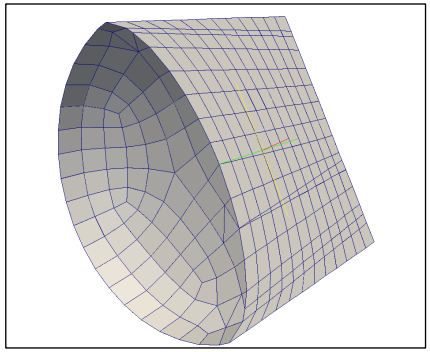
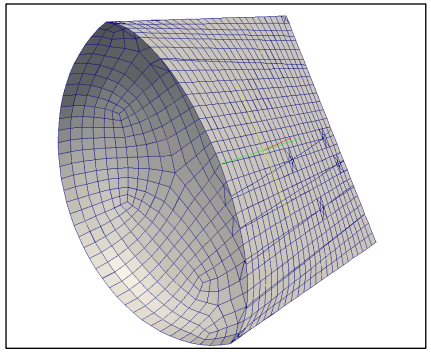
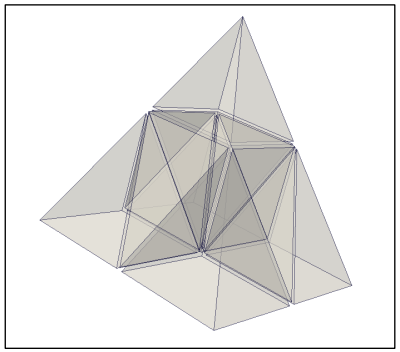
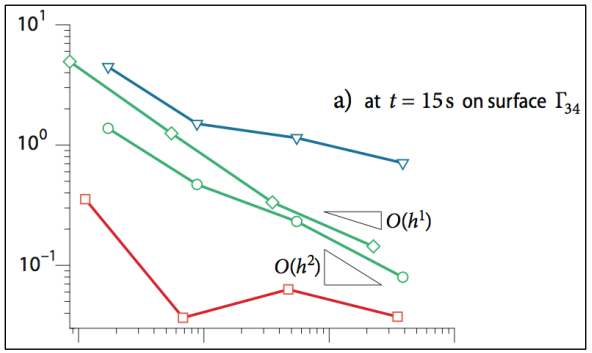
SAND2014-3839P

Exceptional service in the national interest

Sandia National Laboratories

II. Code Verification

Kevin Copps and Brian Carnes, Org. 1544

ESP700

April 2014

U.S. DEPARTMENT OF ENERGY | NNSA National Nuclear Security Administration

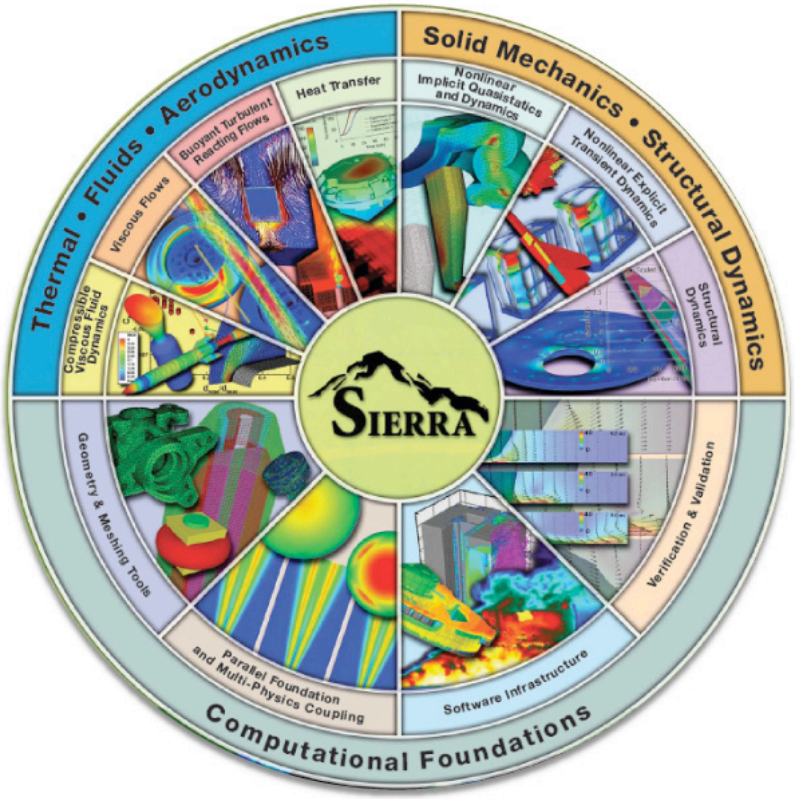
Sandia National Laboratories is a multi-program laboratory managed and operated by Sandia Corporation, a wholly owned subsidiary of Lockheed Martin Corporation, for the U.S. Department of Energy's National Nuclear Security Administration under contract DE-AC04-94AL85000. SAND NO. 2011-XXXXP

