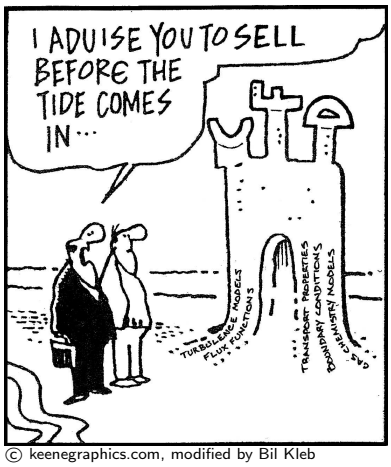
34th AIAA Fluid Dynamics Conference
June 28 – July 1, 2004 / Portland, Oregon
CFD Verification and Validation session 88-FD-22

CFD: A Castle in the Sand?

Bil Kleb and Bill Wood
Aerospace Engineers, Aerothermodynamics Branch
NASA's Langley Research Center, Hampton, Virginia 23681

AIAA Paper 2004–2627

The computational simulation community is not routinely publishing independently verifiable tests to accompany new models or algorithms. A survey reveals that only 22% of new models published are accompanied by tests suitable for independently verifying the new model. As the community develops larger codes with increased functionality, and hence increased complexity in terms of the number of building block components and their interactions, it becomes prohibitively expensive for each development group to derive the appropriate tests for each component. Therefore, the computational simulation community is building its collective castle on a very shaky foundation of components with unpublished and unrepeatable verification tests. The computational simulation community needs to begin publishing component-level verification tests before the tide of complexity undermines its foundation.



© keenegraphics.com, modified by Bil Kleb

1 Introduction

GROWTH in computational power naturally facilitates higher-fidelity computational simulation techniques. But as simulation codes grow more sophisticated, their number of building-block components also increases. The increased complexity is forcing a change from the cottage industry of one person/one code to team software development.[1]

For the continued viability of our computational community we need to be more than clever engineers and mathematicians, we also need to be competent software developers.[2] One distinguishing aspect of competent software developers is their software testing practice. Before inserting a new component into a system, they will perform a set of component-level tests.

There is a tremendous duplication of effort if each development group must independently derive all the component-level test for each model they implement. Further, without repeatable verification, the Hatton studies showing 1 fault per 170 lines for scientific codes[3] highlights the difficulty in achieving consistent implementations. Component-level tests should be published by the original authors who are in the best position to provide these component-level verification tests.

[1] Natalia Alexandrov, et al., *Team Software Development for Aerothermodynamic and Aerodynamic Analysis and Design*, NASA/TM 2003-212421, 2003; L. Cambier and M. Gazaix, *elsA: An Efficient Object-Oriented Solution to CFD Complexity*, AIAA Paper 2002-0108, 2002; and N. Kroll, et al., *MEGAFLOW—A Numerical Flow Simulation System*, ICAS 98-2.7.4, 1998.

[2] James J. Quirk, "Computational Science: Same Old Silence, Same Old Mistakes, Something More is Needed", in *Adaptive Mesh Refinement – Theory and Applications*, edited by Tomaz Plewa, et al., Springer-Verlag, 2004, pp. 1–26.

[3] Les Hatton, "The T Experiments: Errors in Scientific Software", *IEEE Computational Science and Engineering*, 4(2), 1997, pp. 27–38; and Les Hatton and Andy Roberts, "How Accurate is Scientific Software?", *IEEE Transactions on Software Engineering*, 20(10), 1994, pp. 785–797.

Third printing, late July 2004.

This paper explores the lack of component-level testing in the computational simulation community and proposes a course correction that will enable the community to build a solid foundation for increasingly complicated computational codes. The paper revisits the Scientific Method, explores the current practice, proposes a new course of action, and presents test fixture examples.

2 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*.[4]

[4] Roger Bacon, *Opii: Majus, Minus, and Tertium*, c.1267.

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.

3 Current Practice

To develop a CFD 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 the 24 components that comprise the FUN3D flow solver[5] listed in table 1. Now, consider the

[5] fun3d.larc.nasa.gov

Table 1: 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

potential interactions between these components as indicated by the lines in figure 1. 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 system is nevertheless a complicated set of interrelated components.

