SANDIA REPORT
SAND2020-4534
Unclassified Unlimited Release
Printed April 24, 2020

Sandia National Laboratories

Sierra/SolidMechanics Verification Report

SIERRA Solid Mechanics Team
Computational Solid Mechanics and Structural Dynamics Department
Engineering Sciences Center

Prepared by
Sandia National Laboratories
Albuquerque, New Mexico 87185
Livermore, California 94550

Issued by Sandia National Laboratories, operated for the United States Department of Energy by National Technology & Engineering Solutions of Sandia, LLC.

Available to DOE and DOE contractors from

U.S. Department of Energy
Office of Scientific and Technical Information
P.O. Box 62
Oak Ridge, TN 37831

Telephone: (865) 576-8401
Facsimile: (865) 576-5728
E-Mail: reports@osti.gov
Online ordering: http://www.osti.gov/scitech

Available to the public from

U.S. Department of Commerce
National Technical Information Service
5301 Shawnee Road
Alexandria, VA 22312

Telephone: (800) 553-6847
Facsimile: (703) 605-6900
E-Mail: orders@ntis.gov
Online order: https://classic.ntis.gov/help/order-methods

ABSTRACT

This document presents the Sierra/SolidMechanics (Sierra/SM) verification plan. This plan centers on the tests in the Sierra/SM verification test suite, a subset of which are documented in the Sierra/SM Verification Tests Manual. Most of these tests are run nightly with the Sierra/SM code suite, and the results of the tests are checked against analytic solutions. For each of the tests presented in the Verification Tests Manual, the test setup, a description of the analytic solution, and comparison of the Sierra/SM code results to the analytic solution is provided. Mesh convergence is also checked on a nightly basis for several of these tests. This verification plan discusses these various types of tests and what they mean for Sierra/SM verification.

Many other activities also contribute to Sierra/SM quality. These address code and solution quality and range from low-level unit tests, run nightly, up to full-fidelity acceptance tests, used to verify release stability. This acceptance test suite checks that new versions of Sierra/SM continue to yield the same answers for high-resolution analyst problems. Further code quality measures include an extensive suite of intermediate-size regression tests and automated nightly code quality checks. While these additional activities do not fall under a strict definition of verification, they greatly add to the quality, stability, and reliability of Sierra/SM, and are discussed here as well.

1. OBJECTIVES

The audience for this document is rather diverse and as such we seek to both provide strong evidence of the code's "correctness" (tending to have a more mathematical nature), and evidence that has more of a practical bent and can thus have potential utility for the analyst in defining a model (e.g., by seeing how the mesh density affects the accuracy of a contact calculation). Complete verification of Sierra/SM would be a very long-term undertaking, especially since the code is under continuous development. Oberkampf and Roy [4] note that, "V&V are ongoing activities that do not have a clearly defined completion point, unless additional specifications are given in terms of intended uses of the model and adequacy." This alludes to what we call "usage based verification," whereby the verification of the code is driven by the actual use of capabilities by analysts. As such, the current verification manual [10] represents a snapshot of the verification tests that have been more formally documented, but it is under ongoing development to address the evolving applications of the code.

2. SCOPE

To make this document more comprehensive and useful, Section 3 contains introductory material on verification; that section focuses more upon tests that examine convergence (which address two of the verification test types to be explained) than upon the other (five) verification test types that do not address convergence. This emphasis is not because these are the only tests that are important, but rather because they are more complex tests and thus their correct interpretation requires more explanation of issues like "what is the effect of using a linear elastic 'exact solution' when it is not an exact solution to the underlying mathematical model that the code approximates?" Section 3 represents an introductory discussion of these different issues for the the individual test write-ups in the Sierra/SM Verification Tests Manual [10], and the interested reader is directed to that manual for more information on the specific implementation of Sierra/SM verification testing. For further information on verification testing issues, it is recommended to consult more complete treatments of the topics from textbooks such as those that influenced our work [4,5]. A much less comprehensive discussion than the textbooks is presented in a report on the initial efforts to use Sierra tools to perform verification of Sierra/SM using field responses [2], some text of which is incorporated in this introduction.

Further, many activities contribute to the quality and reliability of the Sierra/SM product that are not strictly 'verification' activities. A discussion of these activities is included here in order to provide a comprehensive look at the evidence for the quality of Sierra/SM for production use. Section 4 begins with an overview of the development practices of the Sierra/SM team, including a discussion of unit testing and test driven development. Section 5 describes the 'testing pyramid' and where different Sierra/SM test types fit in the testing architecture. The automated processes that use these tests to ensure quality are discussed in Section 5.1. Additional automated code quality checks are reviewed in Section 6, and the Feature Coverage Tool (FCT), which links analysis results to test suite coverage, is discussed in Section 6.3.

