence class. The equivalence class table is given in *Software Verification Cases and Procedures*.

C.5.1.1.2 Multiple Iterations of Time-Related Functions

Normal range test cases that exercise multiple iterations of time-related functions will be tested in integration testing. These cases will exercise the GCS implementation through frames where critical events take place to see if the software responds properly. These cases are documented in the requirements Traceability Matrix.

C.5.1.1.3 Valid State Transition Testing

The Traceability Matrix (Section 9) documents the high- and low-level requirements. The valid state transitions are included as part of the low-level requirements and are given under the respective functional units in which the state transitions take place. Hence test cases that test valid state transitions are identified in the Traceability Matrix.

C.5.1.1.4 Logical Branching Testing

Paragraph 6.4.2.1 of DO-178B also specifies that normal range test cases should verify the variable usage and the Boolean operators used in logic equations. This coverage criteria is essentially the same as the Multiple Decision/Condition Coverage; and is discussed below as part of the structure-based testing. These test cases will be documented as part of the structure analysis of each implementation.

C.5.1.1.5 Invalid Equivalence Class Testing

Invalid Equivalence testing was discussed earlier in the section with Valid Equivalence Testing. Test cases will be generated to cover all invalid equivalence classes.

C.5.1.1.6 Protection Mechanisms for Exceeded Frame Time Testing

Cases which test protection mechanisms for exceeding frame times are not necessary for GCS, because there are no processing time requirements stipulated in the GCS Specification. Hence there are no time limits to test.

C.5.1.1.7 Invalid State Transition Testing

Invalid state transitions are not part of the traceability matrix because it only lists what the software is required to do and not what the software is NOT suppose to do. Invalid state transition tests also differ from invalid equivalence class tests in that invalid equivalence cases examine the response of a functional unit when given a variable with an out-of-range value. In contrast, invalid state transition cases test possible permutations of state variables that are not specifically addressed in the GCS Specification or provoke transition to an invalid state. The state transition tables in the GCS Specification give information about valid state transitions that the software is suppose to make. These tables do not provide a complete listing of possible input combinations. For completeness, test case listings given in *Software Verification Cases and Procedures* will include separate tables where applicable to test invalid state transitions.

C.5.1.1.8 Software Initialization under Abnormal Conditions Testing

DO-178B also calls for testing software initialization under abnormal conditions. This will not be done because the GCS Specification does not require software initialization to be performed in each implementation.

C.5.1.1.9 Failure Mode of Incoming Data Testing

Paragraph 6.4.2.2 of DO-178B also indicates a need to test the failure mode of incoming data. For GCS implementations, there are two possible interpretations of data failure mode. One interpretation is that data external to a GCS implementation, such as a sensor reading, is corrupt. The second is that data that is computed and passed between functional units is corrupt (e.g., zero or negative values to be divided or square-rooted). In the first case, the GCS Specification stipulates behavior for data failure modes by including various status variables and incorporating the status values in determining the behavior of the software. In this case, testing the failure mode of the incoming data becomes testing the valid and invalid equivalence classes of the status variables. In the second case where data is corrupted during computation, there are also provisions to print warning messages when conditions such as division-by-zero, and negative-square-roots occur. Additionally, the input space of each functional unit is tested by valid and invalid equivalence class tests. Hence, invalid equivalence class test cases also satisfy this coverage criteria.

C.5.1.1.10 Out-Of-Range Looping Testing

DO-178B Paragraph 6.4.2.2 indicates a need to test loops where the loop count is a computed value. If a given implementation has computed loop counters, test cases are generated and documented as robustness test cases in the test case listing of *Software Verification Cases and Procedures* . For loops that do not use calculated loop counters, testing one iteration through the loop will be sufficient since the loop counter is not manipulated to cause out of range conditions. During structural analysis of each implementation, looping decisions that do not involve calculated counters will not be tested.

C.5.1.1.11 Overflow and Underflow Testing

Finally, overflow and under flow conditions must also to be tested, according to DO-178B. For GCS, this involves testing the input space of each functional units with values for variables that are outside the range defined for the variables. These ranges are previously identified as "invalid" equivalence classes. Consequently, testing the invalid equivalence classes for each functional unit also constitutes testing the overflow and underflow conditions.

C.5.1.1.12 Summary

DO-178B specifies that requirements-based test cases are to be divided into two groups. The normal range tests must meet the criteria given in Paragraph 6.4.2.1 of DO-178B and verify that the software is delivering what is required in the GCS Specification. The robustness tests must meet the criteria given in Paragraph 6.4.2.2 of DO-178B and verify that the software does not cause catastrophic failure when encountering abnormal input conditions. For the GCS project, abnormal inputs will be tested to the point as required by the GCS Specification; that is, the GCS Specification dictates how certain exceptions are to be handled. Robustness testing will verify that those mechanisms operate correctly.

To ensure adequate requirements coverage, the verification analyst will review the Traceability Matrix to ensure that there are test cases identified with each requirement in the matrix. If test cases have not been identified for a requirement, then test case(s) must be devised to verify that the software meets that requirement. This step is considered complete when there are test cases associated with every requirement in the Traceability Matrix.

C.5.1.2 Structure-Based Testing

DO-178B also emphasizes structural testing. A set of coverage guidelines is given in DO-178B Table A-7 and deals mainly with structural testing criteria with considerations for the criticality of the software to the mission. The table also specifies whether the test coverage criteria should be satisfied with independence. That is, those individuals who verify the software must be independent from the developers. As stated in other GCS documents, each GCS implementation is considered to be Level A software because it is critical to the successful landing of the spacecraft. For Level A software, Table A-7 specifies that all coverage criteria be satisfied with independence. This has been achieved for each implementation because each implementation team has separate developers and verifiers. Table A-7 specifies that the following structural coverage requirements be achieved for each GCS implementation:

- modified condition/decision coverage
- decision coverage
- statement coverage
- data and control coupling coverage

The sections below will describe the test cases developed based on guidelines given in Paragraph 6.4.2 and Table A-7 of DO-178B. The discussions will focus on how each coverage category is satisfied and the procedure for documenting those test cases.

C.5.1.2.1 Modified Condition/Decision Coverage

The *Modified Condition/Decision Coverage* (MC/DC) requirement specifies that test cases satisfy four criteria. A test case must be derived to test each decision at every possible outcome. For those decisions with multiple conditions, test cases must cover all possible outcomes of each condition in the decision. Each condition must be shown to independently affect the decision outcome. Finally, test cases must invoke each entry and exit point in the software.

This coverage condition will be achieved in several steps for each functional unit in the GCS implementations with the aid of the Analysis of Complexity Tool (ACT). The ACT software (ref. C.8) is a tool developed by McCabe & Associates Inc. to perform complexity analysis on software. The tool can determine the cyclomatic complexity of a software product and provide a basis set of paths that reveal the flow of data and logic in the software. The ACT Software will be used to parse the GCS FORTRAN source code for each functional unit, to identify all the decisions made in the source code for the functional unit, and to graph the decision tree for the source code. For each functional unit, the verification analyst will create a decision table and identify a test case for all possible outcomes of each decision.

For each decision that has multiple conditions, a separate Pairs Table will be created. To clarify, a decision is generally thought of as a complete statement while the conditions are the subparts of the statement that are combined by logical operators such as AND, OR, etc. The Pairs Tables as described in (ref. C.9) list the conditions in the decision and give a test case for all possible outcomes of each condition. Note that this does not imply all possible permutations of the conditions in the decision. The Pairs Tables are also used to indicate the test cases that demonstrate the independent effect of each condition in the decision.

Test cases must be identified for all possible values of each condition. For multiple condition decisions, a separate test case is needed for each combination where any condition has an independent effect on the final decision. The test cases may be selected from the pool of requirements based test cases. If a test case does not yet exist, then one is created. Once the decision table and all pairs tables are complete, then the MC/DC requirement will have been met for a functional unit.

C.5.1.2.2 Decision Coverage

Satisfying the requirement for decision coverage entails that test cases drive each decision in the code to every possible outcome. Since this is already done to satisfy MC/DC, test cases that satisfy that coverage will also satisfy *Decision Coverage*. This is consistent with the Structure-based Test Type Matrix given in (ref. C.10) and reproduced in Section 10.

C.5.1.2.3 Statement Coverage

Statement coverage requires that every statement in a functional unit be executed at least once. This is accomplished by using the ACT software to identify the basis set of paths through the functional unit. According to Pressman (ref. C.11), test cases that traverse through the basis set of paths are guaranteed to execute every statement at least once. This coverage condition is also satisfied when MC/DC is achieved. Hence a subset of MC/DC test cases will also satisfy statement coverage.

C.5.1.2.4 Data and control coupling coverage

Coupling measures the interconnectedness between modules in a software design and indicates the varying degree of interdependence between modules. In the ideal case, the designer strives to minimize the coupling between modules to no more than is specified in the software requirements. However, the GCS Specification requires the use of data, control, and common coupling in the design of each implementation. This is largely due to the constraint that each implementation must run with the GCS simulator.

Common coupling is specified for passing data variables in the four global data stores. Control coupling is specified for passing status variables between modules. Data coupling is specified because variables such as atmospheric temperature and engine temperature are determined in one module and used in another module. To verify the correctness of common coupling, verifiers must first ensure that the global data stores are correctly set up. That is, the ordering of data stores is correct and the ordering of data elements in each data stores is also correct. This is one of the checks in the code review. Then test cases should verify that data written to any global data store does not violate the integrity of that data store. This can be accomplished by having the test driver check all data stores after each test run to verify that only data elements involved in a test case are changed and all other data elements in the global data stores are unchanged. This is one reason why the expected values files for all test cases need to have expected values for all variables of all data stores - even though only a small subset is actually used in any test case.

To satisfy data and control coupling coverage, it is necessary to identify data elements that are assigned a value in one module and used in a subsequent module. To verify correctness for any data element that couples two or more modules, it must be shown that the value generated by the generator module is compatible with what the receiver module is expecting. This can be accomplished by testing the input space of each input variable to a module. This is accomplished with equivalence class test cases. Hence verifying correctness of data and control coupling is implicitly part of equivalence class testing.

C.5.1.2.5 Summary

In summary, structure-based test cases will be implementation specific because it is based on specific coding. Paragraph 6.4.2.1 of DO-178B requires that modified condition/decision coverage be met. This implies that testing cover all combinations of logical variables and all entry and exit conditions. Accordingly, enough test cases will be included to exercise all allowable values of logical variables in the code structure. This requirement for complete logic path coverage can be achieved partly by using ACT tool (described below) to identify all decisions in the code. Test cases can then be created to cover decision paths not traversed by test cases derived from equivalence classes or requirements. Traceability of test coverage to test cases can be established by building decision tables and listing test cases that traverse the logical path of each decision. Tests listed in this table can include those derived from the path coverage analysis as well as those from the equivalence classes.

C.5.2 Test Case Execution Strategy

As previously stated, testing will be conducted based on the model from DO-178B. That is, low-level and the integration-level testing will be performed. This method is similar to what Myers describes as non-incremental testing. That is, tests for each GCS low-level functional unit will be independent. Test cases at this level, as stated before, will be requirements- and structure-based. Each unit will be tested using a driver to call the unit with the appropriate inputs. However, non-incremental testing, as described by Myers, implies the integration step links all the modules together in one step. For the GCS project, instead of linking all modules together in one step, bottom-up integration will be used. The functional units associated with each of the three subframes will first be integrated and tested. Then the three subframes can be integrated and tested. The two levels of integration testing are described below in subframe and frame testing.

C.5.2.1 Low-Level Testing

Low-level testing will concentrate on ensuring that low-level requirements are met by the software functional units. These units, as identified by the GCS Specification are AECLP, ARSP, ASP, CP, CRCP, GP, GSP, RECLP, TDLRSP, TDSP, and TSP. The Low-Level Requirements are those dealing with the functional unit requirements as well as organization and integrity of the Data Stores. This suite will include normal range as well as robustness test cases. Where appropriate, tests at this level will:

- test algorithms to see if they satisfy the requirements
- test loop operations
- test logic decisions
- test for missing or corrupted input conditions
- test that exceptional conditions are detected and handled correctly
- test the sequence of computations
- test the algorithm precision and accuracy
- check that array bounds are not exceeded (using the compiler option: */check*)

This level of testing is considered complete when all test cases are executed and the test outputs analyzed for accuracy and correctness as defined for each test case (see test case format).

C.5.2.2 Software Integration Testing

The Software Integration-level testing will address integration of the functional units into processing blocks that occur during subframes and frames in the simulation. To ensure that integration requirements are met, software integration testing will take place in the following order:

- Subframe processing
- Frame and Trajectory processing

Subframe testing will verify that functional units interact according to the software architecture (data and control flow) and the GCS Specification. The three subframes in each GCS implementation are Sensor Processing (SP), Guidance Processing (GP), and Control Law Processing (CLP). Once subframes are built, tests will:

- ensure that functional units are called in the correct order
- ensure that the rendezvous routine (GCS_SIM_RENDEZVOUS) is called first in each subframe
- ensure that Temperature Sensor Processing (TSP) is called first in SP subframe
- ensure that AECLP is called before CRCP in subframe 3
- ensure that CP is called last in each subframe
- ensure that all functional units are called
- ensure correct parameter passing between units
- ensure that global data store integrity is maintained

The goal of Frame testing is to verify that each implementation will satisfy the overall requirement of the GCS Specification to land the spacecraft. A frame is one iteration of all three subframes. Frame testing verifies that interaction between subframes is as prescribed in the GCS Specification and software design. Frame integration testing will link the subframes and perform testing that will:

- ensure proper system initialization or proper handling of incorrect initialization
- ensure that the subframes are executed in the correct sequence during a frame
- ensure that multiple frames can be executed consecutively
- ensure that an implementation can follow a given trajectory to landing

C.5.3 Test Output Review

Each Verification Analyst must verify the accuracy of the results of the test cases as required by DO-178B. A test log will be kept documenting each test run. Upon completion of the tests, a test output review is performed to find discrepancies between the expected result and the actual result. Any anomalies will be scrutinized by the Verification Analysts to determine if the problem is in the code or the test case. If correction is necessary, a problem report should be used to document and correct the anomalies. In cases where the anomaly warrants no further action, that decision must be documented in the test log. If the test case is found to be in error, a Support Documentation Change Report should be used to correct the error in the test case.

The Problem Reports are given to the software quality assurance personnel for review and relayed to the programmer or verification analyst for correction. When problem reports return to the verification analyst, the test case that revealed the error must be re-executed along with all test cases associated with the corrected code. It is also necessary to re-execute all test cases associated with any other module that has been updated. If the test case was in error, only the test case has to be re-executed. If there are several problem reports generated in a testing phase, it is the responsibility of the verification analyst to determine the sequence of repairs.

C.6 Verification Environment and Tools

According to Paragraph 6.4.1 of DO-178B, the best environment to test the software is the target environment. The target environment on which each GCS implementation is required to run is the computer that executes the GCS Simulator. The Software Life Cycle Environment Configuration Index provides more information on the configuration of the target environment. All testing will be performed on this computer; though not all test cases will require running with the Simulator.

To independently verify the correctness of calculations produced during testing, *Mathematica* (ref. C.12) will be used to model the computations of each function unit and calculate the expected results. *Mathematica* is a software package useful for mathematical modeling and calculations. It is available to GCS project on the SUN platform. *Mathematica* allows complex computations to be placed in a file so that the calculations can be repeated for different data sets. The model of each functional unit will be implemented in this manner.

For test cases which generate output that, according to DO-178B, must be compared with independently calculated values, the Verification Analysts will develop a program that compares the test output with the expected values derived from *Mathematica* models. This analysis program will generate a comparison file which can then be evaluated for problems.

The tool, *ACT* (ref. C.8), is based on McCabe's Cyclomatic Complexity Metric Method (ref. C.13). It will be used to identify all possible paths through the code of each functional unit. As previously described, this will be used for structure-based test case development and Structural Coverage Analysis. The output of interest from this software is the graph of the decision tree from the source code. It will be used to identify any untested decisions and also for documentation purposes.

Generic test drivers will be developed by the Verification Analysts to automate the testing. Drivers will also be used to test for defects outlined in Paragraph 6.4.3.

The FORTRAN Debugger is available for use in the event it is necessary to determine whether failure of a test case execution is due to the test setup or the actual code. It can be used during integration testing for test cases in the requirements and structural categories. It allows tracking transaction flow through subframes and frames. The Debugger can also be used to verify that test cases for statement, decision, and condition coverage are executing the intended target code.

C.7 Transition Criteria

This section describes the condition for progressing verification activities. As previously stated, verification activities are initiated for the artifacts of the software development processes. The three artifacts from the development processes involved are the design, the source code, and the executable modules. Transition criteria deals with the issue of when the verification activities associated with these artifacts are considered completed.

For the GCS project, the first artifact subject to verification is the design for an implementation. Hence the design review is the first activity and must be completed before the development can proceed to the coding process. The design review is considered completed when all deficiencies indicated in problem reports have been addressed and approved.

The code review takes place after coding is complete; that is, when the programmer is satisfied that his source code implements the design and the source code cleanly compiles. Note that due to the experimental aspects of the project, programmers are not allowed to execute their source code. The code review is complete also when all reported deficiencies are addressed and approved.

