[illegible]

IN-14-CR
308265

Research and Development for Onboard Navigation (ONAV) Ground Based Expert/Trainer System

Revised Preliminary Test Plan

Daniel C. Bochsler

LinCom Corporation

April 15, 1988

Cooperative Agreement NCC 9-I6
Research Activity No. *AI.8*

rics

Research Institute for Computing and Information Systems
University of Houston - Clear Lake

T·E·C·H·N·I·C·A·L R·E·P·O·R·T

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information systems in 1986 to encourage NASA Johnson Space Center and local industry to actively support research in the computing and information sciences. As part of this endeavor, UH-Clear Lake proposed a partnership with JSC to jointly define and manage an integrated program of research in advanced data processing technology needed for JSC's main missions, including administrative, engineering and science responsibilities. JSC agreed and entered into a three-year cooperative agreement with UH-Clear Lake beginning in May, 1986, to jointly plan and execute such research through RICIS. Additionally, under Cooperative Agreement NCC 9-16, computing and educational facilities are shared by the two institutions to conduct the research.

The mission of RICIS is to conduct, coordinate and disseminate research on computing and information systems among researchers, sponsors and users from UH-Clear Lake, NASA/JSC, and other research organizations. Within UH-Clear Lake, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business, Education, Human Sciences and Humanities, and Natural and Applied Sciences.

Other research organizations are involved via the "gateway" concept. UH-Clear Lake establishes relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research.

A major role of RICIS is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. Working jointly with NASA/JSC, RICIS advises on research needs, recommends principals for conducting the research, provides technical and administrative support to coordinate the research, and integrates technical results into the cooperative goals of UH-Clear Lake and NASA/JSC.

Research and Development for Onboard Navigation (ONAV) Ground Based Expert/Trainer System

Revised Preliminary Test Plan

Preface

This research was conducted under the auspices of the Research Institute for Computing and Information Systems by LinCom Corporation under the direction of Daniel C. Bocshsler. Terry Feagin, Professor of Computer Science at the University of Houston - Clear Lake, served as the technical representative for RICIS.

Funding has been provided by the Mission Planning and Analysis Division, NASA/JSC through Cooperative Agreement NCC 9-16 between NASA Johnson Space Center and the University of Houston - Clear Lake. The NASA Technical Monitor for this activity was Robert Savely, Head, Artificial Intelligence Section, Technology Development and Applications Branch, NASA/JSC.

Research and Development for Onboard Navigation (ONAV)

Ground Based Expert/Trainer System

PRELIMINARY TEST PLAN

(Deliverable A)

------- revised -------

Prepared For:

Dr. Terry Feagin
Research Institute for Computing and Information Systems
University of Houston - Clear Lake

Prepared By:

Daniel C. Bochsler
LinCom Corporation
18100 Upper Bay Road, Suite 208
Houston, Texas 77058

Performed Under:

Project No. AI.8
Cooperation Agreement no. NCC9-16
Subcontract No. 005

April 15, 1988

TEST PLAN FOR THE ONBOARD NAVIGATION (ONAV) CONSOLE EXPERT/TRAINER SYSTEM

ENTRY PHASE

Revision to Preliminary Version

April 1988

LinCom Corporation
Houston Texas

TABLE OF CONTENTS

TABLE OF CONTENTS (cont.)

Section 1

SUMMARY

This document presents the approach and plans for testing the Entry phase of the Onboard Navigation (ONAV) Console Expert/Trainer System. Included is a discussion of background information and detailed testing approaches.

[illegible] ONboard [illegible] (ONAV)

Section 2

INTRODUCTION

2.1 PURPOSE

This document describes the test plan for the ONAV expert system. Levels of testing are identified and the contributing role of each level for ensuring reliability is described. Also, the objectives of each type of test are identified and a summary of the test methods and the type of testing environment to be used is included.

2.2 BACKGROUND

Prior expert system development efforts [1] began to form the basis for orderly development of expert system software. The testing philosophy for ONAV builds upon the key aspects of those efforts and the following operational necessity:

> "....expert systems must be able to withstand the test of time. Continuous modification and lack of reliability are unacceptable when considering operational integration into vital, on-going operations." [1]

A unique situation for ONAV with regard to testing exists due to the "manned console" nature of the ONAV task. The knowledge requirement document for an ONAV flight phase [2] not only serves as a software knowledge baseline from which to test, but also is playing a role in the class work for console trainees. These classes scrutinize the knowledge base, with emphasis upon identifying any corrections or deficiencies which can then be integrated into the system.

This process is a steady and evolving one such that it is not possible to wait until this effort is complete to begin testing the implemented ONAV expert system. The knowledge baseline is taken as a firm basis upon which to apply testing methods and procedures. The feedback from trainees represents a source of "change requests" for that baseline. Integrating these changes into ONAV regression testing (maintenance) activities is the mechanism that has been adopted to handle this situation, inorder to maximize the verity and controllability of the implemented system. This perspective is important for understanding how to successfully manage the transition from a developmental technology application to configuration controlled operational software support tools for mission operations.