3. BACKGROUND[1]

Verification seeks to prove that a code is "solving the equations right," not "solving the right equations" [4,3,1]. The latter endeavor is the subject of validation. As such, verification seeks to prove that a code will obtain the correct solution of the underlying mathematical model – partial differential equations with corresponding initial and boundary values that define a boundary-initial-value problem (BIVP), or equivalently the weak or variational statements of the BIVP. Of course, the code solutions are based upon approximation theories that 'reduce' the solution of our BIVP to the solution of algebraic equations amenable to computation.

3.1. Convergence

Two categories of tests that will be discussed below incorporate some measure of a code's ability to converge to a solution: convergence to the exact solution, and convergence to a reference solution. That is, we seek to show that successive approximations with finer discretizations (mesh and/or time steps) will be increasingly closer to the exact solution[2], i.e., that we have convergence. The concept of convergence has a rich mathematical foundation, but in this document we merely touch on a few basic definitions to facilitate interpretation of the verification results.

As noted above, we describe convergence as occurring when a sequence of refined numerical solutions becomes increasingly closer to the exact solution. This implies we have a way of measuring the distance between two solutions (a metric, denoted by $d(\bullet,\bullet)$). For our verification of Sierra/SM, we know that our exact solutions live in a function space with additional topological measures for size (a norm, denoted by $\|\bullet\|$) and for angle (an inner product, denoted by $<\bullet,\bullet>$). Our distance measure is then defined in terms of the norm; that is, we measure the distance as the size of the difference between two solutions (i.e., the size of the error):

$$d(u_{approx}, u_{exact}) \equiv \|u_{approx} - u_{exact}\| \tag{1}$$

where u_{approx} is an approximate solution, and u_{exact} is the exact solution. Note that the variable u in this context represents an arbitrary field, not necessarily displacement. These are just generalizations of concepts we are familiar with in three-dimensional Euclidean space.[3] If we have two vectors, one representing the exact solution and one representing the approximate solution, their difference is the error vector, and the magnitude of that vector indicates the size of the error.

The norm used for many of the verification problems is the L_2 norm of the error:

[1] For the reader familiar with verification, the only section that may be of interest is Section 3.2 which describes the classification of verification problems for Sierra/SM.

[2] The weaker category of convergence tests generally deviates from using an exact solution. The issue of using an inexact reference solution will be discussed further in sections below.

[3] A very brief mathematical description that provides additional context for the concepts of function spaces and convergence is presented in [2]. Details were omitted but commonly used terminology was introduced. For more mathematical details on these concepts in the context of boundary value problems see, e.g., [6].

$$\|u_{approx} - u_{exact}\|_2 \equiv \left[\int_\Omega [u_{approx}(x) - u_{exact}(x)]^2 d\Omega\right]^{1/2} \tag{2}$$

Currently, the norm calculations are done with Encore [7] using Gaussian quadrature. For a vector- or tensor-valued quantity, the difference in each component is squared in the integrand. All results given in this document for L_2 norms of symmetric (second order) tensors are based upon "vector" storage of the tensor components, consistent with Voigt notation and the Exodus file storage scheme. As such, the L_2 norm applied to the vector of components reduces the contribution of the off diagonal terms by a factor of 2, since symmetry is exploited to reduce the number of terms in the vector. The Encore input can be modified to yield the true L_2 norm of the tensor, but it complicates the input and the vector form constitutes an equivalent norm.

For some verification tests, we seek to know not only whether the increased resolution of a refined mesh or time step produces better results, but also the rate at which these improvements are realized. For the description below, assume the refinement is in the mesh (i.e., spatial). Ideally the error in the approximation will satisfy a theoretically derived relationship of the form

$$\|e_h\| = \|u_h - u_{exact}\| = ch^p + O(h^{p+1}) \tag{3}$$

for some constant c, where p denotes the theoretical rate of convergence, h is a measure of the element size, u_h denotes the approximate solution for h, and e_h denotes the error vector associated with h. When we apply the above ideas to quantities of interest, like a beam tip displacement, the tensors become scalars and we use an absolute value for the norm. Note that until h becomes sufficiently small, the higher order terms on the right-hand side of Equation (3) can affect the observed rate of convergence when evaluating a sequence of approximations. As h decreases, the right hand side of Equation (3) asymptotically approaches the first term, ch^p. When h is sufficiently small for this first term to dominate, the approximate solutions are described as being in the *asymptotic range*, and thus the theoretical rate of convergence, p, is often referred to as the *asymptotic rate of convergence*. In the V&V literature, the rate obtained from theoretical analysis is also referred to as the *formal order of accuracy* [4]. In the literature for finite element methods, it is also often referred to as the *optimal convergence rate*, or just the *convergence rate*. In a later section, we will describe how an observed convergence rate is measured from a sequence of numerical solutions.

