

SSE Software Test Management STM Capability: Using STM in the Ground Systems Development Environment (GSDE)

JOHNSON

10-61-CR

116929

p.20

N92-33607

Unclas

G3/61 0116929

**Victor E. Church
D. Long
Ray Hartenstein**
Computer Sciences Corporation

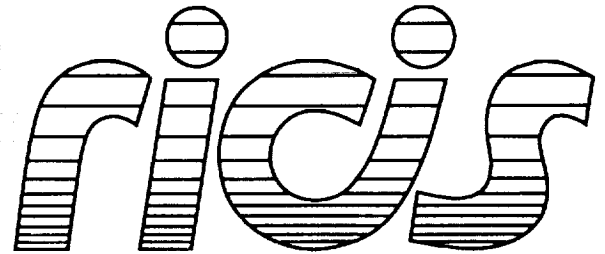
Alfredo Perez-Davila
University of Houston-Clear Lake

February 1992

Cooperative Agreement NCC 9-16
Research Activity No. SE.34

NASA Johnson Space Center
Mission Operations Directorate
Space Station Ground Systems Division

(NASA-CR-190717) SSE SOFTWARE TEST
MANAGEMENT STM CAPABILITY: USING
STM IN THE GROUND SYSTEMS
DEVELOPMENT ENVIRONMENT (GSDE)
(Research Inst. for Computing and
Information Systems) 22 p



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

TECHNICAL REPORT

The RICIS Concept

The University of Houston-Clear Lake established the Research Institute for Computing and Information Systems (RICIS) in 1986 to encourage the NASA Johnson Space Center (JSC) and local industry to actively support research in the computing and information sciences. As part of this endeavor, UHCL 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 continuing cooperative agreement with UHCL 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 UHCL/RICIS mission is to conduct, coordinate, and disseminate research and professional level education in computing and information systems to serve the needs of the government, industry, community and academia. RICIS combines resources of UHCL and its gateway affiliates to research and develop materials, prototypes and publications on topics of mutual interest to its sponsors and researchers. Within UHCL, the mission is being implemented through interdisciplinary involvement of faculty and students from each of the four schools: Business and Public Administration, Education, Human Sciences and Humanities, and Natural and Applied Sciences. RICIS also collaborates with industry in a companion program. This program is focused on serving the research and advanced development needs of industry.

Moreover, UHCL established relationships with other universities and research organizations, having common research interests, to provide additional sources of expertise to conduct needed research. For example, UHCL has entered into a special partnership with Texas A&M University to help oversee RICIS research and education programs, while other research organizations are involved via the "gateway" concept.

A major role of RICIS then is to find the best match of sponsors, researchers and research objectives to advance knowledge in the computing and information sciences. RICIS, working jointly with its sponsors, 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 goals of UHCL, NASA/JSC and industry.