Section 3

TESTING APPROACH

3.1 LEVELS OF TESTING

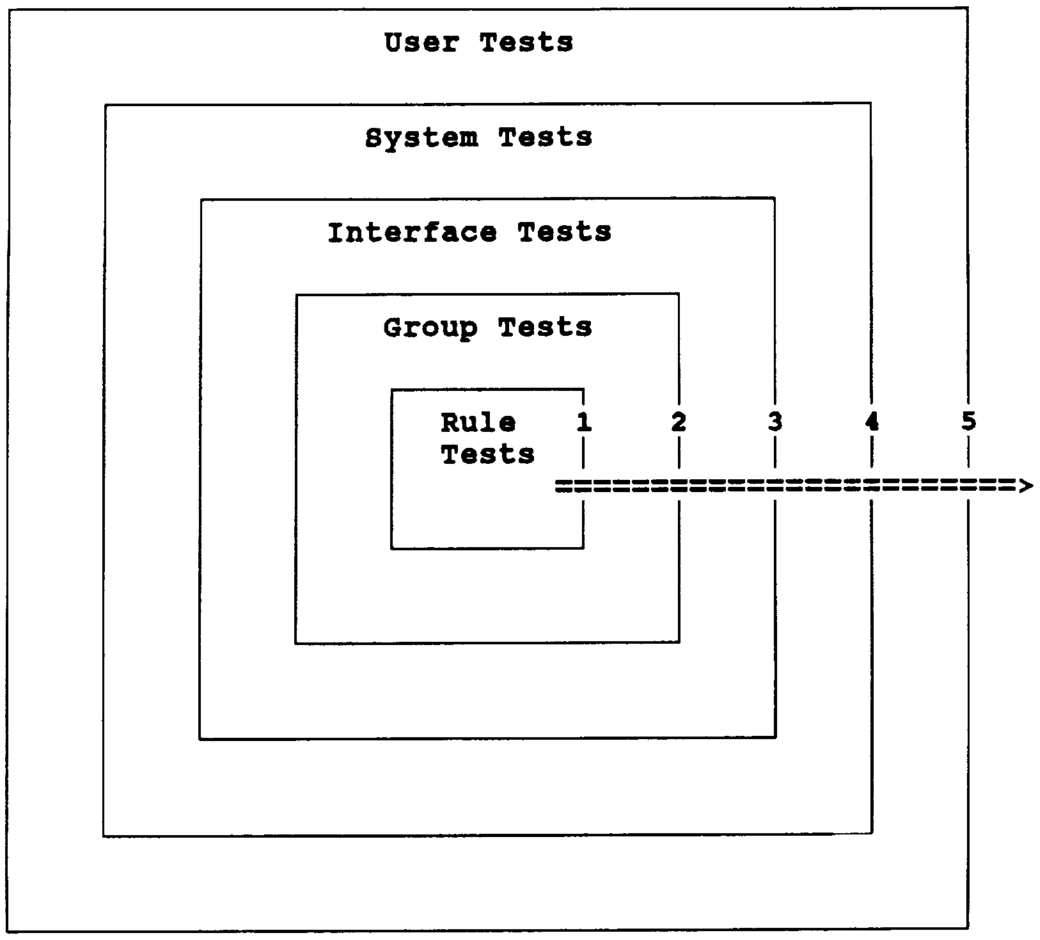
The ONAV expert system testing effort utilizes a multi-level verification approach as illustrated in figure 3-1. Five levels of testing are performed in essentially a serial manner: 1) individual rules, 2) ordered and unordered functional groups, 3) interface rules, 4) system tests, and 5) user tests. The intent is to catch the full range of both programming errors common to traditional programming and those errors characteristic of expert systems including factbase organization and inference engine interactions.

For an expert system there are really two types of errors: 1) errors in rules attributable to the expert information source (e.g., incomplete information, inconsistencies, etc.), and 2) the control structure (including syntax). The validity of what a rule does is defined by the expert. This is different from algorithmic situations which are governed by basic numerical relations. Here the error is one of definition, if you will, where the wrong information is utilized. On the other hand, the implementation problems in structure and syntax of the expert system language(s) are more along the lines of traditional software.The following error types are among those the testing approach attempts to address:

- Incorrect rules: selecting a rule structure that solves a problem incorrectly or badly.
- Errors in analysis; incorrect programming of the design.
- Semantic errors; failure to understand how an implementation language feature works.
- Syntax errors; failure to follow the rules of the programming language.
- Execution errors; failure to predict the possible ranges of rule results.
- Data errors; failure to anticipate the ranges, presence, or absence of data.

Incorrect rule content is solved by involving the experts in knowledge specification reviews. These prove very effective in confirming information as well as pointing out previously unnoticed shortcomings. Inappropriate implementation problems are a function of available experience with expert

Figure 3-1: Expert System Testing Approach



system development tools such as expert system shells and development environments as well as structured methods of software engineering design and development. Here the level of progress is controlled by the expert system programmer skill level.

3.1.1 Rule Tests

Rule tests are designed to verify the accuracy and correctness of the individual rules. This is to be accomplished through the performance of two steps: inspection and compilation.

Inspection consists of comparing the rule code with a design specification statement. This permits traceability from requirement to code and verifies the intent of the rule. This is straight-forward since a requirement for an expert system is usually in the form of a piece of knowledge contained in a rule. Traditional programming does not always have the luxury of such a high degree of compartmentalization of information.

Compilation of each rule will uncover any syntax errors and undefined patterns or functions. For systems where supplemental, often procedural, languages are used as part of the expert system software implementation, functions in the secondary language should be tested concurrently in the same manner as rules.

Given the scope of the ONAV expert system development effort, a third step has been added to the rule tests. This step includes the execution of each individual rule with a predefined set of input fact pattern conditions. This type of testing will ensure completeness of the input conditions within a rule. Performing such a series of tests is obviously useful for any expert system, but significant resources and time are required for a system like ONAV with a large number of rules.

