SAND REPORT
SAND2004-3201
Unlimited Release
Printed July 2004

ITS Strategic Test Plan Revision 1.0

Leonard Lorence, Brian Franke, Ronald Kensek, Thomas Laub, Martin Crawford and Lisa Cordova

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185 and Livermore, California 94550
Sandia is a multiprogramming laboratory operated by Sandia Corporation,
a Lockheed Martin Company, for the United States Department of
Energy under Contract DE-AC04-94AL85000.

Sandia National Laboratories

SAND2004-3201
Unlimited Release
Printed July 2004

ITS Strategic Test Plan

Revision 1.0

Leonard Lorence, Brian Franke, Ronald Kensek, Thomas Laub, and Lisa Cordova

Simulation Technology Research Department 15341
Sandia National Laboratories
P.O. Box 5800
Albuquerque, NM 87185-1179

Martin Crawford
K-Tech Corporation
1300 Eubank Blvd. S.E.
Albuquerque, NM 87123-3336

Abstract

This test plan describes the testing strategy for the ITS (Integrated-TIGER-Series) suite of codes. The processes and procedures for performing both verification and validation tests are described. ITS Version 5.0 was developed under the NNSA's ASC program and supports Sandia's stockpile stewardship mission.

Approval

Document Name: ITS Test Plan

Revision Date:7/04

Prepared by: Leonard Lorence

Approval: ______7/1/04____________________

Leonard Lorence, Manager, 15341

Concurrence: ______7/1/04____________________

Ronald Kensek, ITS PI, 15341

Document Revision Control: The document is configuration managed by Leonard Lorence under Web File Share.

Revision history:

Revision 1.0 (July 2004): Initial Test Plan for ITS Version 5.0

Table of Contents

1. Purpose

This document is the Strategic Test Plan for the ITS radiation transport suite of codes. It was created in 2004 and is the first of several intended software quality process documents for ITS. Other process documents that will be created are:

(1) ITS Strategic Plan for Requirements,

(2) ITS Strategic Plan for Design and Implementation,

(3) ITS Strategic Plan for Release, and

(4) ITS Strategic Plan for Support (Configuration Management, Third Party Software Management, and Training).

It is expected that this document will continue to evolve under configuration control. The Strategic Test Plan describes a snapshot of the current testing activities underway for ITS, or those that we expect to implement in the very near future. Future goals, both near and long term, to improve our testing processes and procedures are also discussed.

This test plan describes the strategy followed by the ITS team for testing. It includes a description of the processes and procedures for performing tests, the test deliverables, and the frequency of the tests. As explained in the ASC Software Quality Engineering Policy,[10] code teams are responsible for code development and verification. Unlike software quality and verification, code teams are not solely responsible for validation. Rather, validation requires:

"a broad set of tasks and participation from a number of different communities: experimental, analysis, code development and customer. For the ASC Program, this wide-ranging scope of activities is the responsibility of the Verification and Validation (V&V) program. The project teams contribute to these validation activities by performing software verification."

In this document, we also recognize that the ITS team is engaged in validation testing. While we do not attempt to address the full range of validation activities here, we recognize some aspects of validation testing resemble those of verification testing (e.g. the notion of a hierarchy of tests). Consequently, we have decided to include some discussion of validation testing in this Strategic Plan because it makes sense to do so. However, it must be recognized that, if software quality activities are evaluated, the ITS team is not solely responsible for validation.

This strategic test plan is not meant to describe or list specific tests. For instance, it will not contain a comprehensive list of all current regression tests. *Our near term goal is to develop ways of generating lists from the test directories in the CVS repository*.

2. Overview

The Integrated-TIGER-Series, or ITS, is a suite of Monte Carlo codes for 1D, 2D, and 3D coupled electron-photon radiation transport. Elements of this code package have been under continuous development at Sandia for over 30 years. In 1992, Version 3.0 of ITS was released world wide with unlimited distribution. Since 1997, the ASC program has funded further development of ITS, including such features as adjoint transport, massively-parallel implementation, and CAD-based transport. Version 5.0 of the ITS code incorporates all of these features, and is internally released within Sandia to support the laboratory's stockpile stewardship mission.

Since the ITS code is used for the design and qualification of new components in the stockpile, it is considered "Class A" software,[1] for which all SQE practices are required. For stockpile stewardship, the ITS code is essential to the modeling of the radiation response of nuclear weapon systems and components in hostile environments. Such analysis supports the design of radiation-hardened weapon systems and the qualification of these systems to their requirements.

For stockpile work, ITS can directly produce useful design information such as dose-rate in electronics, or ITS may be used in combination with other ASC codes to predict the electrical and mechanical response of components to radiation. This strategic test plan focuses only on the verification and validation of a functionally independent ITS, and not ITS in combination with other codes. Validation of the combined predictive capabilities of code combinations are discussed in separate Integrated Verification and Validation Plans.

Some of the major features and functionality of the ITS code suite (code names in the suite are indicated in italics) include:

(1) Fortran with C++ links to CAD software,

(2) Electron and photon coupled transport from 1.0 keV to 1.0 GeV (the upper bound for hostile environment simulations is 20 MeV),

(3) 1D (*TIGER*), 2D, (*CYLTRAN*) and 3D (*ACCEPT*) codes, with options to include externally imposed electromagnetic fields in void regions for the multi-dimensional codes *CYLTRANM* and *ACCEPTM*,

(4) Enhanced ionization relaxation physics (*TIGERP, CYLTRANP, ACCEPTP*),

(5) Both adjoint (multigroup) and forward (continuous-energy or multigroup). The multigroup code in ITS is called *MITS*,

(6) The ITS code package includes the cross section generating codes for multigroup (*CEPXS*) and continuous-energy (*XGEN*) cross sections,

(7) Both combinatorial geometry and CAD geometry capabilities,

(8) Demonstrated links to mechanical codes and electrical codes,

(9) Massively parallel (MPI) simulation capability, and

(10) Operation on multiple platforms, e.g. Linux, IBM AIX, Sandia CPLANT, ASC-Red, White, Q, and all future ASC platforms.

3. Testing and the Software Development Cycle

As will be discussed in the ITS Strategic Plan for Design and Implementation, the ITS team begins to address testing issues early in the design phase of a new software feature. If the software development is to be a formal process, the software development owner (designated by the "Assigned To" field in SourceForge) will create a Feature-Specific Test Plan. Among other things, the Feature-Specific Test Plan will discuss the scope (or requirements) of the functionality of the new feature. Further discussion of requirement creation and management will be the subject of the ITS Strategic Plan for Requirements.

We will follow an IEEE Template[2] for the Feature-Specific Test Plan. The template is shown in Appendix A and an example from the ITS project is shown in Appendix B. The template is made available to team members through Web File Share (WFS). Feature-Specific Test Plans are created early in the design phase. Such plans evolve throughout the design, implementation, and test subphases.[1] The owner of the formal software development will make the Feature-Specific Test Plan available to the entire team through WFS.

The ITS PI (to be defined in Section 14) will determine which new software development activities are formal and require a Feature-Specific Test Plan. The plans are reserved for major software development activities. It is anticipated that each member of the small ITS team will own, at most, one or two formal software development efforts per year. We will use these terms "software owner" and "code developer" interchangeably.