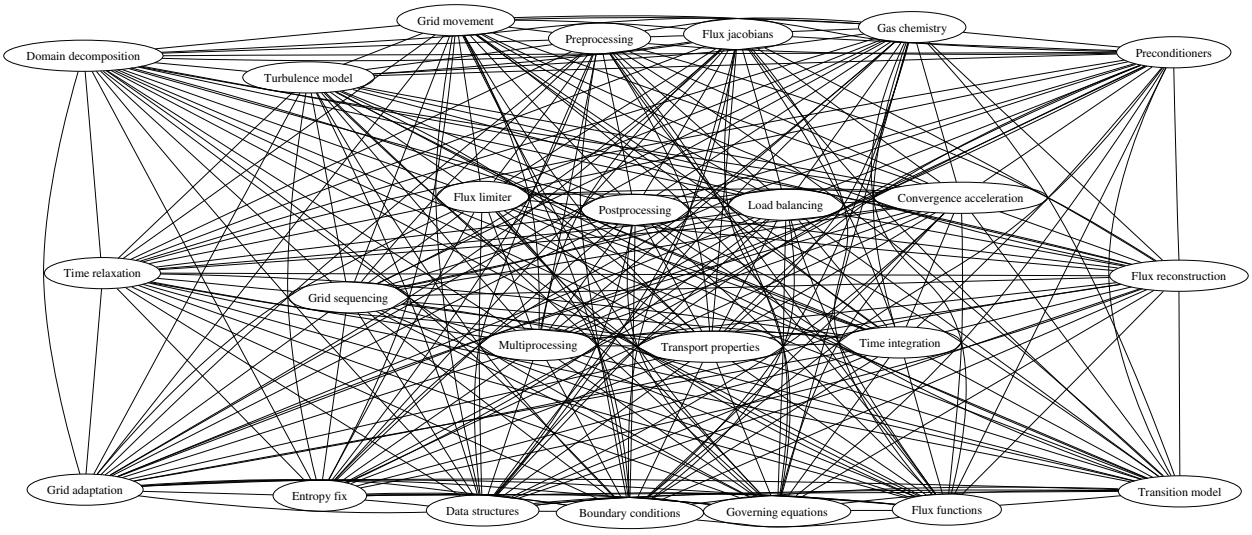


Figure 1: Components interactions in the FUN3D flow solver.

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

The current computational verification and validation community recommends verification on the system level; that is, test the entire collection of components that make up a given code in one shot by using the method of manufactured solutions.[6] This approach typically requires new components be added specifically to accommodate the arbitrary boundary conditions and source functions required by manufactured solutions. In addition, selection of the appropriate basis function for the manufactured solution remains an art, and so far, only smooth-valued solutions have been manufactured.

The goal method of manufactured solutions in this case is to verify the entire system attains its theoretical order-of-accuracy properties. But because this is a system-level test, the potential source of errors from component interactions grows as $n^2/2$. Therefore, before attempting the method of manufactured solutions on a system of components, each component should be independently verified.

As discussed earlier, verification at the component level is the essential ingredient for the advancement of numerical analysis according to the Scientific Method, but currently, component-level tests rarely accompany publications that introduce new models and algorithms. Table 2, a sampling of recent issues of the *Journal of Computational Physics*

[6] Patrick J. Roache, *Verification and Validation in Computational Science and Engineering*, Hermosa, 1998; William L. Oberkampf and Timothy G. Trucano, "Verification and Validation in Computational Fluid Dynamics", *Progress in Aerospace Sciences*, 2002, pp. 209–272; and Christopher J. Roy, "Verification of Codes and Solutions in Computational Simulation", in *Proceedings of CHT-04: International Symposium on Advances in Computational Heat Transfer*, publisher?, 2004.

Table 2: Survey of new component publishing. Upticks indicate articles with component tests, downticks indicate articles lacking component tests, and dot articles did not appear to introduce a new model. The percent of new model articles containing tests is also given.

journal	vol(#)	articles	%
JCP	192(2)	ıı·ıı·ı·ı·ı	0
	192(1)	·ı·ı·ı·ıı·ıı	23
	191(2)	ıı·ı·ı·ı·ı	27
IJNMF	43(10–11)	ıı·····ıı	0
	43(9)	·ı·····ıı·ı	20
	43(8)	·ı·····ıı·ı	67
			22

(JCP) and the *International Journal for Numerical Methods in Fluids* (IJNMF), reveals that only 22% of the 49 new models introduced are published with component-level verification data. A notable exception that provides verification tests is a series of boundary layer papers in IJNMF:43(8).

The omission of component-level verification is also evident in the proposed phases of computational modeling and simulation[7] shown in figure 2. It is important to note that this figure is drawn at the *system* level, that is, from the perspective of the complete computational simulation software system. The boxed step, Computer Programming of the Discrete Model, is the topic of this paper, but we argue that this step must contain component-by-component verification before attempting system-level verification.