3.1.2 Group Tests

Group tests are designed to verify the integration and functioning of groups of rules. Groups of rules are defined as either ordered or unordered with respect to functionality. An ordered group is a set of rules which must execute in a specific order (i.e. procedurally). This sequence of execution is implemented in most systems through the use of salience, control patterns, and/or declarative agendas. Unordered groups are opportunistic (i.e. random) in their execution and independence of a rule with respect to other rules is assumed. All control mechanisms built into the rules of an unordered group are uniform within the group. Group tests are accomplished through the performance of three steps: inspection, compilation, and execution.

First, inspection compares the rule code with the source requirements. Here, though, contrasted with individual rule inspection, the intent is to associate all design specifications with a rule or rules to ensure complete coverage of the specifications.

Next, rules are compiled as a group instead of individually as in the rule tests. This will uncover any rule syntax interactions such as renaming of rules, patterns, schema definitions, or secondary language functions within a group.

Third, each group of rules is executed. The objective here is to verify the proper order of execution of rules in relation to each other. Executions should utilize predefined sets of input facts, conditions, data, etc. The intent for these tests is to verify the functionality of the group and make sure that each rule has been fired with appropriate and explicit fact values, to the extent practical.

Figure 3.1.2-1 lists the results of an analysis that was done on the ONAV Entry phase system with regard to the groups on which testing is to be performed. The group references are keyed to the ONAV Knowledge requirements document [2].

3.1.3 Interface Tests

These tests are similar to group tests in terms of inspection, compilation and execution. But, the key point here is that interfaces between major rule groups or interfaces with users or other processes associated with the expert system as a whole should be verified explicitly. Significant system data flow is characteristic of functional interfaces and warrants special attention. Validating interface specifications are of particular importance during this type of testing.

Testing activities here would include not only ONAV expert system rules, but also data preparation and interface software written in C.

Figure 3.1.2-1: Group Testing Matrix for ONAV Entry Expert System

Group #	Specification Reference No.	Description	No. Rules
A1	3.1	Initial Conditions	10
B1	3.2	Telemetry Status	0
C1	3.3	Runway	5
D1	3.4	IMU	69
D2	3.4.1.1	IMU PASS Availability	10
D3	3.4.1.2	IMU BFS Availability	10
D4	3.4.2.1.1	IMU Velocity Comparisons	4
D5	3.4.2.1.2	IMU Attitude Comparisons	4
D6	3.4.2.1.3	IMU ACC Comparisons	6
D7	3.4.2.2.1	IMU 3-Level Isolation	1
D8	3.4.2.2.2	IMU 2-Level Isolation	17
D9	3.4.2.3	IMU Error Magnitude	4
D10	3.4.2.4	IMU Failure Prediction	3
D11	3.4.3.1	IMU PASS Recommendations	6
D12	3.4.3.2	IMU BFS Recommendations	4
E1	3.5	State Vector	15
E2	3.5.1	SV State Error Status	5
E3	3.5.2	SV Delta State Update	5
E4	3.5.3	SV BFS Transfer	5
F1	3.6	3-String State Vectors	9
G1	3.7	DRAG	5
G2	3.7.1	Drag Flag Status	2
G3	3.7.2	Drag Recommendations	3
H1	3.8	TACAN	86
H2	3.8.1	TACAN Configuration	6
H3	3.8.2	TACAN Availability	9
H4	3.8.3	TACAN LRU Quality	15
H5	3.8.4	TACAN Filter Flag Changes	4
H6	3.8.5	TACAN Toggle Recommendations	6
H7	3.8.6.1	TACAN LRUs For Deselect	8
H8	3.8.6.2	TACAN Deselect Configuration	4
H9	3.8.6.3	TACAN Predicted Availability	3
H10	3.8.6.4	TACAN Compute Config. Data	9
H11	3.8.6.5	TACAN Config. Acceptability	11
H12	3.8.7	TACAN Reselect Recommendation	1
H13	3.8.8	TACAN AIF Change Recommend.	10
I1	3.9	BARO	14
I2	3.9.1	BARO Measurement Quality	7
I3	3.9.2	BARO Flag Status	3
I4	3.9.3.1	BARO With GRND Data Available	4

Figure 3.1.2-1: (cont.)

J1	3.10	MSBLS	33
J2	3.10.1	MSBLS Availability	9
J3	3.10.2	MSBLS Lockon Status	5
J4	3.10.3	MSBLS Error Checks	4
J5	3.10.4	MSBLS Flag Monitoring	4
J6	3.10.5	MSBLS Recommendations	9
J7	3.10.6	MSBLS Effects on State Errors	2
K1	3.11	HSTD Checks	8
L1	---	Control Flow	6
M1	---	Operator Inputs	8
N1	---	Output Management	5
N2	---	. Event Message Mgmt	1
N3	---	. Recommendation Mgmt	3
N4	---	. Status Light Mgmt	1

* From baseline version of the ONAV Entry system; reflects the "implemented" number of rules which is the proper number to refer to in regards to testing.

3.1.4 System Tests

System tests are designed to verify overall system operation. These tests consist of: compilation, performance testing, and stress testing. Compilation of the entire system uncovers any remaining syntax or syntax interaction problems within the system as a whole. The performance of the expert system is demonstrated by defining several typical operational scenarios and then inputting them into the expert system. During these tests, the expert system reasoning process is observed and the verity and reliability of results evaluated. Stress testing could be called "robustness" testing. The intent here is to execute the expert system and observe the system's ability to handle improper operator selections, inputs and other unusual interactions.

3.1.5 User Tests

User tests involve a series of unstructured executions, from a test planning standpoint. The intent here is to involve the users of the system, if the system requires user interaction, so that end user functionality is checked first hand. At this point, the expert system should stand on its own without exceptions, caveats, etc. If the expert system is embedded within another larger software application, without any explicit interaction with a human user, then "User Testing" would take place as part of subsystem and system level validation and verification of the larger, "parent" system.