Scope of Sierra Software Project

What is Sierra?

Sandia, Org. 1500, modeling and simulation of engineering systems

- Loosely coupled multiphysics, solid mechanics, structural dynamics, thermal, aero, fire.
- **99 physics and supporting executables** including supporting tools: aprepro, epu
- **12,827 input options** not including sub-options



- It's own secondary department just to support the developers.
- **50** nightly build, compile and **test platforms** on the dashboard
- **16,485** automated tests (including some serial/parallel redundancy)
- Distribution to multiple HPC platforms across DOE labs and DOD sites.

What do we need to convince ourselves that we trust our software simulations?

Remember the saying
garbage-in, garbage out?
With todays supercomputers, it’s easier than ever to make super-garbage. And it looks lovely, if you don’t dig too deep.

Computer programs are the most intricate, delicately balanced and finely interwoven of all the products of human industry to date. They are machines with far more moving parts than any engine: the parts don't wear out, but they interact and rub up against one another in ways the programmers themselves cannot predict.
- James Gleick (2002)

Weinberg's Second Law: If builders built buildings the way programmers wrote programs, then the first woodpecker that came along would destroy civilization.
- Gerald Weinberg cited in: Murali Chemuturi (2010)

One in a million is next Tuesday.
- Gordon Letwin

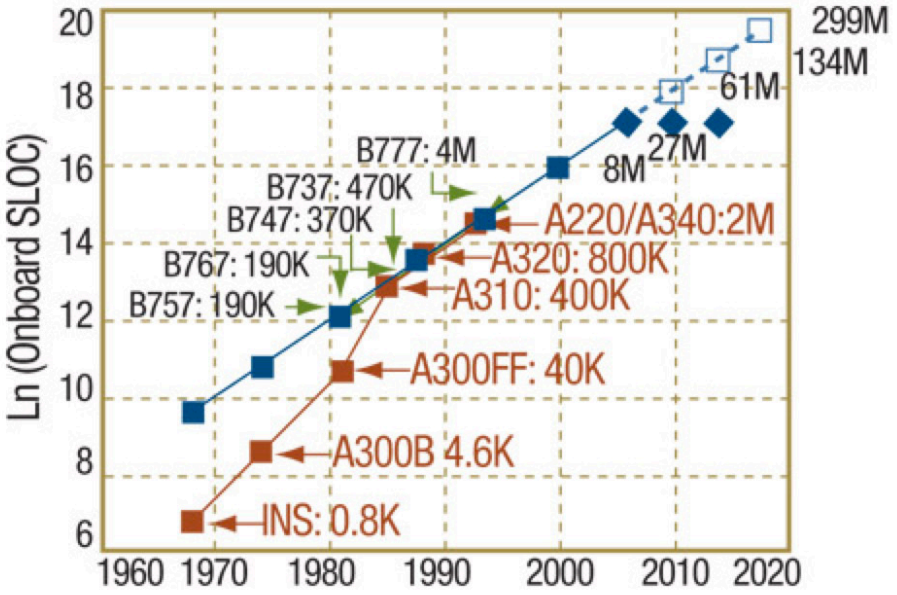
Processes replaced by software

The replacement of processes previously performed by hardware and/or human action is unprecedented. The science fiction future is not filled with robots, but software.