The PI assigns formal projects in the following manner:

(1) at the start of the new fiscal year, when the yearly plans and milestones are finalized for the ASC project office, the PI will record these efforts as Feature Requests with an assigned owner. If he believes that these are formal efforts, he will so indicate in the "Category" field in SourceForge, or

(2) as code team members enter software development activities in SourceForge (Feature Requests or Bugs), they will choose their activities to be either "Formal" or "Informal" using the "Category" Field in SourceForge (the default is "Informal"). SourceForge E-mails are sent to all team members, including the PI, that describe the software development activity and the level of formality. The PI will review the choice of formality. A lack of response will signify concurrence. We note that bug fixes can instigate "Formal" software development, which would necessitate a Feature-Specific Test Plan.

The types of tests included in the Feature-Specific Test Plan can span most of the test hierarchy. However, code development is most commonly associated with unit / integration, regression, and software verification tests. As will be discussed, tests at different levels of the hierarchy can have different acceptance criteria. Specific acceptance criteria (e.g. Pass/Fail criteria) are discussed in the Feature-Specific Test Plans.

After the software development cycle has been completed, the owner of a formal software development effort will generate a Feature-Specific Test Summary Report. All summary reports will conform to the IEEE recommended template[2] shown in Appendix C. An example from the ITS project is shown in Appendix D. The template is available to team members through WFS. The owner of a formal software project will make the associated report available to the entire team through WFS.

The following archives are used for various testing artifacts:

(1) Web File Share (WFS): All formal verification documents (e.g. Feature-Specific Test Plans, Feature-Specific Test Summaries, and AVERTS documentation).

(2) ASC V&V Records and Document Management System (RMS): All formal validation documents (e.g. documents for each AVALTS and V&V Plan). All documents stored here are also automatically stored in WFS. Storage in RMS is a programmatic requirement of the V&V program.

(3) SourceForge: E-mail archive (e.g. results of regression tests and general testing activity such as coverage testing), feature tracking, bug tracking. E-mails to SourceForge are also automatically sent to the entire ITS team.

(4) CVS: Separate directories exist for regression tests, software verification tests (the entire ensemble is called the CVERTS), validation tests (the entire ensemble is called the CVALTS), and the unit / integration tests. The input files necessary to run these tests are stored in CVS. Depending on the acceptance criteria (for the CVERTS) or the comparison metrics (for the CVALTS), the output data from these tests may also be stored. For instance, since the acceptance criteria for regression tests depend on the comparison of new and old output data, the output data is stored. Other tests may have simpler acceptance criteria (e.g. “not crashed” for array bound checking) that do not require storage of the output data. As will be discussed, additional documentation for tests in the CVERTS and CVALTS are also stored in CVS.

4. Testing Hierarchy

The strategy for testing ITS involves a hierarchy of tests in which tests at different levels are performed with different frequencies, for different purposes, and with different deliverables. Deliverables are typically documents or E-mails. Testing activities can be divided into the following categories, which are organized approximately by the frequency with which the tests are repeated (from most to least often):

(1) Commit tests (also known as smoke tests). Deliverable is E-mail to SoureForge regarding the results (either passed or passed with exception with explanation).

(2) Regression tests. Deliverable is E-mail to SourceForge regarding results (either passed or passed with exception with explanation).

(3) General testing activities (coverage, memory, static code analysis). Deliverable is E-mail to SourceForge regarding results.

(4) Software verification tests (CVERTS). Deliverable is E-mail to SourceForge regarding result of running the tests, including an assessment of the acceptance metrics for these tests.

(5) Software validation tests (CVALTS). Deliverable is E-mail to SourceForge regarding result of running the tests, including an assessment of the comparison metrics for these tests.

(6) Application-specific software verification tests (AVERTS). Deliverable is a document stored in WFS that details results of running the tests.

(7) Application-specific software validation tests (AVALTS). Deliverable is a document stored in both WFS and RMS that details result of running the tests.

(8) Installation testing. If run by a team member, deliverable is E-mail to SourceForge regarding results.

(9) Unit / integration testing. No documentation or other deliverable is required.

All tests that are stored in the CVS repository will have, imbedded in their input files, a few comment lines that describe the test. These comments will be constructed in a predictable and organized way that allows lists of the tests to be generated. *It is our near term goal to include such comments in the input files and to create a process for generating lists.* As will be discussed, tests in the CVERTS and CVALTS have additional accompanying documentation.

The tests are performed with the following frequency:

(1) Commit testing: Before new code is checked in.

(2) Regression testing: Every month on the main development platform. Every three months on the other supported platforms (except for the secondary code development platform). *Our near-term goal is to start running regression tests at this stated frequency. Our long-term goal is to eventually increase the frequency of regression tests. More automated tools for regression testing, including scripts that exploit restart data rather than output file, will also be investigated*.

(3) General testing activities: Upon branch release, and only on the code development platform

(4) CVERTS testing: Upon branch release, or when changes occur in the output data from commit or regression tests

(5) CVALTS testing: Upon branch release, or when new or different physics is purposely implemented

(6) AVERTS testing: Upon the request of a customer for V&V documentation for their application

(7) AVALTS testing: Upon the request of a customer for V&V documentation for their application

(8) Installation testing: Upon the port to a new computer

(9) Unit / Integration Testing: During the development time of the original software and then at the discretion of future code developers as needed.

The application-specific test suites, the AVERTS and the AVALTS, are more commonly known as the VERTS and the VALTS.[3] The existence of a CVERTS and a CVALTS arose from our own internal need to manage the V&V testing process. For instance, a CVERTS process was proposed by Ed Boucheron and James Peery [see also Reference 8] for verification testing within the ASC program.

5. Commit Tests

Commit or smoke tests are a subset of the regression tests that are run, only on the main code development platform (see next Section), before each CVS check-in of new or modified code. *Our near-term goal is to establish commit testing*. The check-in test suite has been selected so that the tests can be accomplished quickly. The total time to run commit tests should be kept as small as possible. Our goal is to keep these tests under 30 minutes so as not to impede code development. Also, to aid rapid check-in, we only require commit tests on our primary code development platform, the Linux cluster, Crater.

Any member of the ITS team may contribute a commit test and notify team members by E-mail to SourceForge. A lack of response by the code PI will signify concurrence. To lessen the commit burden, the process is fully automated (only requiring the user to launch the script). *The near-term goal is that acceptance criteria are to be fully automated for Pass/Fail.*

Commit tests are a subset of regression tests. *A near term goal is to develop scripts to run just these commit tests.* It is incumbent on the code developer to insure that all commit tests have been passed before new coding is checked-in.

Commit tests will be automated through scripts. If the commit tests pass, an E-mail that states this is sent to SourceForge. If the commit tests fail, the team member does the following depending on these three scenarios whereby the output from the commit tests can change:

(1) A change occurs because of a change in format of the output file. Presently, the new and old output files are differenced, so that even a trivial format change can cause the output file to

change and the commit tests to fail. In this case, the ITS team member must verify that the output data of the code has not changed, indicate in an E-mail to SourceForge that the tests have "Passed with Exception", and store the new output files in CVS for future commit testing.

(2) A numerical change in the output data occurs (e.g. this could happen because of a change in the random number sequence) without the purposeful introduction of new physics. In that case, the team member is required to run the CVERTS (as we discuss in Section 10, we may decide to use a subset of the CVERTS for this purpose), send E-mail to SourceForge, and store the new output data for the tests in CVS for future commit testing.

(3) If the output data from the regression tests change because of the purposeful introduction of new physics, the team member is required to run both the CVERTS and the CVALTS, send E-mail to SourceForge, and store the new output data in CVS for future commit testing.