***SSE Software Test Management
STM Capability:
Using STM in the Ground Systems
Development Environment (GSDE)***

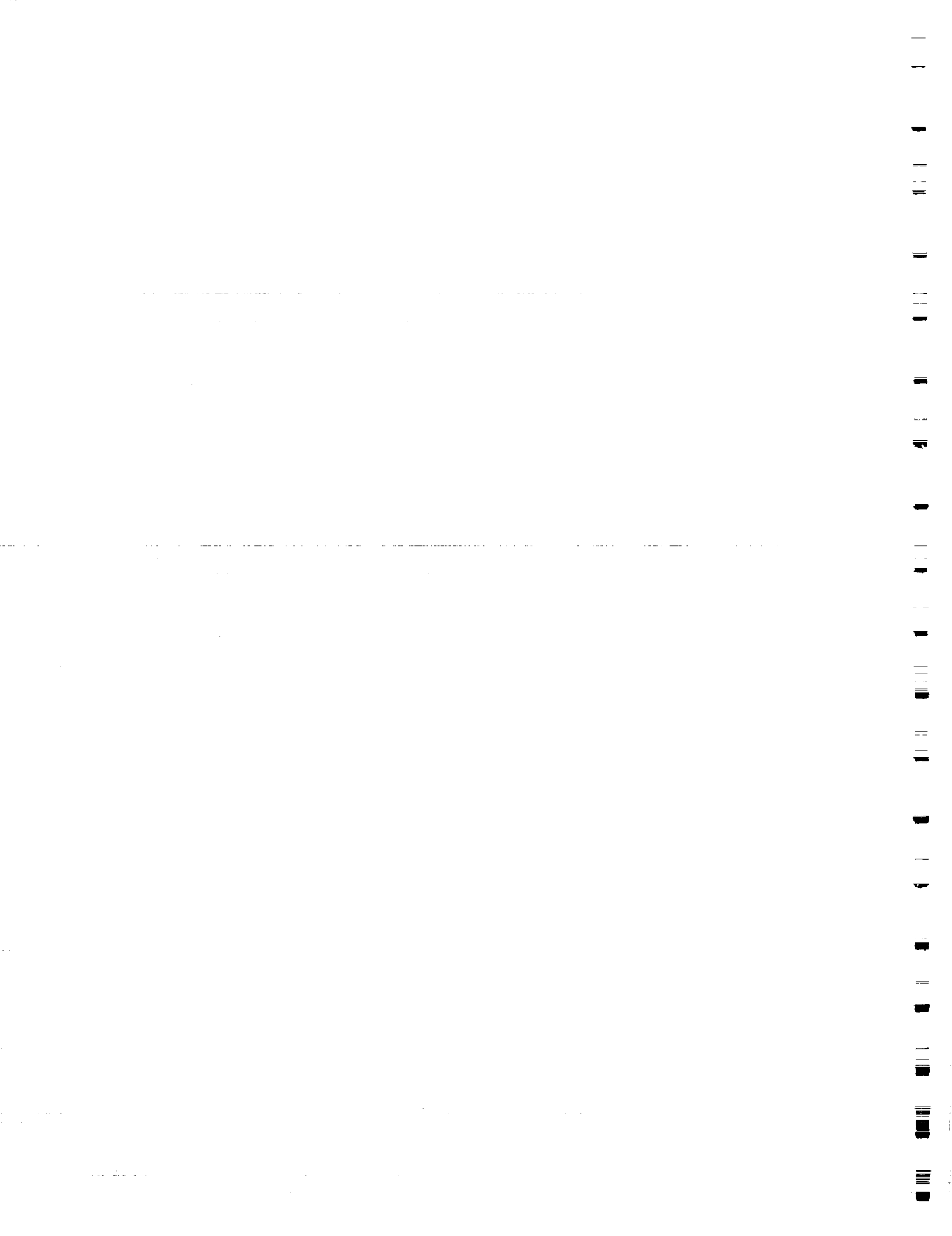


RICIS Preface

This research was conducted under auspices of the Research Institute for Computing and Information Systems by Computer Sciences Corporation in cooperation with the University of Houston-Clear Lake. The members of the research team were: Victor E. Church and D. Long from CSC and Alfredo Perez-Davila from UHCL. Mr. Robert E. Coady was CSC program manager for this project during the initial phase. Later, Mr. Ray Hartenstein assumed the role of CSC program manager. Dr. Perez-Davila also served as RICIS research coordinator.

Funding was provided by the Mission Operations Directorate, NASA/JSC through Cooperative Agreement NCC 9-16 between the NASA Johnson Space Center and the University of Houston-Clear Lake. The NASA research coordinator for this activity was Thomas G. Price of the ADPE and Support Systems Office, Space Station Ground Systems Division, Mission Operations Directorate, NASA/JSC.

The views and conclusions contained in this report are those of the authors and should not be interpreted as representative of the official policies, either express or implied, of UHCL, RICIS, NASA or the United States Government.



**SSE Software Test Management
(STM) Capability:
Using STM in the Ground Systems Development
Environment (GSDE)**

Prepared for

Lyndon B. Johnson Space Center
National Aeronautics and Space Administration
Houston, Texas

by

Computer Sciences Corporation
System Sciences Division
Beltsville, Maryland and League City, Texas

and

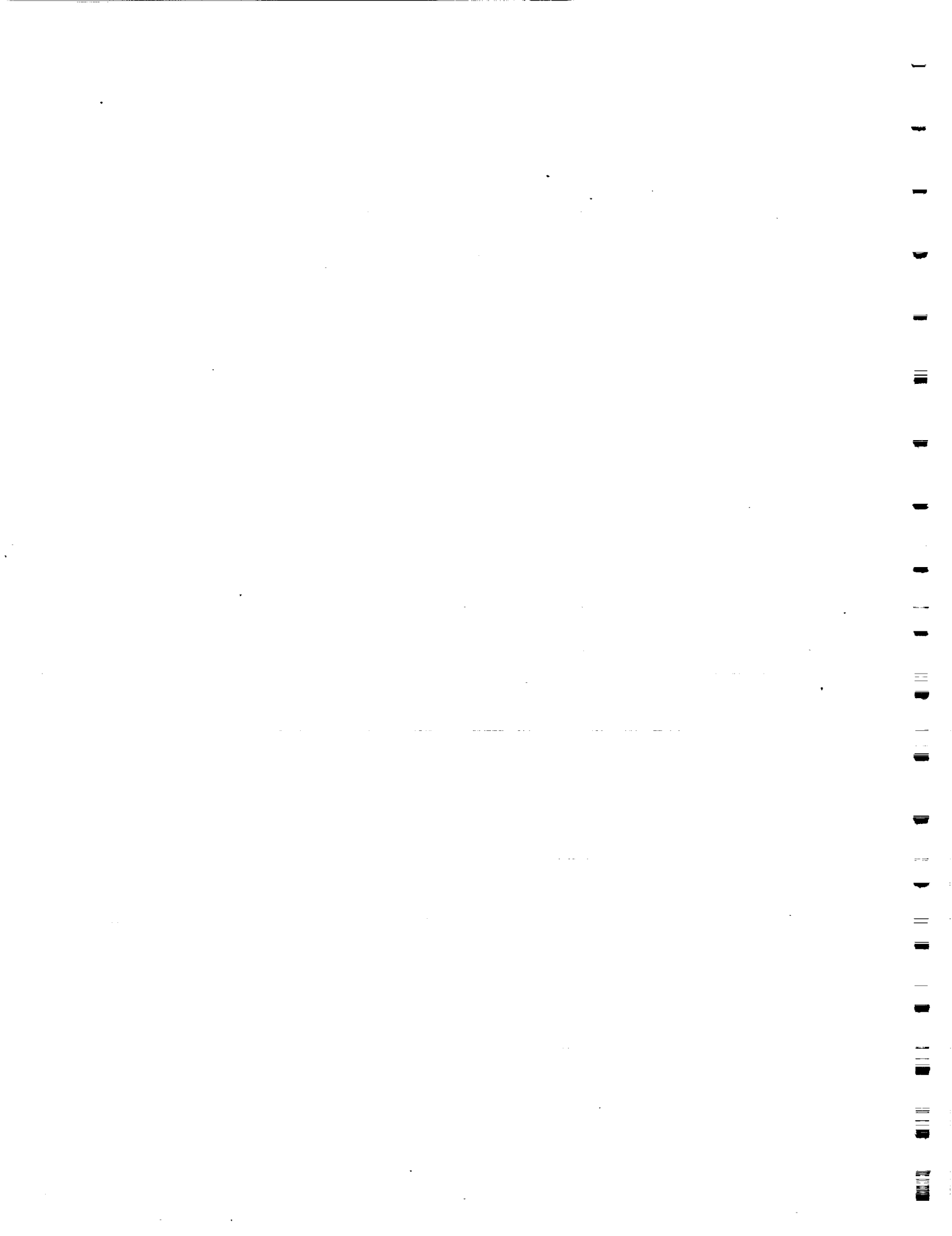
The University of Houston - Clear Lake
Research Institute for Computers and Information Sciences
Clear Lake, Texas

under

Subcontract No. 075
RICIS Research Activity No. SE-34
NASA Cooperative Agreement NCC 9-16

February 1992

Preparation: V. Church
Quality Assurance: D. Long
Approval: R. Hartenstein
A. Perez-Davila



Abstract

This report is one of a series discussing configuration management (CM) topics for Space Station ground systems software development. It provides a description of the SSE-developed Software Test Management (STM) capability, and discusses the possible use of this capability for management of developed software during testing performed on target platforms. This report is intended to supplement the formal documentation of STM provided by the SSE Project. The report describes how STM can be used to integrate contractor CM and formal CM for software before delivery to operations. STM provides a level of control that is flexible enough to support integration and debugging, but sufficiently rigorous to insure the integrity of the testing process.