3.2. Types of Verification Tests

Several types of tests are used in verification, and authors group them differently. For the verification of Sierra/SM, we have adopted the following types of tests. The list of test types is nominally presented in an order ranging from simplest to most complex, with the most complex tests often being considered to be the most rigorous (with respect to being able to reveal subtle code errors[4]).

[4]Note that in referring to the error as "subtle," we are not implying that its effect in an analysis would necessarily be insignificant but rather that the source of the error in the coding is not obvious.

1. **Conservation test** - checks the conservation of physical quantities such as mass, momentum, and energy.
2. **Symmetry test** - checks the preservation of symmetry (symmetries).
3. **Sanity check** - determines if a known qualitative behavior or "sanity" is preserved. An example would be inertial reference frame invariance, i.e., objectivity tests.
4. **Code-to-code benchmark test** - compares results of one code to another code that was previously verified.
5. **Discretization error test** - compares a single numerical analysis to an analytical solution. The analytical solution may or may not be exact. A common solid mechanics test of this type is the patch test, where the reference solution is a constant stress/strain result.
6. **Convergence test** - loosely demonstrates the proper order of convergence at best, or at least demonstrates a tendency to converge to a solution with mesh and/or time step refinement.
7. **Error quantification test** - generates empirical evidence that the code can enter the asymptotic regime and that the computational error trends toward zero (with mesh or time step refinement). This category of test requires the exact analytical solution. They are also referred to as order-of-accuracy tests.

A balanced verification suite would contain tests from each category. The first four categories are rather straightforward and will not be discussed further in this document (see [4] or [5] for additional discussion). As a generic term to address tests that examine the rate of convergence, we will call these tests *convergence-rate tests.* Category (6) tests may be convergence-rate tests, and category (7) tests are always convergence-rate tests.

3.2.1. Reference Solutions

Before discussing the next three categories, we will clarify what we mean by the *exact analytical solution* versus simply an *analytical solution*. Generically, we will refer to any solution used to measure the correctness of numerical solutions as the reference solution. To evaluate the correctness of the code rigorously, we need to have a reference solution that is the exact analytical solution to the mathematical model that the code approximates (in our case, usually the weak statement of the underlying BIVP). This consistency is a key point, because if the analytical solution is for a mathematical model to a "nearby" problem (i.e., a surrogate solution) the verification is weaker.

The common case of adopting a surrogate solution, for solid mechanics, is the use of analytical solutions for linear elasticity problems. Obviously this is the class of solid mechanics problems for which many closed form solutions exist. In the case of Sierra/SM, an analytical solution for linear elasticity problems is not an exact solution to the underlying mathematical model, because the code inherently addresses finite deformations, yielding a nonlinear strain-displacement relationship and enforcement of equilibrium in the deformed configuration, whereas the linear elastic response can only be obtained in the limit of infinitesimal displacements. Even when the goal is to examine how well the code performs for a problem governed by linear elasticity, the underlying nonlinearities can complicate the comparison, because there is the potential for these

nonlinearities to affect the perceived error[5] As such, this issue is most evident for a highly accurate solution where the first significant digit of the error corresponds to many digits into the floating-point word representing the response quantity. For convergence tests, the issue is more important (usually for finer the meshes) and will be discussed further below.

Of course, a challenge is that there are far fewer analytical solutions that are consistent with the underlying mathematical model of Sierra/SM, i.e., for problems with finite deformations. If we consider problems that have other sources of nonlinearities (like contact and nonlinear constitutive behavior) and complexity (like integral operators, e.g., to characterize path dependence), the chance of obtaining an exact analytical solution is reduced further. While we have started to apply the manufacturing of solutions for problems with finite deformations, extension to problems with contact and material models defined in an incremental manner needs further development.

3.2.2. Discretization Error Tests

Note that our current verification test suite is dominated by discretization error tests. This is a rather natural state, since these tests can offer a good balance between verifying the code correctness (or quality) and the investment required to develop the test. These tests are also easy for an analyst to relate to since the comparison of two solutions is often limited to a tabular or graphical representation of quantities of interest or their errors (e.g., a patch test stress state or a load-deflection curve), and the problems are physically meaningful. The reference solution in this case, while analytical, may not be the exact solution. Unfortunately these tests address accuracy alone, and it is often difficult to assess if a level of accuracy is acceptable for a given discretization. As such, these tests can reveal major code errors but are less useful at revealing subtle code errors that error quantification tests can reveal.