Independent of the GCS development is the test case development activities. To ensure a sufficient test suite to satisfy DO-178B testing criteria, a test readiness review will be conducted. The purpose of the test readiness review is to ensure that all the test coverage requirements are met and test cases are documented properly in the traceability matrix and the equivalence class table.

Once the code review and the test readiness review are completed, testing can commence. Testing is conducted according to the strategy outlined above. Testing is considered complete when all test cases have been executed with no errors detected.

C.8 Reverification Activities

As previously stated, the ultimate objective of verification activities is to identify any deviations between the system specification and the artifacts of the software development processes. Once the corrections are made to the artifacts, the verification procedure must be repeated to ensure that the original deficiencies are corrected and that no new problems are introduced during the correction.

For the GCS project, reverification will occur for both reviews and testing. For the design and code review, each deficiency that needs attention will be indicated on a problem report. The corrections made to the design or code are listed on an action report. The verification analyst who initiated the problem report must re-inspect the artifact to ensure all items on the problem report have been addressed correctly. Further, the entire artifact should be reviewed to ensure that the changes do not introduce any conflicts with the original parts.

Testing of the executables will also generate problem reports for bugs discovered during testing. Once the code is debugged, the verification analyst should re-execute the test case(s) that revealed the bugs. In addition, all test cases associated with the unit of level of integration being tested need to be re-executed.

C.9 Requirements Traceability Matrix

Functional Requirements	DESIGN	CODE	TEST CASES
0-1 Specify four separate, globally accessible data stores: EXTERNAL, GUIDANCE_STATE, RUN_PARAMETERS, and SENSOR_OUTPUT.			
2-1 Control flow of the frame processing.			
2-1.1 The appropriate control flow for a frame is: call to GCS_SIM_RENDEZVOUS. Satisfy the Sensor Processing subframe requirements (2-2). call to GCS_SIM_RENDEZVOUS. Satisfy Guidance Processing subframe requirements (2-3). call to GCS_SIM_RENDEZVOUS fulfill Control Law Processing subframe requirements (2-4) or terminate (2-1.2).			
2-1.2 The implementation is to terminate immediately upon completion of the Control Law Processing subframe requirements during the frame in which GP_PHASE is set to 5.			
2-2 Sensor Processing subframe requirements.			
2-2.1 Satisfy the TSP requirements (2.1.5) prior to fulfilling any of the other requirements in (2.1.1 and 2.1.4).			
2-2.2 Satisfy all requirements in the sensor processing requirements hierarchy (2.1).			
2-2.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-2.1.			
2-2.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.			
2-3 The Guidance Processing subframe requirements.			
2-3.1 Satisfy all requirements in the guidance processing requirements (2.2).			
2-3.2 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-3.1.			
2-4 The Control Law Processing subframe requirements.			
2-4.1 Satisfy the AECLP requirements (2.3.1) prior to fulfilling any of the CRCP requirements (2.3.3).			
2-4.2 Satisfy all requirements in the control law processing requirements hierarchy (2.3).			
2-4.3 Satisfy all requirements in the communications processing requirements (2.4) upon satisfying 2-4.1.			
2-4.4 Adhere to the functional unit scheduling in Table 4.3 of the GCS specification.			
2.1 SP -- Sensor Processing			
2.1.1 ASP -- Accelerometer Sensor Processing			
2.1.1-1 Rotate variables.			
2.1.1-2 Adjust gain for temperature.			
2.1.1-3 Remove characteristic bias.			
2.1.1-4 Correct for misalignment.			
2.1.1-5 Determine Accelerations.			
2.1.1-5.1 Acceleration based on current A_COUNTER.			
2.1.1-5.2 Acceleration based on mean of previous accelerations.			
2.1.1-6 Determine Accelerometer Status			
2.1.1-6.1 A_STATUS = healthy			
2.1.1-6.2 A_STATUS = unhealthy			
2.1.2 ARSP -- Altimeter Radar Sensor Processing			
2.1.2-1 Rotate variables.			

2.1.2-2 Determine altitude when echo is received. (based on AR_COUNTER)			
2.1.2-3 Determine altitude when echo is not received			
2.1.2-3.1 Determine altitude based on third-order polynomial.			
2.1.2-3.2 Determine altitude based on previous calculation.			
2.1.2-4 Set altimeter radar status.			
2.1.2-4.1 AR_STATUS = healthy			
2.1.2-4.2 AR_STATUS = failed			
2.1.2-5 Set values of K_ALT.			
2.1.2-5.1 K_ALT = 1			
2.1.2-5.2 K_ALT = 0			
2.1.3 TDLRSP -- Touch Down Landing Radar Sensor Processing			
2.1.3-1 Rotate variables			
2.1.3-2 Determine state for each radar beam.			
2.1.3-2.1 TDLR_STATE = unlocked.			
2.1.3-2.2 TDLR_STATE = locked.			
2.1.3-3 Determine Whether to set FRAME_BEAM_UNLOCKED			
2.1.3-3.1 Set FRAME_BEAM_UNLOCKED to FRAME_COUNTER			
2.1.3-3.2 Leave FRAME_BEAM_UNLOCKED unchanged			
2.1.3-4 Calculate the beam velocities			
2.1.3-5 Process beam velocities based on which beam(s) locked.			
2.1.3-5.1 no beams locked			
2.1.3-5.2 Beam1 locked			
2.1.3-5.3 Beam2 locked			
2.1.3-5.4 Beam3 locked			
2.1.3-5.5 Beam4 locked			
2.1.3-5.6 Beam1 & Beam2 locked			
2.1.3-5.7 Beam1 & Beam3 locked			
2.1.3-5.8 Beam1 & Beam4 locked			
2.1.3-5.9 Beam2 & Beam3 locked			
2.1.3-5.10 Beam2 & Beam4 locked			
2.1.3-5.11 Beam3 & Beam4 locked			
2.1.3-5.12 Beam1, Beam2, & Beam3 locked			
2.1.3-5.13 Beam1, Beam2, & Beam4 locked			
2.1.3-5.14 Beam1, Beam3, & Beam4 locked			
2.1.3-5.15 Beam2, Beam3, & Beam4 locked			
2.1.3-5.16 Beam1, Beam2, Beam3, & Beam4 locked			
2.1.3-6 Convert to body velocities.			
2.1.3-7 Set values in K_MATRIX.			
2.1.3-7.1 Kx = 0			
2.1.3-7.2 Kx = 1			
2.1.3-7.3 Ky = 0			
2.1.3-7.4 Ky = 1			
2.1.3-7.5 Kz = 0			
2.1.3-7.6 Kz = 1			
2.1.3-8 Set TDLR_STATUS.			
2.1.4 GSP -- Gyroscope Sensor Processing			
2.1.4-1 Rotate variables.			
2.1.4-2 Determine the vehicle rotation rates along each of the vehicle's three axes.			
2.1.4-2.1 Adjust gain.			
2.1.4-2.2 Convert G_COUNTER.			

2.1.4-3 Set gyroscope status to healthy.			
2.1.5 TSP -- Temperature Sensor Processing			
2.1.5-1 Calculate solid state temperature			
2.1.5-2 Calculate Thermal Temperature			
2.1.5-3 Determine which Temperature to use (SS or Thermocouple)			
2.1.5-3.1 Calculate the Thermo sensor upper limit			
2.1.5-3.2 Calculate the Thermo sensor lower limit			
2.1.5-4 Determine Atmospheric Temperature			
2.1.5-5 Set status to healthy.			
2.1.6 TDSP -- Touch Down Sensor Processing			
2.1.6-1 Determine status of touch down sensor.			
2.1.6-2 Determine whether touch down has been sensed.			
2.2 GP -- Guidance Processing			
2.2-1 Rotate variables.			
2.2-2 Determine the attitude, velocities, and altitude.			
2.2-2.1 Set up the GP_ROTATION matrix.			
2.2-2.2 Calculate new values of attitude, velocity, and altitude.			
2.2-3 Determine if the engines should be on or off.			
2.2-3.1 Engines on			
2.2-3.2 Engines off			
2.2-4 Set FRAME_ENGINES_IGNITED			
2.2-5 Determine velocity error.			
2.2-6 Determine optimal velocity			
2.2-7 Determine if contour has been crossed.			
2.2-8 Determine guidance phase.			
2.2-8.1 GP_PHASE = 1			
2.2-8.2 GP_PHASE = 2			
2.2-8.3 GP_PHASE = 3			
2.2-8.4 GP_PHASE = 4			
2.2-8.5 GP_PHASE = 5			
2.2-9 Determine which set of control law parameters to use.			
2.2-9.1 CL = 1			
2.2-9.2 CL = 2			
2.3 CLP -- Control Law Processing			
2.3.1 AECLP -- Axial Engine Control Law Processing			
2.3.1-1 Generate the appropriate axial engine commands when AE_CMD=ON.			
2.3.1-1.1 Determine engine temperature			
2.3.1-1.1.1 AE_TEMP = COLD			
2.3.1-1.1.2 AE_TEMP = WARM			
2.3.1-1.1.3 AE_TEMP = HOT			
2.3.1-1.2 Compute limiting errors for pitch			
2.3.1-1.3 Compute limiting error for yaw			
2.3.1-1.4 Compute limiting error for thrust			
2.3.1-1.5 Compute pitch, yaw, and thrust errors.			
2.3.1-1.5.1 CHUTE_RELEASED = 1			
2.3.1-1.5.2 CHUTE_RELEASRD = 0			
2.3.1-1.5.3 CONTOUR_CROSSED = 1			
2.3.1-1.5.4 CONTOUR_CROSSED = 0			
2.3.1-1.6 Compute INTERNAL_CMD			
2.3.1-1.7 Compute axial engine valve settings (AE_CMD).			

2.3.1-1.7.1	when INTERNAL_CMD < 0.0			
2.3.1-1.7.2	when 0.0 £ INTERNAL_CMD ≥ 1.0			
2.3.1-1.7.3	when 1.0 < INTERNAL_CMD			
2.3.1-2	Generate the appropriate axial engine commands when AE_CMD=OFF.			
2.3.1-2.1	Set AE_CMD = 0			
2.3.1-3	Set axial engine status to healthy.			
2.3.2	**RECLP -- Roll Engine Control Law Processing**			
2.3.2-1	Generate the appropriate roll engine command.			
2.3.2-2	Set roll engine status to healthy.			
2.3.3	**CRCP -- Chute Release Control Processing**			
2.3.3-1	Determine appropriate parachute release command.			
2.3.3-1.1	AE_TEMP = COLD			
2.3.3-1.2	AE_TEMP = WARM			
2.3.3-1.3	AE_TEMP = HOT			
2.3.3-1.4	CHUTE_RELEASED = 0			
2.3.3-1.5	CHUTE_RELEASED = 1			
2.4	CP -- Communications Processing			
2.4-1	Set communicator status to healthy.			
2.4-2	Get synchronization pattern.			
2.4-3	Determine sequence number.			
2.4-4	Prepare sample mask.			
2.4-4.1	Subframe 1 mask			
2.4-4.2	Subframe 2 mask			
2.4-4.3	Subframe 3 mask			
2.4-5	Prepare data section.			
2.4-5.1	Use subframe 1 data			
2.4-5.2	Use subframe 2 data			
2.4-5.3	Use subframe 3 data			
2.4-2.5	Calculate checksum.			

C.10 Structure-based Test Type Matrix

The following matrix is reproduced from a presentation titled "Testing Techniques" given by Steve Paasch at the FAA-ACS Software Standardization Conference. It is included in this document to support the assertion made in the test coverage section that Modified Condition/Decision Coverage is a super set of Decision Coverage, Condition Coverage, and Decision/Condition Coverage.

Table C.1. Structure-based test type matrix.

	Every path executed	Every statement executed at least once	Each decision takes on each possible outcome at least once	Each entry & exit point invoked at least once	Each condition in a decision takes on each possible outcome once	Each condition shown to independenty effect decision outcome	Each combinition of conditions in a decision takes on each possible outcome once
Path Coverage	■	▒	▒	▒	▒	▒	▒
Statement Coverage		■					
Decision Coverage		▒	■	■			
Condition Coverage		▒		■	■		
Decision/ Condition Coverage		▒	■	■	■		
Modified Condition/ Decision Coverage		▒	■	■	■	■	
Multiple Condition Coverage		▒	▒	▒	▒	▒	■

C.11 References

C.1. Finelli, George B.: Results of Software Error-Data Experiments. In *AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference,* Atlanta, GA, September 1988.

C.2. "Software Considerations in Airborne Systems and Equipment Certification", Document No. RTCA/DO-178B, Dec. 1992.

C.3. Myers, Glenford J., *The Art of Software Testing, ,*Wiley-Interscience Pub. N.Y., N.Y., 1979.

C.4. Derek J. Hatley and Imtiaz A. Pirbhai. *Strategies for Real-Time System Specification ,* Dorset House Publishing Company, New York, New York, 1987.

C.5. De Marco, Tom. *Structured Analysis and System Specification.* Prentice-Hall, Englewood Cliffs, N.J., 1978.

C.6. Ward, Paul T., and Stephen J. Mellor. *Structured Development for Real-Time Systems.* Prentice-Hall, Englewood Cliffs, N.J., 1985.

C.7. team*work*/SA team*work*/RT User's Guide, Release 4.0. Cadre Technologies Inc. 1990.

C.8. Analysis of Complexity Tool User's Instructions, McCabe Associates Inc., Redwood City, Ca.,1992

C.9. Chilenski, John Joseph, Miller, Steve P.; Applicability of Modified Condition/Decision Coverage to Software Testing; The Boeing Company, and Rockwell International Corporation.

C.10. Paasch, Steve; "Testing Techniques"; FAA ACS-Software Standardization Conference; July 26-28, 1994.

C.11. Pressman, Roger S. Software Engineering, A Practitioner's Approach; McGraw-Hill Inc. N.Y., N.Y.;1992.

C.12. Wolfram, Stephen,. *Mathematica, A System for Doing Mathematics by Computer, Second Edition*. Addison-Wesley Publishing Company, Inc., 1991

C.13. McCabe, Thomas j., Structural Testing: A Software Testing Methodology Using the Cyclomatic Complexity Metric, McCage & Associates, Inc, 1982.

Appendix D: Software Configuration Management Plan for the Guidance and Control Software Project

Authors: Laura J. Smith, Kelly J. Hayhurst, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

D. Contents

D. List of Tables

D. List of Figures

D.1 Introduction

According to the RTCA/DO-178B "Software Considerations in Airborne Systems and Equipment Certification", configuration management is the "process of identifying and defining the configuration items of a system, controlling the release and change of these items throughout the software life cycle, recording and reporting the status of configuration items and change requests, and verifying the completeness and correctness of configuration items" (ref. D.2). This configuration management plan establishes the methods to be used to achieve the objectives of the software configuration management (SCM) process throughout the software life cycle for the Guidance and Control Software (GCS) project in accordance with the DO-178B guidelines. Specifically, this document provides a description of the SCM environment that will be used throughout the GCS project (including the methods, tools, standards, and procedures) and a description of the SCM activities in the software life cycle.

As described in Subsection 7.1 of the DO-178B guidelines, the SCM process, along with the other software life cycle processes, assists in meeting the following general objectives for the certification authority:

- provide a defined and controlled configuration of the software throughout the software life cycle;
- provide the ability to consistently replicate the executable object code or to regenerate it if needed;
- provide control of process inputs and outputs during the software life cycle that ensures consistency and repeatability of process activities;
- provide a known point for review, assessing status, and change control by establishing baselines for configuration items;
- provide controls that ensure problems receive attention and changes are recorded, approved, and implemented;
- provide evidence of approval for the software;
- aid the assessment of the software product compliance with requirements; and
- ensure that secure physical archiving, recovery and control are maintained for the configuration items.

D.1.1 The Role of SCM in the GCS Project

The GCS project involves independent production of two implementations of a guidance and control application where the development process for each implementation follows the DO-178B guidelines. The two GCS implementations are referred to by planetary names: Mercury and Pluto. When there is a need to distinguish multiple implementations, the word *planet* will be used to refer to Mercury or Pluto. For this project, the configuration environment and activities must provide for the management of the life cycle data for one set of development processes and must also provide a mechanism to preserve the independence of the life cycle data for the multiple implementations. This plan will address the configuration management process for life cycle data from both GCS implementations.

According to Uczekaj and Hughes of Honeywell (ref. D.3), a configuration management system is a tool that is critical for tracking all phases of the software development cycle. A detailed description of the software development cycle for the GCS project is found in the *Plan for Software Aspects of Certification*. As described in that document, the development of three GCS implementations was started at the Research Triangle Institute (RTI), along with the generation of the documentation of the software development process (ref. D.1). The original software development processes for the GCS project included the following:

- software design,
- software coding, and
- integration.

All three RTI-developed implementations of the GCS were developed under the DO-178A guidelines and went through the design and coding processes. Due to a number of factors, the project was transferred to NASA Langley Research Center to complete under the DO-178B guidelines. Upon delivery to NASA, new development teams were assigned and the project restarted with a review of the software requirements. Consequently, the software development processes for the in-house GCS project will include the following processes (see the *Plan for Software Aspects of Certification* for more details):

- transitional software requirements development (focusing on the review and modification of the existing software requirements),
- transitional software design,
- software coding, and
- integration.

