

Computational Simulations and the Scientific Method

Bil Kleb* and Bill Wood*
NASA Langley Research Center, Hampton, Virginia, 23681

AS scientific simulation software becomes more complicated, the scientific-software implementer's need for component tests from new model developers becomes more crucial. The community's ability to follow the basic premise of the Scientific Method requires independently repeatable experiments, and model innovators are in the best position to create these test fixtures. Scientific software developers also need to quickly judge the value of the new model relative to other models, i.e., the new model's cost-to-benefit ratio in terms of gains provided by the new model and risks such as implementation time and software quality.

This letter asks two questions. The first is whether other scientific software developers would find published component tests useful, and the second is whether model innovators think publishing test fixtures is a feasible approach.

In Refs. 1 and 2, we argue that as computational models become more complex, software unit-testing practices become essential for advancing simulation capabilities. Those papers called for model and algorithm innovators to publish succinct test fixtures, i.e., sample input and output, so that subsequent implementers can independently verify they have correctly translated the new innovation to source code, i.e., so the Scientific Method's notion of independently-verifiable experiments can be used. This letter provides an alternative presentation of those ideas in light of copious feedback.

As growth in computational power facilitates higher-fidelity computational simulation techniques, the number and variety of building-block components also increases. While this increased complexity is forcing a change from the cottage industry of one person/one code to team software development to address increasing software system size,³ the community is not yet routinely publishing independently verifiable tests for new models or algorithms to address the code-verification complexity. The survey results of Table 1 show that only 22% of new models published are accompanied by tests suitable for independently verifying the new model.

To sustain our growing numerical simulation capability, we need to become competent software developers;⁴ and one measure of software development competence is sound software testing practices.^{5,6} For example, before inserting a new component into a system, software developers will perform a set of component-level tests. Based on feedback from previous conference presentations,^{1,2} many agree with the need for component-level testing in the computational simulation community but there is disagreement about how to implement it.

While each development group could derive component-level tests for each model they choose to implement, this duplication is unnecessary and would not likely catch the special cases that the original innovator would likely know intimately. Besides, the Hatton studies of scientific codes underscores the difficulty in achieving consistent implementations: 1 fault per 170 lines.^{7,8}

This letter calls for institutionalizing component-level testing in the computational simulation community and offers one possible route toward implementation. The letter begins by exploring the current practice, recalls basic tenets of the Scientific Method, proposes a course of action, gives a couple brief examples, and finishes with some concluding remarks.

Received 5 December 2004; revision received 28 December 2005; accepted for publication 29 December 2005. This material is declared a work of the U.S. Government and is not subject to copyright protection in the United States. Copies of this paper may be made for personal or internal use, on condition that the copier pay the \$10.00 per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923; include the code 1542-9423/04 \$10.00 in correspondence with the CCC. This material is a work of the U.S. Government and is not subject to copyright protection in the United States.

* Aerospace Engineer, Research and Technology Directorate, Mailstop 408a; AIAA lifetime member.

Table 1 Survey of new component publishing that includes the percent of articles introducing new models that provide tests.

Journal	Vol(#)	Articles* %
JCP	192(2)	0
	192(1)	23
	191(2)	27
IJNMF	43(10–11)	0
	43(9)	20
	43(8)	67
		22

*Upticks indicate articles with component tests, downticks indicate articles lacking component tests, and dots indicate articles that did not appear to introduce a new model.

I. Current Practice

For the sake of discussion, consider the components of a Computational Fluid Dynamics (CFD) code. While developing such a code, a team will pull components, such as flux functions, boundary conditions, turbulence models, transition models, gas chemistry models, data structures, and so on—each from a different original publication. For example, consider 24 components that comprise the FUN3D flow solver[†] listed in Table 2. Now, consider the potential interactions between these components as indicated by the lines in Fig. 1, if none of the components have been verified. While arguments can be made about whether all components necessarily influence all the other components (as drawn), even the most ardent detractor has to concede that this is a complicated system.

As the number of components increases, the potential interactions grow as $n^2/2$, where n is the number of components. The task of finding an error in a system of interrelated components is daunting, but this task becomes untenable if the components have not been independently verified. Rational verification of this complicated system must proceed in two steps: (1) verification of components and then (2) verification of their interactions.