3.2.3. Convergence Tests

These tests are the weaker of the tests that yield information on convergence. The source of their weakness is typically either that they: (1) adopt an inexact reference solution; or, (2) demonstrate a tendency to converge without reference to another solution. One type of convergence test that the verification test suite adopts is an asymptotic analysis to estimate the rate of convergence. This will be discussed more below, but it can be thought of as adopting an inexact reference solution, since the analysis follows the asymptotic approach of Richardson's extrapolation and obtains an estimate of the exact solution that is one order higher than the numerical analysis. While a test in this category may indicate a tendency to converge, and may even loosely demonstrate convergence at the proper order of convergence, it does not show that the convergence is to the exact solution; we can only say the approximation appears to be converging to a solution. Detail on the characteristics of convergence tests adopting a surrogate reference solution or using asymptotic analysis will be discussed more in separate sections that follow.

[5]Technically, any difference between an inexact reference solution and a numerical solution is not an **error**, but herein this reference is occasionally made, and it should be interpreted as a **difference**.

3.2.4. Error Quantification Tests

This category of tests contains the strongest tests for convergence. They adopt an exact analytical reference solution, consistent with the underlying mathematical model of the code and clearly demonstrate the ability of the code to exhibit the asymptotic rate of convergence (with mesh or time step refinement). While mathematical proofs of convergence generally address measures of errors in fields, we include both field and quantity of interest measures of convergence in this category.

3.3. Observed Convergence Rate

Verification has been referred to as being an inherently empirical process [4] in the sense that we seek, via numerical experiments, to determine if the code works as intended. In the context of order-of-accuracy tests, we seek to show that in the asymptotic range, the observed rate of convergence matches that theoretically predicted. If so, confidence is increased that the code is correctly approximating the underlying mathematical model. In the V&V literature, the rate inferred from multiple numerical analyses with different levels of discretization is referred to as the *observed order of accuracy* [4]. In this document, we will tend to use the more common FEM phrases, *observed convergence rate* or just *convergence rate* (in the latter case, the distinction between the theoretical rate and observed rate is determined by the context).

To estimate the convergence rate from multiple finite element analyses, we assume that Equation (3) is valid, and that the $O(h^{p+1})$ terms are not significant, i.e., that we are obtaining the asymptotic rates. Taking the log of both sides of the asymptotic part of Equation (3) gives

$$\log(\|e_h\|) = \log(c) + p\log(h). \tag{4}$$

Thus on a log-log plot of error versus element size, the slope of the line gives the observed rate of convergence. Often the results for the coarser meshes are not in the asymptotic range, and then the slopes obtained by sequences of results from two meshes changes, giving more accurate convergence rates with finer meshes. For two results from a FEA where the exact solution is known, we can estimate this convergence rate by comparing the errors from these two meshes, and solving for p. For the problems that follow, most refinements involve halving the element size, h, which leads to the following relation for estimating the convergence rate:

$$p = \log(\|e_{h/2}\|/\|e_h\|)/\log(1/2) \tag{5}$$

where $e_{h/2}$ is the corresponding error for uniform (half-size) mesh refinement. When we have multiple levels of refinement, we could apply linear regression to all of the results on a log-log plot obtaining the rate of convergence over a larger range of meshes. However, obtaining rates of convergence from sequences of two results provides an indication of the extent to which the results are in the asymptotic range.

The above discussion of observed rate of convergence is based upon the assumption that we have the exact solution and is thus applicable to error quantification tests. For tests that fall in the

convergence test category, we usually do not have the exact solution, but we may still seek to estimate the rate of convergence. If so, we attempt to obtain the rate of convergence either by using a surrogate solution or by using asymptotic analysis. Both approaches can provide more confidence in the code correctness for some problems, but they also have limitations that will be discussed below.

3.4. Convergence Tests using a Surrogate Solution

As previously noted, a surrogate solution is not an exact solution to the underlying mathematical model that the code approximates, but rather is the solution of a "nearby" problem. As such, the surrogate solution has *mathematical modeling errors* due to the differences in the problem it solves. The surrogate solution may be useful for estimating the rate of convergence when the numerical modeling errors are greater than the mathematical modeling errors, that is, for sufficiently coarse meshes it provides an accurate surrogate for the exact solution. When the converse is true (i.e., the mesh is relatively fine), strictly speaking we are faced with the uncertainty of whether the difference in solutions is due entirely to the inexactness of surrogate solution, or due to a subtle error in the implementation that verification is designed to reveal. Unfortunately for coarser meshes where the surrogate solution is sufficiently close to the exact solution, the numerical solution may not be in the asymptotic range. As such, for a surrogate solution to be useful in estimating the rate of convergence, we need a range where the surrogate is sufficiently accurate and the numerical results are in the asymptotic range. For some sequences of solutions, this range will not even exist.