At the end of the transitional software requirements development process, Version 2.2 of the *Software Requirements Data*, referred to as the GCS specification, was created. The transitional software design process is complete when the design has been verified and approved by the SQA. The coding process is complete when the code has been verified and approved by the SQA. Integration is broken down into two types of testing: functional (consisting of unit testing, subframe testing, frame testing, and trajectory testing) and structural. The integration process ends when the functional and structural testing are complete.

For the GCS project, the general plan for configuration management is to use a set of software tools, already available at Langley, and some paper forms to identify, control, baseline, and archive all life cycle data associated with the development of the GCS implementations. Table D.1 gives a list of the life cycle data for the GCS project as discussed in Section 11 of the DO-178B guidelines. This life cycle data consists of planning and support documents and the actual products from the software development process (e.g., design description and source code). Configuration management is responsible for maintaining all changes made to this life cycle data throughout the GCS project.

Since the guidance and control software is classified as Level A software according to DO-178B, Control Category 1 (CC1), described in Subsection 7.3 of DO-178B, will be applied to the necessary software life cycle data. Table D.1 shows the life cycle data and its associated control category (CC1 or CC2). In accordance with the control categories, the SCM process for the GCS project must provide for the objectives listed in Table D.2. All of the processes listed in Table D.2 will be managed through the use of software tools, with the exception of the problem reporting process. Problem reporting will be managed by paper forms and is discussed in detail later in this document.

Table D.1. DO-178B Life Cycle Data for the GCS Project

Software Life Cycle Data	Subsection Reference in DO-178B	Control Category
Plan for Software Aspects of Certification	11.1	1
Software Development Plan Software Design Standards Software Code Standards Software Requirements Standards	11.2 11.7 11.8 11.9	1 1 1 1
Software Verification Plan	11.3	1
Software Configuration Management Plan	11.4	1
Software Quality Assurance Plan	11.5	1
Software Requirements Data	11.6	1
Design Description	11.10	1
Source Code	11.11	1
Executable Object Code	11.12	1
Software Verification Cases and Procedures	11.13	1
Software Verification Results	11.14	2
Software Configuration Index Software Life Cycle Environment Configuration Index	11.16 11.15	1 1
Problem Reports	11.17	2
Software Configuration Management Records	11.18	2
Software Quality Assurance Records	11.19	2
Software Accomplishment Summary	11.20	1

The following chapter describes the software configuration management environment for the GCS project and gives a general overview of the primary tools used in the development cycle.

Table D.2. SCM Objectives for Control Category 1 and Control Category 2

SCM Process Objective	DO-178B Subsection reference	CC1	CC2
Configuration Identification	7.2.1	•	•
Baselines	7.2.2 a, b, c, d, e	•	
Traceability	7.2.2 f, g	•	•
Problem Reporting	7.2.3	•	
Change Control -- integrity and identification	7.2.4 a, b	•	•

Change Control -- tracking	7.2.4 c, d, e	•	
Change Review	7.2.5	•	
Configuration Status Accounting	7.2.6	•	
Retrieval	7.2.7 a	•	•
Protection against Unauthorized Changes	7.2.7 b(1)	•	•
Media Selection, Refreshing, Duplication	7.2.7 b(2), (3), (4), c	•	
Release	7.2.7 d	•	
Data Retention	7.2.7 e	•	•

D.2 SCM Environment

According to Subsection 7.2 of DO-178B, configuration management should be provided throughout the software development process for configuration identification, change control, baseline establishment, and archiving of the software life cycle data. This chapter describes the SCM environment to be used for the GCS project, including descriptions of procedures, tools, methods, standards, organizational responsibilities, and interfaces.

Since the development of the GCS implementations is part of a research project, the development environment for the software is the same as the target environment of the implementations; that is, the GCS implementations will not be included in a "real" hardware system intended for space flight. The environment for most of the software development of the GCS implementations is a microVAX 3800 computer system running the VAX/VMS 5.5-2 operating system. This computer system is physically located at NASA Langley in Building 1220, Room 214. This computer system is referred to as "AIR19" by personnel working on this project and will be referred to as such in project documentation.

For the GCS project, the VAX DEC/Code Management System (CMS) will be used as the primary tool to aid in the configuration management activities. CMS will be used for the configuration management of all life cycle data shown in Table D.1 (with the exception of the Problem Reports, Software Configuration Management Records and Software Quality Assurance Records) and other software artifacts related to the project, including the GCS simulator and its user's guide. In general, the DO-178B life cycle data for the development of the GCS implementations can be divided into three different categories: support documentation (shown in Table D.3); development products (shown in Table D.4); and records, results, and reports (shown in Table D.5). The support documentation and the development products are under CC1; the records, results, and reports are under CC2.

Table D.3. Support Documentation

Plan for Software Aspects of Certification
Software Development Plan
Software Verification Plan
Software Configuration Management Plan
Software Quality Assurance Plan
Software Requirements Standards
Software Design Standards
Software Code Standards
Software Requirements Data
Software Verification Cases and Procedures
Software Life Cycle Environment Configuration Index
Software Configuration Index
Software Accomplishment Summary

Table D.4. Development Products

Design Description
Source Code
Executable Object Code

Table D.5. Records, Results, and Reports

Software Verification Results
Problem Reports
Software Configuration Management Records
Software Quality Assurance Records

Table D.6 shows the project member who is responsible for each element of the life cycle data. Since two GCS implementations are being independently developed, there will be data from each implementation in some cases. For example, each implementation will have its own source code (e.g., Mercury Source Code and Pluto Source Code). The data that will be replicated for each implementation are denoted with a * in Table D.6.

Table D.6. DO-178B Life Cycle Data and Organizational Responsibilities

Software Life Cycle Data	Project Member Responsible for that Data
Plan for Software Aspects of Certification	Project Leader
Software Development Plan	Project Leader
Software Requirements Standards	Project Leader
Software Design Standards	Project Leader
Software Code Standards	Project Leader
Software Accomplishment Summary	Project Leader
Software Verification Plan	Verification Analysts
Software Verification Cases and Procedures*	Verification Analysts
Software Verification Results*	Verification Analysts
Software Configuration Management Plan	Configuration Manager
Software Life Cycle Environment Configuration Index	Configuration Manager
Software Configuration Index	Configuration Manager
Software Configuration Management Records	Configuration Manager
Software Quality Assurance Plan	Software Quality Assurance Representative
Problem Reports*	Software Quality Assurance Representative
Software Quality Assurance Records*	Software Quality Assurance Representative
Software Requirements Data	System Analyst
Design Description*	Programmer
Source Code*	Programmer
Executable Object Code*	Programmer

The following section gives a general overview of the CMS tool as it will be used on the GCS project, and the procedures for its use.

D.2.1 CMS Description

CMS is an on-line library system (located on AIR19, the microVAX 3800 computer system) that helps track the software development process. A CMS library is a VMS directory that contains specially formatted files. CMS stores files called elements in a library, tracks changes made to these elements, and monitors access to the elements. An element may contain text, source code, object code, test cases, etc. To help preserve the integrity of the configured items, direct access to the CMS libraries is limited to the configuration manager and the project leader.

The configuration manager has the primary responsibilities for all configuration management activities; however, the project leader will have full access to the CMS libraries in order to perform the configuration manager's duties in case of an emergency. Further information on change control is contained in the section "Change Control".

The basic structural unit of a CMS library is called an *element*. An element consists of one file and all of the file's successive versions. A *generation* of an element is a specific version of that element. The first element that is created and placed into a library is version one of that element; each time an element is reserved and replaced into the CMS library, a new generation of that element is created. CMS stores the entire text of the first generation of the element. For each successive generation of that element, CMS stores only the lines that change from one generation to the next.

Elements can be combined into *groups* which are manipulated as a single unit. For example, an element could be a single test case developed to test a functional unit and a group could be all of the test cases to test that module. Even if an element is in a group, the element can still be manipulated on an individual basis. A group may consist of other groups; but a group may not be a member of itself. Specific generations of elements can be clustered into a *class* and manipulated as a single unit. For example, the Post-Code Review class could represent the specific generations of elements that comprise the code resulting after the Code Reviews. Only one generation of an element can be in a class, but each element can have a different generation number indicating that some elements have been modified more than others. Classes will be used to identify the software life cycle data at specific phases in the development process.

A reference copy directory has been established for each CMS library. A reference copy directory is a nonlibrary directory that contains the latest generation of each element; CMS automatically updates the reference copy directory every time a new generation of an element is created. CMS also maintains a delta file for each element stored in a library. A delta file is a file that contains the contents of all of the generations of a single element; it contains the actual data and the control records. The control records tell CMS which data records are valid for which specific generation of the element. Backup information is maintained by CMS which allows CMS to recover from an incomplete transaction in the event of a system failure.

CMS manages the change process by using a system of reservations and replacements. Since most participants of the GCS project do not have direct access the CMS libraries, only a few basic commands need to be known to communicate with the configuration manager about their life cycle data. For more information about available CMS commands, refer to the *Guide to VAX/DEC Code Management System* (ref. D.4). Knowledge of the following commands will be helpful in understanding the procedures for configuration management of the GCS life cycle data.

Fetch -- A copy of one or more specified element generations is placed in a directory for use by the project participant. No changes are made to the element within the CMS library. For example, a copy of the element generations that comprise the version of code to be reviewed at the Code Review may be fetched for all participants in the Code Review to examine in preparation for the Review.

Reserve -- A copy of one or more specified element generations is placed in a directory so that it can be modified by the project participant. The latest version of the element will be reserved unless otherwise specified. After the file has been modified, the file should be returned to the library (using the replace command) and the changes will be made to the library copy and the reference copy. As an example for this command, a programmer should reserve a particular element of source code in order to make a change in response to a Problem Report.

Replace -- An element that has been reserved can be replaced to the CMS library. The element that is replaced may be completely different from the element that was reserved. A new generation of that element is created. As in the example where the programmer has reserved an element to make a change in response to a Problem Report, the element will be replaced after the SQA representative has signed the PR indicating all necessary changes have been made.

Unreserve -- If the wrong element has been reserved, the element can be unreserved without changing the library element.

Once an element has been placed under configuration control, there must be a valid justification to change that element. If an element needs to be changed, it must be reserved, changed, and replaced. When an element generation is reserved from a CMS library, a reservation exists. This reservation ends when the generation is replaced or unreserved. If the element is replaced, the contents in the CMS library are updated; if the element is unreserved, no new generation is created and CMS records the cancellation in the library history. The unreserve command is useful if the wrong element has been reserved.

Every action that takes place in the CMS library is recorded in a history file, along with the name of the person requesting the action, the date, and a remark. For the GCS project, the configuration manager and the project leader are the only people who can request an action within a CMS library. Whenever a reservation of a CMS library element is made, CMS prompts for a remark. This remark provides a permanent record of the transaction in the library's history file. CMS does not record transactions that do not alter the library.

D.2.1.1 CMS Libraries

Table D.7 shows the CMS library names associated with the project data.

Table D.7. CMS Libraries for Project Data

Project Data	**CMS Library Name**
Plan for Software Aspects of Certification, Software Development Plan	DISK$HOKIE:[GCS.CMS.CERT_PLAN]
Software Verification Plan	DISK$HOKIE:[GCS.CMS.VER_PLAN]
Software Requirements Traceability Data	DISK$HOKIE:[GCS.CMS.TRACE_DATA]
Software Configuration Management Plan	DISK$HOKIE:[GCS.CMS.CM_PLAN]
Software Quality Assurance Plan	DISK$HOKIE:[GCS.CMS.SQA_PLAN]
Software Requirements Standards, Software Design Standards, Software Code Standards	DISK$HOKIE:[GCS.CMS.DEV_STAND]
Software Requirements Data	DISK$HOKIE:[GCS.CMS.SPEC]
Modifications to Requirements Data	DISK$HOKIE:[GCS.CMS.SPEC_MODS]
Design Description*	DISK$HOKIE:[GCS.CMS.DES_DESCRIP.*planet*]

Source Code*	DISK$HOKIE:[GCS.CMS.SOURCE_CODE.*planet*]
Executable Object Code*	DISK$HOKIE:[GCS.CMS.EXEC_OBJ_CODE.*planet*]
Software Verification Cases*	DISK$HOKIE:[GCS.CMS.VER_CASES]
Software Verification Procedures	DISK$HOKIE:[GCS.CMS.VER_PROC]
Software Verification Results*	DISK$HOKIE:[GCS.CMS.VER_RESULTS.*planet*]
Software Life Cycle Environment Configuration Index, Software Configuration Index	DISK$HOKIE:[GCS.CMS.CONFIG_INDEX]
Software Accomplishment Summary	DISK$HOKIE:[GCS.CMS.ACCOMP_SUM]
Simulator User's Guide	DISK$HOKIE:[GCS.CMS.SIMULATOR.USER_GUIDE]
Simulator Source Code	DISK$HOKIE:[GCS.CMS.SIMULATOR.SOURCE_CODE]

* These project data are implementation specific. The Verification Cases library only has a few elements that are implementation specific; therefore, there will be a naming convention to distinguish between the two implementations.

D.2.1.2 Procedures for Using CMS

The configuration manager will use CMS libraries to manage project data. CMS can be invoked from the DCL command level, from the CMS subsystem command level, or from the DECwindows user interface.

In order to fetch, reserve or replace an element using CMS, it is easiest to have the directory set to the specific directory in which the element will be placed or retrieved. The fetch command is issued when a copy of the element is needed for examination purposes only; no changes may be made to this copy of the element. For example, after issuing the fetch command, the element name is entered in the appropriate place. If this transaction needs to be recorded in the history log, a remark must be entered before the command is executed; otherwise, no transaction will be recorded. Once the fetch command has been issued, the element will reside in the VMS default directory that was set prior to issuing the command. The reserve and replace commands work in a similar manner, except these transactions are always recorded in the history log, even if no remark is entered along with the command. The reserve command places a working copy of the element in the directory; the latest version of the element is reserved unless otherwise specified. If the noconcurrent qualifier was issued at the time of reservation, no other reservations of that element are allowed until after the element has been replaced. Once the reserve command has been issued, the element name is entered, along with a remark, and then the reservation is executed. The replace command can only be executed if a reservation exists. The replace command, along with the element name and remark, are entered and executed. If there is more than one version of a file in the default directory, the replace command will use the highest version number for the replacement of an element.

The wildcard character, “*”, may be used for multiple reservations, replacements, or fetches if the elements are similar in name. The * may be used in place of one or more characters.

The following section describes the tool Team*work*, which will be used by the programmers for the development of their detailed designs in addition to CMS.

D.2.2 Teamwork

For the GCS project, each programmer is required to use the Computer Aided Software Engineering (CASE) tool, Team*work* (ref. D.5), to develop the detailed design description. The Team*work* tool is used to aid in the structured design of the applications, and certain parts of the output from Team*work* will be required for design and code reviews. Team*work* is composed of several tools that are available to the designer. Each programmer may choose to use any of the following Team*work* components:

SA --- The baseline structured analysis tool (ref. D.6),

RT --- An extension of SA that allows description of real-time systems (ref. D.6), and

SD --- A parallel tool that follows the Ward and Mellor approach (ref. D.7).

The Team*work* tool provides its own configuration management capability. Each of the pieces of the design is stored by Team*work* as individual files, and each file has a version number appended to it. Whenever a file is changed, the old file is kept and a new file is created with a higher version number. Team*work* also has a baselining capability which saves files with a specific version number under a baseline name. For this project, the programmers will be allowed to use the configuration management capabilities of Team*work* as they choose, since the Team*work* designs will be configured using CMS at the appropriate milestone versions (see section "Baselines and Traceability").

D.2.3 Other SCM Tools

The GCS programmers and verification analysts are required to use VAXnotes to request CMS library elements from the configuration manager; other GCS project participants may use any means available to request elements. For example, if a programmer needs to reserve his source code, he should provide the configuration manager with the element name and the PR#. The PR# is needed so that this information may be recorded in the CMS library history log. See the *Software Development Standards* for more information about communication protocol.

Problem Reports, which are paper forms, will be kept in binders by the configuration manager once the SQA representative has approved them; a binder will also be kept for the Support Documentation Change Report forms. These binders will be physically located in the Configuration Manager's office in Building 1220 of NASA Langley. A status log binder will also be kept in the configuration manager's office. This binder will have transactions affecting the life cycle data recorded in it. This is maintained as an alternative to entering CMS and searching the history log to see when the item was configured or at what baseline the item is now under. See the section on "Configuration Status Accounting" for more details on these binders.

For information about other tools used on the GCS project see the *Software Life Cycle Environment Configuration Index.*

The following chapter describes the SCM activities to be performed during the life cycle of the GCS project.

D.3 SCM Activities

This chapter describes the SCM activities required for the GCS project according to Subsection 7.2 of DO-178B. The following SCM activities will be addressed in this chapter:

- configuration identification;
- baselines and traceability;
- problem and change reporting;
- change control;
- change review;
- configuration status accounting;
- archive, retrieval and release; and
- software load control.

The software life cycle environment controls are discussed in the *Software Configuration Index*.

D.3.1 Configuration Identification

According to Paragraph 7.2.1 of DO-178B, the objective of the configuration identification activity is to label unambiguously each configuration item (and its successive versions) so that a basis is established for the control and reference of configuration items. This section describes the methods used to identify the software life cycle data; Table D.8 gives the unambiguous labels for each configuration item.