6. Regression Tests

The ASCI V&V Guidelines defines regression testing as:[3]

"Regression testing is the activity of regularly building the code and executing a series of tests designed to verify that the code works as expected for all computational platforms supported. Minimally, such testing should be done when either the code or operating platform changes. This activity includes the development and maintenance of a regression test suite. This test suite should be designed to exercise as many of the code features as possible."

We use regression testing[1] for several different purposes:

(1) To ensure, in a more comprehensive manner than commit tests, that defects are not introduced by changes to the code and that the changes function properly,
(2) To detects changes in code output stemming from changes in the OS, compilers, or third-party libraries,
(3) To provide the tests for coverage analysis, and
(4) To provide the tests for installation on new computing platforms.

We support ITS on all the main ASC platforms, and upon our code development platforms. The supported platforms for ITS are currently:

1. Q (Tru64)
2. White (AIX)
3. Red (SUNMOS)
4. Crater, main code development platform (Linux)
5. Scorpio, secondary code development platform (AIX)

Regression tests are periodically run on all of these platforms at the stated frequency. The stated frequency of regression testing on the main code development platform is much greater than on the other supported platforms. Presently, we are only doing regression testing on our main code development platform. *It is our near term goal to implement regression testing on all supported platforms, except for the secondary platform.* The code PI is responsible for insuring that these regression tests are performed at the stated frequency.

While ITS may be built on other platforms, the ITS team is not responsible for maintaining the code on these platforms and they do not perform regression tests on these other machines. Users of ITS on non-supported platforms are informed of this policy.

Any member of the ITS team may contribute a regression test and notify team members by E-mail to SourceForge. A lack of response by the code PI will signify concurrence. Regression tests are stored in a separate directory of the CVS repository.

Regression tests are partially automated through scripts. The user can launch the script of all regression tests. However, acceptance criteria may be more complex than Pass/Fail, and require the team member to inspect the output data and make comparisons to the old output data. If the regression tests pass, an E-mail that states this is sent to SourceForge. If the regression tests fail, the team member does the following depending on these three scenarios whereby the output from the regression tests can change:

(1) A change occurs because of a change in format of the output file. Presently, an output file is examined, so that even a trivial format change can cause the output file to change and the regression tests to fail. In this case, the ITS team member must verify that the output data of the code has not changed, indicate in an E-mail to SourceForge that the tests have "Passed with Exception", and store the new output files in CVS for future regression testing.

(2) A numerical change in the output data occurs (e.g. this could happen because of a change in the random number sequence) without the purposeful introduction of new physics. In that case, the team member is required to run the CVERTS (as we discuss in Section 10, we may decide to use a subset of the CVERTS for this purpose), send E-mail to SourceForge, and store the new output data in CVS for future regression testing.

(3) If the output data from the regression tests change because of the purposeful introduction of new physics, the team member is required to run both the CVERTS and the CVALTS, send E-mail to SourceForge, and store the new output data in CVS for future regression testing.

Regression tests are not periodically run for the non-transport codes associated with ITS since these codes do not change frequently. These codes include the cross section generating codes (XGEN & CEPXS), tools (e.g. ITS2EXO, CG2ACIS, MAPPER, and eventually CONTRAST) and scripts. Of these, only XGEN & CEPXS have regression tests, but most of the tools have at least one software verification test. Any such tests are stored in the same directory with the tool. These are run at the developer's discretion. Since the Monte Carlo regression tests make use of much of the scripting software, some of these other software elements that are stored in CVS along with ITS are implicitly tested each time regression tests are run.

7. General Testing

General testing[1] covers tests that need to be conducted on all software products to meet specific SQE goals, e.g. code coverage, memory testing, and static source code tests. We will perform these general tests only on either the main or alternate code development platforms.

According to Sandia SQE practices:[1]

> Evidence must be provided demonstrating that at least 80% of the software source statements have been executed through testing. Applying an automated tool that uses a specified set of tests (such as the regression tests) typically provides this evidence.

The ITS team uses the Aprobe software to assess code coverage of the regression tests. This is done only on the main code development platform. Since the tool covers commercial third-party code, shared libraries and components that are linked to ITS, the reported coverage tends to be reduced. The tool can also be used to identify which portions of the code (over 60,000 lines) are not tested. The present set of regression tests cover much of ITS. The most significant uncovered sections of the code currently involve error messages. Coverage testing is currently done manually and is not an automated process.

One of the challenges of doing coverage analysis for a suite of codes like ITS is to test all of the various codes that comprise ITS. Over 50 different types of executables can be created from the ITS suite of codes! For instance, different executables can be generated from the following combinations:

1. Geometry
    - a. TIGER (1-D)
    - b. CYLTRAN (2-D)
    - c. ACCEPT (3-D)
        - i. Combinatorial geometry
        - ii. CAD (ACIS, Versions 4 or 6)
2. Physics
    - a. ITS (Continuous-energy)
    - b. MITS (Multigroup cross sections). 1-D and 3-D only
    - c. Mcodes (Applied Magnetic and Electric Fields), 2-D and 3-D only
    - d. Pcodes (Enhanced Ionization and Relaxation)
3. Processing
    - a. Serial
    - b. MPI
        - i. Dynamic Load Balancing
        - ii. Static Load Balancing
    - c. Random Number Generator (1, 2, or 3)
4. Compiler flag combinations (we will choose one scenario, per computing platform, for each of our various types of tests.)

Also, over 50 keywords with numerous sub-keyword options are used to specify the input of the code.

Currently, coverage analysis is only performed for the Monte Carlo portion of ITS. The cross section generating codes, CEPXS and XGEN, which serve as preprocessors, have not been subject to Aprobe coverage analysis, although they have regression tests and are under CVS management. *Our near term goal is to do coverage testing for these codes as well*. Other in-house developed software, such as scripts, are maintained under CVS, but are not regression tested. We do not intend to perform coverage analysis for such ancillary software.

Coverage analysis is done whenever a branch release occurs. This is done by the code PI or his designate on the main development platform. *Our long-term goal is to further automate coverage testing and increase the frequency*. The results of the coverage analysis will be sent by E-mail to the entire team and the E-mail will be archived in SourceForge.

The ITS team performs memory testing with special built-in IBM functions that work only on AIX platforms, such as the secondary code development platform. We intend to investigate other memory testing tools. The software has been useful to track down memory leaks. The guidelines indicate that memory testing should be done prior to check-in. This is too onerous for the ITS team since most code modifications do not have the potential to introduce memory leaks. Therefore, memory testing will be done whenever a code release occurs. This is done by the code PI or his designate on the secondary code development platform (IBM AIX). However, the need for more frequent memory checking will be left up to the developer, particularly if their modifications have the potential to introduce memory leaks. *Our long-term goals are to automate memory testing, increase the frequency of these tests, and use better tools that can operate on our primary code development platform.* The results of the memory analysis will be sent by E-mail to SourceForge.

The ITS code development team does not currently do static source code testing. Tools such as FLINT and CLEANSCAPE will be investigated for this purpose. *Our long-term goal is to do static source code testing.*

8. Application-specific Software Verification Tests (AVERTS)

According to the ASCI V&V Plan Guidelines,[3] verification is “the process of determining that a computational software implementation correctly represents a model of a physical process.” In other words, verification is the process of determining that the equations are solved correctly.