For the case of an exact reference solution, one expects the code to yield the asymptotic rate with increasing accuracy upon mesh refinement. For the case of a surrogate reference solution, if the FEM solution is approaching the exact solution, one would expect the difference to approach a constant value that quantifies the mathematical modeling error. That is, in the limit, the difference is an indicator of the error in the surrogate solution, not the FEM solution. Generally however, we do not know that the FEM solution is approaching the exact solution, so the constant difference that the FEM solution approaches could be a combination of code error and mathematical modeling error.

Another characteristic of using a surrogate reference solution is that the convergence to the constant difference is not necessarily monotonic. This is true for field quantities and quantities of interest, and can be illustrated in terms of a solution in a function space or on the real line, respectively. For simplicity, consider a description for a quantity of interest, the values of which are on the real line. Assume for example that our quantity of interest is a force response, and that the exact solution for the response is 1000. Also assume that the surrogate solution gives a force response of 1001. If the sequence of results from the FEM starts at 1400 (a 40% error) and monotonically decreases toward the exact solution, apparent non-monotonic convergence can be obtained relative to the surrogate solution. Note that at a load level of 1400, the surrogate solution provides a reasonable measure of the error (~39.8%). For example, consider a sequence of FEM force predictions having linear convergence given by {..., 1006, 1003, 1001.5, 1000.75, 1000.375,...} and that are converging to the exact solution. The actual sequence of errors are simply {..., 6, 3, 1.5, 0.75, 0.375,...} which exhibits monotonic convergence. However, the

perceived errors (actually differences) obtained relative to the surrogate solution are {..., 5, 2, 0.5, 0.25, 0.625,...} (not monotonic). These differences approach a difference value of 1 – the mathematical modeling error. Note that if the surrogate solution gave a value of 999, the convergence would be monotonic, so the relative values of the exact and surrogate solutions can determine the nature of the convergence.

3.5. Convergence Tests using Asymptotic Analysis

Asymptotic analysis is used in convergence tests when we are seeking an estimate of the rate of convergence, sometimes when we have a surrogate solution and other times when we do not. We tend to use them when we have a surrogate solution, for cases where we do not have an obvious range for estimating the rate of convergence. As such, it can strengthen the convergence argument, though it is still weaker than having an error quantification test. When we do not have any surrogate solution, it provides an estimate of the rate of convergence when otherwise we could only observe a tendency of the results to converge to some value (hopefully the exact solution). In the latter case, it is being applied identically as one does for solution verification.

The asymptotic analysis can be considered as consisting of two steps. First, the results from sequences of analyses based upon three mesh refinements, where each refinement halves the characteristic length of the element (e.g., each hex is approximately subdivided into eight hex elements) are used to estimate the rate of convergence. Second, the convergence rate obtained from the finest sequence of meshes may be assumed to be accurate, and then is used with Richardson extrapolation to obtain a higher-order estimate of the exact solution. The Richardson's extrapolated estimate is then often adopted as the reference solution to analyze the results, sometimes with log-log plots of a difference measure versus an element size measure as would be done with an analytical reference solution. For problems like contact, where we cannot define the expected rate of convergence exactly, we have chosen to use the rate obtained in the first step as the rate applied in the second step and solve three equations for three unknowns (as we will outline below), but the rate is not an integer. The alternative is to use the rate obtained in the first step to provide an estimate of the rate and round it to the next integer. If the formal rate of convergence is known for the numerical method, that rate should be used in the Richardson extrapolation.

Consider an outline of the asymptotic analysis as two steps. For more detail, see references [4] or [5]. First consider a sequence of three scalar results for a quantity of interest or norm of a field that will be denoted as $\{S_i, S_{i+1}, S_{i+2}\}$, where S_i denotes the scalar value for the coarsest mesh, and S_{i+1} and S_{i+2} denote the scalar values for one and two uniform mesh refinements, respectively. These results correspond to meshes such that $h_{i+1} = h_i/r$ where $r = 2$.[6] As with Equation (3), we assume that error can be expressed in a power series in h, as

$$S_i = S_{exact} + ch_i^p + O(h_i^{p+1}) \tag{6}$$

for a p^{th}-order method. Combining the higher order error terms with the exact solution gives

[6]Uniform mesh refinement as specified here is not a requirement of the methodology, but it is the approach that has been adopted for all problems in the manual to date.