Table of Contents

Introduction	1
GSDE software development context.....	2
Overview of the STM capability	3
STM data structure.....	4
STM operations.....	8
Staging a testbed	9
Results Capture	10
Checkout/Checkin	10
STM Reporting	11
Implications for use in the GSDE.....	11
Requirements for test planning and definition	12
Requirement for staging and target scripts	13
Comparison of STM and CM interfaces to targets	13
Integrating contractor CM during testing.....	14
Summary	15
Glossary and Abbreviations	15

List of Figures

1. Ground Software Development Environment.....	2
2. Four steps in target-based testing	4
3. Example of a Test Configuration Specification	5
4. Test plans are based on software design	6
5. Development of test definition scripts	7
6. Sample of a test process script (in TPL).....	8
7. Staging elements in a testbed	9



Introduction

This report briefly describes the operation and use of the Software Test Management (STM) system, a capability that will be provided by the Space Station SSE* in OI 6.0, scheduled for April 1992. This description is based on material presented by SSE Project personnel; it is tailored to the development of ground systems software. The intent of the report is to identify potential impacts to the process of testing software for the Space Station Control Center (SSCC) and the Space Station Training Facility (SSTF).

This report is one in a series of reports on configuration management issues in the development of Space Station ground systems software. The reports are part of a study of GSDE interfaces being conducted by Computer Sciences Corporation and the University of Houston - Clear Lake for the Lyndon B. Johnson Space Center Mission Operations Directorate (DJ).

STM is a SPF host-based capability that supports the definition, staging, and reporting of tests performed on target platforms. (Test execution is supported only indirectly, with target platform command scripts that can be managed by STM.) STM can support testing during development (e.g., unit testing that requires the target platform) as well as formal testing performed for qualification and operational delivery.

STM is not an alternative to formal SPF-based CM. While it does provide controlled storage and tracking of component status, it is intended to work *with* the SSE CM system and the Rational-based CMVC system--not to replace them. STM will also provide a development path to the use of the SSE Build Process system that will be available with OI 7.0 in April of 1993.

This report is intended to supplement the formal documentation of the STM by addressing the particular circumstances of SSTF and SSCC development. The reason for this supplement is as follows: The primary focus of the SSE Project (and of SSE documentation) is the production of *flight* software. The primary flight software target platform is a Standard Data Processor (SDP), an Intel 80386 processor with the Lynx real-time Unix operating system. *Ground* software systems, in contrast, will use IBM mainframes and various Unix-based workstations as the primary target platforms. Some of the mechanisms and procedures described in SSE documentation must be modified or interpreted to apply to ground systems software. This report focuses on use of STM in the context of the Ground Systems Development Environment (GSDE).

* Abbreviations that are in common use in the Space Station Freedom Program are not spelled out in the text, but are defined in the Glossary at the end of this report.

GSDE software development context

Ground systems software, specifically the SSCC and SSTF, will be developed and tested using combinations of development computers and workstations (called Software Production Environments, or SPEs), a Ground Systems SPF (GS/SPF), and target platforms that are essentially the operational target environments. It is worth noting that there is no separate ITVE for either ground system. Figure 1 shows the basic development context. The target environments include IBM mainframe computers, Unix-based workstations, and mission-specific special-purpose hardware.

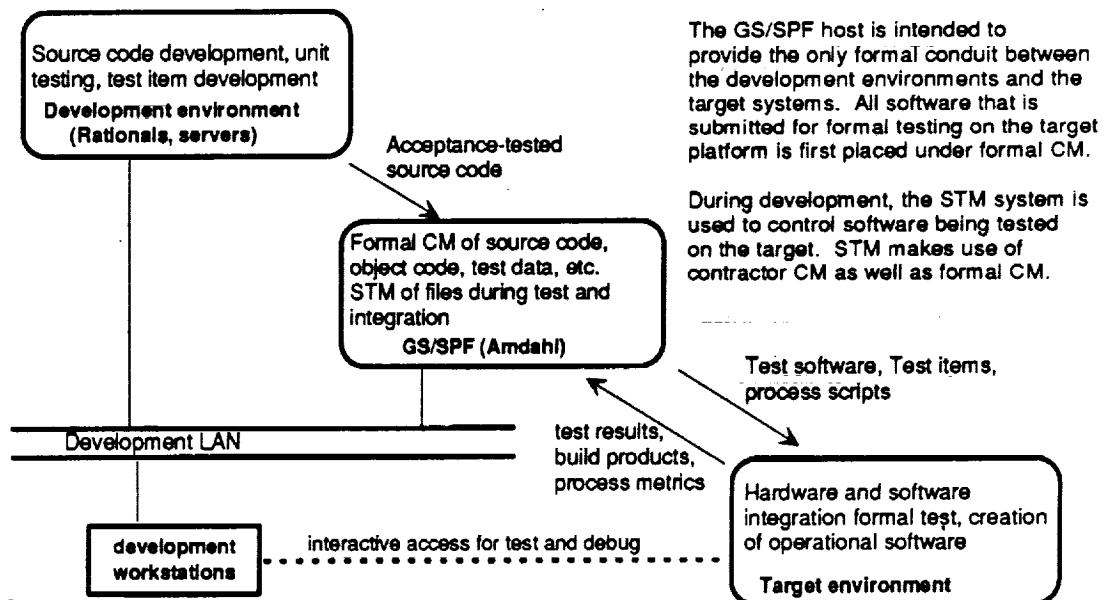


Figure 1. Ground Software Development Environment

The SPEs will include Rational computers to support Ada* development. It is also probable that a substantial amount of non-Ada code (primarily C-language) will be developed (or reused) and supported. Workstations and file servers will be used along with the Rationals to support development.