Software verification testing[1] is conducted to demonstrate that specific modeling capabilities function properly without the use of experimental or real data for comparison of results. Tests may include analytical solutions, semi-analytical solutions, and idealized solutions. The manufactured solution testing approach may also be used to demonstrate specific algorithm implementation.

According to the ASCI V&V Plan Guidelines[3], an application-specific verification test suite (which we refer to as the AVERTS) generally consists of three tiers, or categories, of test problems:

> **"Tier I** - tests with exact analytical solutions, including unit tests;
>
> **Tier II –** tests with semi-analytic (reduction to quadrature, to simple ordinary differential equations, etc) solutions;
>
> **Tier III –** idealized problems suitable for code comparison exercises.

This structure for verification testing is advocated in the AIAA V&V Guide."[6]

Examples in each of these Tiers for the ITS code can be found in the code-specific V&V plan.[4] Acceptance criteria for tests in the AVERTS may vary. Simple automation of Pass/Fail criteria for these tests is unlikely. Instead, considerable judgment may be required of the analyst. There are other integrated V&V plans that pertain to the use of several codes to simulate a phenomenon. Verification activities for codes that are used in combination are not discussed here.

Also, according to the Guidelines,[3] different AVERTS are constructed for the different applications or stockpile drivers. Each AVERTS is tied to the application via a PIRT, or phenomena ranking table. The PIRT provides the requirements for the AVERTS. An example of a PIRT for a specific stockpile driver can be found in the ITS V&V plan.[4] The important thing to note is that, for each application, a unique AVERTS needs to be assembled, the latest version of the code run, and the results documented for the customer.

We propose to achieve this in the following manner. As software development proceeds (or other needs arise), the code team populates the code verification test suite or CVERTS in CVS. If a customer requests documentation of V&V testing for their application, the code PI appoints an application investigator who:

(1) Adds a new PIRT to the V&V plan, if needed,
(2) Assembles the AVERTS, if needed, from either culling the CVERTS and/or adding new tests to the CVERTS directory
(3) Reruns the tests in the AVERTS (to insure that the latest version of the code has been used). The application investigator decides which supported computing platforms to use for these tests
(4) Creates a document of test results and stores in WFS for access by the customer and the team.

The AVERTS documentation includes an assessment of these tests relative to their acceptance criteria. See Appendix E for a template for the AVERTS document. Such an AVERTS document was created for adjoint CAD capability of ITS to calculate dose.[7]

9. Application-specific Validation Tests (AVALTS)

According to the ASCI V&V Plan Guidelines,[3] validation is "– The process of determining the degree to which a computer model is an accurate representation of the real world from the perspective of the intended model applications." In other words, validation is the process of determining that the equations are appropriate for the application. It is also the process of quantifying the uncertainties of using a computer model to predict the real world.

According to the Guidelines,[3] an application-specific validation test suite (which we refer to as the AVALTS) generally consists of four tiers or categories, of test problems:

> **"Tier I –** Assess the degree of accuracy to which separable effects (or single phenomena) in the code correctly represent the real world.
>
> **Tier II –** Assess the degree of accuracy to which coupled effects between distinctly identified phenomena in the code correctly represent the real world.
>
> **Tier III –** Assess the degree of accuracy to which integral phenomena in the code, in which many coupled effects may be present, correctly represent the real world.
>
> **Tier IV –** A "Qualification Experimental Campaign" or confirmatory experimental activity designed to assess the readiness of the code for stockpile computing and application to stockpile problems associated with the chosen driver. It is essential that this validation activity be performed in conjunction with the Sandia weapon design community and in coordination with additional experimental opportunities that may arise in the normal course of nuclear weapon program work at Sandia."

"Tiers I through III reflect the AIAA validation test hierarchy[6] of Unit Problems, Benchmark Problems, Subsystem Problems and Full System Problems. Unit and Benchmark problems are mainly contained in our Tier I specification. Our Tier IV has no analog in the AIAA classification. Tiers I through III should emphasize the phenomena and their couplings that are elements of the PIRT. Tier IV problems are not clearly expressed by the PIRT in all likelihood. Rather, this is a termination activity aimed at qualification of a frozen code for the stockpile driver."

Examples in each of these Tiers for the ITS code can be found in the ITS V&V plan.[4] According to the Guidelines,[3] a new AVALTS is constructed for each application or stockpile driver. Each AVALTS is tied to the application via a PIRT, or phenomena identification ranking table. The PIRT provides the requirements for the AVALTS. An example of a PIRT for a specific stockpile driver can also be found in the ITS V&V plan. The important thing to note is that, for each application, a unique AVALTS needs to be assembled.

We propose to achieve this in the following manner. As software development proceeds (or other needs arise), the code team populates the code validation test suite or CVALTS under CVS management. If a customer requests documentation of V&V testing for their application, the code PI appoints an application investigator who:

(1) Adds a new PIRT to the V&V plan, if needed,
(2) Assembles the AVALTS, if needed, from either culling the CVALTS and/or adding new tests to the CVALTS directory

(5) Reruns the tests in the AVALTS (to insure that the latest version of the code has been used). The application investigator decides which supported computing platforms to use for these tests
(3) Creates a document of test results and stores in WFS for access by the customer and the team. The mangers will store in RMS to meet a requirement of the V&V program.

The AVALTS documentation includes ties to the requirements in the PIRT. Rather than having acceptance criteria, validation tests have comparison metrics to compare computational results to experimental data. Validation metrics for tests in the AVALTS may vary. Considerable judgment of the analyst may be required to assess the code's predictive capabilities relative to the metrics that have been established. The AVALTS documentation includes an assessment of comparison metrics. See Appendix F for a template for the AVALTS document.

10. Comprehensive Verification Tests (CVERTS)

The CVERTS is the comprehensive ensemble of verification tests for the code that are stored and managed in the CVS repository. Such a CVERTS was originally proposed in a draft document from Ed Boucheron, et. al. [Ref. 8]. The CVERTS contains every AVERTS, plus additional tests.

The tests in the CVERTS are exercised, on at least one supported computing platform:

1. when a new version of the code is branch released, or
2. when the output data of a commit or a regression test change (e.g. such changes can be induced by changes in the random number sequence or the introduction of new physics). Depending on the acceptance criteria, newer output data may also need to be committed to the repository at this time.

New branch releases are anticipated to occur about once a year (see the ITS Strategic Test Plan for Release). At that time, the code PI or his designate will rerun the tests in the CVERTS and send E-mail to SourceForge. For the second situation described above, if the CVERTS population grows too large, we may consider creating a subset of the CVERTS that can be run more quickly. This reduced verification set of tests would be analogous to the commit tests which are a subset of the regression tests. However, we still propose running the entire CVERTS when a code is branch released.

The running of the tests in the CVERTS is a manual process, and requires assessment of the code's accuracy relative to the acceptance criteria by the application investigator. These assessments are part of the Test Description Document that is stored together with the test.

Each test, or test suite, in the CVERTS will have a document, also stored in the same CVS repository as the test itself, that describes the test. The test description document will have the following format:

1. Name (or identifier) of the test or test suite

2. Owner (who put the test in the CVERTS)
3. Brief description of the test problem (energy regime, materials, geometry)
4. Code capabilities tested
5. Acceptance metric
6. Output data of the code
7. Assessment of the output data relative to the acceptance metric
8. All other relevant information pertaining to running the test (specialized scripts, read-me files, etc.)