For the GCS project, configuration identification is established for each configuration item, for each separately controlled component of a configuration item, and for combinations of configuration items that comprise a software product.

The life cycle data that will be kept in CMS libraries were combined into one plan if related, otherwise the data was maintained as an individual plan. The plans were then labeled according to their content. For example, the Project Standards include the software standards for requirements, design, and code. These were combined into one plan since they all involve standards of the GCS project.

For implementation specific data, some elements in the libraries may have the same names. Since each implementations' elements are mainly kept in separate libraries there will be no confusion as to which elements are being referenced; however, for the verification cases, some elements are distinguished by preceding the element name with the first letter of the planet name followed by an underscore. For example, the guidance processing test case driver for Mercury would be named m_test_gp.for. The source code is maintained in a CMS library named DISK$HOKIE:[GCS.CMS.SOURCE_CODE.*planet*]. The programmers do not have access to the other programmers source code so it does not matter if elements have the same name.

Table D.8 shows the configuration item labels associated with the software life cycle data; see the section of DO-178B referenced for a description of the contents of each document.

D.3.2 Baselines and Traceability

The DO-178B guidelines define a baseline as the approved, recorded configuration of one or more configuration items that, thereafter, serves as the basis for further development. Hence, the objective of baseline establishment is to define the base configuration for all configuration items in such a manner as to allow reference to, control of, and traceability between configuration items. For the GCS project, baselines are established in CMS software libraries (by creating classes at appropriate phases) to ensure that their integrity is maintained. The baselining capabilities of the CMS tool will be used to group generations of files at certain major life cycle phases. Baselines will be established for configuration items used to demonstrate that all certification requirements have been satisfied according to DO-178B.

Table D.8. Configuration Identification for the DO-178B Life Cycle Data

Configuration Items	**Software Life Cycle Data**	**Subsection Reference in DO-178B**
Plan for Software Aspects of Certification	Plan for Software Aspects of Certification Software Development Plan	11.1 11.2
Verification Plan Software Requirements Traceability Data	Software Verification Plan	11.3
Configuration Management Plan	Software Configuration Management Plan	11.4
Software Quality Assurance Plan	Software Quality Assurance Plan	11.5
Software Development Standards	Software Requirements Standards Software Design Standards Software Code Standards	11.6 11.7 11.8
GCS Specification	Software Requirements Data	11.9
Team*work* Model* Design Overview*	Design Description	11.10
Source Code*	Source Code	11.11
Executable Object Code*	Executable Object Code	11.12
Verification Cases* Verification Procedures	Software Verification Cases and Procedures	11.13
Software Verification Results*	Software Verification Results	11.14
Software Configuration Index	Software Life Cycle Environment Configuration Index Software Configuration Index	11.15 11.16
Problem and Action Reports* Support Document Change Reports Formal Modifications to the Specification**	Problem Reports	11.17
Configuration Management Records*	Software Configuration Management Records	11.18
Software Quality Assurance Records*	Software Quality Assurance Records	11.19
Software Accomplishment Summary	Software Accomplishment Summary	11.20

* These configuration items will be implementation specific, the labels should refer to the implementation as appropriate.

** Formal modifications 2.2-1 through 2.2-26 of the GCS Specification were not recorded on a Support Documentation Change Report (SDCR) form. All remaining modifications to the GCS Specification will be recorded on a SDCR form.

Baselines can be changed only through change control procedures. Specifically, baselines can be changed only through the process of:

- recording the change,
- reviewing and evaluating the change,
- approving or disapproving the change, and
- coordinating the change.

Since the GCS project was originally started at RTI and then transferred to NASA, all of the documents and source code as brought in from RTI are kept in a "cms_old" library so that the capability exists to reconstruct the data as received from RTI. The support documentation, GCS specification, and source code from RTI are maintained in three separate CMS libraries. The support documents are kept in the library named DISK$HOKIE:[GCS.CMS_OLD.DO178A.DOCS], the GCS specification is located in the library DISK$HOKIE:[GCS.CMS_OLD.DO178A.SPEC], and the source code is located in the library DISK$HOKIE:[GCS.CMS_OLD.CODE.*planet*]. The in-house part of the GCS project is starting with the revision of RTI's Version 2.1 of the GCS Specification during the transitional requirements development phase. The transitional design phase will start with RTI's Post-Code Review version of the design. Hence, Version 2.1 of the GCS specification, and the Post-Code Review version of the design for each implementation will be the starting point for all development activities for the in-house GCS project.

The baselines for the design description and source code of each GCS implementation are derived from the milestones of the development and verification processes (see the *Software Verification Plan* for more details on the verification activities). In general, the design description and source code will be baselined after the SQA representative completes the review following the verification activity that takes place during each development phase. The milestones will comprise the classes in the CMS libraries. For example, after the subframe test completion review, all elements in the source code libraries will have their latest generations clustered into a class called SF. These elements can then be manipulated as a single unit.

The Team*work* designs from each implementation are located in the CMS library DISK$HOKIE:[GCS.CMS.DES_DESCRIP.*planet*] and will be baselined according to the schedule shown in Table D.9. The source code for each implementation will be contained in FORTRAN files, and these files are located in the CMS library DISK$HOKIE:[GCS.CMS.SOURCE_CODE.*planet*] and will be baselined according to the schedule shown in Table D.10.

Table D.9: Milestones for Design

Milestone	**CMS Class Name**
Post-Design Review	DR
Post-Code Review	CR
Post-Requirements-based Test	RBT
Post-Structure-based Test	SBT

The design libraries will be created from the Post-Code Review version of the designs received form RTI. The source code libraries and the executable object code libraries will start after the design process is completed. The GCS specification library was started with RTI's Version 2.1 of the GCS specification converted to a Microsoft Word document. All other life cycle data will enter the configuration management process as an in-house version.

In some cases, a new baseline may be established for a support document if numerous modifications have been made (since no predefined milestone exists). For example, when the GCS specification was first developed, Version 1.0 was created. There were a few interim versions of the GCS specification (Version 1.1, 1.2, etc.) created before it was classified as Version 2.0. After verification of the GCS specification, it was updated to Version 2.0. After a significant number of specification modifications, the GCS specification was updated to Version 2.1. Upon transfer to NASA, more modifications will be made to the GCS specification, and Version 2.2 will be released at the end of the transitional software requirements development phase. The library DISK$HOKIE:[GCS.CMS_OLD.DO178A.SPEC] contains Versions 1.0 through 2.1 of the GCS specification as received from RTI. The library DISK$HOKIE:[GCS.CMS.SPEC] starts with Version 2.1 of the GCS specification.

Table D.10: Milestones for Source Code

Milestone	CMS Class Name
Initial Clean Compile (before Code Review)	ICC
Post-Code Review	CR
Post-Requirements-based Test	RBT
Post-Structure-based Test	SBT

D.3.3 Problem and Change Reporting

According to Paragraph 7.2.3 of DO-178B, there should be a mechanism within the software development processes for problem reporting, tracking and corrective action in order to:

- record process non-compliance with software plans and standards,
- record deficiencies of the outputs of the life cycle processes,
- record anomalous behavior of the software products, and
- ensure resolutions of these problems.

An effective problem reporting and tracking system is also extremely important in terms of the project goals, because one of the major objectives of the GCS project is to collect software error data which can be used to help assess the reliability of the resultant software and also assess the effectiveness of different development and verification methods for generating reliable software. In the context of the GCS project, a problem is a question or issue raised for consideration, discussion, or solution regarding some artifact of the software development process. In the software development process, problems can be identified in practically all life cycle data, including the software requirements, software design and code, and test cases.

The tables in Annex A of DO-178B specify that certain life cycle data are classified under Control Category 1 (CC1), which means that the project must provide a formal system of problem reporting, change control, and change review for that data. Other life cycle data are classified under Control Category 2 (CC2), indicating that formal problem reporting and change control procedures are not required for certification. For the purposes of developing an efficient problem and change reporting system, the DO-178B life cycle data has been divided into three different categories: development products (shown in Table D.3); support documentation (shown in Table D.4); and records, results, and reports (shown in Table D.5). The life cycle data in the development products and support documentation categories are all under CC1. A unique problem and change reporting system has been established for each category under CC1.

D.3.3.1 Problem Reporting for Development Products

This section addresses the content and identification of problem reports for the development products, time frame for initiating problem reports, the method of closing problem reports, and the relationship to the change control activity in compliance with Subsection 11.4 of DO-178B. The GCS Problem Report (PR) and Action Report (AR) forms, shown in Figures 1 and 2, respectively, will be used to document any problems and subsequent changes to the development products that arise during the development of the GCS implementations. The PR form is used to capture data concerning a possible problem that is identified during the software development process. The Problem Report contains

- information about when (in the development processes) the problem was identified,
- the configuration identification of the artifact
- a description of the problem (such as non-compliance with project standards or output deficiency), and
- a history log for tracking the progress and resolution of the problem.

All problems are investigated to determine if indeed a fault has been detected, in which case corrective action is taken and properly documented. Each identified fault is traced to determine the source where the fault was introduced. The AR form is used to capture relevant information about the action that is taken in response to a Problem Report. The Action Report will contain the configuration identification of the artifact affected and a description of a change that is made to an artifact in response to the Problem Report. In the case that no change is required in response to the PR, the AR form will contain the justification for not making any changes.

D.3.3.2 Instructions for Problem and Action Reports

In general, a project participant who identifies, in the course of his prescribed activities, something in a development product that may be regarded as a problem (such as a violation of a software requirement or project standard) is responsible for initiating a Problem Report. However, during those verification activities where a Moderator is present, the Moderator will have the authority to determine whether issuing a Problem Report is appropriate. Figure D.3 shows the flow of the problem reporting process, starting with the initiation of a PR to the final signature from the SQA representative indicating that the problem has been resolved. The following procedure, as shown in the flow chart, should be followed. During the development cycle,

1. The initiator of the PR form fills out the form from Section 2 through Section 8. The Continuation form should be used if additional space is required for further explanation.

2. The PR form is given to the SQA representative who assigns a PR number to it and logs this PR as an outstanding PR.
3. The SQA representative keeps the original PR form and gives a copy to the most appropriate member of the development project for examination.
4. The project member receiving a copy of the PR form should examine the appropriate artifact to determine if a change should be made. The response to the PR is made on an Action Report. If one or more changes are necessary, the change(s) are made and Action Reports describing the changes are written. When completing the Action Report, the respondent should contact the SQA representative to get the appropriate AR number. The respondent should refer to the AR number when requesting the appropriate configuration item from the configuration manager. This number should also be placed in the artifact comments when a change has been made. It is also important to make the change at this time.
5. The project member will return the PR form to the SQA representative with either one or more Action Reports. The SQA representative checks that the report(s) are properly filled out and contain an adequate description of the change or an adequate explanation for making no change. At this time the SQA representative may deem it necessary to give a copy of the PR form to a different member of the project. This process may repeat itself until the SQA representative decides no further changes are necessary without further review by the PR initiator. It is the responsibility of the SQA representative to make sure that each problem is properly traced back to its origin. The SQA representative notes the sequence of the PR distribution in the history section of the original PR form.
6. When all parties have responded to the PR, the SQA representative gives the original PR form and the Action Report(s) to the initiator. If the initiator feels that the problem is resolved, he signs off on the PR form and gives it to the SQA representative for final approval. If the initiator does not feel the problem is resolved, the initiator can seek further changes through the SQA representative. The SQA representative should make note of any problems in the History Log.
7. The SQA representative then reviews the Problem and Action Reports. If further modification is deemed necessary, the reports should be distributed for further action. Upon final approval of the reports, the SQA representative notes the total number of changes and the total number of no changes on the original PR form and signs and dates it signifying resolution of the problem. The SQA representative then indicates the resolution of this PR on the master list of PRs. The Action Report forms should be attached to the original PR form.
8. The SQA representative should notify the configuration manager that the configuration items that were modified have been approved and should be replaced in the CMS libraries.

D.3.3.3 Number System for the Problem and Action Reports

This section discusses the identification system for the Problem and Action Reports. Each GCS implementation will have its own set of Problem and Action Reports for the development products. The identification numbers for the Problem and Action Reports are of the form:

a.b where
a is the chronological number of the Problem Report
b is the chronological number of the action made in response to Problem Report "a"

The Problem Reports will be numbered: 1.0

2.0
3.0
...

The subsequent responses made (via Action Reports) to a Problem Report would be numbered:

<PR#>.1
<PR#>.2
<PR#>.3
...

For example, consider the third problem found with an implementation and suppose that 2 responses are made to the Problem Report. The Problem Report number would be 3.0 and the Action Report numbers would be 3.1 and 3.2

See Section D.7 for instructions on how to complete the Problem Report form, the Action Report form, the Support Documentation Change Report form, and the Continuation form.

D.3.3.4 Problem Reporting for Support Documentation

The problem and change reporting for the support documentation will be conducted through the use of Support Documentation Change Reports. Although the Support Documentation Change Report form shown in Figure D.4 does not capture as much detailed information as the Problem Report, this form does capture the information necessary to comply with Paragraph 7.2.3 of DO-178B. Once a support document enters the configuration management system, all further changes to that document will be controlled through the Support Documentation Change Reports; that is, all changes to any support documentation must be accompanied by an approved Support Documentation Change Report. Each configuration item that is a part of the support documentation will have its own set of change reports. The SQA representative will keep a log of all change reports for each configuration item.

The following procedure, as shown in the flow chart in Figure D.5, should be followed for initiating and completing the Support Documentation Change Report for all support documentation.

1. The author of the support documentation fills out Sections 1, 2, 4, and 5 of the Support Documentation Change Report form. The Continuation form should be used if additional space is required for further explanation.
2. The form is given to the SQA representative who determines if the change request is reasonable and assigns a modification number to the report if the request is approved.
3. The SQA representative logs this as an outstanding change report for the particular configuration item and returns the form to the author to implement the change.
4. The author requests to reserve the affected configuration item and must refer to the modification number when making the request.
5. The author implements the requested change to the configuration item.
6. When the modification is completed, the author completes Section 6 of the form, places the configuration item in the appropriate place for the configuration manager to retrieve, and returns the form to the SQA representative for review.

7. The SQA then reviews the change for consistency and compliance with project plans and standards. If the change is not acceptable, the SQA representative can work with the author to implement the necessary modifications. The project leader will arbitrate if the author and SQA representative cannot reach consensus.

8. When the change has been completed and approved by the SQA representative, the SQA representative should notify the configuration manager that the configuration item that was modified has been approved and should be replaced in the appropriate CMS library.

D.3.4 Change Control

According to the DO-178B guidelines, change control is "the systematic evaluation, coordination, approval or disapproval, and implementation of approved changes in the configuration of a configuration item after formal establishment of its configuration identification or to baselines after their establishment" (ref. D.2). The objective of the change control activity is to provide for recording, evaluation, resolution and approval of changes throughout the software life cycle. Change control will preserve the integrity of the configuration items and baselines by providing protection against change. Change control ensures that any change to a configuration item requires a change to its configuration identification. Changes to baselines and configuration items under change control should be recorded, approved and tracked.