The GS/SPF is an IBM-compatible mainframe with an instance of the SSE SPF software for each major system (SSCC and SSTF). The GS/SPF will host both the formal CM system and the STM system, along with disk storage supporting both systems. STM will serve to support target-based testing prior to operational delivery. Formal CM will be used to manage software following Acceptance Test. For reasons of security and

* Ada is a trademark of the U. S. Department of Defense, Ada Joint Program Office

software integrity, the GS/SPF will serve as the conduit for moving software from the SPEs to the targets.

There are no planned interactions between the two ground system development efforts (SSCC and SSTF). However, there are a number of common interfaces that will be implemented to support the interfaces between SPE and target.

Overview of the STM capability

For the most part, development and initial testing of ground systems software will take place on Rational Ada development computers or on workstations supporting C-language development. Current experience with Ada suggests that it is productive to keep development activities in the Rational environment as long as possible before transitioning to the target environment. There is also some value to performing as much development as possible in the less formal SPE (compared with the more rigorously controlled target environments) to facilitate changes and debugging. The transition from SPE to the target system is usually late in the development cycle.

The transition must eventually be made, however, and software testing must be performed on the target platform. Integration of components, either with test harnesses or with other integrating software, is performed (or repeated) on the target. Where the software must interface with special purpose hardware devices, this target-based testing is even more essential.

Effective testing (of complex systems) generally requires that a *testbed* be assembled, comprised of integrating software, data, test plans, compile-and-build commands, and operator instructions. Testing also requires a mechanism for recording the results of each test in a sequence, and for capturing products (executable images, test output, etc.) for further examination and/or use. Finally, a mechanism is needed to record and make use of relationships among components, such as "After test A is passed, perform test B with the output from test A". These requirements are addressed by the SSE STM system.

STM provides support for defining and controlling test elements, for assembling test resources in a testbed, and for capturing results and products. Through the use of command scripts (which are themselves controlled), STM also supports the transfer of test elements to target platforms, the initiation of compile-build-test sequences, the recovery of products, and the baselining (into formal CM) of tested components. STM does *not* directly support the execution of tests on the target; that function is left to the test operator (or ITVE system, if one exists). Figure 2 shows in broad strokes the support provided by STM for testing on target platforms.

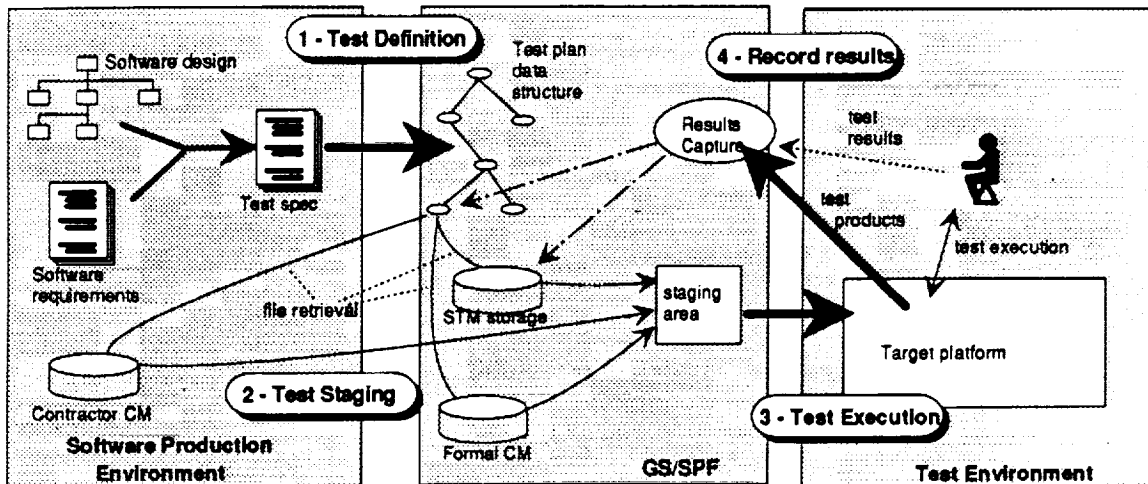


Figure 2. Four steps in target-based testing.

STM data structure

The SSE STM system is based on a structure of data objects and relationships recorded in an Oracle-based data structure. A test plan is actually stored as the set of objects and relationships that comprise the plan. (Some of the possible relationships are predefined; others can be added by the STM administrator on a particular SPF). The STM system deals with both logical and physical objects.

STM objects include operation scripts as well as test objects such as source files and test data. These scripts amplify the test information embodied in the object relationships.

STM objects (logical and physical)

A primary element of the STM system is the data structure used to represent logical objects. STM distinguishes between *physical* objects, such as source files, and *logical* objects, which are definitions and descriptions of physical objects. The STM data structure itself contains logical objects, which may include pointers to physical objects. (The physical objects may or may not reside on the STM host computer.) The logical object data structure in the STM system provides support for recording relationships and attributes of physical objects.

The STM system uses the host-computer file system to provide storage for physical objects. This capability can be used to free up data storage in the SPEs. But the physical

location of objects is relatively unimportant to the STM, which provides a measure of location independence through its logical object definitions.

The STM system includes a mapping between storage location data and the commands that STM uses to retrieve the specified file. For example, the storage location "SSE CM" maps to a "Get component" command with appropriate parameter substitution. This mapping can be augmented by the STM administrator to include any accessible storage (e.g., a file server in an SPE). The specification of a physical object can be as precise (e.g., name, type, version) as is supported in the storage system.

Figure 3 provides examples of storage location specifications in object definitions.