The current computational verification and validation literature recommends verification on the system level by using the Method of Manufactured Solutions (MMS).⁹ Although this necessary step in every code-verification process can verify the whole system of components, it has not yet been widely practiced due to implementation overhead[‡] and because if this system-level test fails, the debugging task could encompass any of n components *in addition* to the roughly $n^2/2$ component interactions. Therefore, before attempting MMS on a system of components, each component should be independently verified.

Table 2 Components in the FUN3D flow solver. Data provided by Eric Nielsen of NASA.

Turbulence model	Transition model	Boundary conditions	Flux limiter
Flux reconstruction	Time relaxation	Convergence acceleration	Flux functions
Entropy fix	Transport properties	Data structures	Gas chemistry
Time integration	Preconditioners	Flux Jacobians	Governing equations
Multiprocessing	Domain decomposition	Preprocessing	Postprocessing
Grid sequencing	Grid adaptation	Grid movement	Load balancing

[†] See fun3d.larc.nasa.gov, last accessed April 14th, 2006.

[‡] MMS typically requires the addition of arbitrary boundary conditions and source functions. In addition, selection of the appropriate basis function remains an art, and so far, only smooth-valued solutions have been manufactured.

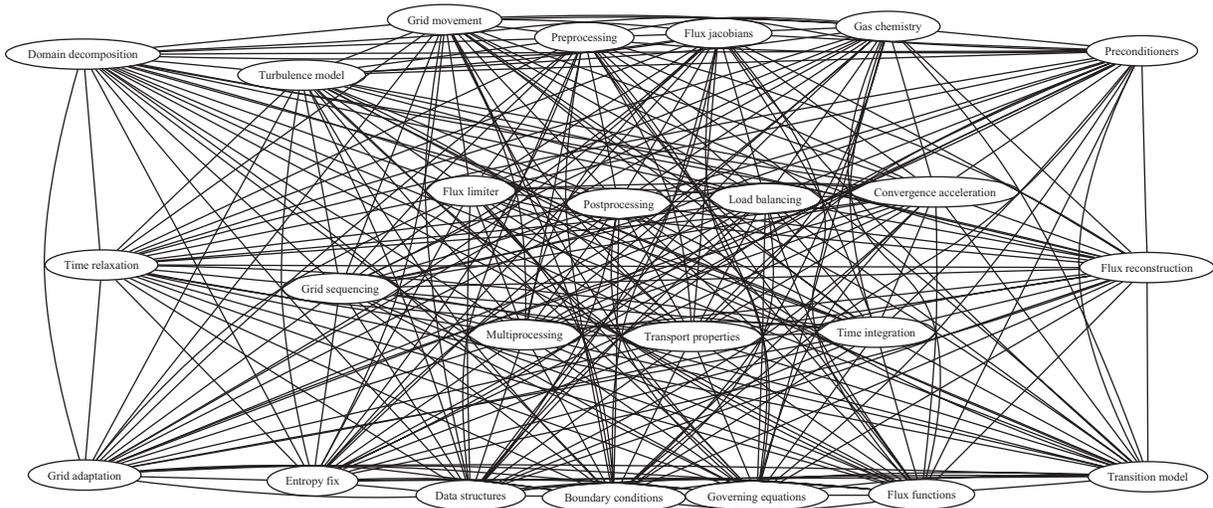


Fig. 1 Potential component coupling for the FUN3D flow solver.