Any member of the ITS team may contribute a CVERTS test and notify team members by E-mail to SourceForge. A lack of response by the code PI will signify concurrence. Verification tests are stored in a separate directory of the CVS repository.

A CVERTS directory in CVS has not yet been created for ITS. It is our near term goal to create such a directory and include a document for each test.

11. Comprehensive Validation Tests (CVALTS)

The CVALTS is the comprehensive ensemble of validation tests for the code that are stored and managed in the CVS repository. The CVALTS contains all the tests in all the AVALTS, plus additional tests. Rather than having acceptance criteria, validation tests have comparison metrics to compare computational results to experimental data. Validation metrics for tests in the CVALTS may vary. Considerable judgment of the analyst may be required to assess the code's predictive capabilities relative to the metrics that have been established. These assessments are part of the Test Description Document that is stored together with the test.

The tests in the CVALTS are exercised on at least one supported computing platform:

1. if a new version of the code is branch released, either internally or externally, or
2. if the output data of a commit or regression test changes because of a change in the physics. The newer output data is also committed to the repository at this time.

New branch releases are anticipated to occur about once a year. At that time, the code PI or his designate will rerun the tests in the CVALTS and send E-mail to SourceForge. For the second situation described above, if the CVALTS population grows too large, we may consider creating a subset of the CVALTS that can be more quickly run. This reduced validation set of tests would be analogous to the commit tests which are a subset of the regression tests. However, we still propose running the entire CVALTS when a code is branch released.

Each test, or test suite, in the CVALTS will have a document, also stored in the same CVS repository at the test itself, that describes the tests. The test description document will have the following format:

1. Name (or identifier) of the test or test suite
2. Owner (who put the test in the CVALTS)
3. Brief description of the test problem (energy regime, materials, geometry)

4. Experimental data, plus references
5. Code capabilities tested
6. Comparison metric
7. Output data of the code
8. Assessment of the output data relative to the comparison metric
9. All other relevant information pertaining to running the test (specialized scripts, read-me files, etc.)

Any member of the ITS team may contribute a CVALTS test and notify team members by E-mail to SourceForge. A lack of response by the code PI will signify concurrence. Validation tests are stored in a separate directory of the CVS repository.

While a CVALTS directory in CVS has been created for ITS, a document for each test has not yet been done. However, some documentation for a test suite that is part of the CVALTS[9] , does exist.

12. Installation Tests

According to the SNL Quality Practices docuemnt[1], "installation testing is conducted to confirm that the software installation on a new target platform occurred correctly. Installation tests are typically delivered with the software for execution by the end user. These tests form the basis of customer acceptance tests."

An ITS team member may run installation tests, either at the request of a customer, or for their own purposes. In either case, E-mail is sent to SourceForge regarding the success of the installation tests. Installation tests may also be delivered with the software and may be executed by non-team members who are not required to send E-mail acknowledgement to SourceForge.

Our current policy is that the regression tests are the installation tests. Rounding differences need to be addressed on a case-by-case basis on each new platform.

13. Unit/integration Tests

Unit/integration testing[1] covers low-level structural testing of modules and integrated modules prior to full software product testing. The ASCI V&V Plan Guidelines[3] present further definition:

> Unit testing is the activity of testing code units against their requirements, specifications, and design. This activity involves the development and documentation of unit-test drivers and test-case inputs. Unit testing should be developed and performed by the software developers during the software development life cycle.

The extent of unit testing will be left up to each software development owner. Unit testing will also be done only on the main code development platform. We do not propose to recreate unit tests for all the legacy parts of ITS, but to create unit tests as the need arises (e.g. during formal software development activities). Many unit tests will become regression or verification tests. If so, the code developer will store them in the appropriate CVS directory. Otherwise, the code developer will store these unit/integration tests in a separate CVS directory.

If the software development effort is formal, the software development owner will document his/her unit tests, including the tie to requirements, in the Feature-Specific Test Plan. The results of unit/integration tests are listed in the Test Summary for these efforts. Acceptance criteria are also discussed in Test Plans.

The software development owner will also include, in the CVS directory for unit/integration tests, some documentation that describes the test, its purpose and tie to requirements, and the expected results. We do not specify a template for this associated documentation for unit/integration tests, but leave this up to the discretion of the software development owner.

While unit/integration tests are archived in CVS, they are not typically repeated unless the need arises. They can be used to guide future development activities. Of all the tests discussed, ITS unit testing is done least frequently, only during development.

Any member of the ITS team may contribute a unit/integration test and notify team members by E-mail to SourceForge. A lack of response by the code PI will signify concurrence. Unit/integration tests are stored in a separate directory of the CVS repository.

14. Testing Roles and Responsibilities

Manager:

(a) Maintains all software quality strategic plans in WFS
(b) Maintains V&V plan in WFS
(c) Insures that all formal validation documentation for the AVALTS is stored in RMS, in addition to WFS

PI: Principal Investigator for the ITS code project:

(d) Insures that regression tests for ITS are run at the stated frequency on supported platforms
(e) Insures that general tests for ITS are done at the stated frequencies
(f) Designates (or approves by default) software development owners (designated in the “Assigned to” field of SourceForge) and level of formality attached to a software project
(g) Reviews all tests archived to CVS and approves by default.
(h) Designates application investigator, if a customer requests documentation of V&V testing for their application
(i) Insures that CVERTS and CVALTS are run with every new release and sends E-mail to SourceForge regarding the results

Team member: a code developer (also known as a software development owner) and/or someone who runs ITS tests (with the exception of installation tests which may be run by non-team members):

(j) Before check in of new code, runs commit tests and verifies successful completion
(k) Adds new tests to CVS as needed and sends E-mail to SourceForge, thereby alerting the team to the new test
(l) If the regression or general tests are run, sends E-mail to SourceForge regarding results
(m) Runs installation tests on new machines and sends E-mail to SourceForge regarding results
(n) Runs CVERTS if new or different physics is added to the code and/or if a commit or regression test reveals that the random number sequence has changed. Sends E-mail to SourceForge regarding results
(o) Runs CVALTS if the physics is changed and sends E-mail to SourceForge regarding results
(p) Before modifying a section of code, examines the suite of unit and integration test to see which may be useful for developmental testing
(q) If owner of a formal software development effort, creates a Feature-Specific Test Plan as early in design phase as possible. Continue to update throughout the formal effort and creates a Test Summary Report at the conclusion. Stores these documents in WFS and sends E-mail to SourceForge regarding the creation of the plan and any new revisions

Application investigator: an ITS team member, who, at the request of a customer, performs V&V tests and documents results for a specific application:
(r) Creates a PIRT and expands the V&V plan if needed
(s) Creates an AVERTS if needed, runs tests, assesses, documents, and archives documentation in WFS
(t) Creates an AVALTS if needed, runs tests, assesses, documents, and archives for in WFS and RMS.