<pre> Test_Configuration MODULE_1_Test --Attributes Description "This is a test procedure" Date_Created "December 1, 1991" Creator "Sam" --Relations Uses_Component MODULE_1 Uses_Build_Instructions BUILD_MODULE_1 Uses_Host_Load COMMON_HOST_LOAD Uses_Test_Script TEST_RUN_DELTA_1 END MODULE_1_TEST </pre>	<pre> Component MODULE_1 --Attributes Description "This module is in Formal CM" --Location data Location="SSECM" Location_data="CSCI_1.CSC_2.CSU_3, 3.2.1" END MODULE_1 Component MODULE_2 --Attributes Description "This component is on a server" --Location data Location "VAXServer_1" Location_Data "[CSC1.SAM]module_2_1.ada" END MODULE_2 </pre>
---	--

Figure 3. Example of a Test Configuration Specification

A typical logical object can be a specific test configuration: a specification of elements required in testing a particular software component. The test configuration could include the following elements and attributes:

- a reference (pointer) to the software component to be tested, including the *component name* (and CCB version number, if appropriate), the *location* (e.g., formal CM, a Rational, or other storage), and a specific *reference* used in retrieving the component from storage (e.g., a file name)

- references to test data, related libraries, test scripts, compile/build scripts--to each of the items needed to perform the specified test
- reference to a *staging script*, which is used to collect all items into a staging area and ensure that all requisite items (including satisfactory completion of prerequisite testing) are available
- status of the test procedure itself (Ready for Test, Passed, Failed, etc.)

Figure 3 shows a sample test configuration in script form. Within the STM system, this definition would be stored as objects, attributes, and relationships.

The STM logical structure is designed to support the natural interrelationships of tests in a software structure. It supports a hierarchy of testing, with user-specified conditions.

A test procedure can be defined to be dependent on the passage of other, prerequisite tests. Using these dependency relationships, a tree structure is defined that models the build-up and integration process for major elements such as CSCIs, subsystems, and systems. Figure 4 represents the graph structured nature of test procedures compared with an integration graph structure.

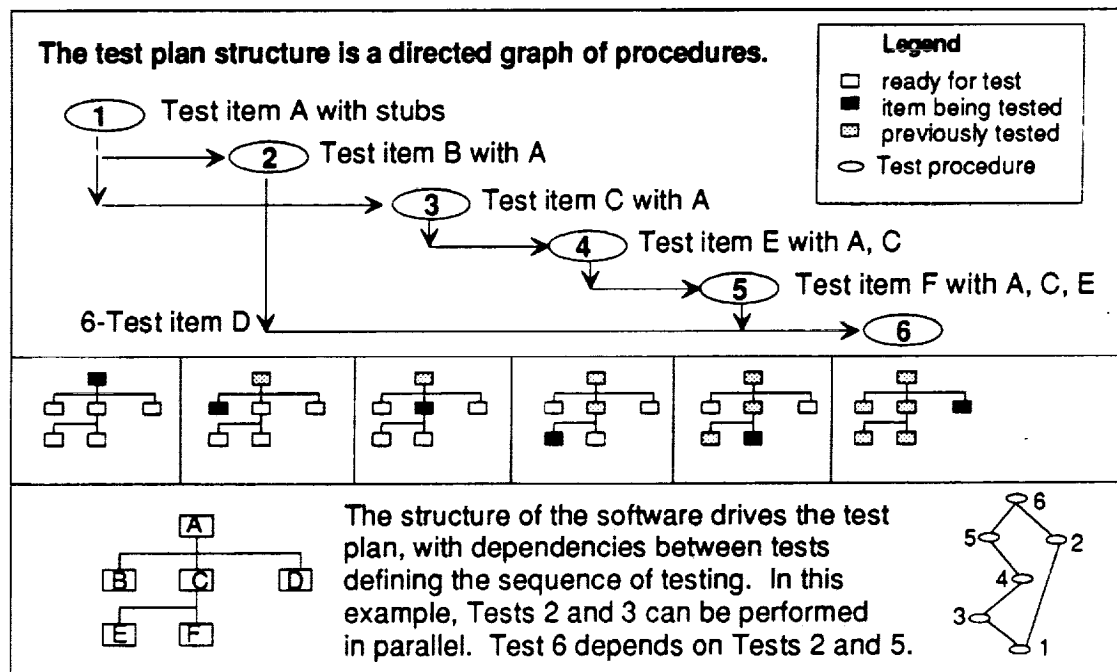


Figure 4. Test plans are based on software design

The STM uses Oracle to store logical objects in relations and attributes. The database is constructed from one or more *definition scripts*, created as text files on workstations and interpreted by the STM Resource Definition tool (TResDef). The object definition scripts can be regenerated from the database, for example to serve as templates for different build structures. Figure 5 illustrates the definition process for object definitions as well as for operation scripts (discussed below).