Consider for example, the dilemma created by the debut publication of the popular Spalart-Allmaras turbulence model.¹⁰ The document contains a mathematical description of the model and then shows comparisons with experimental boundary layer profiles that require a complete CFD code system. This scenario is sketched in Fig. 2, in which New Component is the mathematical description of the new turbulence model and the author's code

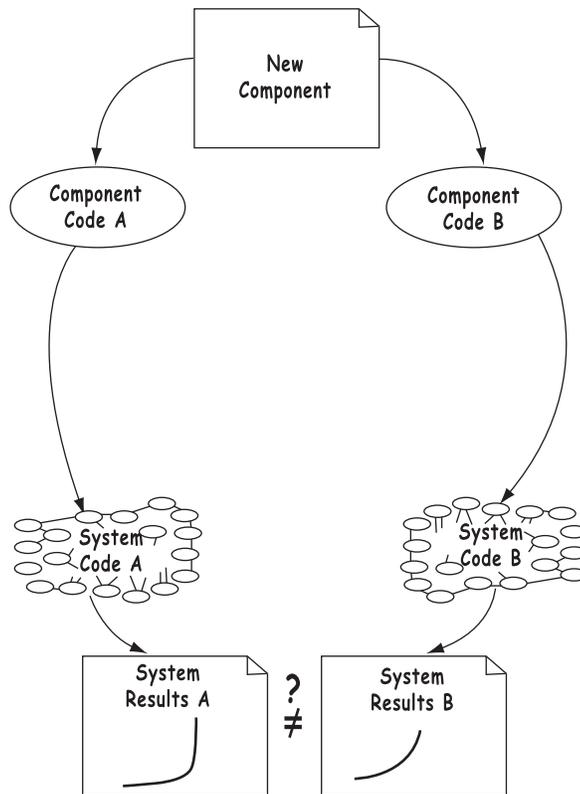


Fig. 2 Current method of translating the “paper” model to numerical results.

are indicated by Component Code A and System Code A. The boundary layer profile output appears at the bottom.

The dilemma is that no isolated tests of the turbulence model itself, either mathematical or numerical, are presented. So, when another CFD code development team (path B) elects to install this new model in their system, a comparison with boundary layer profiles does not assure the model was implemented in the same way as the original because the other code components are completely different. The specific effects of the turbulence model become lost in the large computational simulation infrastructure, and there is no credible means to determine that both codes are using precisely the same model. As a consequence of this implementation risk and the lack of test data, the implementer is unable to quickly determine the value of the new model.

II. The Scientific Method

In a computational context, component-based verification testing is the engine behind the Scientific Method that Roger Bacon first described in the thirteenth century: a repeating cycle of *observation, hypothesis, experimentation*, and the need for *independent verification*.¹¹

Popularized by Francis Bacon and Galileo Galilei, the Scientific Method has since become a means of differentiating science from pseudoscience. The Scientific Method is fueled by the idea that hypotheses and models must be presented to the community *along with* the description of experiments that support the hypothesis. The experiments that accompany a hypothesis must be documented to the extent that others can repeat the experiment—Roger Bacon’s independent verification.

This last notion, that others should be able to repeat an experiment is currently not widely practiced by computer simulation software community. In part, this is due to the large body of software required by a modern simulation capability. While electronic documentation methods such as Quirk’s Amrita system[§] can go a long way toward solving this issue, the fact remains that our experiments must be small enough and isolated enough to be independently repeatable and widely applicable. Ultimately, an implementer should be able to come to the same conclusion as another implementer based solely on the numerical evidence.[¶]

III. Proposed Practice

How can the computational simulation community leverage the Scientific Method?—by having innovators publish a set of tests when they present a new model or algorithm so implementers can gage the innovation’s value with respect to similar models and reliably make the transition from the mathematics to the numerics.

This notion is depicted by the pages labeled **Component Verification** in Fig. 3, where model innovators publish component test fixtures so that developer B can verify the numerical implementation of the mathematical model or algorithm in isolation before inserting it into her simulation system.

The test fixtures, or numerical experiments, should consist of simple input/output combinations that document the behavior of the model. In particular, boundary cases and any other special cases should be documented. For example, the temperature domain of Sutherland’s viscosity law or the non-realizable initial states for a linearized Riemann solver flux function.

Wherever possible, tests should be written at the mathematical level,[#] but some actual discrete values should also be presented. The latter is particularly advantageous if the experiments are designed to expose boundary areas that are sensitive to divided differences, nonlinear limiters, or truncation error. (Examples are given in the next section.) Further advantages of electronically published tabular test fixture data are that verification can be automated and typographic or mathematical errors in the verification process can be discovered and rectified.