3.2 TEST SUPPORT

A development environment is a necessity when building an expert system. Many tools are used during traditional software implementation to assist source code entry, maintain consistency, and to look at processing and data flow in great detail. Comparable tools are required for expert systems. Features in a development environment should include such things as:

- Source code editors for changing and modifying code, along with adequate file access and storage capabilities.

- Debugging programs or capabilites that permit execution and examination of expert system processing activity. Examples would be features for monitoring or watching facts, rule executions, agendas, etc. associated with the inferencing process within the expert system.

- Support software or features that simplify the implementation of user interfaces, if required, for the expert system.

Without these types of capabilities, sufficient access and traceability to low level activity within an expert system for testing purposes cannot be achieved. If expert system development utilizes an expert system shell, often some or all of the above mentioned features are included in some form. Here, too, the extent of required development environment facilities depends upon the scope and complexity of the expert system.

For the ONAV system, the testing environment was defined early on in requirements definition to be that included in the CLIPS system. In addition, a cross reference program with the capability to read and cross reference rule and fact information in available and will be used to support various levels of testing.

3.2.1 Test Configurations

Several general software configurations will be utilized during testing. What follows are descriptions of each:

1) Raw CLIPS Environment:
This configuration consists of running a test using the command interface provided by CLIPS. At this level, detailed access to expert system processing is available for low level checking.

2) Manual ONAV Environment:
This configuration consists of running a test by invoking the ONAV expert system at the UNIX command level. Here, the screen interface is not used. At this level, detailed access to CLIPS system capabilities is available and the functionality of the ONAV rules are present.

3) Full ONAV Environment:
This configuration consists of running a test using the entire ONAV system, including the screen interface. At this level, no detailed CLIPS features are available. The full functionality of the ONAV system is utilized.

SECTION 4

TEST PROCEDURES

This section describes the approaches and test procedures to be followed during test activities for the ONAV system.

4.1 RULE TESTS

4.1.1 Inspection

4.1.1.1 Unused Fact Patterns

This test is intended to check for unused fact patterns. Such unused or unfamiliar patterns quite likely represent typographical errors.

Procedure:

1) Run XREF program for all rule sets (can be run on each set individually or on all sets together at the same time; there should be no impact on testing validity).

2) Examine the entire list to verify that no strangely named patterns exist.

4.1.1.2 Valid Literal Values

This test checks that no typographical errors exist in the literal values used in fact pattern fields.

Procedure:

1) Run XREF program for all rule sets.

2) Examine each relation in the relation summary section of XREF. Verify that each literal value makes sense with respect to the knowledge specification and functional use of the pattern.

4.1.2 Compilation

This test checks that all rules compile and that no duplicate names for different rules are present in the rule base.

Procedure:

1) Run XREF for all rule sets.

2) Load all rule sets into a manual ONAV environment.

3) List a "(rules)" command to obtain a list of all rules compiled into the CLIPS environment.

4) Examine "(rules)" list against the XREF listing to ensure the same rules appear on both lists. Note that the XREF program has limited capability to find duplicate rules. This feature can be utilized to the extent practical to supplement this test.

4.1.3 Execution

These tests check that each rule will execute given a set of input patterns that match at least one set of possible left hand side combinations on that rule.

4.1.3.1 Default Data Activations

This test checks for default data that cause activations of rules. The idea is to verify that any spontaneous executions that occur should in fact occur; in most cases no executions are expected. It should be noted that some default data is of a general nature and is included in a separate file. Other, more rule group specific defaults, are included in the rule files.

Procedure: For each set of rules (i.e., TACAN, BARO, etc), do the following.

1) Prepare a CLIPS batch command file that specifies a series of commands to CLIPS like the following:

```
(dribble-on <some file name such as "default-results">)
(watch all)
(load <tables.r file containing many ONAV defaults>)
(load <control.r and output.r files>)
(load <name of first rule set to be checked>)
(reset)
(run)
```

```
(clear)
(load <tables.r,control.r,output.r>)
(load <name of next rule set to be checked>)
(reset)
(run)
  .
  .
  .
(dribble-off)
(exit)
```

2) Batch the above command file into a manual ONAV environment.

3) List the dribble file. Then:

   a) Verify that all rule sets were loaded, reset, run and cleared properly.
   b) Examine each set and make assessment of how rules executed.

4.1.3.2 Single/Multiple Rule Interaction

These tests check execution of each rule, multiple executions of the same rule, and execution of more than one rule with a given input data pattern set. This will check infinite loop rules (though some types of looping may be appropriate to parts of the control flow design) and identify similarities/relationships between rules.

4.1.3.2.1 Individual Rule Sets Loaded

Procedure:

1) Prepare a test deffacts file for each rule in each rule set (one deffacts statement for each rule in a separate file so that each deffacts can be individually loaded).

2) Set up a CLIPS command file as follows:

```
(dribble-on <some file name "rule-results")
(watch all)
(load <name of first rule set to be checked>)
(load <name of test deffact file for first rule>)
(reset)
(run)
(undeffacts <name of test deffact statement>)
(load <name of test deffact file for second rule>)
(reset)
(run)
```

```
(undeffacts <name of test deffact statement>)
  .
  .
  .
(clear)
(load <name of file for second rule set>)
(load <test deffact file name for first rule in group>)
(reset)
(run)
(undeffact <name of test deffact statement>)
  .         .
  .         .
  .         .
(dribble-off)
(exit)
```

3) Batch command file into a manual ONAV environment.

4) List dribble file and examine results.