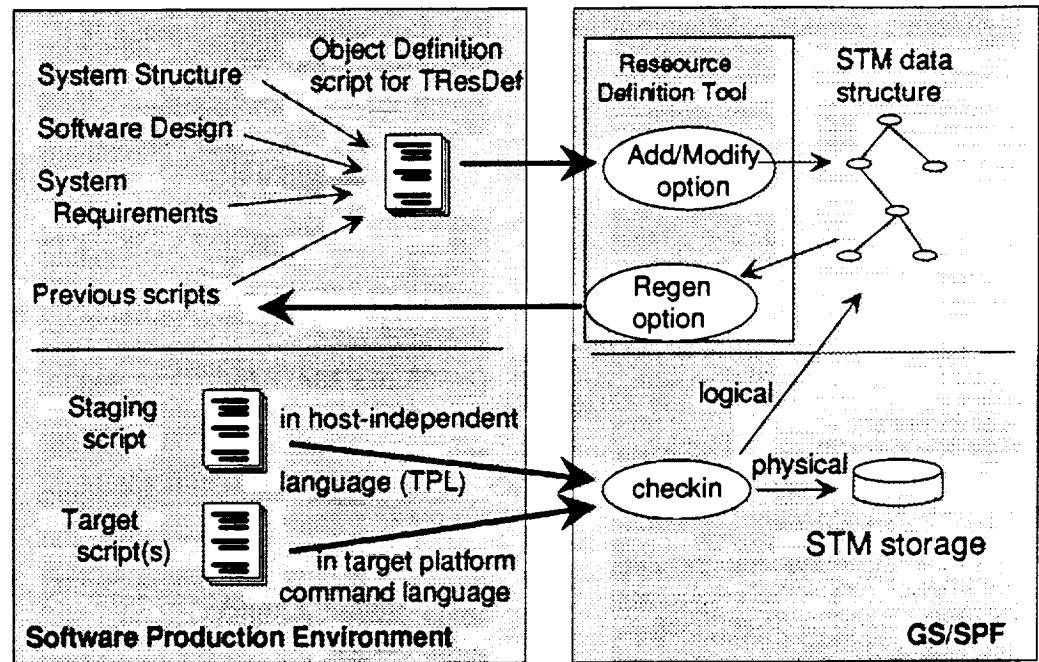


Figure 5. Development of test definition scripts

The object definition scripts define objects that make up the test plan. The Resource Definition tool also supports definition of new types of objects and relationships, such as user-defined storage locations for physical storage of objects.

Operation scripts

In addition to the object definition script, two other types of scripts can be created and made known to the STM. *Staging scripts* are used by the STM to assemble all items of the testbed and transfer them to the target. Staging scripts are written to be executed on the STM host computer; they can be written in the command language of the host (REXX for the GS/SPF) or in the host-independent TPL format that supports translation of STM object names to host-based file names (as required). *Target scripts* are invoked

on the target platform to perform compilation, linking, and possibly testing of components. Target scripts are written in the command language of the target platform.

Figure 5 illustrates the construction of operation scripts, along with the creation of object definition scripts. Figure 6 shows a sample script.

```
--Test procedure for MODULE_1

LET COM_STATUS := FAIL
LET LINK_STATUS := FAIL

PERFORM COMPILE WITH SOURCE_ID = ADA_1_S,
LIB_NAME = 'CONSOLE_LIB'

PERFORM COMPILE WITH SOURCE_ID = ADA_1_B,
LIB_NAME = 'CONSOLE_LIB'

IF COM_STATUS == GO THEN
PERFORM LINK WITH UNIT_NAME = MAIN,
LIB_NAME = 'CONSOLE_LIB'

IF LINK_STATUS == GO THEN
PERFORM RUN_SCRIPT WITH SCRIPT_ID = 'WALKTHRU.TEST',
OBJECT_ID = ADA_1_S
```

Figure 6. Sample of a test process script (in TPL)

STM operations

The primary operations supported by the test management tool are:

- staging all necessary test files, and invoking any supplied scripts (copying them to the target),
- capturing the results of tests, including both status and products,
- checkout/checkin and tracking of change status on controlled files, and
- providing reports on test status.

Other operations include processing specific inquiries about status and structure, setting test status without performing tests (e.g., "failing" a component to make it unusable to other tests that require it), import and export of definition scripts, and facilitating component checkin to formal CM.

Staging a testbed

Staging a test procedure uses the test procedure definition to determine the status (e.g., Ready for Test or Passed Test) of all necessary files, and to assemble the files in a *staging area*. The files to be retrieved may reside on the host (for ground systems software, the GS/SPF) under STM control; they may be under formal CM (also on the GS/SPF); they may be under CMVC control on a SPF-connected Rational; or they may reside in some user-defined location (e.g., contractor CM) accessible over the LAN. The staging area is implemented in the GS/SPF file system.

The staging script is written to be executed on the SPF (that is, the STM host computer); not, for example, on the target platform. The staging script can submit a target-platform script to manage the test setup and execution process, but it does not directly execute on the target.

Figure 7 illustrates the test staging process.

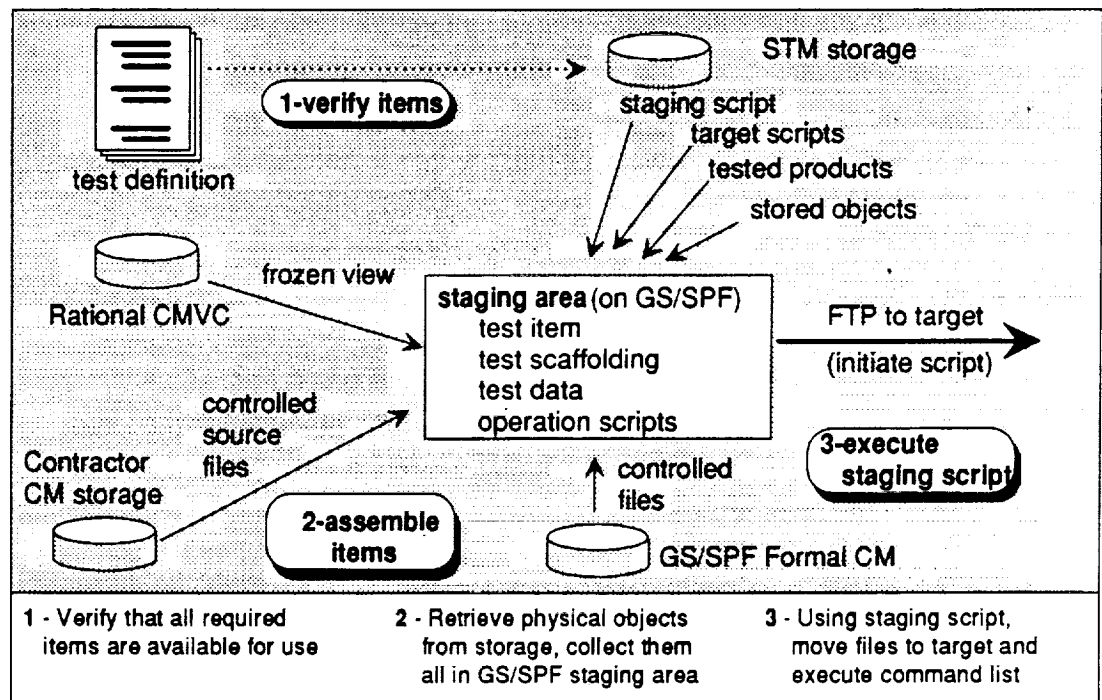


Figure 7. Staging elements in a testbed

Testing is performed on the target. Testing may include compilation and linking, definition of resources (such as hardware devices or communications streams), and execution of the component being tested. STM is not directly involved in this process.