GCS Problem Report

page 1 of ____

1. PR #:	2. Planet:	3. Discovery Date:	4. Initiator & Role:

5. Activity at Discovery:

Development Phases \ Activity	Design Review	Code Review	Reading Code	Reading Specification	Test Readiness Review	Test Completion Review	Test Case Creation	Test Case Execution	~ #	Other
Design		■	■		■	■	■	■		
Code	■				■	■	■	■		
Unit Testing										
Functional	■	■								
Structural	■	■								
Subframe Testing	■	■								
Frame Testing	■	■								
Top-Level Simulator Integration Testing	■	■								

6. Description of Problem:

7. Artifact Identification:

□ Design Description □ Support Documentation ______

□ Source Code □ Other ______

□ Executable Object Code ______

8. Test Case Identification:

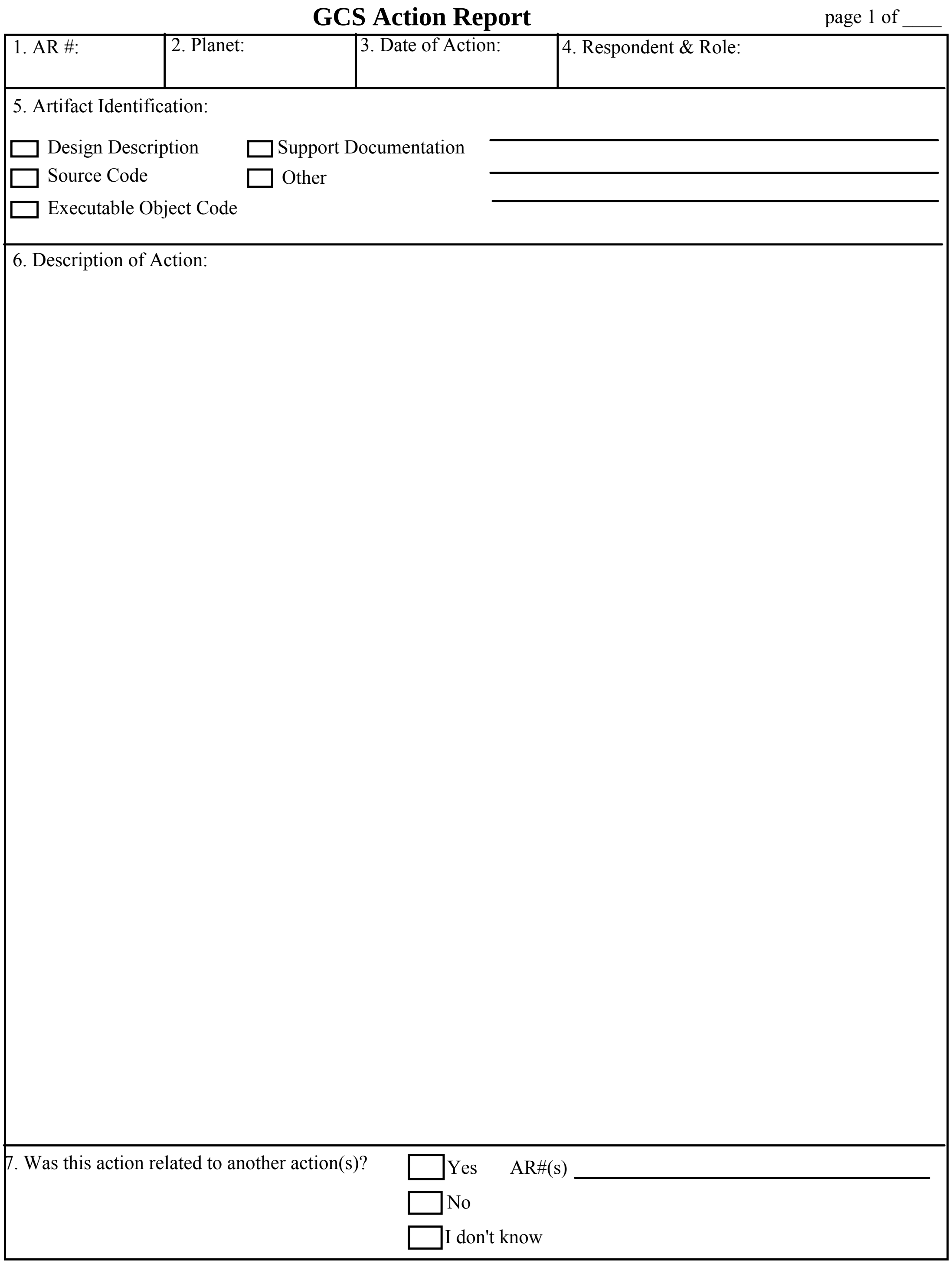
9. History Log:

Date To	Date From	Person	Comments	AR#

10. Total # of Changes: □ 11. Total # of No Changes: □

12. Initiator Signature & Date	13. SQA Signature & Date

Figure D.1. GCS Problem Report Form



GCS Action Report

page 1 of ____

1. AR #:	2. Planet:	3. Date of Action:	4. Respondent & Role:

5. Artifact Identification:

□ Design Description □ Support Documentation ________________

□ Source Code □ Other ________________

□ Executable Object Code ________________

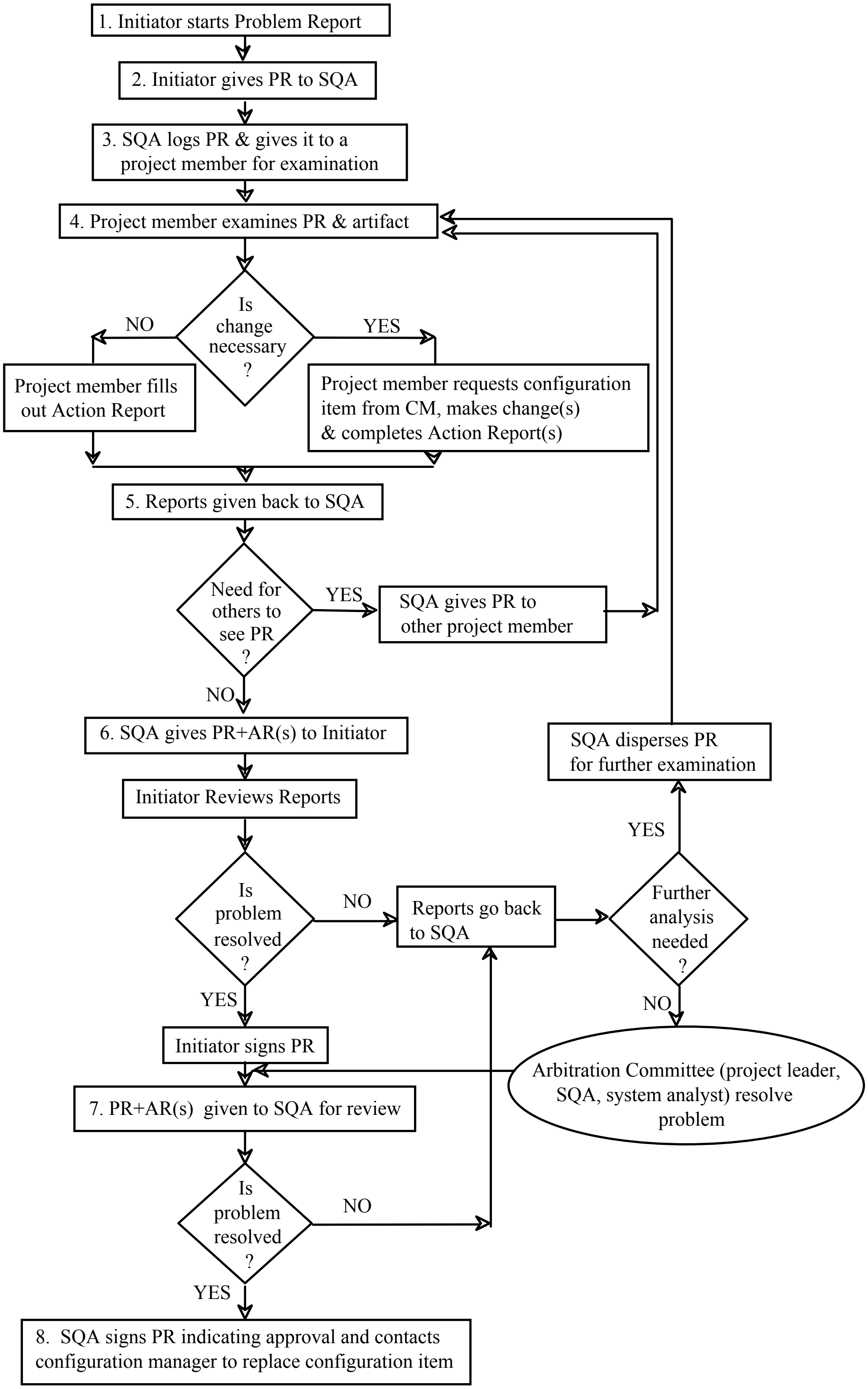
6. Description of Action:

7. Was this action related to another action(s)? □ Yes AR#(s) ________________

□ No

□ I don't know

Figure D.2. GCS Action Report Form



1. Initiator starts Problem Report
2. Initiator gives PR to SQA
3. SQA logs PR & gives it to a project member for examination
4. Project member examines PR & artifact
Is change necessary ?
NO
YES
Project member fills out Action Report
Project member requests configuration item from CM, makes change(s) & completes Action Report(s)
5. Reports given back to SQA
Need for others to see PR ?
YES
SQA gives PR to other project member
NO
6. SQA gives PR+AR(s) to Initiator
SQA disperses PR for further examination
Initiator Reviews Reports
YES
Is problem resolved ?
NO
Reports go back to SQA
Further analysis needed ?
YES
NO
Initiator signs PR
Arbitration Committee (project leader, SQA, system analyst) resolve problem
7. PR+AR(s) given to SQA for review
Is problem resolved ?
NO
YES
8. SQA signs PR indicating approval and contacts configuration manager to replace configuration item

Figure D.3. Flow of Problem Reporting Process for the Development Products

Support Documentation Change Report page 1 of ____

1. Configuration Item:	2. Date:	3. Modification #:
4. Part of Configuration Item Affected:		
5. Reason for Modification:		
6. Modification		
7. SQA Signature & Date:		

Figure D.4. Support Documentation Change Report Form

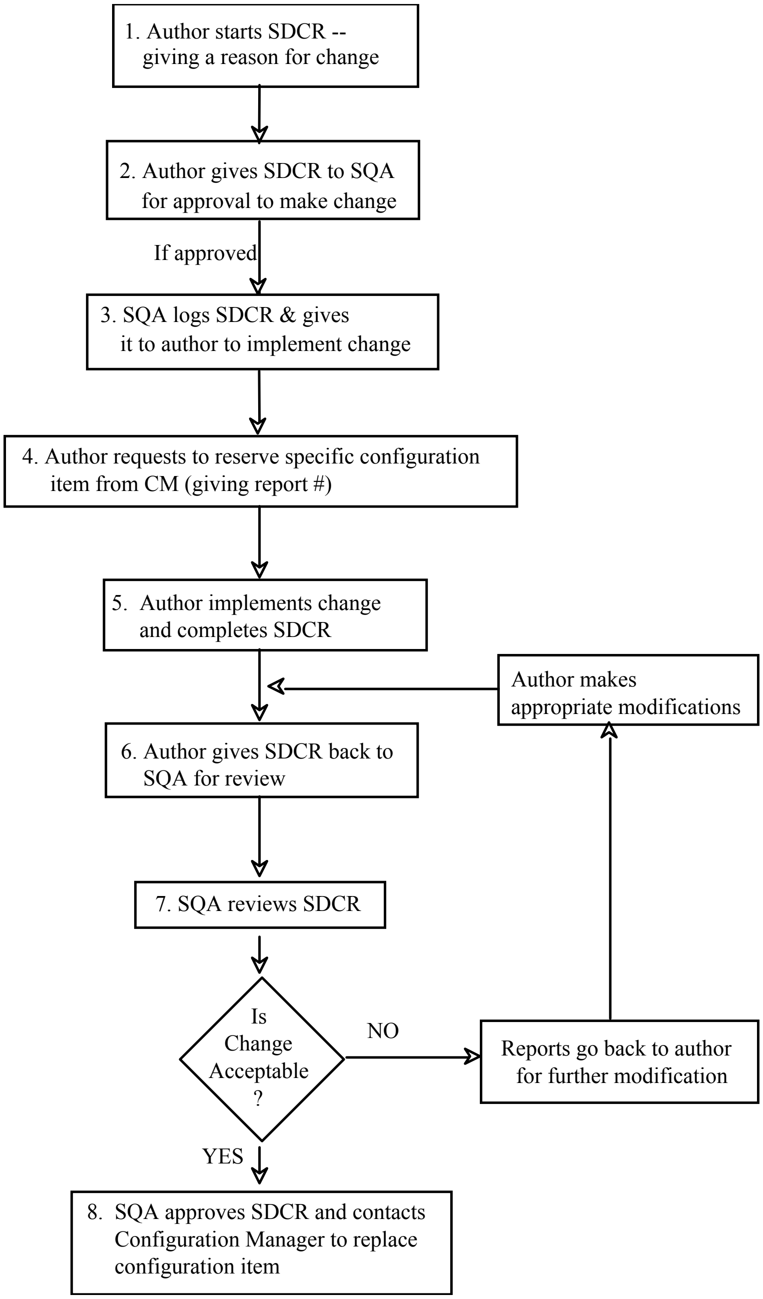

1. Author starts SDCR -- giving a reason for change
2. Author gives SDCR to SQA for approval to make change
If approved
3. SQA logs SDCR & gives it to author to implement change
4. Author requests to reserve specific configuration item from CM (giving report #)
5. Author implements change and completes SDCR
Author makes appropriate modifications
6. Author gives SDCR back to SQA for review
7. SQA reviews SDCR
Is Change Acceptable ?
NO
Reports go back to author for further modification
YES
8. SQA approves SDCR and contacts Configuration Manager to replace configuration item

Figure D.5. Flow of Change Reporting Process for the Support Documentation

The support equipment and software used in this project are discussed in the *Software Life Cycle Environment Configuration Index*. Changes to these tools, in the form of version updates, must be approved by the project leader. Procedures for ensuring that a tool change does not adversely affect the development process, testing process, or any existing design, code, or other document vary with the tool and are developed as needed. Approval of the procedure by the project leader is required prior to implementing the procedure.

The procedure for controlling the effects of changing a tool vary with the importance of a tool as it relates to the actual design or code. A FORTRAN compiler change requires full regression testing of the code to look for changes in output. This involves re-running all of the requirements-based and structural test cases. Once the tests have been re-run, the output must be approved by the SQA representative. A change to the CMS tool would require an intensive review of the documented changes and several trials to ensure that the changes indeed work properly and that the libraries are not corrupted with the use of the new tool. There are no plans to upgrade the CMS tool or the Team*work* tool during the project life cycle.

Change control of the development products and support documentation will follow the guidelines of problem and change reporting. Some changes recorded on PRs and SDCRs may effect other software life cycle data. If it does, the life cycle data will be changed according to the problem and change reporting process and will go through the SQA representative for approval as necessary. All changes will be in accordance with DO-178B. For example, if a programmer needs to reserve his source code, he should provide the configuration manager with the element name and the PR#. The PR# is needed so that this information may be recorded in the CMS library history log. If a support document needs to be modified, a formal modification number and the configuration item name should be provided to the configuration manager to reserve the item. Before the item can be replaced in the CMS library, the SQA representative must sign the SDCR form and contact the configuration manager to replace the item. To maintain the integrity of the CMS libraries, only the configuration manager and the project leader will have access to them.

The GCS simulator's change control procedures are handled differently than the off-the-shelf tools. The same version of the GCS simulator will be used throughout the development of the implementations. Because this tool directly affects the output from the testing, any change to the simulator would require regression testing and approval by the project leader.

D.3.5 Change Review

The objective of the change review activity is to ensure problems and changes are assessed, approved or disapproved, approved changes are implemented, and feedback is provided to affected processes through problem reporting and change control methods defined during the software planning process.

The change review process for development products and support documentation is directly related to the problem and change reporting process. Once these items have been placed under configuration control, the problem reporting procedures will be the only mechanism for initiating changes to these items. For development products, there will be a Problem Report associated with all changes. Each PR will be reviewed by the SQA representative to ensure that the change is necessary. If the change has been approved and implemented, the SQA representative consults with the initiator of the PR to assure the problem has been resolved. If no change has been made to a GCS artifact, an Action Report is filled out to explain the reason for no change. Once a PR has been initiated and approved, the project member responsible for the change needs to contact the configuration manager to reserve the artifact affected. The requester needs to supply the configuration manager with the element name(s) and the PR#; this will assure that the configuration item(s) being requested has been configuration identified (reference the flow chart for the Problem Reporting process, Figure D.3). If a programmer or verification analyst is requesting a GCS artifact, he must use VAXnotes to communicate with the configuration manager (see the *Software Development Standards* for communication protocol). If support documentation requires a change, a Support Documentation Change Report (SDCR) form must be filled out. The SQA representative will assign a formal modification number to the SDCR form and then the document may be requested from the configuration manager; the requester should provide the configuration item name and formal modification number to reserve the item. The SQA representative must sign the SDCR form and contact the configuration manager before the configuration item may be replaced in the CMS library.

The system safety assessment process is beyond the scope of the GCS project and will therefore not be addressed.

D.3.6 Configuration Status Accounting

The objective of the status accounting activity is to provide data for the configuration management of software life cycle processes with respect to configuration identification, baselines, problem reports, and change control.

The configuration manager will keep binders for each implementations' PRs signed off by the SQA representative; any Action Reports associated with a PR will be attached to that PR. These binders will be labeled "Problem Reports for *Planet*" and will be located in the Configuration Manager's office in Building 1220 of NASA Langley. A binder labeled "Change Reports" will also be kept by the configuration manager. This binder will contain all Support Documentation Change Reports signed off by the SQA representative.

A binder with the status of each CMS library will also be kept. The log kept in this binder will contain the CMS library name, the date an item is acted on, the action performed on the item, the name(s) of the artifact(s) requested, the requester's user name, and a remark as to why the element(s) is being reserved. The log sheets in the binders have the following format where XXX is the specific library name:

LIBRARY: DISK$HOKIE:[GCS.CMS.XXX]

Date	**Action**	**Element**	**Requester (initials)**	**Remarks**

.
.
.

In case of an unusual occurrence, a "*" will be entered in the log with an explanation of the occurrence. The binder will contain dividers to separate configuration items. The dividers will be labeled with only the last directory name of the CMS library, except in the cases where the library is planet specific. For example, for the library DISK$HOKIE:[GCS.CMS.SPEC], the divider will be labeled "spec". For the Pluto Source Code library, DISK$HOKIE:[GCS.CMS.SOURCE_CODE.PLUTO], the divider will be labeled "source_code.pluto". This binder will be labeled "CM Status Log" and will be kept in the Configuration Manager's office in Building 1220 of NASA Langley.

D.3.7 Archive, Retrieval and Release

The objective of the archive and retrieval activity is to ensure that the software life cycle data associated with the GCS project can be retrieved in case of a need to duplicate, regenerate, retest, or modify the software product. The objective of the release activity is to ensure that only authorized software is used, in addition to being archived and retrievable.

All of the software life cycle data associated with the GCS project are retrievable by contacting the project leader. The project leader will then contact the configuration manager and request a copy of all life cycle data needed for delivery to the requester.