Estimated Onboard SLOC Growth

Slope: 0.1778 Intercept: -338.5
Curve Implies SLOC doubles about every 4 years

NEXT



This line fit is pegged at 2705 M SLOC because the SLOC sizes for 2010-2020 are not affordable. The COCOMO II estimated costs to develop that much software is in excess of $10B

- Straight Line curve fit
- Boeing aircraft
- Airbus aircraft
- Not affordable extrapolation

Acronyms:
SLOC: software lines of code
COCOMO II: COnstructive COst MOdel II

Airbus data source: J.P. Potocki De Montalk, "Computer Software in Civil Aircraft," Sixth Annual Conference on Software Assurance (Compass '91), Gaithersburg, MD, June 24-27, 1991 Boeing data source: J.J. Chilenski, 2009

According to Boeing and Airbus, the size and complexity of avionics software is projected to grow geometrically—with the current rate of software growth as 400 percent every two years.

Image 1 of 2

CLOSE ✕

Group Exercise

- Think of a software tool that you use regularly
  1. How much credibility do you put into the results?
  2. What is the evidence or why
  3. How accurate are the results?

Ideas:

- Microsoft Excel
- Sierra Mechanics
- Google Maps or your favorite GPS
- Search Engine
- Speech Recognition

Code Verification

Considerations:

- Modeling and simulation software running on today's supercomputers is an extremely complex system, which in turn is designed to model another complex system, in our case: nuclear weapons.
- Our main difficulty in the design and maintenance of modeling and simulation software is: ***managing complexity***.
- Basic SQA (Software quality assurance) practices are critical, and provide the foundation for ***code verification***.

What is Code Verification

- An activity to ensure our modeling and simulation software is **adequate for its intended purpose**.
- Ensuring that the approximation to the model equations is constructed correctly.
- Ensuring defects do not exist in the software, in context.
- Verification is part of a larger SQA (software quality assurance) effort.
- There are many activities and tools for SQA, however, testing the *order of convergence* of numerical approximations is the essential tool for verification.

Order of convergence tests, on production compute servers, are a specialized type of *integration*, or *system tests*.

Defect Detection Rates

Technique	Modal Rate
Informal design reviews	35%
Formal design inspections	**55%**
Informal code reviews	25%
Formal code inspections	**60%**
Modeling or prototyping	**65%**
Personal desk-checking of code	40%
Unit test	30%
New function (component) test	30%
Integration test	35%
Regression test	25%
System test	40%
Low-volume beta test (<10 sites)	35%
High-volume beta test (<1000 sites)	**75%**

This table refers only to *detection*, not actually fixing or resolving the defects.

Testing is not the most effective technique, but it is critical for mod-sim software. Testing can be tracked and automated in some sense.

From Programming Productivity (Jones 1986), "Software defect-Removal Efficiency (Jones 1996) and "What We Have Learned About Fighting Defects" (Shull et al. 2002)

Defect Detection Rates

Conclusions from the table:

- Modal rates don't rise above 75%
- Average effectiveness of techniques is around 40%
- The most commonly used, unit testing and integration testing, are only 30%-35%.
- The typical organization uses a test-heavy defect removal approach and often achieve only 60% defect removal efficiency.
- Order of convergence testing on various installed platforms is a *system test*.
- Defect detection methods work better in combination.

Software Quality Key Points

- SQA requires a reallocation of resources, a major component of development time and money must go to verification.
- Prefer early cheap defect prevention rather than expensive fixes later.
- Not all facets of the software and all quality assurance goals are achievable within a fixed budget. Therefore, we must identify the goals, limits and scope of the activities explicitly. Communicate these with developers, analysts and their customers.
- No single technique is effective by itself. Testing by itself is not optimally effective at removing defects.