4.1.3.2.2 <u>All Rule Sets Loaded</u>

Procedure:

1) Use deffacts for each rule that were prepared for the "Individual Rule Sets Loaded" tests.

2) Set up command file as follows:

```
(dribble-on <some file name "allrules-results")
(watch all)
(load <first rule set>)
(load <second rule set>)
  .
  .
(load <last rule set>)
(load <test deffact for 1st rule in 1st rule set>)
(reset)
(run)
(undeffacts <deffact for 1st rule in 1st rule set>)
(load <deffact for 2nd rule in 1st rule set>)
(reset)
(run)
  .
  .    for 1st through nth rule in each set
(dribble-off)
(exit)
```

3) Batch command file into a manual ONAV environment.

4) Print listing and verify results.

4.2 GROUP TESTS

4.2.1 Functional Inspection

This check consists of comparing the design information to the expert system code to ensure that the function of the design was completely implemented. A particular rule may relate to all or part of more than one design item. Therefore, where individual rule inspection makes sure a rule relates to a design specification, rule group inspection looks for "holes." Parts of a design specification may have been missed.

Procedure: For each rule group identified in Table 3.1.2-1, do the following.

1) Obtain a listing of the code for that rule group.

2) Using the corresponding knowledge requirements specification section for each rule group, ensure that all specification functions are covered by the rule group code.

3) Document and implement necessary corrections.

4.2.2 Functional Execution

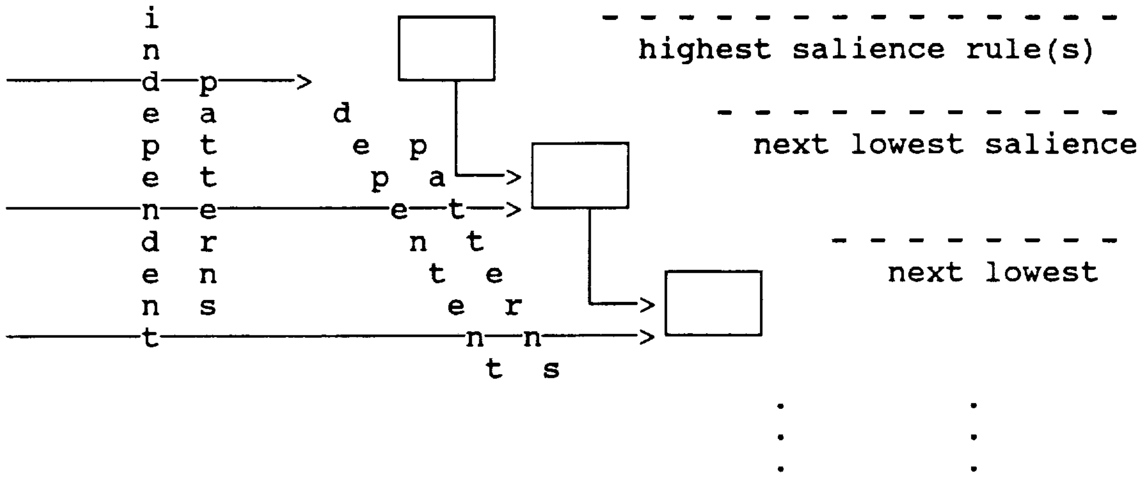
These tests represent the first step in the verification process where parts of the overall system "function" can be examined. While individual rules have a function, a grouping of rules gives more meaning and relevance with respect to the overall system. At this level of testing, salience is sometimes needed for some rules which, although carrying out similar functions, need to be executed in a particular sequence to insure logical and unambiguous results.

4.2.2.1 Ordered Groups

This test checks that all rules in a group can execute in the proper sequence, based on salience or other sequencing mechanisms.

Procedure: For each rule group specified in Table 3.1.2-1, perform the following steps.

1) Prepare a deffacts data set that will satisfy the independent patterns of the rules in the group, as illustrated in the following figure:



2) Setup a CLIPS command file as follows:

```
(dribble-on <some file name "ordered group">)
(watch all)
(load <essential overhead files for ONAV>)
(load <name of rule set to be checked>)
(load <name of test deffacts file>)
(reset)
(run)
(dribble-off)
(exit)
```

3) Batch command file into a manual ONAV Environment.

4) List dribble file and examine results.

4.2.2.2 <u>Unordered Groups</u>

This series of tests will be much like those in section 4.2.2.1, except that the deffacts data set for each rule group should consist of patterns that form a "functional input data set." In dealing with these groups at a functional level, the function results are the focus of the testing, rather than the satisfaction of rule patterns.

4.3 INTERFACE TESTS

4.3.1 End to End Data Flow

This test is intended to verify that all required data from input files, user inputs, and all other sources (e.g., deffacts, functions, or derived facts) get into the fact base of ONAV.

4.3.1.1 Input File Sources

Procedure:

1) Prepare a repairs file with each of 600 or so data items set to a unique value (e.g., item(1)=1, item(2)=2, ..., item(n)=n). The resulting value of the fact in the fact base does not have to be in the same format as the actual fact. The intent is to verify that the flow of information is there.

2) Run LOGCOMPS for at least one cycle to prepare a data file which can be read by the ONAV system.

3) Set up an ONAV run (without the user interface screen). Load only those rule sets that are considered essential to handling control flow, etc. Do not load any of the functional rule sets. A rule to halt CLIPS processing may need to be added to one of the control phases of ONAV so as to facilitate an orderly termination of the run after the data has been asserted into the fact base.

4) Run XREF for all rule sets. This will be used to chech off rule data pattern sources when analyzing the results of this test.