$$S_{RE} = S_{exact} + O(h_i^{p+1}) \tag{7}$$

where S_{RE} denotes an approximation of the exact solution that, if the h_i^{p+1} term exists, is one order higher in accuracy than the original p^{th}-order method would give. The notation of the RE subscript denotes the Richardson Extrapolated value for S, which will be solved for in the second step of the analysis. Combining Equations 6 and 7 gives

$$S_i = S_{RE} + ch_i^p. \tag{8}$$

If we write this relationship for meshes i, $i+1$, and $i+2$, we have three equations and three unknowns. Eliminating S_{RE} and c from the three equations and solving for p gives

$$p = \frac{\ln\left(\dfrac{S_1 - S_2}{S_2 - S_3}\right)}{\ln(r)}. \tag{9}$$

Note that the above analysis can be used for successive sequences of three meshes, as is done for many verification problems, and consistency in the results for p then gives an indication if the results are in the asymptotic range. If the results are not consistent, the asymptotic analysis is not conclusive, though the result from the finest set of meshes may suggest a tendency in the rate of convergence.

The second step of the analysis corresponds to the generalized Richardson extrapolation, where the extrapolated value is given by

$$S_{RE} = S_3 + \frac{S_s - S_2}{r^p - 1}. \tag{10}$$

As previously noted, this value can now be used as a reference solution. We have done that in many of the tests to give a graphical representation of the results from the asymptotic analysis. These types of graphical results must be interpreted carefully, because in most cases S_{RE} is considered to be more useful as an indicator of the uncertainly in the solution than as a proper surrogate solution. Use of this solution, when p is obtained directly from Equation (10), also tends to instill false confidence in the results. This is due to the fact that by definition the convergence plot will show the results for the finest three meshes as lying perfectly on a straight line. When this occurs with an exact solution, we infer that we are in the asymptotic range; when it occurs in this case it is simply a result of solving the corresponding three equations to make it occur. In this case, we need four values to lie along a line, which corresponds to getting consistent p estimates from two overlapping sequences (as previously mentioned). Another caveat in plotting the results for these tests is that when multiple tests are plotted on the same plot we have to keep in mind that they each have their own reference solution, so comparisons of relative accuracies can be questionable, though potentially meaningful if we know the extrapolated results are based upon data from the asymptotic range.

4. CODE DEVELOPMENT PRACTICES

The first step to a well verified code is code development practices that ensure all new code features are properly tested. The Sierra/SM team follows the laws of test driven development (TDD) coding practice as outlined in Clean Code [9]. The three laws of TDD are

1. You may not write production code until a failing unit test[7] is written.
2. You may not write more of a unit test than is sufficient to fail.
3. You may not write more production code than is sufficient to fix the currently failing test.

Following these laws ensures that all new capability is covered by tests, and that all capability modified through user stories or corrected by user support is also covered by tests. However, these practices fail to ensure that *all* legacy capability is adequately covered, or that all permutations of capability are well verified. The Sierra/SM process for covering permutations of capability is outlined in 6.3. In addition to the enumerated TDD practices the Sierra/SM development team also uses code reviews, pair programming, and external beta testing as additional safeguards to prevent coding errors.

5. OVERVIEW OF TESTING PYRAMID

In order to efficiently maintain code quality a properly organized suite of tests must be used, a large number of small tests of individual capabilities building up to smaller numbers of large and complex tests. There are many types of tests for Sierra/SM: Unit, Fast (Continuous), Performance, Verification, Regression, and Acceptance. For tests to have value they must be run regularly and in an automated fashion. With the exception of a few large acceptance tests the entire Sierra/SM test suite is run nightly.

Unit Tests: a test of an individual source code function. Unit tests are generally run through the Google Test framework. A unit test can be used to verify a given function has the correct behavior for every possible input. Unit tests are typically very fast, since they do not involve user input files or mesh databases.

Fast Tests: a test that must run in under ten seconds. Fast tests are run every hour on the 'master' development branch of the Sierra code base. This high run frequency allows quickly pinpointing any issues introduced into the code base. The fast test suite is designed to give a broad coverage of all core Sierra/SM features.

Verification Tests: a test that compares test outputs to an analytic result or confirms the test has some expected property (such as a convergence rate.) Verification tests are one of the most valuable test types, and the verification test suite is continually expanded over time.

Regression Tests: a test that confirms the code produces an expected output, but without rigorous mathematical demonstration that the output is indeed correct. Generally a test case is produced and then engineering judgment used to confirm the test case is behaving as

[7] Unit testing is discussed in Section 5.

expected. The test then confirms this approved behavior is maintained. Regression tests are necessary, but the code team works to balance regression testing with more valuable unit and verification categories.

Performance Tests: a test used to confirm Sierra/SM maintains acceptable run time and memory use bounds. These tests are expensive.