What is a Verification Test?

Verification Test A high quality test of the code, evaluating the accuracy and precision of the approximation, preferably on a problem with a known analytic exact solution, or manufactured solution.

Aiming for higher Levels of Rigor as opposed to:

- Code-to-code comparisons
- Benchmarks
- Comparing to the solution of a nearby simpler model (beam/shell theory)

Verification Test Procedure

1. Construct an analytic solution to a problem
   a) Find exact solution on a simple domain, or
   b) manufacture a solution and reverse engineer boundary conditions, source terms
2. Compute errors in suitable metrics, the difference between they analytic and approximate solution produced by the code
3. Examine convergence behavior as mesh size becomes uniformly smaller

Aside: Metrics of Numerical Error

Error in your output **Quantity of Interest** or **QoI**

- Directly relevant to your problem
- Examples:
  - Coefficient of drag in fluid problems
  - Average temperature of a part at a certain time in transient heat conduction
  - Stress intensity factors in linear elastic fracture mechanics
  - Mode shapes, or eigenvalues, in structural dynamics
- Often a result of extra post processing of the fields in the code following the algebraic solve

Aside: Metrics of Numerical Error

Error measured in a **Global Norm**

- Includes field information across the whole simulation
- Scalar quantity
- Much can be proven about their convergence behavior
- Remain finite even when the model allows for infinite field values for example: linear elasticity allows infinite stresses
- Often behave monotonically as mesh size shrinks
- Can be estimated with simple and computable error estimates
- Can often be computed as a post-process outside the code as they only require field values

L2 norm of a field function u(x)

$$\|u\|_{L^2}^2 = \int_\Omega |u(x)|^2 \, dx$$

H1 norm of a field function u(x)

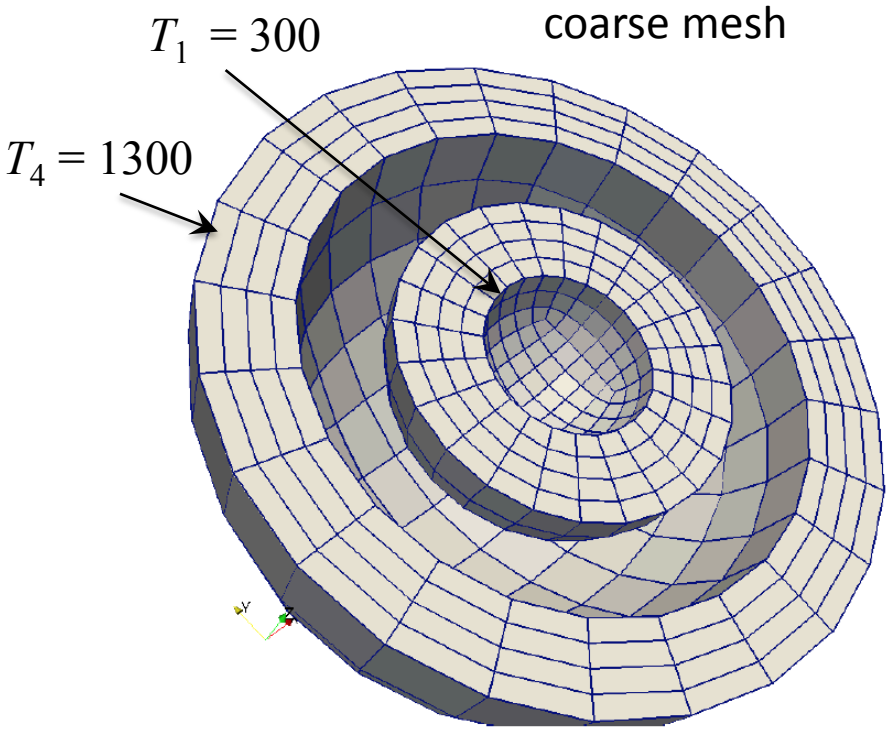
$$\|u\|_{H^1}^2 = \|u'\|_{L^2}^2 + \|u\|_{L^2}^2$$