15. Future Work

The ITS team endeavors to continuously improve its testing processes. In addition to continuous revision of the ITS Strategic Test Plan, the team has the following goals:

Near term (< one year from revision date):
(1) Add comment lines to all tests stored in CVS
(2) Create scripts to generate list of tests
(3) Establish commit testing and scripts, including automated acceptance criteria
(4) Do regression testing on all supported platforms
(5) Implement coverage testing for XGEN and CEPXS
(6) Create a CVERTS directory with documentation for each test
(7) Add documentation for each test in the CVALTS

Far term (> one year from revision date):
(1) Increase frequency of regression testing by improving automation (investigate using output data in restart file)

(2) Automate coverage testing and increase frequency
(3) Automate memory testing and devise memory testing techniques on our primary code development platform, Crater
(4) Investigate static code analysis tools such as CLEANSCAPE and FLINT

GLOSSARY

ACCEPT	3-D code module of ITS
ACCEPTM	3-D code module of ITS with external electrical and magnetic fields
ACCEPTP	3-D code module of ITS with enhanced ionization/relaxation physics
ACIS	Commercial CAD libraries (Spatial Technology)
ASC	Advanced Simulation and Computing program (formerly ASCI)
ASCI	Accelerated Strategic Computing Initiative (now ASC)
AVALTS	Application specific VALidation Test Suite
AVERTS	Application specific VERification Test Suite
CAD	Computer Aided Design (geometry models)
CEPXS	Multigroup coupled-electron photon cross section generating code module in ITS
CVALTS	Comprehensive VALidation Test Suite
CVERTS	Comprehensive VERification Test Suite
CYLTRAN	2D (cylindrical geometry) code module of ITS
CYLTRANM	2D (cylindrical geometry) code module of ITS with external electrical and magnetic fields
CYLTRANP	2D (cylindrical geometry) code module of ITS with enhanced ionization/relaxation physics
CVS	Concurrent Version System (configuration control)
keV	Kilovolt (1000 electron volts), unit of energy

GeV	Giga electronvolts, unit of energy (10^9 electron volt)
ITS	Integrated TIGER Series code (SNL)
MeV	Mega electron volts, unit of energy (10^6 electron volts)
MITS	Multigroup transport code module of ITS
MPI	Message Passing Interface (parallelization libraries)
OS	Operating System
PI	Principal Investigator
PIRT	Phenomenon Identification Ranking Table
RMS	Records and Document Management System (ASC V&V program)
TIGER	1-D code module of ITS
TIGERP	1-D code module of ITS with enhanced ionization/relaxation physics
SQE	Software Quality Engineering
V&V	Verification and Validation
WFS	Web File Share
XGEN	Continuous-energy cross section generating code module in ITS

APPENDIX A. Template for Feature-Specific Test Plan

Software System Test Plan

Template

Approvals:

Table Of CONTENTS

Introduction

Purpose

The Software System Test Plan (SSTP) document provides details of activities required to prepare for and conduct system test, defines sources of the information used to prepare the plan, defines the software system tests, and defines the test tools and environment needed to conduct the tests.

Scope

Summarize the softaer items and software fratures to be tested. The need for each item and its history may be included. This section should describe the software product at a very high level. It should also name the primary customer(s) for the product. If this is a modification of an existing program or programs it should be so stated in this section.

Definitions, Acronyms and Abbreviations

Often only a referencesuch as"See Project X SRS. [2]".
The Glossary in Appendix 1 contains definitions of technical terms and phrases used in this specification. Acronyms and abbreviations also appear in the Glossary.

References

This section should provide a complete list of all documents referenced in the SSTP. This could include the project plan, quality assurance plan, configuration plan, design specification, users guide, operations guide, installation guide, and any relevant policies and/or standards. The Software Requirements Specification (SRS) is also a major input to the SSTP. The References section should identify each document by title, report number, date, and publishing organization. It should also specify the sources from which the references can be obtained. This information may be contained in Appendix 2 in which case this section should reference Appendix 2. The references should include:

1. **The Institute of Electrical and Electronics Engineers, Inc.,** *IEEE Standard for Software Test Documentation,* IEEE Standard 829-1983, New York, 1983.
2. ***SRS***
3. ***SDD***

Overview

This section should describe the contents of the following sections of the SSTP and explain how the SSTP is organized.

Features To Be Tested

Identify all software features and combinations of software features to be tested. Identify the test design specification associated with each feature and combination of features.

Features Not To Be Tested

Identify all features and significant combinations of features that wil not be tested and the reasons.

Approach

This section should describe the overall approach to testing. For each major group of functions (e.g., interface, security, recovery, performance, etc.) or function combinations, specify the approach that will ensure that these function groups are adequately tested. It should also specify the major activities, techniques and tools that are used to test the designated groups of functions. The approach should be described in sufficient detail to permit identification of the major testing tasks and estimation of the time required to perform each task. This section should specify the minimum degree of comprehensiveness desired. It should identify the metrics that will be used to judge the comprehensiveness of the testing effort. In addition, it should specify any additional completion criteria and significant constraints on testing such as test-item availability, testing-resource availability, and deadlines. The techniques used to trace requirements should also be specified. This section should reference Appendix 3.

Item Pass/Fail Criteria

Specify the criteria to be used to determine whether each test item has passed or failed testing
.

Suspension Criteria and Resumption Requirements

Specify the criteria to be used to suspend all or a portion of the testing activity on the test items associated with theis plan. Specify the testing activities that must be repeated when resumed.

Test Deliverables

Identify the deliverable documents. The following documents should be included:

1. Test Plan (or Plans)
2. Test Design Specifications
3. Test Case Specifications
4. Test Procedure Specifications
5. Test Item Transmittal Reports
6. Test Logs

7. Test Incedent Reports
8. Test Summary Reports

Testing Tasks

Identify the set of tasks necessary to prepare for and perform testing. Identify all intertask dependencies and any special skills required.

Test Environment

This section and its associated subsections should specify both the necessary and desired properties of the test environment. These sections should also identify the source for all testing environment requirements that are not currently available to the test group.

Hardware

This section should describe the hardware requirements for the test environment including the hardware configuration.

Software

This section should describe the software requirements for the test environment. Versions of external or commercially available software should be specified.

Communications

This section should discuss any communications software required for the test environment such as a network software package.

System

This section should discuss the system software required for the test environment.

Security

This section should specify the level of security that must be provided for the test facilities, system software, and proprietary components such as software, data, and hardware.

Tools

This section should identify any special tools that are required. For example, ***automated test tools*** *may be used.*

Documentation

This section should identify any documentation that is required.

Environment Usage

This section should contain estimates of the test environment usage required to complete the software system test phase.

Responsibilities

Identify the groups responsible for managing, designing , preparing, eceduting, witnessing, checking and resolving. In addition identify the groups responsible for providing the test items identified in 4.2.3 and the test environmental needes identified in 4.2.11.

These groups may include developers, testers, operational stall, user representatives, technical support staff, data administration staff, and quality support staff

Staffing and Training Needs

Specify the test staffing needs by skill level. Identify training options for providing necessary skills.

Schedule

Include test milestones identified in the software project schedule, and test item test schedule, as well as all item transmittal events.

Define any additnal test milestones needed. Estimate the time required to do each testing task. Specify the schedule for each testing task and test milestone. For each testing resource (facilities, tools, and staff) specify the period of use)

Risks and contingencies

Identify the high-risk assumptions of the test plan. Specify contingency plans for each (e.g., delayed delivery of test items might required increased nifht shift scheduling to meet the delivery date.

Appendix 1: Glossary

The definitions of technical terms and phrases, acronyms and abbreviations used in this document are specified in the Glossary.

Appendix 2: References

Appendix 2 should contain a list of all pertinent references.