Acceptance Tests: a test of a full analysis use case provided by an analyst. Acceptance tests are the largest and most complex tests in the system. An acceptance test ensures that the work flow for an entire realistic, complex analysis chain maintains functionality.

5.1. Nightly Testing Process

Every night the entire code base is built on multiple platforms using a variety of compiler configurations. Some subset of the nightly tests are run on each supported platform, while unit, fast, verification, and regression tests are run on every production platform (i.e., platforms on which production use of Sierra/SM is supported). Production platforms include (as of mid-2018) a development platform, compiled with both debug/release and GNU/Intel compilers, along with TLCC2, CTS-1, Trinity (Haswell and Knights Landing chips), and Darwin (macOS). The performance tests are additionally run nightly in the Intel-release configuration on the primary HPC production platform dedicated to Nuclear Deterrence (CTS-1 as of mid-2018). A subset of the tests are also run on experimental platforms, such as Ride (a GPU-accelerated platform), and upcoming compiler releases. The experimental test results prepare the code for new platforms to move into production and may identify software quality issues that don't cause problems in the current production platforms.

5.2. Nightly Verification Testing

A verification test suite representing the types of tests outlined in Section 3 is run nightly to verify the correctness of capabilities in Sierra/SM. Many of these tests are documented in the Sierra/SM Verification Tests Manual [10], with each test being a member of a distinct verification "test group." In some cases, the test group contains a single test, and in other cases it contains a group of related tests (e.g., a patch test applied all of the hex elements). The verification test groups verify some aspect of that suite of capabilities. The test files for these problems may be found in the Sierra regression test repository, in the sub-directory

```
adagio_rtest/VerificationTestManual
```

On many Sierra-supported platforms, the latest versions of these tests can be accessed at

```
/sierra/Dev/nightly/Sierra.tests.master/adagio_rtest/VerificationTestManual
```

5.3. Unit Testing

A key component of modern software development is unit testing. Unit tests are in-code tests that define and preserve behavior of individual routines. In Sierra/SM they are used both to define desired or expected behavior ahead of the actual implementation and to preserve behavior during changes that do not affect functionality (known as "refactoring"). One analogy for unit testing is with double-entry bookkeeping [8]. By keeping two records of each transaction, and balancing accounts using these records, errors can be caught quickly. Unit tests serve as a second record (in addition to the routine itself) of how the developer intended the code to behave, and running the tests checks that the tests and routines are still in agreement. The nightly test suite of Sierra/SM runs all of the unit tests.

Another tenet of modern software development is that the set of tests that define the behavior of a product should map to the pyramid structure discussed at the beginning of this section. Simple unit tests should be the most numerous type of test and represent the base. More complex in-code integration tests (which combine multiple routines) and simple user facing tests (whether verification tests or tests that simply check for 'regression', *i.e.* behavior changes) should represent the smaller middle, and exceptionally complex 'acceptance' tests (see Section 5.4) that mimic actual analyst use cases should be few in number and represent the peak. This paradigm has not always been followed throughout the legacy of development in Sierra/SM, and the middle of the pyramid (integration and regression tests) are over-represented. Unit tests are not as ubiquitous as ideal. The large backlog of work to change this state is being addressed by the development team, who are continually improving the test suite and re-architecting the tests to better conform with modern development practices.

5.4. Acceptance Testing

Acceptance testing is an important part of delivering a high quality, reliable analysis product. The basic definition of an acceptance test is one that defines how a feature, an algorithm, or a combination should behave from an analyst's perspective. Here we use the term to refer to tests that are as close as possible to the actual problems that analysts are successfully solving with our product. This means that the problems will be complex, combine numerous capabilities, and likely require large compute resources to complete. For these reasons, the number of these tests should be small relative to unit and regression tests, as discussed at the beginning of this section.

A set of acceptance tests taken directly from the largest projects which the Sierra/SM Team supports keeps the features identical or as close as possible to the original intent. These tests are not run nightly as other test types are, since some require more than a day to run, but they are run regularly to ensure that each major Sierra/SM release successfully completes this suite of problems. Successful completion entails clean execution of the analysis with results that agree with those previously accepted by analysts using prior code versions. Any changes in behavior are analyzed by the code team in conjunction with relevant analysts to verify that the change is expected. These tests provide confidence that the most important user analyses, with all the attendant complications, will execute with comparable or better accuracy and performance in

each new version of Sierra/SM.

6. OTHER SQA TOOLS

In addition to the nightly testing process, other software quality tools are run nightly to check for possible code errors or gaps in testing coverage. These tools include the memory checker Valgrind, the Line Coverage Tool (LCOV), and the Feature Coverage Tool (FCT).