Often this term is left out, which is technically called the H1-seminorm

Example Verification Test

Goal: test whether code achieves expected order of convergence

- Heat conduction and enclosure radiation
- Formulated in such a way that an analytic solution can be represented in Matlab
- Exact solution provided to the code



initial condition $T_{t=0} = 300$

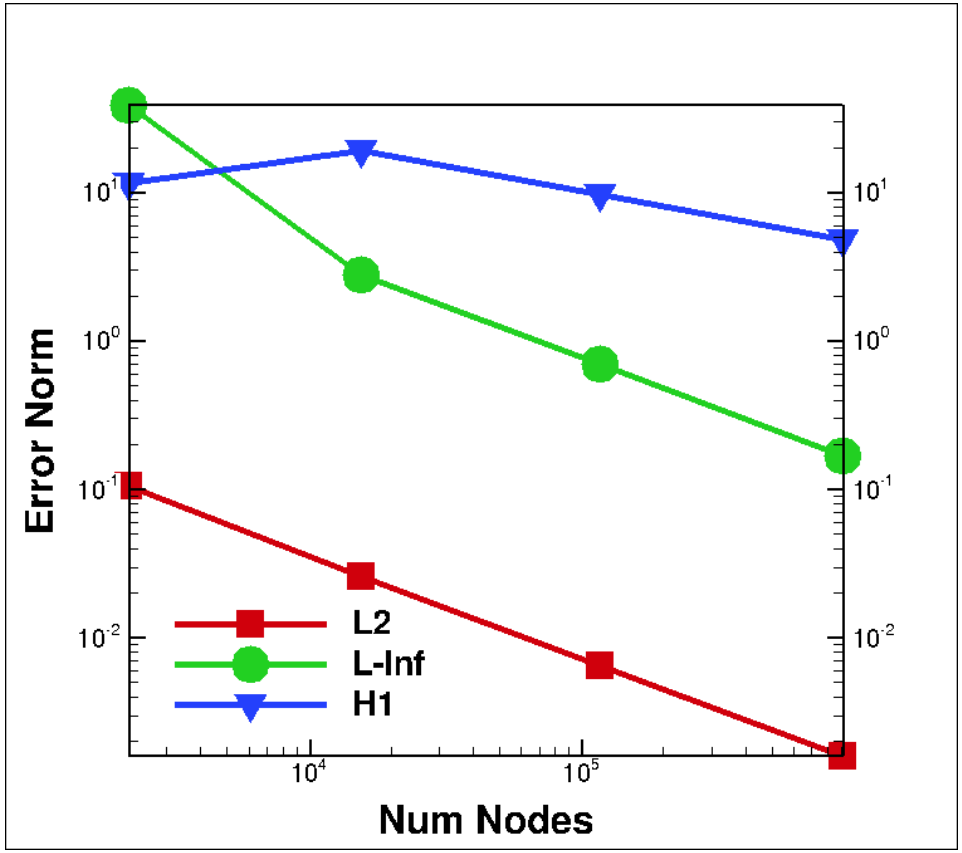
Thermal conductivity (block_1)	κ_1	2.0
Thermal conductivity (block_2)	κ_2	0.35
Density	ρ	1.0
Specific heat	C_p	1.0
emissivity (surface_2)	ε_2	0.50
emissivity (surface_3)	ε_3	0.55
Stefan-Boltzmann constant	σ	5.6704e-8

radius of surface_1	r_1	0.01
radius of surface_2	r_2	0.02
radius of surface_3	r_3	0.03
radius of surface_4	r_4	0.04

Example Verification Test

Passing the test

- The problem is run by the code on a sequence of four meshes using uniform refinement.
- Each subsequence mesh has 8X the elements as the previous.
- From theory, the expected rates of convergence in the L^2, L^∞, and H^1 norms are 2, 2, and 1, respectively.