All subsequent developers that implement the model and publish their results would be required to document which of the original verification experiments they conducted. Over time, the popular techniques could have a suite of tests formally sanctioned by a governing body such as the AIAA so that any implementation would have to pass the standard tests to be considered verified. Otherwise, the community is left to suffer the fate of unquantified uncertainties as described by Youden’s two seminal works.^{13,14}

[§] See www.amrita-cfd.org, last accessed April 14th, 2006.

[¶] Michael Hensch’s restatement of Shewhart’s 1st Law of Data Presentation—for the original, see page 58 of Ref. 12.

[#] These tests could also be published in terms of Method of Manufactured Solutions at the component level.

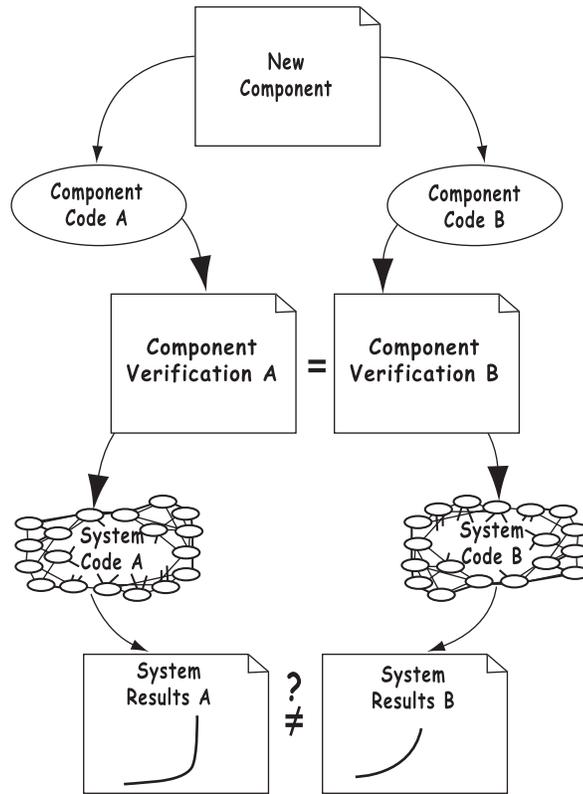


Fig. 3 Proposed method of translating the “paper” model to numerical results.

IV. Examples

Kleb and Wood¹ contains examples for the CIR flux function and the Van Albada limiter function. In this letter, two simple components are discussed: the Sutherland viscosity law and the modified Newtonian law.

Table 3 shows a succinct component test fixture for Sutherland’s viscosity law, which gives the viscosity of air as a function of temperature. The mathematical form is presented along with boundary points and a value from the middle of the domain. While this example is trivial, it demonstrates the very localized level at which components should be considered.

Table 3 Sutherland’s viscosity law component test fixture.

input T (K)	output μ (kg/s-m)
$200 \leq T \leq 3000$	$K^* \frac{T^{1.5}}{T + 110.4}$
199	error
200	1.329×10^5
2000	6.179×10^5
3000	7.702×10^5
3001	error

*where $K = 1.458 \times 10^{-6}$.

Table 4 Modified Newtonian Law component test fixture.

θ (deg)	input		output
	M_∞	γ	C_p
$0 \leq \theta \leq 90$	$1 \leq M_\infty$	$1 \leq \gamma \leq 3$	Eq. 1
0	100	1	2.000
45	100	1	0.500
0	100	1.4	1.839
0	5	1.4	1.809
0	1.0	\forall	error
91	\forall	\forall	error
-91	\forall	\forall	error

Where \forall indicates “for all valid values”.

For example, the flux function example presented in Ref. 1 was attacked on the grounds that it would be impossible to cover all the discretization settings in which it would be applied, e.g., finite volume, finite difference, or finite element. These considerations are an indication that the component is being defined at too high a level.