Results Capture

STM supports capture of test results after the test execution. The tester first copies to the STM host any product files that are to be captured. (The STM system does not have the capability to retrieve files from another platform). The tester then logs into STM and invokes the Results Capture tool to report on the status of the test. The tool prompts for products that should be placed under STM control on the SPF, and then retrieves those products (e.g., compiler output, executable files, output data) and places them under STM control. The Results Capture tool does not perform any analysis on the results; it merely records the test operator's report.

The products captured are a special type of object, derivative of some other controlled object. They cannot be checked out for editing. If the base object is checked out or otherwise modified, the captured products are marked "unavailable for test".

The STM can be directed to perform a component checkin to the CM system upon successful completion of testing. In this function, the STM system essentially acts as a user of the CM system; STM does not interact directly with CM tables or data structures. In order for the checkin to succeed, the STM user must have appropriate CMO authority and relevant change-instrument ID(s), just as would be needed in a user-directed checkin to formal CM.

Checkout/Checkin

STM provides a level of configuration management for object under its control. Objects can be checked out for editing and checked back in after changes. Note that since the *physical* objects may reside in the development area, the checkout/checkin is primarily a *logical* control procedure.

When an object is checked out, it becomes "Not ready for Test" and so cannot be used in any testbed which might require it. Also, as noted above, any derivative product of the item, such as an executable image that has been captured from the target, is marked unusable for testing. (This prevents related tests from using an obsolete product.) When the component is checked back in, it may be marked "Ready for Test".

STM Reporting

STM provides a set of predefined reports on the contents of the database. It also provides a limited capability for user-defined reports (the STM ad-hoc report) to be specified and generated. The internal structures of the STM system are not published information, to maintain the integrity of the STM security model; there is no capability for users to create SQL access commands for reporting. (This differs from the capability provided by the SSE CM system). The ad-hoc report format, however, does provide for user-specified selection and reporting parameters. Detailed information on the reporting process (e.g., specific report definitions) is not yet available.

Implications for use in the GSDE

STM clearly will play a significant role in the later stages of target based testing of ground system software. Depending on the particular processes, it may play a role in early (unit-level) target testing as well. There are several implications for the overall development process that stem from the capabilities provided, and those not provided, by STM. This report addresses the following four implications: need for advance planning, need for target-command-language scripts, availability of a "low-cost" controlled conduit to the target, and integration of STM with contractor CM.

Each of these topics is discussed in a separate section below. Briefly, the four implications are:

- A substantial amount of advance planning and test definition is required to make effective use of STM. (This planning will reap dividends in using the SSE Build Process capability, to be provided in OI 7.0).
- To use STM as a coordinator for target-based activities (or at least as an initiator), scripts written in the target command languages will have to be created and recorded in the STM system. Some general standards and templates for target-based scripts would greatly simplify this process.
- The STM system provides a controlled conduit between the SPEs and the target platforms that does not require the rigor of the formal CM system. This conduit should obviate the need for any "short-circuits" bypassing the GS/SPF.
- STM can make use of files stored under contractor CM as well as those stored on the GS/SPF. This will simplify the integration of STM with the development process.

Each of these considerations is discussed in the followings sections of this report.

Requirements for test planning and definition

Testing software elements of a complex system involves testing a large number of interfaces. These include program structure ("will various components compile together?"), functional interplay ("are the services provided precisely what other components expect?"), data compatibility, and timing considerations. Because of the complexity of these interfaces, it is often more effective to use the developed software as its own test scaffold than to build complete test scaffolding for each component. As the subsystem under test grows (i.e., more elements are integrated), stubs are replaced with working software and the entire subsystem is tested.

STM is specifically designed to support such gradual integration testing, but the overall test sequence and goal (a description of the completed system) must be specified in some detail. This advance planning has an added benefit in complex systems development: since many subsystems will be built and tested by teams of programmers, the test definition can usefully serve as a governing framework to control and monitor the testing.

Object definition scripts (which are loaded with the Resource Definition Tool) must define precisely where every required component is stored. For components managed by formal CM, a specific CCB version can also be specified. As changes are made during integration and test, the test definitions may have to be changed to reflect new components (i.e., new versions). The test configuration can be defined to retrieve the latest version, or some specified version of a file.

STM is intended to operate on the most recent edition of a file. It does not save multiple versions of logical objects in its own controlled storage. Although it can save sequential physical files, it ordinarily provides *access* only to the latest version. (Earlier versions of physical objects can be retrieved with the Checkout procedure.)

For components that are *logically* managed by STM but *physically* stored outside the STM system (for example, in contractor CM), changes made are only known to STM when a changed component is checked back in. If such components are changed *without* using the checkout/checkin procedure, the integrity of the test process will be compromised. STM presumes that externally stored files are under some form of CM, and does not exert its own control.

The design of STM is intended to provide maximum flexibility in the use of the tool, without overly constraining the *manner* of use. It is based on the assumption that components are placed under some form of trusted storage. Files can be checked in to the STM for controlled storage, or kept under contractor CM in the SPE. However, the carefully constructed test plans will not be useful unless *some* control is applied to the files named in the plan.

From a system operations perspective, it is critically important that the software that gets *delivered* is the same as the software that was successfully *tested*. The Build Process capability that will be provided in OI 7.0 for formal delivery will likewise require a detailed, rigorous definition of how components are integrated to make up a delivery. The Build Process will make use of the same support structure as the STM. Test plans constructed for the STM should therefore be useful in constructing the build scripts for software delivery.