5) Run the test; when execution is halted, enter the following commands to get a list of all data received by the fact base:

```
(dribble-on <some file name "endtoend-results">)
(facts)
(dribble-off)
(exit)
```

6) List out the dribble file and check the list of facts that were in the fact base against the fact patterns given in the XREF listing (identify each XREF list fact as being received from "input stream").

4.3.1.2 Default Data Sources

Procedure:

1) Load all rule files into a raw CLIPS environment.

2) Enter the following commands:

```
(reset)
(dribble-on <some file name "default-results">)
(facts)
(dribble-off)
(exit)
```

3) Obtain a listing of the dribble file and check off facts against the XREF listing (same listing as for the "input source" test, identifying each XREF fact as being from a "default" source.

4.3.1.3 User Input Data Sources

These tests ensure that data originating from user interface commands result in appropriate facts being asserted into the ONAV fact base.

4.3.1.3.1 Non-HSTD Inputs

This test checks the following operator inputs: stop, subsystem, selection, delta-state, bfs nogo, runway, and toggle tacan.

Procedure:

1) Enter the following CLIPS commands into a Manual ONAV Environment:

```
(dribble-on <some file name "nonhstd-input-check">)
(watch all)
(load <essential ONAV rule sets>)
(load <operator.r rule set>)
(reset)
(run)
```

2) As the system runs, enter the following series of key strokes, pausing ~10 seconds between each key stroke.

... to be determined ...
(must include all possible key strokes, both small and capital letters)

Wait 30 seconds after the last key stroke before interrupting CLIPS and getting back to the CLIPS command prompt.

3) Enter the following commands:

```
(facts)
(dribble-off)
(exit)
```

4) List the dribble file and ensure that facts for each key stroke have been put into the fact base.

4.3.1.3.2 HSTD Inputs

This test checks the HSTD related operator inputs which are handled by the HSTD rule set.

Procedure:

1) Prepare a command list like that in 4.3.1.3.1, except that the hstd.r file is loaded instead of the operator.r file.

2) List the dribble file and see that each key stroke was received and the proper fact was put into the fact base.

4.3.1.4 Function Call Sources

4.3.1.4.1 Identify Function Calls

This procedure identifies which rules contain external function calls.

Procedure:

1) Run XREF program for each major rule group (i.e., any group number that ends with the number "1", like A1, G1, H1, etc.). The rule summary will identify the number of external function calls in each rule, and the external function summary will list the names of all functions referenced in the XREF file.

2) Verify that each external function is defined by comparing the function list with ONAV C language code.

4.3.1.4.2 Function Call Returns

This test checks that function calls, which return data as part of fact assertions, return the expected type of data. In addition, a check is made to ensure that the use of that returned value in the rule is consistent (e.g., as to data type that the rest of the rule expects.

Procedure:

Using the list of external function calls generated in 4.3.1.4.1, this test is to be done as part of the other tests as each of the rules with an external function is executed and results obtained.

4.3.2 Status Light Indicators

This test checks the status light indicators to ensure all lights lightup, all lights are indicated at the proper location on the user interface screen, and all possible values of a light can be indicated correctly.

Procedure:

1) Identify all available status lights, range of value, and status light fact formats (i.e., expert system fact name and field definitions).

2) Prepare a test deffacts that specifies a fact for each light and each possible value/status for each light. Make a hardcopy listing of the test deffacts for later use in this test.

3) It is likely that a delay loop of some sort (e.g., a while loop with a large number?) may need to be inserted in the status output rule. This would cause repetitive rule firing to slow down and enable test personnel to recognize each of the display changes.

4) Load the Manual ONAV Environment, but only load the output rule set along with the essential control flow rules (do not load any functional rule sets).

5) Load the test deffacts file through the screen interface prior to the "recycle, and onav" commands are given.

6) Keeping the deffacts list available, note on that list the sequencing of each status light through the proper values and conditions.

4.3.3 <u>User Interface Command Acceptance</u>

This test will verify that the user interface configuration commands associated with logging, etc. are received and processed correctly by the ONAV system.

Procedure:

1) Execute the full ONAV environment.

2) Invoke the logging command and observe the confirmation message that appears on the screen.

3) Sequentially invoke each of the other interface configuration commands. These can be monitored by observing the respective messages as they appear on the screen.

4) After all configuration commands have been invoked, the logging off command can then be invoked. Then terminate the run.

5) List the resulting logging file and verify that all messages and corresponding rules executed.

4.4 ACCEPTANCE TESTING

Operational considerations for ONAV utilization leads to the combining of system and user tests into a single test category. This effort is patterned after the current procedures used to validate human console operators. Table 4.4-1 lists the key parts of the testing approach to be followed. Figure 4.4-1 shows the overall acceptance test activity flow and illustrates how the ONAV system fits into current training activities. Table 4.4-2 shows the checkoff list used to record test completions. Tables 4.4-3 and 4.4-4 are examples of summary data from a test using sim data.

Table 4.4-1: ONAV Entry Acceptance Test Approach

I. <u>ONAV Experience Data Base</u>

- o The data base consists of the errors each ONAV operator has experienced while on console.
- o The errors of each sim will be evaluated to determine if the sim will fill a slot in the ONAV test case library.
- o For each test case in the library, a copy of the CCSLOG tape will be kept on the UNIVAC and the HP 9000 computers.

II. <u>ONAV Paper Sims</u>

- o Each test case will be processed by the DELOG program to generate reproductions of the ONAV displays.
- o A certified ONAV operator will review the displays to provide the correct ONAV responses for the case.
- o The test cases will provide inexperienced ONAV trainees with a meaningful way to go through the ONAV checklists.
- o The correct responses will be used to evaluate the expert system responses and can be used to evaluate the trainees responses to the ONAV training sims.