The items under CMS configuration management for the GCS project are kept on-line on a DEC VAX cluster, running the VMS operating system. The following describes the backups of this system to ensure the integrity of the data:

- a full backup of all items located on the system will be performed once a week;
- a duplicate copy will be made of each full backup tape and stored in a physically separate archive to minimize the risk of loss in the event of a disaster;
- no unauthorized changes can be made to any of the backup tapes;
- all tapes will be verified for regeneration errors (by using the backup/verify command);
- incremental backups are run on a daily basis for a four week cycle to lessen the probability of losing any information.

After a full backup has been performed, a duplicate copy of the tape will be made. The duplicate tapes are verified when copied to ensure that accurate copies have been produced. The components of the GCS project will be authorized for release to the certification authority after the integration testing has been completed. All data will be archived for future references.

Since Problem Reports are not kept electronically, they will be archived in a binder by the configuration manager. Only PRs that have been approved and signed by the SQA representative will be archived. There will be a separate binder labeled "Problem Reports for *Planet*" for each implementation. See the section on "Configuration Status Accounting" for more details on the PR binders.

D.3.8 Software Load Control

The objective of the software load control activity is to ensure that the Executable Object Code is loaded into the airborne system or equipment with appropriate safeguards. This activity is non-applicable to the Guidance and Control Software Project since the implementations will not be included in a "real" hardware system intended for space flight. Instead, the software will run with the GCS simulator which is located on the microVAX 3800 computer system with the rest of the software product.

D.4 Transition Criteria

This section defines the transition criteria by specifying which items will have configuration identification and when they enter the configuration management process. The software life cycle data that requires approval by the project leader will enter the configuration management process after approval has been received. The following processes are performed concurrently with the software development process throughout the software life cycle:

- the software verification process,
- the software configuration management process, and
- the software quality assurance process.

Each software life cycle process performs activities on inputs to produce outputs. A process may produce feedback to other processes and receive feedback from others. Feedback includes how information is recognized, controlled, and resolved by the receiving process. For example, a verification activity (such as executing a test case) may identify that a problem exists and then the problem reporting procedures bring it to the attention of other processes. Many different processes may be effected by the resolution of the problem and will therefore need to be modified and re-approved.

The support documents enter CMS when the initial draft of the document has been approved by the project leader, with the exception of the GCS specification. The GCS specification enters the configuration management process at Version 2.1 received from RTI after being converted to a Microsoft Word document. The design descriptions enter the configuration management process at the Post-Code Review version received from RTI. Each programmer is responsible for modifying the original design of his implementation (developed at RTI) so that the new design meets the requirements of Version 2.2 of the GCS specification and the development standards. After the design phase has been completed, the source code and executable object code are generated and then enter the configuration management process after the source code cleanly compiles and is ready for initial code review. Table D.11 shows the transition criterion for entering the configuration management process for the project data.

Table D.11. Transition Criterion for Project Data

Configuration Item	**Transition Criterion**
support documentation	initial draft approved by SQA
GCS Specification	GCS Specification version 2.1 received from RTI
Design Description	Post-Code Review version received from RTI
Source Code	Design Phase Completion
Executable Object Code	Design Phase Completion
Verification Results	after first verification activity (Design Review)
Problem and Action Reports	after first verification activity (Design Review)
Configuration Management Records	initial CMS activities
Software Quality Assurance Records	after first verification activity (Design Review)

D.5 SCM Data

The SCM data is the life cycle data produced by the configuration management process. This data includes SCM Records, the Software Configuration Index, and the Software Life Cycle Environment Configuration Index.

The results of the SCM process activities are recorded in SCM Records. Table D.1 showed the configuration identification list for all of the life cycle data that will be maintained in the CMS libraries. The remaining life cycle data will be kept either in paper or electronic form. Baselines for the source code will be established after each review or test phase and baselines for documentation will be established as needed. CMS keeps a history file that logs all changes made to the libraries for the life cycle data; this information will also be kept in the CM Status Log binder.

The *Software Configuration Index* identifies:

- the configuration of the GCS project,
- the executable object code and instructions for building it,
- each source code component,
- software life cycle data,
- archive and release media, and
- procedures used to recover the software for regeneration, testing, or modification.

The Software Life Cycle Environment Configuration Index identifies the development environment:

- the software life cycle operating system,
- the software development tools,
- the test environment used to verify the GCS project, and
- qualified tools and their associated qualifications.

D.6 Supplier Control

The supplier control is the means of applying the software configuration management process requirements to the sub-tier suppliers. This is non-applicable for the Guidance and Control Software Project.

D.7 Completing the Problem Report Form

In this section, instructions for completing the fields of the PR form are stated. Specific instructions or further explanation for each section of the PR form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of attached pages. As many Continuation forms as necessary may be used.

1. **PR#**: to be assigned by the SQA representative
2. **Planet:** the name of the planet in whose development process this problem was identified

3. **Discovery Date**: date when this problem was identified. It is important to issue a PR form at the time a problem is identified.

4. **Initiator & Role**: name of the person who has identified the problem and the role (programmer, verification analyst, SQA representative, or system analyst) that person is fulfilling at the time of problem identification.

5. **Activity at Discovery**: The development cycle for each GCS implementation can be decomposed into 6 distinct phases. In this section, indicate the phase by placing an **X** in the appropriate box that corresponds to the development phase in which this problem was identified and the specific activity that was being performed at that time. If the Other category is appropriate, please put an explanation in Section b of the Continuation form.

6. **Description of Problem**: Provide an adequate description of the issue in question.

7. **Artifact Identification**: Check the box that corresponds to the artifact under consideration when the problem was identified. The label for the configuration item should be given along with the information in Table D.12 for each artifact. If a PR is being generated because the actual results from the execution of a test case did not agree with the expected results, the initial artifact under consideration would be the executable object code. The test case that surfaced the anomalous behavior would be identified in Section 8. If more space is needed, use Section b of the Continuation form.

8. **Test Case Identification**: If the failure of a test case is the reason for initiating this PR, fill in the appropriate test case number, including its configuration item label, element name(s), and generation #; otherwise, indicate Not Applicable (N/A).

9. **History Log**: to be filled in by the SQA representative. The SQA representative should log the sequence of dispersals of the PR, logging all ARs related to the PR and noting date of issuance, date of return, and the person receiving the PR form. The SQA representative should also note any anomalies in the resolution of the problem, such as disagreements in resolution between the initiator and the person making the change.

10. **Total # of Changes**: to be filled in by the SQA representative when all Action Reports are closed and the problem has been resolved. A total of 0 indicates that no change was made.

Table D.12. Information for Artifact Identification

Artifact	Information
Design Description	diagram, P-Spec #, C-Spec #, or M-Spec #
Source Code	element name & generation #
Executable Object Code	element name & generation #
Support Documentation	specific chapter, section, and table or figure reference, as appropriate
Other	be as specific as possible

11. **Total # of No Changes**: to be filled in by the SQA representative when all Action Reports are closed and the problem has been resolved.

12. **Initiator Signature & Date**: The person who initiates the PR should sign and date the original PR form here when the problem has been resolved.

13. **SQA Signature & Date**: After checking that the problem is satisfactorily resolved and all necessary changes have been properly made, the SQA representative should sign and date the original PR form indicating closure of this PR.

D.8 Completing the Action Report Form

In this section, instructions for completing the fields of the AR form are stated. Specific instructions or further explanation for each section of the AR form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

1. **AR#**: to be assigned by the SQA representative. The respondent should contact the SQA representative to get the appropriate AR number. When a change is indicated, the AR# can be incorporated in the comments which describe this change in the code or design.
2. **Planet:** the name of the planet associated with the person making this action.
3. **Date of Action**: date when this action was taken. In case of changes, it is important to complete the AR form at the time a change is being made.
4. **Respondent & Role**: name of the person who is making the response and his role (programmer, verification analyst, SQA representative, or system analyst).
5. **Artifact Identification**: Check the box that corresponds to the artifact in question. The information in Table D.6 should be specified for each artifact. In case of responses made to the support documentation, the label for the configuration item should be cited. If more space is needed, use Section b of the Continuation form.
6. **Description of Action**: provide a general description of the change that was made or an explanation of why no change is necessary. In case of responses made to the support documentation, the appropriate modification number from the Support Documentation Report Form should be cited.
7. **Was this action related to another action(s)?:** Check the appropriate box to indicate whether this action is related to another action. If yes, indicate the relevant AR#(s).

D.9 Completing the Support Documentation Change Report Form

In this section, instructions for completing the fields of the Support Documentation Change Report form are stated. Specific instructions or further explanation for each section of the Support Documentation Change Report form are given below.

page 1 of __: Fill in the total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

1. **Configuration Item:** the label for the configuration item that needs to be changed.
2. **Date:** date that this change report is being initiated.

3. **Modification #:** to be provided by the SQA representative. The author should give the form to the SQA representative to get the number and corresponding authorization to implement the change.
4. **Part of the Configuration Item Affected:** describe the location of the proposed change. Chapter and section references should be included as appropriate.
5. **Reason for Modification:** explanation detailing why the configuration item should be changed.
6. **Modification:** description of the change including the following information as appropriate: original text (that is to be changed), action (such as deletion, addition, or modification), and modified text (the correct text to be inserted). If substantial changes are made, the affected pages should be attached to the form.
7. **SQA Signature and Date:** After checking that the change is acceptable and has been properly made, the SQA representative should sign and date the form indicating approval of this change.

D.10 Completing the Continuation Form

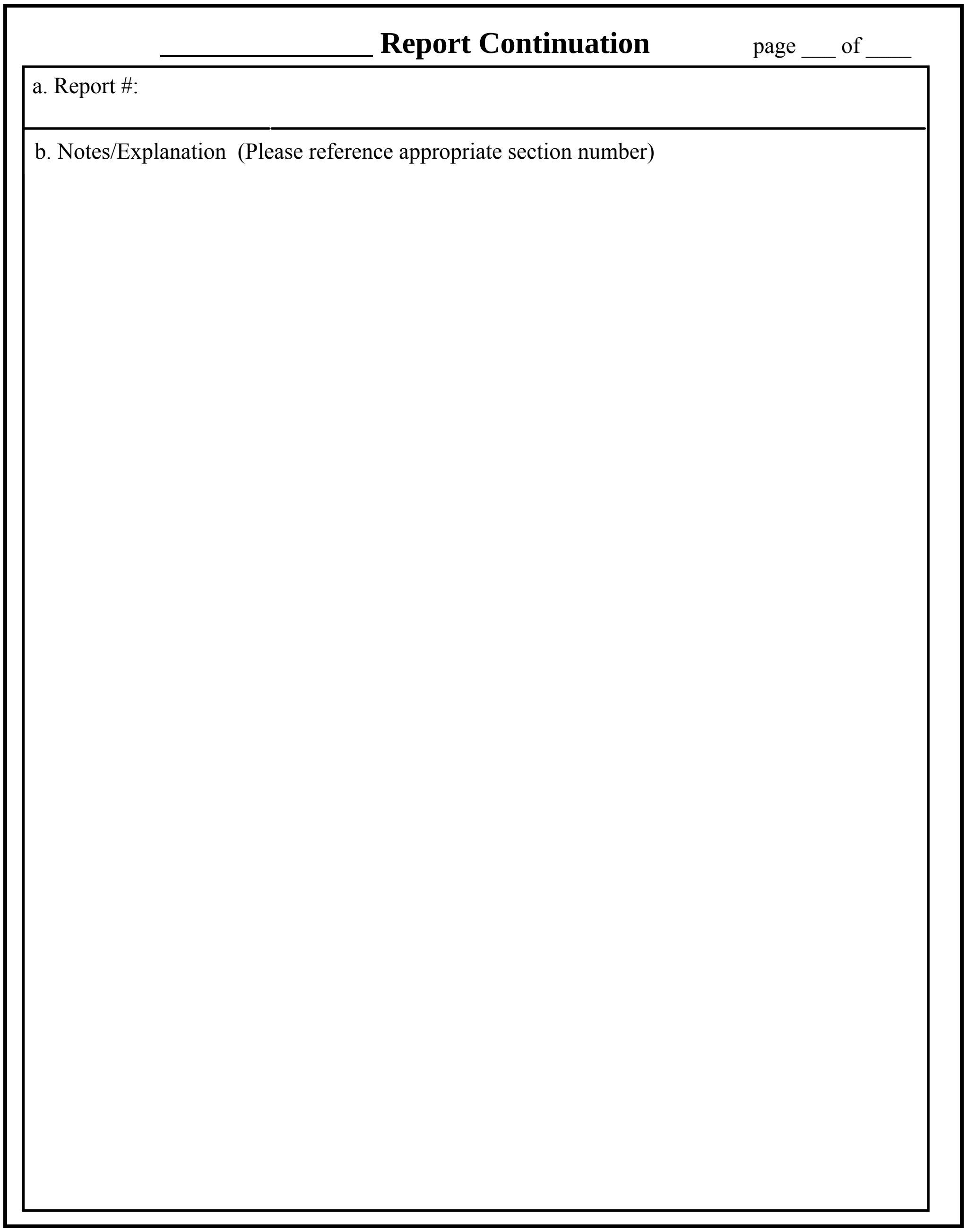
The Continuation Form provides extra space in addition to the PR, AR, and Support Documentation Change Report forms. Figure D.6 shows the Continuation Form. Specific instructions or further explanation for each section of the Continuation form are provided below.

______________ Report Continuation: Fill in the blank with the name of the form that is being continued.

page__ of __: Fill in the page number and total number of pages on each form to help avoid the loss of any attached pages. As many Continuation forms as necessary may be used.

a. **Report #**: the number of the report that is being continued

b. **Notes/Explanation**: This section is to be used to continue comments or descriptions from any section of a report.



______________ **Report Continuation** page ___ of ____

a. Report #:

b. Notes/Explanation (Please reference appropriate section number)

Figure D.6. Report Continuation Form

D.11 References

D.1. Finelli, George B.: Results of software error-data experiments. In *AIAA/AHS/ASEE Aircraft Design Systems and Operations Conference,* Atlanta, GA, September 1988.

D.2. RTCA Special Committee 167. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178B, Requirements and Technical Concepts for Aviation, December 1992.

D.3. Uczekaj, John and Hughes, Banni: Tailoring Configuration Management Tools for Development of Avionics Software.

D.4. *Guide to VAX DEC/Code Management System.* Digital Equipment Corporation, Maynard, Massachusetts, April 1987.

D.5. Team*work* Environment Reference Manual. Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1991.

D.6. Team*work*/SA Team*work*/RT User's Guide. Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1991.

D.7. Team*work*/SD User's Guide. Cadre Technologies, Inc., Providence, Rhode Island, Release 4.0, 1991.

Appendix E: Software Quality Assurance Plan for the Guidance and Control Software Project

Author, Kelly J. Hayhurst, NASA Langley Research Center

This document was produced as part of Guidance and Control Software (GCS) Project conducted at NASA Langley Research Center. Although some of the requirements for the Guidance and Control Software application were derived from the NASA Viking Mission to Mars, this document does not contain data from an actual NASA mission.

E. Contents

E.1 Introduction

As described in the Requirements and Technical Concepts for Aviation RTCA/DO-178B guidelines, "Software Considerations in Airborne Systems and Equipment Certification," (ref. E.1) the Software Quality Assurance (SQA) process provides evidence that the software life cycle processes satisfy their objectives and that the resultant software conforms to its requirements. The primary means that SQA provides this evidence is by assuring that the software life cycle processes are performed in compliance with the approved software plans and standards. For the Guidance and Control Software (GCS) project, three objectives of SQA process (as given in Table A-9 of Annex A in DO-178B) are to be obtained:

- that software development processes and integral processes comply with approved software plans and standards,
- that the transition criteria for the software life cycle processes are satisfied, and
- that a conformity review of the software product is conducted.

In conducting the SQA process, two other objectives are to be obtained. First, deficiencies in the development and integral processes and project artifacts are to be detected, evaluated, tracked and resolved. Second, assurance is to be provided that the software products and software life cycle data conform to certification requirements. This plan defines the means by which these software quality assurance process objectives will be satisfied. In compliance with section 11.5 of DO-178B, this document contains the following:

- a description of the SQA environment,
- a statement of the SQA authority, responsibility, and independence,
- a description of the SQA activities,
- the transition criteria for entering the SQA process,
- the timing of the SQA activities, and
- a definition of the SQA records to be produced.

E.2 Software Quality Assurance Environment

The GCS project has been undertaken as part of a series of studies conducted by NASA Langley Research Center to characterize the software failure process and provide data on which to base the development of methods for assessing software reliability (ref. E.2). For this project, two implementations of GCS are to be developed based on a common specification of the software requirements and in compliance with the DO-178B guidelines. Additional details about the rationale underlying the study can be found in the *Plan for Software Aspects of Certification* .

The SQA process for the GCS project is administered by one person, hereafter called the SQA representative. Because the scale of the software product is small (the source code is expected to be approximately 2000 non-commented source lines), one individual should be able to conduct all necessary SQA activities. The *Software Quality Assurance Plan* outlines all procedures, controls, and audits to be carried out by the SQA organization to ensure adherence to documented procedures and standards. This plan was written according to the guidelines contained in DO-178B, with the assumption that GCS represents *Level A* software because all GCS functions are classified in the DO-178B *catastrophic* category. All quality assurance activities and reports described in this plan are intended to support this level of software certification.