Requirement for staging and target scripts

The staging scripts (typically written in TPL) created for the STM can be used to orchestrate the build-up of executable images on a target system. The staging scripts can direct the use of predefined, controlled target scripts to compile and link source files and libraries. The target scripts direct the actual target processes. Predefined target scripts, for example, would be used to select a library, compile a source file into that library, and generate an executable image. The TPL syntax includes a PERFORM command that can be extended by the STM administrator for special purposes. Platform-unique functions (e.g., compiling a module) can be performed through this tailoring mechanism.

These target scripts might also serve as the build definition scripts for formal build and delivery. When the Build Process becomes available with OI 7.0, these scripts may be convertible to the build control lists for formal delivery.

The capabilities for tailoring and templates provides an opportunity to standardize the interface between the SPEs and the target environments. The availability of standard scripts (e.g., "compile item *x* into library *l*") and templates for target-based processes would help in simplifying the cross-platform testing activity. Since the SSE is currently concentrating on the SDP as a target, and not on ground system target platforms, we recommend that an effort to create standard scripts and templates for use in ground system testing be initiated.

Comparison of STM and CM interfaces to targets

STM provides a mechanism for transferring software to the target for compilation and test without the rigor of the formal CM system, while providing enough control to ensure that development proceeds in a manageable fashion.

There are compelling reasons why there should be restrictions on the process of moving source files onto target platforms. Physical security of the target environment is one reason; ensuring integrity and reliability of delivered software is another. NASA has directed that all source file transfers to the targets are to occur through the GS/SPF.

There has been some concern voiced that formal CM on the GS/SPF could become a bottleneck during some stages of testing, such as during hardware/software integration. The number of steps required to change a controlled component is daunting, even when approval authority resides with the contractor (i.e., no NASA review is required). Formal CM was not designed to support the frequent small changes that may occur during rehosting and testing with new hardware.

The suggested alternative of *bypassing* control (to facilitate development) is obviously unacceptable on projects as large as the SSTF and the SSCC. STM provides a means of using the GS/SPF as the transfer point *without* the elaborate formal control provided by the CM system.

The emphasis in STM is on *recording* operations, rather than on limiting and controlling them. Files must be defined to STM before they are transferred to the target. STM relies on the file access and security mechanisms of the host or of the SPE to ensure that only valid files are checked in. In that sense, STM acts as an interface between contractor CM system and the target environment.

After successful testing under STM control, tested components can be checked in (or back in) to formal CM for build-up and delivery.

Integrating contractor CM during testing

As noted above, STM can utilize user-defined file storage for physical objects that are logically checked-in to the STM data structure on the GS/SPF. This means that developers can keep files in the contractor CM system during the testing process, promoting the files into formal CM only when necessary for integration purposes. This capability should reduce the concern about the onerous requirements of formal CM for testing purposes. It does, however, entail some risks. STM has no way of detecting whether a *user-controlled* physical object has been changed following a successful test.

(Before an object passes its tests, changes do not affect other objects and so are not of concern to STM. After passing its tests, however, an object becomes available for use by other tests. Such an object therefore should not be changed without notification).

To mitigate this risk, we recommend that STM in the GSDE be restricted to using *controlled* files in the SPE for physical object storage ("trusted storage" in STM parlance). The STM should not be permitted to retrieve files from any and all locations (e.g., a developer's workstation), but only from configuration managed storage.

Since STM does not provide version control for parallel testing of different versions of the same software, developers will have to use version-specific names to maintain integrity of files. This implies that project-wide naming conventions should be developed to ensure consistent use of file names. (Another approach would be to write a CR to the STM to support such versioning).

Summary

In spite of its flight software orientation, the Software Test Management capability appears to be a good match for a number of unresolved problems in ground systems software development. There are some shortcomings, particularly in the area of parallel development of different versions of software, but workarounds do exist. The major benefit of using STM is that controlled access to the target environment can be provided without the excessive rigor of formal CM.

We recommend that the SSE STM system be adopted as an integral element of the GS/SPF. We also recommend that all transfers of source files to the target platforms make use of either formal CM or STM. (This would not affect interactive access to the target environment, which is required for testing and debugging.)

We recommend that CRs be written to require that STM support multiple, parallel versions of software, and that the STM system be able to retrieve products from platforms other than the SPF.

We also recommend that an inventory be performed of the platforms to be used in the SSCC and SSTF target environments. If there is sufficient commonality in the command languages used (e.g., the Bourne shell), a CR should be written to require that the TPL support one or more of those target command languages along with DCL and REXX.

Glossary and Abbreviations

CCB	configuration control board
CM	configuration management
CMO	CM Office (or Officer)
CMVC	Code Management and Version Control (a Rational subsystem)
contractor CM	a CM system managed by the contractor for software which has not been delivered to formal CM; part of the SPE
CSC	Computer Sciences Corporation; also computer software component
CSCI	computer software configuration item
DCL	command language used for DEC VAX computers
DEC	Digital Equipment Corporation
formal CM	the SSE-provided CM system residing on the GS/SPF; it contains the software which has been delivered to NASA, and forms the basis for delivery to operations
GS/SPF	Ground Systems/Software Production Facility
GSDE	Ground Systems Development Environment
IBM	International Business Machines Corporation
ITVE	integration, test, and validation environment

JSC	Lyndon B. Johnson Space Center, Houston, Texas
LAN	local area network
OI	operational increment (the added functionality in a new release)
REXX	command language used for IBM VM mainframe systems
RICIS	Research Institute for Computers and Information Systems
SDP	standard data processor (for Space Station Freedom)
SPE	software production environment
SQL	structured query language (a database access language)
SSCC	Space Station Control Center
SSE	software support environment
SSTF	Space Station Training Facility
STM	Software Test Management (an SSE capability)
TPL	Test Process Language (defined for the STM)
TResDef	Resource Definition Tool (used to define objects in the STM system)
UHCL	University of Houston - Clear Lake