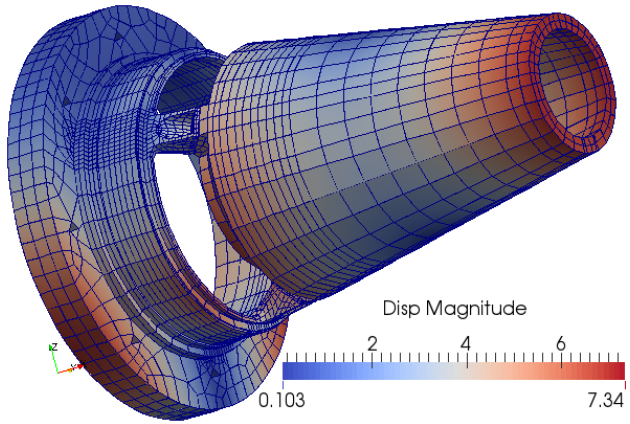
Feature Coverage Assessment

- Can we provide confidence that the code is tested well in the context of your problem domain?
  That is, are the *features* of the code you are actually using *verified* by verification tests?
- A feature is any single class of input to the simulation code:
  - Activating an emissivity boundary condition
  - Traction boundary condition
  - Choice of algebraic solver
  - Contact tolerances, etc.
- In the past any such evidence was only word of mouth, subjective developer claims, or collected by hand

The Sierra feature coverage tool is intended to provide an automated, objective, and independent assessment of how well the features you using are verified

An FCT Analysis

Example: the structural dynamics cone problem.




```
SOLUTION
//    eigen nmodes=20
// uncomment all the lines below this to the END and comment the
line above to run a nonlinear blast analysis
    NLtransient
    time_step 2.0e-5
    nsteps 8192
    nskip 1
    rho 0.9
    solver = gdsw
END

[...]

GDSW
   max_iter=1000
   solver_tol 1e-10
   krylov_method=1  //0
   overlap = 2
   orthog = 1000
//   orthog_option = 2
END
```

The main input file.
Specifies the problem domain, a grid, boundary conditions, material properties, algebraic solver, etc.

An FCT Analysis

1. **Execute the tool:**

```
$ module load sierra
$ fct salinas mycone.inp
```

Executing the fct command line tool.

- Compares the parsed input file to the CoverageCertificate for the version of the code that you are running. A few seconds to run.

2. **Examine Output:**
   1-way coverage report (*.html) - open in your browser
   2-way coverage report (*.xls) - open in Excel

FCT 1-way coverage

```
verified
  * one-way:93%
  * two-way:66%
tested
  * one-way:100%
untested
ignored
```

One Way
Percent of features (non-commented lines) in your input file that are covered by at least one verification test.

Two Way
Percent of pairs of every two features in the input file that were present in one or more verification tests.