E.2.1 Organization Responsibilities

For a host of reasons, the SQA representative should be at a level in the hierarchy of the organization equal to or above the project leader. However, due to constraints on project resources, the project leader will also have to perform the function of the SQA representative. Due to the project objective of collecting data from a life cycle process that complies with DO-178B, the project leader/SQA representative will be compelled to faithfully conduct the SQA activities and will not have conflicting priorities such as real industrial schedules and deadlines to meet. The organizational environment for GCS is shown in Figure E.1.

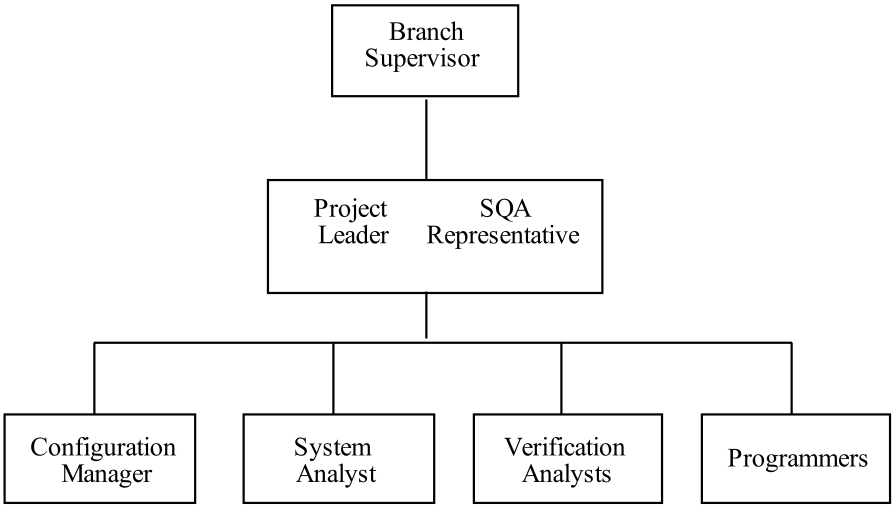


Figure E.1. GCS Project Organizational Chart

One programmer and one tester are assigned as a team to independently develop and verify each GCS implementation. All GCS implementations will independently undergo the same software development processes, as well as the software verification process. Due to the experimental nature of this project and resource constraints, the software integral processes of Software Configuration Management and SQA will be administered independently across the implementations, but the systems and individuals used to carry out these processes will be the same. For example, one configuration management system will store all data items for all implementations, one person will do configuration management for all implementations, and one person will do SQA for all implementations. Further, there will not be a certification liaison process for the GCS project.

Because the entire project only consists of seven people who are physically located at Langley Research Center, most of the communication, in addition to the project documentation, among project members will be through project meetings. Electronic mail will be used for informal communication. *The Software Development Standards* provides a more detailed discussion of communication protocol for project participants.

E.2.2 Scope and Organization of the SQA Plan

This plan is largely organized by life-cycle phase. For the GCS project, there are four development processes that follow a modified waterfall life cycle model: software requirements, design, code, and integration. The *Plan for Software Aspects of Certification* provides more detail on the software development plan. The development of the software requirements, however, is beyond the scope of this plan with respect to compliance with DO-178B. For each of the other processes, this plan lists the life cycle data that is produced, the associated verification activities, all applicable and documented standards and procedures (including procedures governing conduct of verification activities), and the SQA representative's role in ensuring adherence to those standards and procedures.

For this project, the integration phase includes two types of testing: requirements-based testing (at the functional unit, subframe, frame, and trajectory levels) and structure-based testing. Conduct of these tests is governed by the *Software Verification Plan*. For each phase in integration process, this plan gives a brief description of the testing to be conducted and of the applicable policies contained in the *Software Verification Plan*. There is also a description of the test readiness and test completion reviews to be conducted by the SQA representative.

The SQA representative is responsible for ensuring that all problems identified during the various verification activities are documented and corrected and that all change control procedures are followed. This plan contains a section on problem reporting and correction as well as a section on software configuration management.

Finally, the SQA representative is responsible for reviewing all deliverable life cycle data for adherence to DO-178B guidelines. The final section of this plan summarizes the set of reports and approvals to be provided by the SQA representative.

E.3 Software Quality Assurance Authority

The SQA representative for the GCS project has the full set of authorities to grant or deny the necessary approvals for all life cycle data. The SQA representative will ensure that the project objective of developing software that is compliant with DO-178B is accomplished.

E.4 Software Quality Assurance Activities

The following sections describe the SQA activities to be performed for each software life cycle including: a brief description of the life cycle process and data, corresponding verification activities, timing and transition criteria, and SQA actions.

E.4.1 Requirements Process

As described in the *Plan for Software Aspects of Certification*, there is no real guidance and control system to be developed nor documentation of real system requirements. The GCS project is solely a research effort to investigate the faults that occur in the development and operation of software. The GCS implementations will only be executed in a simulated operational environment to collect software failure data. Consequently, the GCS project started with the definition of software requirements for a specific component of a guidance and control system, namely the terminal descent phase.

The requirements process for the GCS project basically consists of revising the software requirements previously developed by the Research Triangle Institute (RTI), see the *Plan for*

Software Aspects of Certification for further details. The software requirements are contained in a document entitled *Guidance and Control Software Development Specification,* which is referred to as the GCS specification and serves as the *Software Requirements Data* for this project. The GCS specification will be placed under configuration control (as described in the Software Configuration Management section of this document) with the release of version 2.2. Version 2.2 represents the completion of the modifications made to the GCS specification that was delivered by RTI. Requirement standards to be applied to the modification of the GCS specification are discussed in the *Software Development Standards*.

E.4.1.1 Verification

The software requirements developed at RTI were subjected to a verification process which was outside of the scope of the formal verification procedures carried out by the verification analysts currently assigned to the project. The correctness and completeness of the requirements were verified in two ways: by conducting walk-throughs and peer reviews at RTI and by coding two prototype programs from the requirements. The results of the original requirements review are summarized in the *GCS Development Specification Review Description* (ref. E.3). The project development activities that are intended to comply with the DO-178B guidelines start with the release of version 2.2 of the GCS specification. Prior to the release of version 2.2 of the GCS specification, the current system analyst, project leader, and an in-house consultant will review the requirements; however, no formal review or analysis is planned.

E.4.1.2 Quality Assurance

The initial development and verification of the GCS specification was beyond the scope of the SQA process for this project. Only modifications to the GCS specification after release 2.2, driven by problem reports, are subject to review by the SQA representative. All questions raised by any member of the development team regarding the GCS specification are brought to the system analyst. The system analyst reviews all questions and determines if changes to the GCS specification are required. When changes are deemed necessary, the system analyst submits a Support Documentation Change Report (SDCR) describing the modification to the SQA representative for review and approval. For more details on the change control process see the *Software Configuration Management Plan*.

E.4.1.3 Transition Criteria

Once the project leader has approved version 2.2 of the GCS specification, the GCS specification will be released to the project programmers to begin the design process. No formal SQA record will be issued because this activity is not intended to be within the scope of compliance with DO-178B.

E.4.2 Design Process

The purpose of the design process is to produce a description of the low-level requirements and software architecture from the high-level requirements given in the GCS specification. Each of the independently produced GCS designs will be contained in an implementation specific Design Description document. For example, the design for the Mercury implementation should be entitled *Design Description for Mercury*. These documents will be placed under configuration control prior to the design review. Design standards and guidelines are contained in the document entitled *Software Development Standards*.

E.4.2.1 Verification

For each GCS design, a design review will be conducted to verify that the high-level requirements have been correctly translated into the low-level requirements and software architecture and that the design standards have been followed. The procedures to be followed during the review are outlined in the *Software Verification Plan*. That document also contains the Requirements Traceability Matrix, that is used to verify that all requirements are addressed by the design; the Design Review Checklist, that is used to verify that the design adheres to all applicable standards; and the inspection log, that is used to record any suspected problems.

The design review is divided into an overview meeting and one or more inspection meetings. The overview meeting is used to distribute the design and the procedures and tools for reviewing it and answer any procedural questions. The inspection meetings examine all parts of the design description to determine if there are any problems.

E.4.2.2 Quality Assurance

The SQA representative will serve as the moderator of the design review meetings and will be responsible for ensuring that the Requirements Traceability Matrix is completed during the design process. The SQA representative will also make sure that problem reports are filled out for all cases of missing, excess, or incorrect functionality or for nonconformance to standards. The SQA representative will track all problem reports through to completion.

E.4.2.3 Transition Criteria

Approval from the SQA representative is required before a programmer is permitted to proceed from the design phase to the coding phase. Before this approval is granted, all problems identified during the design review must be corrected. The design changes (or any changes to the GCS specification) made in the course of these corrections must be documented on action reports and approved by the SQA representative.

The SQA representative generates a Design Review Report, which contains a summary of SQA activity and the list of problem and action reports.

E.4.3 Code Process

The purpose of the code process is to implement the low-level requirements and software architecture given in the design description into source code. The GCS Fortran source code and the commands used to compile and link it will be contained in implementation specific documents. For example, the code for the Mercury implementation should be entitled *Source Code for Mercury*. These documents will be placed under configuration control prior to the code review. The compiler and linker commands will be such as to produce a listing and map with the information specified in the *Software Verification Plan.* The listing and map will be used as tools during the code review. Coding standards and guidelines are contained in the document entitled *Software Development Standards*.

E.4.3.1 Verification

The purpose of code reviews is to verify that the source code has properly implemented the low-level requirements and software architecture as specified in the design description and that it meets coding standards. Code reviews are scheduled after all modules (a module consists of a single function or subroutine) have been written and compiled without errors (but not executed).

The procedures to be followed during the review are outlined in the *Software Verification Plan*. That document also contains the Requirements Traceability Matrix, that is used to verify that all requirements are implemented in the source code; the Code Review Checklist, that is used to verify that the code adheres to all applicable standards; and the inspection log, that is used to record any suspected problems.

The code review is divided into an overview meeting and one or more inspection meetings. The overview meeting is used to distribute the code and the procedures and tools for reviewing it and answer any procedural questions. The inspection meetings examine all parts of the source code to determine if there are any problems. A problem report is filled out for all cases of missing, excess, or incorrect functionality or for nonconformance to standards. If it is determined that a problem originates in the design, the programmer is responsible for filling out an action report prior to making the design correction and generating a second action report for the code. All completed problem reports must be approved by the SQA representative.

E.4.3.2 Quality Assurance

The SQA representative will serve as the moderator of all code review meetings and will be responsible for ensuring that the Requirements Traceability Matrix is completed during the code process. The SQA representative will track any problem reports generated.

E.4.3.3 Transition Criteria

Due to the experimental aspects of this project, the programmers are not allowed to execute or test their own code. All problems identified during the Code Review must be corrected and all problem reports completed before the source code is approved for testing in the integration process. The designated verification analyst will conduct all testing of his assigned GCS implementation.

The SQA representative generates a Code Review Report, which contains a summary of SQA activity and the list of problem reports.

E.4.4 Integration Process

Due to the nature of this project, there is no special hardware or additional software needed for integration. The integration process for the GCS project consists of two major types of testing: requirements-based testing (at the functional unit, subframe, frame, and trajectory levels), and structure-based testing. For all phases of the integration process, each verification analyst will be required to execute the appropriate test cases and maintain a test log. A problem report must be filled out whenever the expected results do not match the actual results. The Requirements Traceability Matrix will be used to cross reference the requirements-based test cases to the software requirements.

The programmer is responsible for making code changes to correct all problems identified in problem reports. If it is determined that a problem originates in the design, the programmer must fill out an action report prior to making the design correction and generate a second action report for the corresponding code changes. This second report must be completed by the programmer after the code corrections have been made. All completed problem reports must be approved by the SQA representative.

E.4.4.1 Requirements-Based Testing

For this project, requirements-based testing is equivalent to requirements-based low-level testing as described in DO-178B. Every requirement will be covered by at least one requirements-based test case and these will be listed in the Requirements Traceability Matrix. The two verification analysts will design the requirements-based test cases together primarily using equivalence class partitioning and boundary value analysis techniques. The actual testing of each implementation will be carried out independently by the tester assigned to that implementation. The requirements-based testing will start at the functional unit level and proceed through subframe, frame and trajectory levels.

E.4.4.2 Structure-Based Testing

Upon the completion of requirements-based testing and the correction of all outstanding problem reports, each verification analyst will structurally analyze his source code using the Analysis of Complexity (ACT) tool (ref. E.4) to determine path structure and corresponding decision points in the code. Multiple Condition/Decision Coverage (MC/DC) tables (ref. E.5) will be constructed for all decision points. The requirements-based test cases will be reviewed to determine if any additional test cases are needed to reach 100% MC/DC for each GCS implementation. Any necessary structure-based test cases for each of the GCS implementations will be designed and carried out by the verification analyst assigned to that implementation. The *Software Verification Plan* contains a more detailed description of the structure-based testing.

E.4.4.3 Quality Assurance

Prior to the start of requirements-based testing, the SQA representative is responsible for conducting a test readiness review, where the SQA representative will verify that:

- all test cases are documented, including all inputs and expected results, and placed under configuration control.
- the set of requirements-based test cases meet the coverage criteria outlined in the *Software Verification Plan*. The SQA representative will ensure that the Requirements Traceability Matrix with the identification number of the test case(s) associated with each requirement is completed.

At the conclusion of requirements-based testing, the SQA representative will conduct an informal review of the requirements-based testing results to ensure that all requirements-based test cases ran successfully (that is, all output and expected results matched). When all problem reports issued during requirements-based testing are completed, the verification analyst can proceed with the structure-based testing. Prior to executing any structure-based test cases, each verification analyst must present the MD/DC decision tables, structure graphs of the source case, and test case identification to the SQA representative for review. Once the structure-based test cases are approved and placed under configuration control, the verification analyst can execute the test cases.

At the conclusion of structure-based testing, the SQA representative is responsible for holding a test completion review according to the following procedures:

- The SQA representative will check to ensure that the actual test results are recorded in the test logs.
- The SQA representative will verify that all changes to the test cases (including the addition of new cases) are documented in support documentation change reports.

- The SQA representative will verify that all discrepancies between actual and expected test results have been documented in a problem report and that all problem reports have been completed and approved.
- The SQA representative will produce the Test Completion Review Report signifying approval the completion of the integration process.

E.4.4.4 Transition Criteria

The integration process is considered to be complete when all of the requirements-based and structure-based test cases successfully run and all problem and action reports are completed.

E.5 Problem Reporting and Correction

One of the cornerstones of an effective software quality program is a systematic, disciplined set of procedures for problem reporting and correction. These procedures ensure that all problems are documented, that problem status at any given time can be readily determined, and that all changes to documentation and code resulting from problem correction follow established configuration control procedures. The problem reporting and correction procedures to be used on the GCS project are outlined in this section, and a more detailed description is given in the *Software Configuration Management Plan*.

For the purposes of developing an efficient problem and change reporting system, the GCS project life cycle data has been divided into three different categories: development products (shown in Table E.1); support documentation (shown in Table E.2); and records, results, and reports (shown in Table E.3). The life cycle data in the development products and support documentation categories are all under Control Category 1 (CC1) according to DO-178B; and, the records, results, and reports are under CC2. A unique problem and change reporting system has been established for each category under CC1. The problem reporting system for the development products requires the documentation of more information (to aid in the data analysis for the experiment objectives of the project) than for the support documentation. There is no formal change reporting system for the CC2 items.