III. <u>Expert System Printout</u>

- o The HP9000 copy of the CCSLOG tape will be run through the expert system with necessary tape repairs incorporated.
- o The printed record of the expert system responses will be compared to the paper sim responses.
- o The expert system should be tested as if it was a new ONAV trainee. This means it should be subjected to every type of error in the ONAV data base in at least two different situations.

IV. <u>ONAV Trainer Sims</u>

- o The HP9000 copy of the CCSLOG tape will be run through the HP DELOG program to generate a near-real time simulation.
- o The log of the trainees will be compared to the responses suggested in the paper sims. This will indicate if the trainees saw all the errors and if they saw the errors in a timely manor.

Figure 4.4-1: Overall Flow of Activity

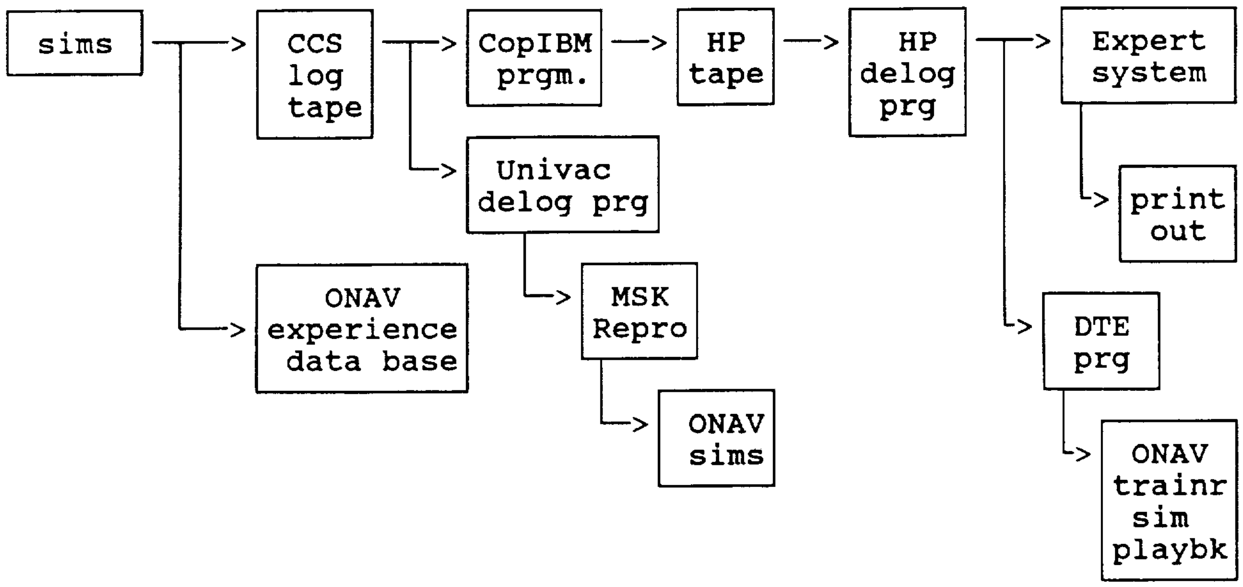


Table 4.4-2: Training Log

DATE:

TYPE	ERROR	1	2	3	4	5	6
SITE DATA	OB RUNWAY	000	000	000	000	000	000
	BFS RUNWAY	000	000	000	000	000	000
	GND RUNWAY	000	000	000	000	000	000
	TAC CHANNEL NO.	000	000	000	000	000	000
	MLS CHANNEL NO.	000	000	000	000	000	000
BFS	ENGAGE	000	000	000	000	000	000
	NO GO	000	000	000	000	000	000
GND	NONE	000	000	000	000	000	000
	NO GO	000	000	000	000	000	000
OPS	SPLIT SET	000	000	000	000	000	000
	STRING DOWN (PASS)	000	000	000	000	000	000
	RESTRING (PASS)	000	000	000	000	000	000
	STRING DOWN (BFS)	000	000	000	000	000	000
	RESTRING (BFS)	000	000	000	000	000	000
STATE VECTORS	O DELTA STATE	000	000	000	000	000	000
	DELTA T UPDATE	000	000	000	000	000	000
	DELTA STATE (PASS)	000	000	000	000	000	000
	MANUAL DELTA STATE (PASS)	000	000	000	000	000	000
	WHOLE STATE PASS	000	000	000	000	000	000
	DELTA STATE (BFS)	000	000	000	000	000	000
	MANUAL DELTA STATE (BFS)	000	000	000	000	000	000
	WHOLE STATE (BFS)	000	000	000	000	000	000
	BFS TRANSFER	000	000	000	000	000	000
IMU	FAIL (PASS)	000	000	000	000	000	000
	COMMFAULT (PASS)	000	000	000	000	000	000
	DESELECT (PASS)	000	000	000	000	000	000
	DILEMMA	000	000	000	000	000	000
	FAIL (BFS)	000	000	000	000	000	000
	COMMFAULT (BFS)	000	000	000	000	000	000
	DESELECT (BFS)	000	000	000	000	000	000
	DESELECT THEN RESELECT (BFS)	000	000	000	000	000	000
	DRIFT	000	000	000	000	000	000
	RESOLVER	000	000	000	000	000	000
	BIAS	000	000	000	000	000	000
	SCALE FACTOR	000	000	000	000	000	000
DRAG	ATMOSPHERE	000	000	000	000	000	000
	EDITING	000	000	000	000	000	000

Table 4.4-2 (cont.)