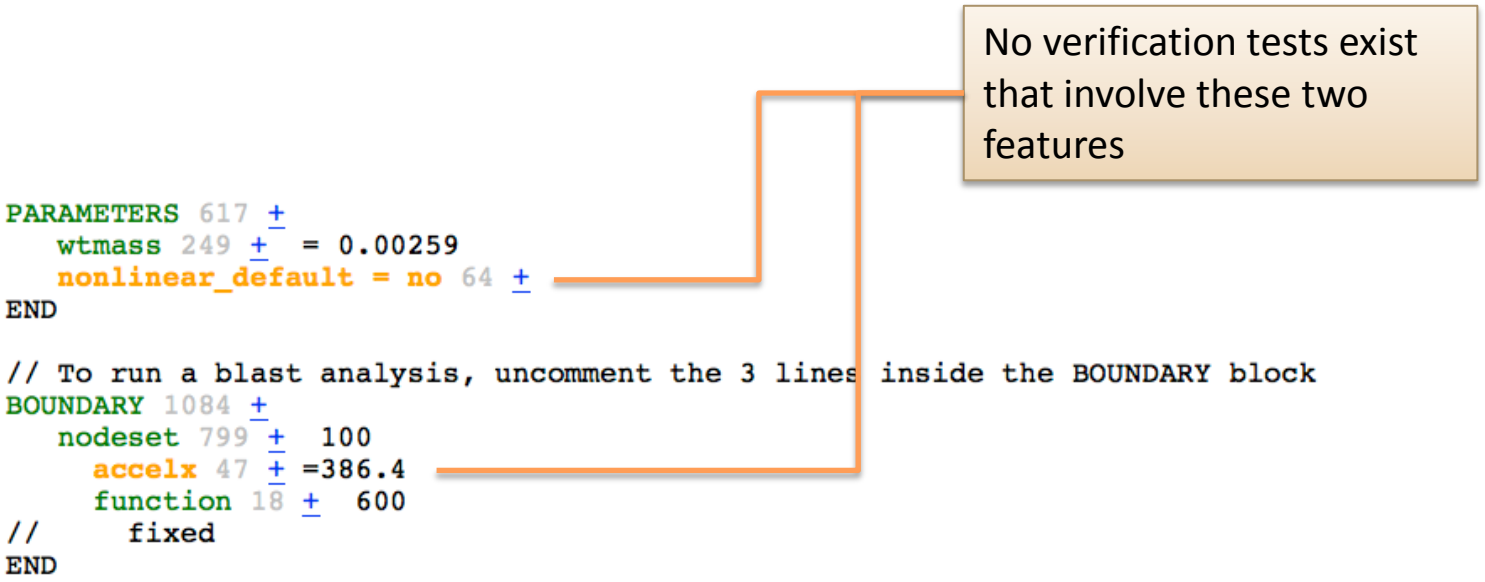
Input File

```
SOLUTION 1087 +
//    eigen nmodes=20
// uncomment all the lines below this to the END and comment the line above to run a nonlinear blast
    NLtransient 31 +
    time_step 31 +  2.0e-5
    nsteps 31 +  8192
    nskip 10 -
Salinas_rtest/verification/visco/visco_poissons_ratio_nl.test|visco_poissons_ratio_nl.np1_feti-dp
Salinas_rtest/verification/visco/visco_poissons_ratio_nl.test|visco_poissons_ratio_nl.np1_sparsepak
Salinas_rtest/verification/visco/visco_poissons_ratio_nl.test|visco_poissons_ratio_nl.np1_gdsw
Salinas_rtest/verification/visco/visco_poissons_ratio_nl.test|visco_poissons_ratio_nl.np1_cf-feti
Salinas_rtest/verification/visco/visco_stress_relaxation_nl.test|visco_stress_relaxation_nl.np1_gdsw
Salinas_rtest/verification/visco/visco_stress_relaxation_nl.test
Salinas_rtest/verification/visco/visco_stress_relaxation_nl.test
Salinas_rtest/verification/visco/visco_stress_relaxation_nl.test
Salinas_rtest/verification/visco/visco_poissons_ratio_nl.test|vi
Salinas_rtest/verification/visco/visco_stress_relaxation_nl.test
 1
    rho 27 +  0.9
    solver = gdsw 292 +
END
```

Clicking on the expansion +/- symbol opens a list of the verification tests.

Items in the list will soon link to documentation and a directly of all the test inputs and outputs.