Appendix 3: Test Case List

Typically what the customer really wants – the beef

TEST CASE ID	**Description**
TC 1 Module 1	
TC 1 Module 1	

Test cases can be number to represent the module and test using Outline Numbering with prefix. E.g.,

TC-Mod1	*Module 1*
TC-Mod2	*Module 2*
TC-Mod 2.1	*Module 2.1*
TC-Mod 2.1.1	*Module 2.1.1*
TC-Mod3	*Module 3*

Edit the Numbering Prefix to fit your code

Appendix 4: Matrix of Numbered Requirements vs. Test Identifiers

This appendix should contain a matrix of numbered requirements versus test cases. This is used as a quick reference to indicate which test cases covers which requirements.

The example matrix shows each requirement is covered by one or more of the test cases. The exception is requirement 20.00, which was not implemented.

TEST REQUIREMENT	*abc000*	*abc010*	*abc020*	*xyz010*	*xyz020*
10.00	*x*				
20.00					
30.00		*x*			
40.00			*x*		
50.00				*x*	*x*
60.00					*x*

APPENDIX B. Example of Feature-Specific Test Plan

XGEN
Software System Test Plan

Software Version 6.1
Document Rev. 0.0

Approvals:

1 Introduction

1.1 Purpose

The Software System Test Plan (SSTP) document provides details of activities required to prepare for and conduct system test, defines sources of the information used to prepare the plan, defines the software system tests, and defines the test tools and environment needed to conduct the tests.

1.2 Scope

The software to be tested is XGEN Ver. 6.1. The change from Ver 6.0 to 6.1 is implementation of dynamic allocation. This is to include a full regression test. Where direct comparison to XGEN 5.0 output is not practical, the CONTRAST program which compares numbers within a specific accuracy will be used.

1.3 Definitions, Acronyms and Abbreviations

See Glossary in the ITS Strategic Test Plan

1.4 References

See Refernces in the ITS Strategic Test Plan

1.5 Overview

N/A. This is a small project.

2. Features To Be Tested

This is a full regression test. In addition, additional test files shall be used or created to verify that “large” input files (e.g., 100 materials with 100 elements each, existing customer data files) do not cause program failure.

3. Features Not To Be Tested

This is a full regression test.

4. Approach

This is a full regression test. The program for test purposes shall be compiled with debug and array bound checking turned on.

5. Item Pass/Fail Criteria

Output between 6.0 and Ver. 6.1. should be identical in all cases.

Because there are small rounding errors between the Version 5.0 output and 6.0 output, where difference in output file data occur, CONTRAST, a custom program which compares accuracy to within a specified tolerance (e.g., 0.00001%) may be employed. These rounding errors can be introduced by compiler differences across machines, or between versions of compilers. Where differences of more than 0.001 are encountered, the output shall be run through ITS to ensure negligible impact.

6. Suspension Criteria and Resumption Requirements

Any allocation error or exceeding any array bounds in any file shall suspend testing. Full regression will restart.

7. Test Deliverables

1. Test Plan (Filed in WFS)
2. Test Case Specifications (regression test script updated in XGEN test directory)
3. Test Summary Reports (Filed in WFS)

8. Testing Tasks

Run the ./regtest.pl script

9. Test Environment

See the ITS Version 5.0 User Manual manual for environment requirements.

10. Hardware

Hardware shall be utilized on Scorpio, Crater, and Andromeda.

11. Software/System

Scorpio – AIX O/S, Xlf compiler
Crater – Linux O/S, ifc V7 compiler
Andromeda – Linux O/S, ifc V8 compiler

12. Responsibilities

See Testing Roles and Responsbilities in ITS Strategic Test Plan

13. Staffing and Training Needs

N/A. This is a small project.

14. Schedule

N/A. This is a small project.

15. Risks and contingencies

N/A. This is a small project.

Appendix 1: Test Case List

TEST CASE ID	Description	Pass/ Fail
100el-100mat.inp	Proposed regression test: 100 materials with 100 elements each	
100elements.inp	Proposed regression test: 1 material with 100 elements	
100el-mat.inp	Proposed regression test: 1 material with 100 elements	
100x100.inp	Proposed regression test: 1 material with 100 elements, 99 other materials with 1 element each	
aluminum.inp	Proposed regression test: User input	
starsat.inp	Proposed regression test: User input	
xgen.inp	Existing regression test	
xgenbrems.inp	Existing regression test	
xgenbremsp.inp	Existing regression test	
xgenmat.inp	Existing regression test	
xgenp.inp	Existing regression test	
xgenstep.inp	Existing regression test	

Note:
The xgen*.inp files have output to compare to.
The remaining input files must run only, and need only prove a “well behaved system” – e.g., not crash.
These are to be rolled into the the standard set of regression tests hereafter.

APPENDIX C. Template for Feature-Specific Test Summary Report

Software System Test Summary Report

Template

Approvals:

Table of Contents

Introduction

<A unique identifier should be provided for each separate test summary report.>

Summary

<Summarise the evaluation of the test items. List the items covered by this summary report, and the features evaluated during this period of testing. Provide cross-references to the relevant test documents.>

Variances

Report any variances of the test items from specifications. Also indicate any variances from the approved test plans, or instances where defined test procedures were varied.>

Comprehensive assessment

<Give an evaluation of the comprehensiveness of the testing against the criteria specified in the test plan. Identify any items or features that were not sufficiently tested and explain the results.>

Summary of results

<Summarise the results of testing. Describe and record the number of incident reports raised and resolved, and identify any reports still unresolved.>

Evaluation

<Give an overall evaluation of the status of each test item, focussing on whether the pass/fail criteria have been met. If possible, give an estimate of failure risk.>

Summary of activities

<Summarise the major testing activities and events, and give an overview of the resources applied to testing.>

Appendix 1: Test Case Results

Typically copied straight from the test plan

TEST CASE ID	**Description**	**Pass/Fail**
TC 1 Module 1		
TC 1 Module 1		

Test cases can be number to represent the module and test using Outline Numbering with prefix. E.g.,

TC-Mod4	*Module 1*
TC-Mod5	*Module 2*
TC-Mod 5.1	*Module 2.1*
TC-Mod 5.1.1	*Module 2.1.1*
TC-Mod6	*Module 3*

Edit the Numbering Prefix to fit your code

APPENDIX D. Example of Feature-Specific Test Summary Report

XGEN
Software System Test Summary Report

Software Version 6.1
Document Rev. 0.0

March 5, 2004

Martin J. Crawford

Approvals:

1. Identifier

XGEN 6.1 Test Summary Report

2. Summary

All regression tests passed. The results of the output matched with those of XGEN version 6.0 except where the MATERIAL or ELEMENT number was increased to 3 digits to handle the number 999.

100el-100mat.inp
100elements.inp
100el-mat.inp
100x100.inp
aluminum.inp

3. Variances

See Pass with exception.

4. Comprehensive assessment

Ready for production use.

5. Summary of results

The results of the output matched with those of XGEN version 6.0 except where the MATERIAL or ELEMENT number was increased to 3 digits to handle the number 999.

Upon comparing the output to XGEN Version 5.0, the rounding errors were still present. These were proven insignificant in XGEN version 6.0 by running the XGEN output through ITS.