TACAN	POWER OFF	000	000	000	000	000	000
	FAIL	000	000	000	000	000	000
	COMMFAULT	000	000	000	000	000	000
	DESELECT	000	000	000	000	000	000
	RESELECT	000	000	000	000	000	000
	BAD GROUND STATION	000	000	000	000	000	000
	NO GO	000	000	000	000	000	000
	FORCE	000	000	000	000	000	000
	SELFTEST	000	000	000	000	000	000
	DILEMMA	000	000	000	000	000	000
	40 DEGREE GLITCH	000	000	000	000	000	000
	BIAS	000	000	000	000	000	000
	NOISE	000	000	000	000	000	000
	TIMMING	000	000	000	000	000	000
ADTA	NO GO	000	000	000	000	000	000
	DILEMMA	000	000	000	000	000	000
	BIAS	000	000	000	000	000	000
	NOISE	000	000	000	000	000	000
MLS	POWER OFF	000	000	000	000	000	000
	FAIL	000	000	000	000	000	000
	COMMFAULT	000	000	000	000	000	000
	NO GO	000	000	000	000	000	000
	FORCE TACAN	000	000	000	000	000	000
	BIAS	000	000	000	000	000	000
	NOISE	000	000	000	000	000	000
LOOK OUTS	IMU AFTER TOWER ROLL	000	000	000	000	000	000
	NAV AFTER FILTER STOP	000	000	000	000	000	000
	NAV DURING OMS BURNS	000	000	000	000	000	000
	NAV WITH DELTA T IN PASS-BFS	000	000	000	000	000	000
	TACAN CONE OF CONFUSION	000	000	000	000	000	000
	ADTA DURING ROLL REVERSAL	000	000	000	000	000	000
	ADTA DURING MACH JUMP	000	000	000	000	000	000

	ONAV1	ONAV2	INST	CCSLOG	RUNTYPE	LANDED AT
1						
2						
3						
4						
5						
6						

COMMENTS (CONSOLE OPS; COMMUNICATION:)

Table 4.4-3: Sim Summary Example

PHASE:	ASCENT
TYPE:	NOMINAL
TIME:	T-1 MINUTE TO 6:30

ERROR SUMMARY:

- O IMU 2 RESOLVER ERROR
- O IMU 2 FAILS IMU RM
- O IMU 2 DESELECTED IN THE BFS
- O IMU DILEMMA
- O IMU 3 DESELECTED

Table 4.4-4: Example List of Calls

TIME	WHO CALLED	CONTENT OF CALL
-0:45	GDO	GND and ONBOARD have KSC15 selected and they are the correct runways.
-0:30	GDO	The IMUs are less than 1 sigma.
0:00	GDO	PASS and BFS nav states agree.
0:30	GDO	PASS and BFS nav states are go.
1:00	GDO, LOG	IMU 2 has a resolver error. It should fail RM shortly.
1:15	-	Static data.
2:45	-	End of static data.
	GDO, LOG	IMU 2 has failed (failed at 1:20). IMU 3 has a velocity error of about 500 mirco G's. (Note REF IMU 2) BFS is on IMU 2 (crew deselected IMU 2 in the BFS). PASS and BFS nav states are go.
3:00	GDO, LOG	IMU 3's velocity error is a scale factor and it should fail RM shortly. (Note REF IMU 3 and there is about 6500 mirco G's in the Z axis).
3:30	GDO, LOG	There is an IMU DILEMMA. Recommend that you deselect IMU 3. (Note that this appears only on MSK 1417. This is explained in the paper.)
3:45	-	(Note the dilemma shows up on MSK 547 now.)
4:00	GDO, LOG	The crew has deselected IMU 3. (Note that we have no insight if they deselected IMU 3 in the BFS as per procedure.)
4:15	-	(Note deselection appears on MSK 547.)
5:00	GDO	PASS and BFS nav states are go.
6:30	GDO, LOG	PASS is go; BFS has a 3000 ft. downrange error.
6:45	-	Static data.

4.5 TEST CONDUCT NOTES

Several practical considerations should be kept in mind while performing several of the above specified test procedures:

1) Watch the terminal screen carefully for signs of possible infinite loops on rules so that such an error can be halted. If this occurs, the problem should be logged, noted, corrected, and the entire test repeated.

2) All input command file created for the above tests can be accumulated, documented, and kept for future regression testing.

3) Keep all ONAV system files in a configuration controlled area on the computer system used during testing. Keep a log of all changes to the test configuation system (resulting from rule corrections, etc.). This will assist in maintaining the integrity of the test results given the assumption that some changes to the ONAV system will result from testing.

4) The extent to which CLIPS has been verified affects testing in general. Determination and recognition of known bugs and problems in CLIPS should be considered at all times.

Section 5

REFERENCES

1) "Test Report for the Rendezvous/Proximity Operations Trajectory Control Expert System (RENEX), NASA Johnson Space Center, Internal Note # JSC-22528, April 1987.

2) "Knowledge Requirements for the Onboard Navigation (ONAV) Console Expert/Trainer System," NASA Johnson Space Center, Internal Note #JSC-22657, October 1987.

3) "Guidelines and System Requirements for the Onboard Navigation (ONAV) Console Expert/Trainer System," NASA Johnson Space Center, Internal Note #JSC-22433, December 1986.

4) "Test Plan for the Onboard Navigation (ONAV) Console Expert/Trainer System - Entry Phase," Preliminary version, LinCom Corporation, December 1987.

5) "User's Guide for the Onboard Navigation (ONAV) Console Expert/Trainer System - Entry Phase," Final Version, LinCom Corporation, April 1988.

END OF DOCUMENT