Another component examined is the Modified Newtonian law, which gives pressure coefficient as a function of surface inclination according to,

$$C_p = C_{p_{\max}} \sin^2 \theta \quad (1)$$

where θ is the surface inclination angle in degrees, i.e., the angle between the incoming flow and the surface normal vector. The stagnation pressure coefficient is governed by shock relations,

$$C_{p_{\max}} = \frac{2}{\gamma M_\infty^2} \left\{ \left[\frac{(\gamma + 1)^2 M_\infty^2}{4\gamma M_\infty^2 - 2(\gamma - 1)} \right]^{\gamma/(\gamma-1)} \left[\frac{1 - \gamma + 2\gamma M_\infty^2}{\gamma + 1} \right] - 1 \right\}$$

where M_∞ is the freestream Mach number and γ is the ratio of specific heats. A sample component test fixture for this law is shown in Table 4. Again, it begins by defining the valid input domains with pure math. Next, certain limiting cases are provided along with a sampling of interior points. Finally, boundary cases are shown and suggested error messages are given.

An advection-diffusion code developed using extensive component-based testing is documented in Ref. 15, and is available from the authors.

V. Concluding Remarks

To sustain our growing numerical simulation capabilities, we need to become competent software developers by increasing our use of component testing practices. The implementation path offered here is to have model innovators publish simple, component-level verification test fixtures so that implementers can verify their implementation according to the basic premise of the Scientific Method—independently-verifiable experiments.

Based on feedback from prior publications^{1,2}, most readers agree that component-level testing should be standard practice in the computational simulation software development community. However, two questions remain:

Do scientific software developers want published component tests?

Is the proposed solution palatable by model innovators?

If the answer to either is “no,” then how should we proceed?

References

¹Kleb, B. and Wood, B., “CFD: A Castle in the Sand?” AIAA Paper 2004-2627, June 2004.

²Kleb, B. and Wood, B., “Computational Simulations and the Scientific Method,” AIAA Paper 2005-4873, June 2005.

³Alexandrov, N., Atkins, H. L., Bibb, K. L., Biedron, R. T., Gnoffo, P. A., Hammond, D. P., Jones, W. T., Kleb, W. L., Lee-Rausch, E. M., Nielsen, E. J., Park, M. A., Raman, V. V., Roberts, T. W., Thomas, J. L., Vatsa, V. N., Viken, S. A., White, J. A., and Wood, W. A., "Team Software Development for Aerothermodynamic and Aerodynamic Analysis and Design," NASA/TM 2003-212421, Nov. 2003.

⁴Quirk, J. J., "Computational Science: Same Old Silence, Same Old Mistakes, Something More is Needed," *Adaptive Mesh Refinement—Theory and Applications*, edited by T. Plewa, T. Linde, and V. G. Weirs, Springer-Verlag, 2004, pp. 1–26.

⁵Beck, K., *Test Driven Development: By Example*, Addison-Wesley, 2002.

⁶Sommerville, I., *Software Engineering*, Addison Wesley, 7th ed., 2004.

⁷Hatton, L., "The T Experiments: Errors in Scientific Software," *IEEE Computational Science and Engineering*, Vol. 4, No. 2, 1997, pp. 27–38.

⁸Hatton, L. and Roberts, A., "How Accurate is Scientific Software?" *IEEE Transactions on Software Engineering*, Vol. 20, No. 10, Oct. 1994, pp. 785–797.

⁹Roy, C. J., "Review of Code and Solution Verification Procedures for Computational Simulation," *Journal of Computational Physics*, Vol. 205, No. 1, 2005, pp. 131–156.

¹⁰Spalart, P. R. and Allmaras, S. R., "A One-Equation Turbulence Model for Aerodynamic Flows," AIAA Paper 92–0439, Jan. 1992.

¹¹Bacon, R., *Opii: Majus, Minus, and Tertium*, c.1267.

¹²Shewhart, W. A., *Economic Control of Quality of Manufactured Product*, D. Van Nostrand Company, 1931.

¹³Youden, W. J., "Systematic Errors in Physical Constants," *Physics Today*, Sept. 1961, pp. 32–43.

¹⁴Youden, W. J., "Enduring Values," *Technometrics*, Vol. 14, No. 1, Feb. 1972, pp. 1–11.

¹⁵Wood, B. and Kleb, B., "Exploring eXtreme Programming for Scientific Research," *IEEE Software*, Vol. 20, No. 3, May 2003, pp. 30–36.