6. Evaluation

Test Case Results:

Test Case ID	**Description**	**Pass/ Fail**
Regression tests		
xgenbrems.inp	Existing regression test	Pass*
xgenbremsp.inp	Existing regression test	Pass*
xgenmat.inp	Existing regression test	Pass*
xgenp.inp	Existing regression test	Pass*
xgenstep.inp	Existing regression test	Pass*
mat10.inp	Existing regression test	Pass*
mat10p.inp	Existing regression test	Pass*

Test Case ID	Description	Pass/ Fail
mat10egrid.inp	Existing regression test	Pass*
mat10egridp.inp	Existing regression test	Pass*
100el-100mat.inp	New regression test: 100 materials with 100 elements each	Pass
100el-100matp.inp	New regression test: 100 materials with 100 elements each	Pass
100elements.inp	New regression test: material with 100 elements	Pass
100elementsp.inp	New regression test: 1 material with 100 elements	Pass
100el-mat.inp	New regression test: 1 material with 100 elements	Pass
100el-matp.inp	New regression test: 1 material with 100 elements	Pass
100x100.inp	New regression test: 1 material with 100 elements, 99 other materials with 1 element each	Pass
100x100p.inp	New regression test: 1 material with 100 elements, 99 other materials with 1 element each	Pass
aluminum.inp	New regression test: User input	Pass
aluminump.inp	New regression test: User input	Pass
starsat.inp	New regression test: User input	Pass
starsatp.inp	New regression test: User input	Pass

Pass * = Pass with exception.
Rounding differences present in XGEN Version 6.0 still existed in XGEN Version 6.1

7. Summary of activities

7.1 Test Cases

The testing was completed using the existing regression testing.
All tests were run using the regression test script “regtests.pl”. This script was modified to include the following test cases created for regression tests:

1. 100el-100mat.inp
2. 100el-100matp.inp
3. 100elements.inp
4. 100elementsp.inp
5. 100el-mat.inp
6. 100el-matp.inp
7. 100x100.inp
8. 100x100p.inp
9. aluminum.inp
10. aluminump.inp
11. starsat.inp
12. starsatp.inp

These are now part of the regression tests.

7.2 Compilers and Operating Systems Tested

The software was compiled and run on the following computers:

Computer	Operating System	Compiler
1. Crater	Linux	IFC Version 7.1
2. Scorpio	AIX	XLF90
3. Andromeda	Linux	IFC Version 8.0

In addition, a comparison was made to XGEN 5.0. No significant excursions were detected.

7.3 Output Files Updated

To reduce time testing future versions, the output from these tests, which have been proven reliable, have now been placed in the NewOutput directory. Future versions should not have variances. These new files in the xgen/Tests/RegTests/Output directory were created on Crater.

7.4 IEEE Compliance

Approximately 4 hours effort was extended to see if setting the compiler flags to strict IEEE floating point math conformance would reduce variation between compilers. This reduced variation approximately 10%. The compiler flags are still set.

APPENDIX E. Template for AVERTS document

1. Overview
2. Requirements
3. Design of Verification Test Suite
4. Verification Test Matrix
    a. Name
    b. Feature
    c. Description
    d. Relation to PIRT (and other requirements documents)
    e. Status
    f. Acceptance Metric
5. Detailed Description of Tests
6. References
7. Appendices

APPENDIX F. Template for AVALTS document

1. Overview
2. Requirements
3. Design of Validation Test Suite
4. Validation Test Matrix
   g. Name
   h. Feature
   i. Description
   j. Relation to PIRT (and other requirements documents)
   k. Status
   l. Comparison Metric
5. References
6. Appendices

References

[1] John Zepper, Kathy Aragon, Molly Ellis, Kathleen Byle, and Donna Eaton , "Sandia National Laboratories ASCI Applications Software Quality Engineering Practices Version 2.0", SAND2003-0962 (April 2003). https://wfsprod01.sandia.gov/groups/srn-uscitizens/documents/document/wfs084575.pdf

[2] The Institute of Electrical and Electronics Engineers, Inc., IEEE Standard for Software Test Documentation, IEEE Standard 829-1983, New York, 1998.

[3] Martin Pilch, Timothy Trucano, Jamie Moya, Gary Froehlich, Ann Hodges, David Peercy (2000), "Guidelines for Sandia ASCI Verification and Validation Plans – Content and Format: Version 2.0", Sandia National Laboratories, SAND2000-3101. https://wfsprod01.sandia.gov/groups/srn-uscitizens/documents/document/wfs036244.pdf

[4] Leonard Lorence, David Beutler, William Barrett, Clifton Drumm, Wesley Fan, Brian Franke, Ronald Kensek, Thomas Laub, and Jennifer Powell, "CEPTRE and ITS Verification and Validation Plan, Version 1.0", SAND2004-2073 (2004).

[5] ITS Version 5.0: The Integrated TIGER Series of Coupled Electron/Photon Monte Carlo Transport Codes, Brian C. Franke, Ronald P. Kensek, Thomas W. Laub, SAND2003-4173 (not yet published). https://wfsprod01.sandia.gov/groups/srn-uscitizens/documents/document/wfs099592.pdf

[6] American Institute of Aeronautics and Astonautics (1998), "Guide for the Verification and Validation of Computational Fluid Dynamics Simulations," AIAA-G-077-1998.

[7] R.P. Kensek, T.W. Laub, B.C. Franke, and L.J. Lorence, "Application Verification Test Suite (AVERTS) for ITS 5.0 for Adjoint CAD Prediction of Dose", Revision 1.0", Draft Sandia National Laboratories Report (2002) Unpublished, https://wfsprod01.sandia.gov/groups/srn-uscitizens/documents/document/wfs166641.pdf

[8] E. A. Boucheron, P. J. Griffin, S. N. Kempka, M. L. Kiefer, L. J. Lorence, J. S. Peery, S. D. Wix, D. Womble, J. D. Zepper, "ASCI Applications Program Code Verification Testing," Draft Sandia National Laboratories Report (2001) Unpublished.

[9] L. Cordova and R. P. Kensek, "Doe Validation Test Suite (Lockwood Data) for ITS 5.0," Draft Sandia National Laboratories Report (2003) Unpublished, https://wfsprod01.sandia.gov/groups/srn-uscitizens/documents/document/wfs166645.pdf

[10] E. Boucheron, R. Drake, H. C. Edwards, M. Ellis, C. Forsythe, R. Heaphy, A. Hodges, C. Pavlakos, J. Sturtevant, J. Schofiled, C. M. Williamson, "Sandia National Laboratories

Advanced Simulation and Computing (ASC) Software Quality Plan, Part 1: ASC Software Quality Engineering Policy, Version 1.0," Draft Sandia National Laboratories Report (2004)

DISTRIBUTION:

1	MS 0376	J. D. Fowler, 09226
1	MS 0378	W. J. Bohnhoff, 09231
1	MS 0525	C. Bogdan, 01734
1	MS 0651	L. A. Cordova 06323
1	MS 0828	M. Pilch, 09133
1	MS 1146	P. J. Griffin, 06871
1	MS 1146	K. R. DePriest, 06871
1	MS 1146	B. A. Miller, 06871
1	MS 1152	M. L. Kiefer, 01642
1	MS 1152	C. D. Turner, 01642
1	MS 1166	G. J. Scrivner, 15345
1	MS 1166	C. R. Drumm, 15345
1	MS 1166	W. C. Fan, 15345
1	MS 1179	J. R. Lee, 15340
5	MS 1179	L. J. Lorence, 15341
1	MS 1179	B. C. Franke, 15341
5	MS 1179	R. P. Kensek, 15341
1	MS 1179	T. W. Laub, 15341
1	MS 1179	S. D. Pautz, 15341
1	MS 1179	J. L. Powell, 15341
1	MS 1179	M. Crawford, 15341
1	MS 9018	Central Technical Files, 8945-1
2	MS 0899	Technical Library, 9616