Table E.1. CC1 Development Products

Design Description
Source Code
Executable Object Code

Table E.2. CC1 Support Documentation

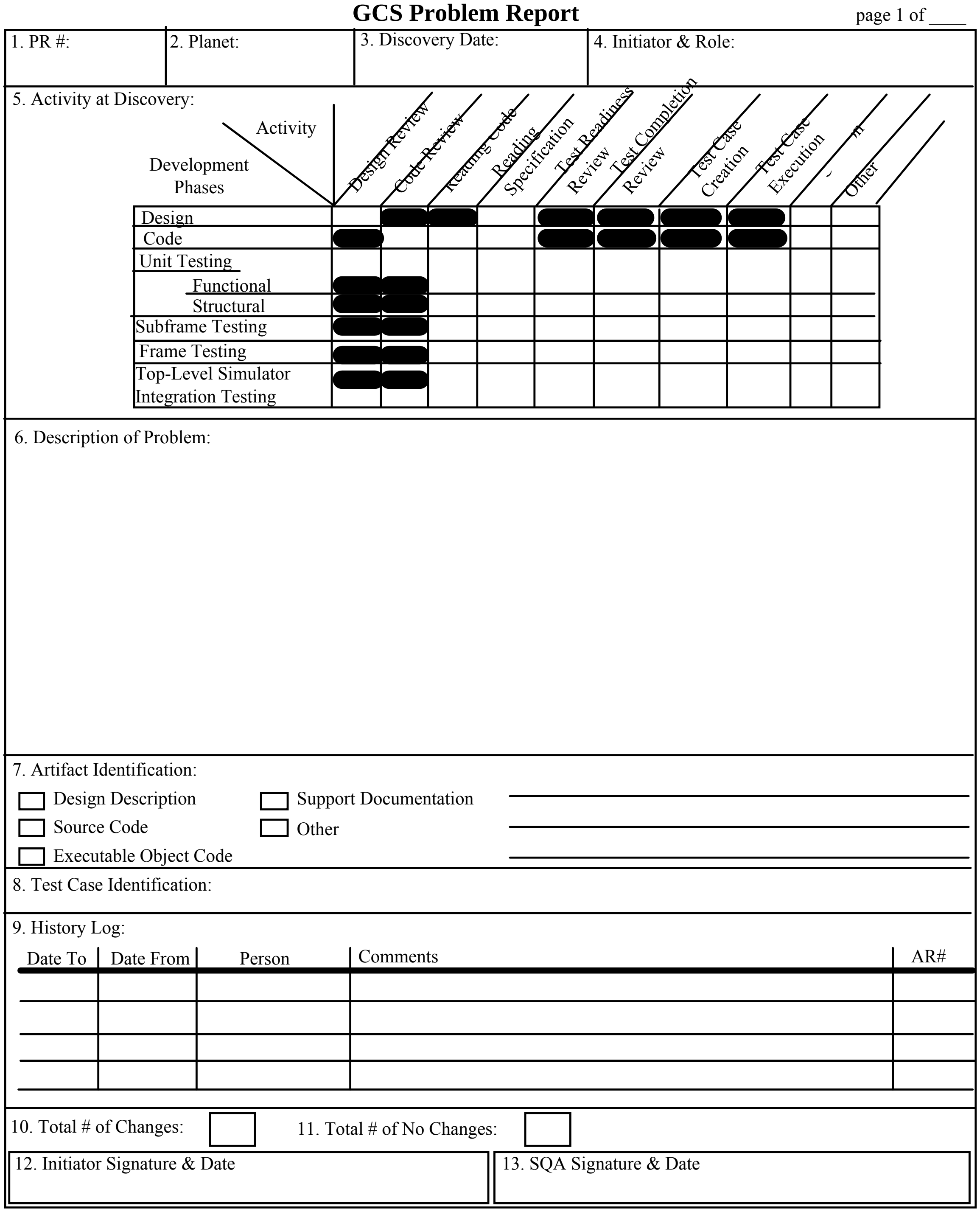
Plan for Software Aspects of Certification
Software Development Plan
Software Requirements Standards
Software Design Standards
Software Code Standards
Software Accomplishment Summary
Software Verification Plan
Software Verification Cases and Procedures
Software Quality Assurance Plan
Software Configuration Management Plan
Software Life Cycle Environment Configuration Index
Software Configuration Index
Software Requirements Data

Table E.3. CC2 Records, Results, and Reports

Software Verification Results
Software Quality Assurance Records
Problem Reports
Software Configuration Management Records

For the development products, a two-form problem and action reporting system will be used. The GCS Problem Report (PR) and Action Report (AR) forms, shown in Figures E.2 and E.3, respectively, will be used to document any problems and subsequent changes to the development products that arise during the development of the GCS implementations. A separate set of PRs and ARs will be kept for each implementation.

In general, the person who identifies a problem is responsible for initiating a problem report. The problem report must be given to the SQA representative who will assign a number to it and assign at least one project member to examine it. Each project member assigned to examine the problem report will generate an action report describing required change(s) or why no change is required. In cases where significant changes are made (e.g., more than 20 lines of source code are changed), the system analyst will be required to review and approve the change. The approval of the SQA representative is needed to complete each problem report. When the resulting change has been approved by the SQA representative, the new version of the development product will be placed in the configuration management system. All problem reports are turned over to the SQA representative for approval, but the configuration manager will store the original reports. For more details on problem reporting and correction see the *Software Configuration Management Plan*.



GCS Problem Report

page 1 of ____

1. PR #:	2. Planet:	3. Discovery Date:	4. Initiator & Role:

5. Activity at Discovery:

Development Phases \ Activity	Design Review	Code Review	Reading Code	Reading Specification	Test Readiness Review	Test Completion Review	Test Case Creation	Test Case Execution		Other
Design		■	■		■	■	■	■		
Code	■				■	■	■	■		
Unit Testing										
Functional	■	■								
Structural	■	■								
Subframe Testing	■	■								
Frame Testing	■	■								
Top-Level Simulator Integration Testing	■	■								

6. Description of Problem:

7. Artifact Identification:

□ Design Description □ Support Documentation ______________________

□ Source Code □ Other ______________________

□ Executable Object Code ______________________

8. Test Case Identification:

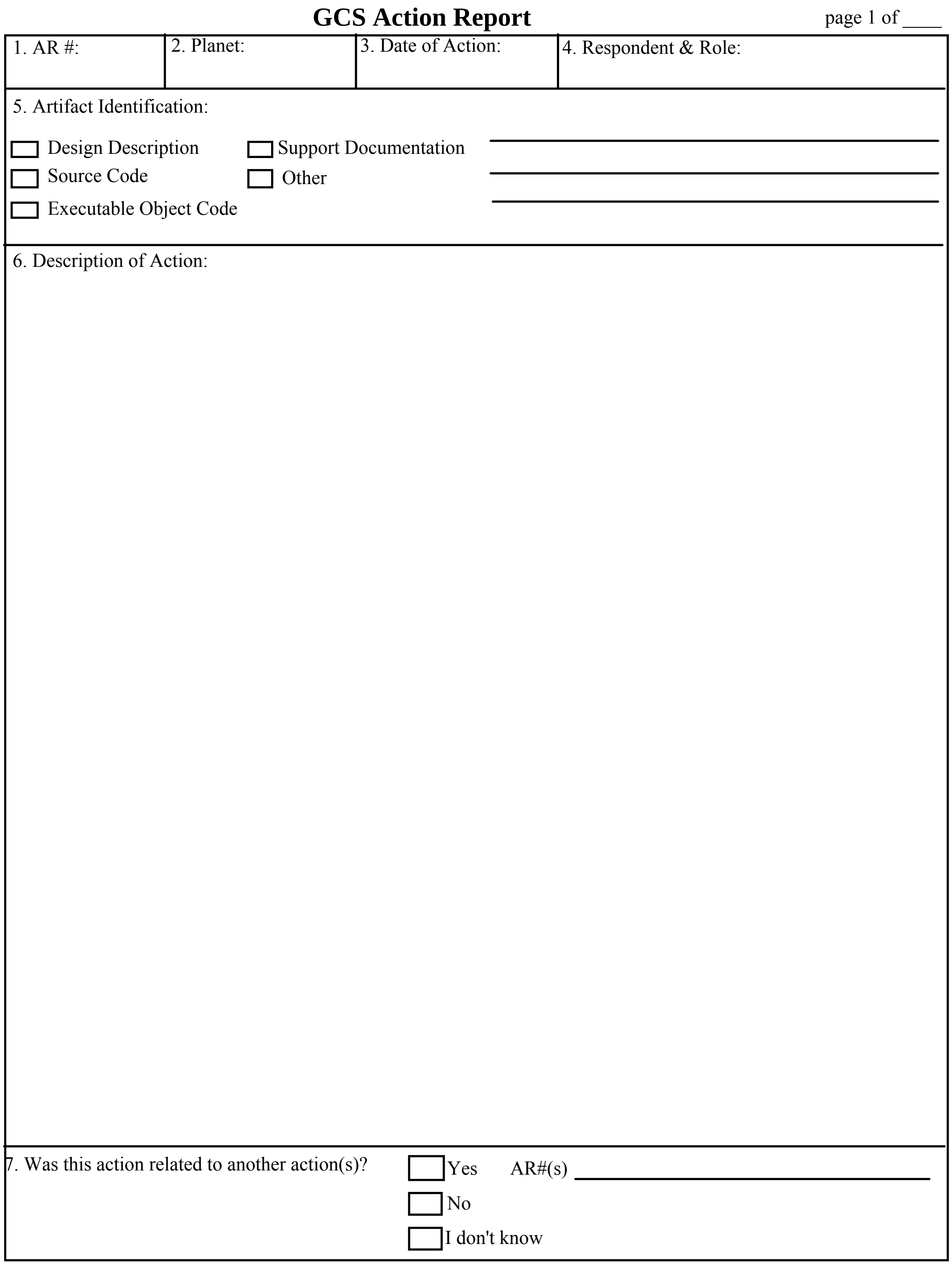
9. History Log:

Date To	Date From	Person	Comments	AR#

10. Total # of Changes: □ 11. Total # of No Changes: □

12. Initiator Signature & Date	13. SQA Signature & Date

Figure E.2. GCS Problem Report Form



GCS Action Report page 1 of ____

1. AR #:	2. Planet:	3. Date of Action:	4. Respondent & Role:

5. Artifact Identification:

□ Design Description □ Support Documentation ____________________

□ Source Code □ Other ____________________

□ Executable Object Code ____________________

6. Description of Action:

7. Was this action related to another action(s)? □ Yes AR#(s) ____________________

□ No

□ I don't know

Figure E.3. GCS Action Report Form

For support documentation, a Support Documentation Change Report (SDCR) form shown in Figure E.4 is used. A separate set of SDCRs will be generated for each support document. The SQA representative will log and assign a number to all SDCRs. Approval of the SQA representative is needed to complete the SDCR. The configuration manager will keep the original SDCR forms for all support documents.

Support Documentation Change Report page 1 of ____

1. Configuration Item:	2. Date:	3. Modification #:
4. Part of Configuration Item Affected:		
5. Reason for Modification:		
6. Modification		
7. SQA Signature & Date:		

Figure E.4. Support Documentation Change Report Form

The SQA representative will keep status logs to track all change reports and ensure that all are approved before entering the next development phase. These logs include the report number, the date it was assigned, the name of the assignee, the date it was returned, and date the SQA representative approved the report. Figure E.5 shows the form used for the status logs for the problem and action reports, and Figure E.6 shows the form of the status logs for the SDCRs.

Problem Reports Assigned for Action

Implementation: ______________________

PR #	Date Assigned	Assigned to:	Date Received (by Project Leader)	Date Approved (by SQA)	# of Action Reports	Comments

Figure E.5. Status Log for Problem Reports

Support Documentation Change Reports Assigned for Action

Configuration Item: _____________________________

SDCR #	Date Assigned	Assigned to:	Date Received (by Project Leader)	Date Approved (by SQA)	Comments

Figure E.6. Status Log for Support Documentation Change Reports

E.6 Configuration Management

The *Software Configuration Management Plan* outlines the procedures to be followed to control access and changes to documents. The configuration management procedures are supported by Digital Equipment Corporation's Code Management System (CMS). CMS allows one to define various libraries, each of which contains all versions of the documents within that library that can be easily retrieved. The *Software Configuration Management Plan* outlines the access and change authorizations for the documents within each library and contains a list of the configuration labels for all documents to be placed within the various libraries.

Specific users can be authorized to access but not change the documents within a library while other users can be authorized to make changes as well. Change control will be achieved by authorizing only the SQA representative and the configuration manager to *reserve* and *replace* documents, allowing new versions of documents to be placed under configuration control. Once a document has been placed under configuration control, there must be a change report that has been logged by the SQA representative before any item can be reserved from the CMS libraries; that is, no changes can be made to any item under configuration control unless a change has been authorized by the SQA representative.

As the small scale of the project permits easy communication, only two audits of the configuration are planned. The first is a conformity review to ensure that all elements of the project are of the proper release prior to review by the certification authority. The second audit has a similar purpose and will be held at the end of the project to ensure that all changes requested by the certification authority have been completed. Results of the audits will become part of the *Software Quality Assurance Records*. Additional audits may be requested by the configuration manager or by the SQA representative. See also the *Software Configuration Management Plan*.

E.7 SQA Records

The SQA records for the GCS project consist of the status logs for all of the change reports for the project's life cycle data and reports from reviews that are held during each of the development processes. There will be an SQA report at the closure of each development process. All reports become part of the *Software Quality Assurance Records*. The basic form of all the reports is an introduction followed by the overview of the review sessions and any problem reports that are issued. Below is a brief synopsis of each report. Each report documents the SQA approval for a particular stage of the implementation's development, and contains an acceptance statement signed by the SQA representative as part of the introductory comments.

- **Design Review Report**
  The Design Review Report is the formal acceptance of the design, signifying that the design process has ended and the coding process can begin. This report is generated when all problem reports and action items generated during the design reviews and any subsequent investigations have been closed.

- **Code Review Report**
  The Code Review Report is issued when all problem reports and action items generated during code reviews and any subsequent investigations have been closed, including any problem reports and action items for the design. This report is the formal acceptance of the code and indicates the inception of the integration process.

- **Test Readiness Review Requirements-based Testing**
  This report records that all test cases necessary for requirements-based testing at all levels (functional unit, subframe, frame, and trajectory) have been developed and are recorded in the Requirements Traceability Matrix. The requirements-based testing can be begin after approval of the test cases.

- **Test Completion Review Report for Integration Testing**
  This report is issued when SQA representative has determined that all verification procedures have been adhered to, all problem reports and action items initiated through requirements-based and structure-based testing have been closed, and all tests have successfully completed.

E.8 Software Conformity Review

The SQA representative will conduct a software conformity review of all project life cycle documentation prior to delivery to the project manager for submission for certification. A list of project documents that will be reviewed is contained in the Preface to this and all documents. The conformity review will meet the following objectives:

- ensure that all planned life cycle process activities have been completed,
- check the traceability of the software requirements through the design, code, and test cases,
- ensure that all life cycle data complies with the plans and standards and is properly controlled in compliance with DO-178B,
- ensure that all problem reports and support documentation change reports have been completed,
- ensure that all deviations from plans and standards have been approved and recorded, and
- check that the executable object code to be delivered can be regenerated from the archived source code.

E.9 Supplier Controls

All individuals working on the GCS project fall under the jurisdiction of this SQA plan. All project participants must use the processes and tools delineated in the full set of GCS documentation. Therefore, there is no need to set up additional means to assure that sub-tier suppliers processes and outputs will comply with this GCS SQA plan.

E.10 References

E.1 RTCA Special Committee 167. Software Considerations in Airborne Systems and Equipment Certification. Technical Report RTCA/DO-178B, Radio Technical Commission for Aeronautics, December 1992.

E.2 George B. Finelli. Results of software error-data experiments. In AIAA/AHS/ASEE Aircraft Design, Systems and Operations Conference, Atlanta, GA, September 1988.

E.3 Anita M. Shagnea and Janet R. Dunham. GCS Development Specification Review Description. Technical report, Research Triangle Institute, Research Triangle Park, NC, August 1990. Prepared under NASA Contract NAS1-17964; Task Assignment No. 8.

E.4 Thomas J. McCabe. A software complexity measure. IEEE Transactions on Software Engineering, SE-2, No. 6:308-320, December 1976.

E.5 John Joseph Chilenski and Steven P. Miller. Applicability of Modified Condition/Decision Coverage to Software Testing.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*
01- 12 - 2008

2. REPORT TYPE
Technical Memorandum

3. DATES COVERED *(From - To)*

4. TITLE AND SUBTITLE
Guidance and Control Software Project Data - Volume 1: Planning Documents

5a. CONTRACT NUMBER

5b. GRANT NUMBER

5c. PROGRAM ELEMENT NUMBER

6. AUTHOR(S)
Hayhurst, Kelly J. (Editor)

5d. PROJECT NUMBER

5e. TASK NUMBER

5f. WORK UNIT NUMBER
457280.02.07.07.06.02

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
NASA Langley Research Center
Hampton, VA 23681-2199

8. PERFORMING ORGANIZATION REPORT NUMBER
L-19548

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
National Aeronautics and Space Administration
Washington, DC 20546-0001

10. SPONSOR/MONITOR'S ACRONYM(S)
NASA

11. SPONSOR/MONITOR'S REPORT NUMBER(S)
NASA/TM-2008-215550

12. DISTRIBUTION/AVAILABILITY STATEMENT
Unclassified - Unlimited
Subject Category 61
Availability: NASA CASI (301) 621-0390

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Guidance and Control Software (GCS) project was the last in a series of software reliability studies conducted at Langley Research Center between 1977 and 1994. The technical results of the GCS project were recorded after the experiment was completed. Some of the support documentation produced as part of the experiment, however, is serving an unexpected role far beyond its original project context. Some of the software used as part of the GCS project was developed to conform to the RTCA/DO-178B software standard, "Software Considerations in Airborne Systems and Equipment Certification," used in the civil aviation industry. That standard requires extensive documentation throughout the software development life cycle, including plans, software requirements, design and source code, verification cases and results, and configuration management and quality control data. The project documentation that includes this information is open for public scrutiny without the legal or safety implications associated with comparable data from an avionics manufacturer. This public availability has afforded an opportunity to use the GCS project documents for DO-178B training. This report provides a brief overview of the GCS project, describes the 4-volume set of documents and the role they are playing in training, and includes the planning documents from the GCS project.

15. SUBJECT TERMS
Software engineering; Computer programming; Software reliability; DO-178B

16. SECURITY CLASSIFICATION OF:			**17. LIMITATION OF ABSTRACT**	**18. NUMBER OF PAGES**	**19a. NAME OF RESPONSIBLE PERSON**
a. REPORT	**b. ABSTRACT**	**c. THIS PAGE**			STI Help Desk (email: help@sti.nasa.gov)
					19b. TELEPHONE NUMBER *(Include area code)*
U	U	U	UU	164	(301) 621-0390