6.1. Valgrind

Valgrind is a tool used to check for memory leaks and memory errors. A memory leak is when memory is allocated, but never freed while the program is still running. The existence of memory leaks within loops can lead to a simulation taking an increasing amount of memory as simulation time increases, eventually leading to code failure. A memory error represents the executable accessing memory that has not been allocated, or is otherwise out of bounds. A memory error generally results in unpredictable behavior, and can lead to fatal segmentation faults. Valgrind is run nightly on most of the "nightly" and "fast" tests. All memory leaks and errors are eliminated prior to each release version of Sierra/SM.

6.2. Line Coverage Tool (LCOV)

For Sierra/SM, LCOV, measures the code source line coverage of unit, fast, and nightly testing. The LCOV tool reports how many times each line of code is executed over the respective test suite. For each file, folder, and executable in Sierra/SM, LCOV reports the percentage of lines in the code that are covered by at least one test. The development team must ensure that all new features are well covered. The Sierra/SM development team strives to improve test code coverage over time; however, 100.0% coverage is a challenge to obtain. Examples of uncovered code may be non-released research capability or deprecated legacy capability. Additionally, many error checks and messages do not currently have corresponding tests.

6.3. Feature Coverage Tool (FCT)

The FCT is used to map an analysis input file to the Sierra/SM test suite in order to guide verification and results quality assessment. The FCT creates three documents from an input file: the annotated input file, the two way coverage graph, and the list of best matching tests. The FCT can be used by analysts to assess the Sierra/SM verification rigor for a specific analysis. Additionally the Sierra/SM development team can use output of the FCT to prioritize needs for verification test suite improvement. The FCT should be used in conjunction with subject matter expertise to positively verify that the identified tests do in fact apply to the physics regimes being modeled. Many code features apply to multiple regimes (e.g., plasticity material models, which

have elastic and plastic regimes), and not every verification test that uses a feature will exercise all regimes.

The annotated input file shows the features (corresponding to input deck lines) that are used in verification tests (in green), regression tested (yellow) or untested (red). Developers and analysts can use this tool to determine if a particular analysis uses untested features and take action to mitigate or explain them. One mitigation strategy is to create a new verification test for the feature.

The second document produced by FCT is the two way coverage chart. The two way coverage chart indicates for any two features if a verification or regression test exists that uses both of those features simultaneously. It can be impractical to add a verification test every possible feature combination; however, the two way coverage report can be used to determine if certain key feature combinations are tested together, such as damping in a transient analysis or strain output on shell elements. Lack of a two way coverage test may indicate additional verification testing is needed, though engineering judgment must be applied to identify the most critical feature combinations.

The third FCT output is a of list the top five (5) most similar verification tests to the input file, based on combination of features used. A closely matching, rigorous verification test can give high confidence that the use case of the analysis and its feature combinations are well verified.

7. REFERENCES

1. F. G. Blottner, "Navier-Stokes results for the hypersonic flow over a spherical nosetip", *Journal of Spacecraft and Rockets*, 27(2), 113-122, 1990.
2. J. V. Cox and K. D. Mish, "Sierra Solid Mechanics Example Verification Problems to Highlight the use of Sierra Verification Tools", *SAND Report*, SAND2013-2390, Sandia National Laboratories, 2013.
3. B. W. Boehm, **Software Engineering Economics**, Prentice-Hall, 1981.
4. W. L. Oberkampf and C. J. Roy, **Verification and Validation in Scientific Computing**, New York, NY: Cambridge University Press, 2010.
5. P. J. Roache, **Verification and Validation in Computational Science and Engineering**, Albuquerque, NM: Hermosa Publishers, 1998.
6. I. Stakgold, **Green's functions and Boundary Value Problems**, New York, NY: John Wiley & Sons, 1979.
7. K. D. Copps and B. R. Carnes, "Encore User Guide", *SAND Report*, DRAFT, Sandia National Laboratories, 2009.
8. R. Martin, "The Sensitivity Problem", Web blog post, *www.butunclebob.com*, 5 October, 2005. Web. 8 April, 2018, *http://www.butunclebob.com/ArticleS.UncleBob.TheSensitivityProblem*.
9. R. Martin, **Clean Code: A Handbook of Agile Software Craftsmanship**, Prentice Hall, 2008.
10. Sierra Solid Mechanics Team, "Sierra/SolidMechanics 4.46 Verification Tests Manual", *SAND Report*, SAND2017-9776, Sandia National Laboratories, 2017.

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology & Engineering Solutions of Sandia LLC, a wholly owned subsidiary of Honeywell International Inc., for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-NA0003525.