FCT 1-way coverage

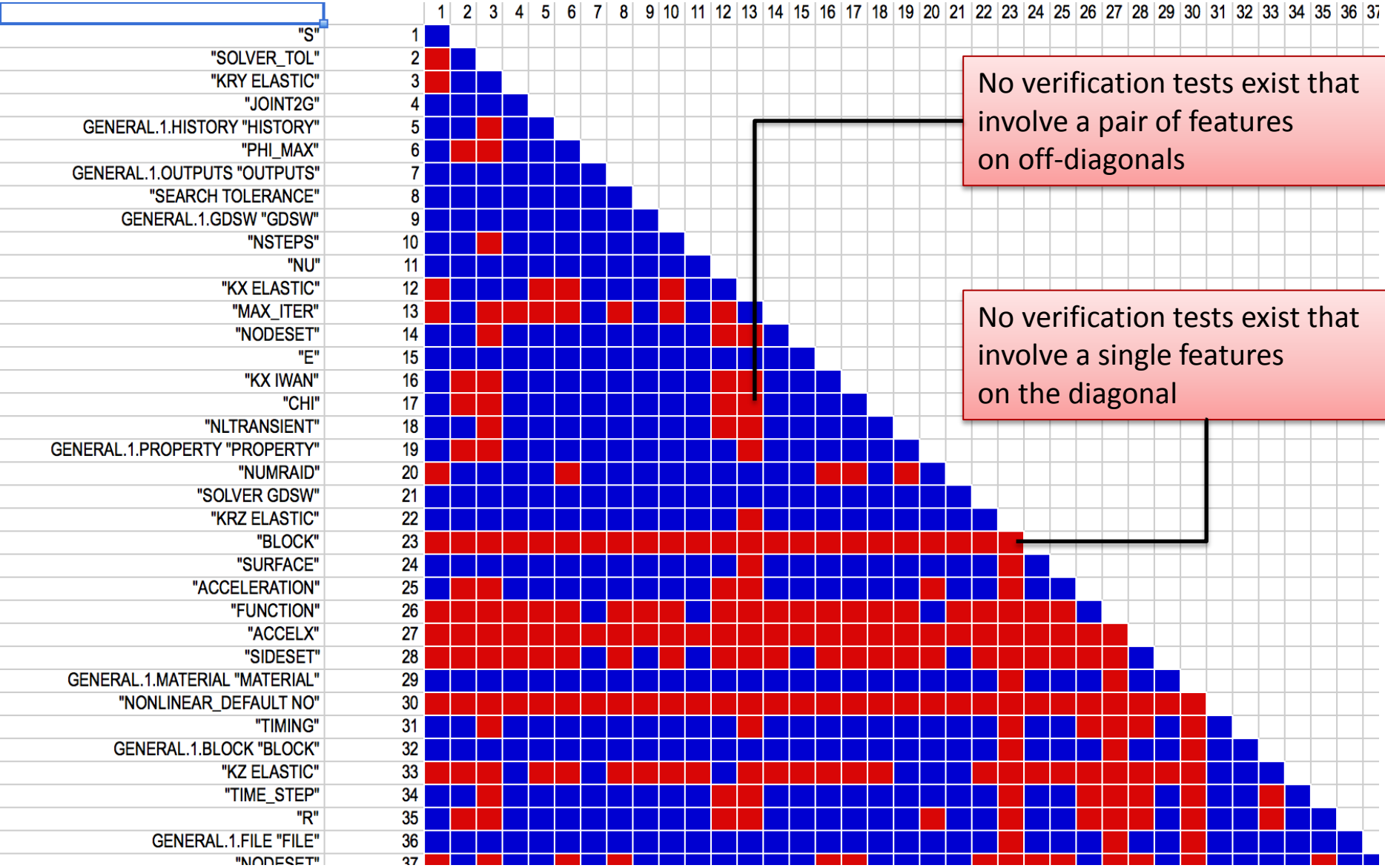


Amber color signifies a limited form of testing: a *regression test* (ensures the feature works the same as it did yesterday).

Red color signifies no test of any kind was found that included that feature.

- Results are an opportunity to discuss these features with developers and whether additional or higher quality testing would be useful.
- 1-way coverage snapshot can be pasted into reports and documentation of your analysis.

FCT 2-way coverage (Excel table)



Summary on Code Verification

- Code verification improves credibility
- Desire to ensure the software works as intended
- Demonstrate there are no defects in how you use it
- Testing and SQA is expensive, but better to prevent issues cheaply than to discover them later
- The feature coverage tool, FCT, can generate evidence as a pathway to communication between users and developers