This distinction is important not only because component-level testing is simpler than testing the entire system, but also because it is the necessary first step when building a complex system. By first testing at the component level, developers avoid what Steve McConnell, author of *Code Complete* and *Rapid Software Development* has declared the absolute worst software development practice: code-n-fix.[8]

Consider, for example, the publication of the Spalart-Allmaras turbulence model.[9] The document contains a mathematical description of the model and then shows comparisons with experimental boundary layer profiles that require the use of a complex computational simulation system like the one portrayed earlier in figure 1. This scenario is sketched in figure 3, in which New Component is the

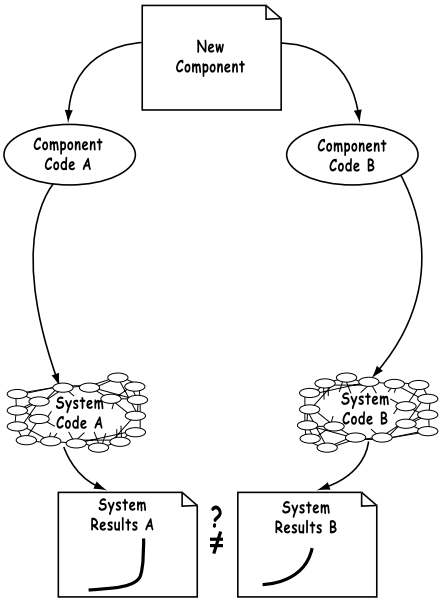


Figure 3: Current method of translating the "paper" model to numerical results.

Table 2: Repeated for convenience.

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

[7] William L. Oberkampf, et al., "Error and Uncertainty in Modeling and Simulation", *Reliability Engineering and System Safety*, 2002, pp. 333–357.

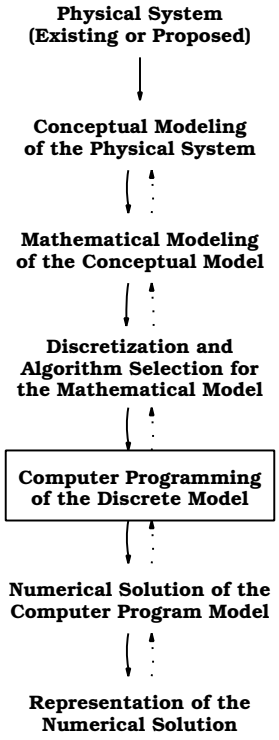


Figure 2: Proposed phases of computational modeling and simulation.

[8] Rick Wayne, "While Rome Burns?", *Software Develpment*, 2004, pp. 48–49; and Steven McConnell, *Code Complete*, Microsoft Press, 1993.

[9] P. R. Spalart and S. R. Allmaras, *A One-Equation Turbulence Model for Aerodynamic Flows*, AIAA Paper 92–0439, 1992.

mathematical description of the new turbulence model and the author's code are indicated by Component Code A and System Code A. The boundary layer profile output appears at the bottom.

The issue is that no isolated tests of the turbulence model itself, either mathematical or numerical, are presented. So, when another CFD developer (path B) implements this new model in her system, a comparison with boundary layer profiles does not assure the model was implemented in the same way as the original because most of the code components are completely different. The specific effects of the turbulence model become lost in the large sea of computational simulation infrastructure in which it has been cut adrift, and there is no credible means for someone else to be assured that they too are employing precisely the same model in their code.

4 *Proposed Practice*

How can the computational simulation community realign itself with the Scientific Method—by publishing a set of tests when a new model or algorithm is presented so that when others make the leap from the mathematics to the numerics they can have a means to verify their component's implementation before inserting it into their system. This notion is depicted by the pages labeled Component Verification in figure 4.

The tests, or numerical experiments, should consist of simple input/output combinations that document the behavior of the model. In addition, any limiting cases should also be documented; for example, the temperature range of Sutherland's viscosity law or the nonrealizable initial states for a linearized Riemann solver. Wherever possible, tests should be written at both the mathematical and numerical levels and could be given in terms of the method of manufactured solutions on the component level. 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 section 5.)

All subsequent developers that implement the model and publish their results would be required to document which of the original verification experiments they conducted and the results of those experiments. 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.

Once the errors and limits of each component can be quantified, error analysis can be used to build a notion of the entire system's uncertainty levels.[10]

[10] W. J. Youden, "Systematic Errors in Physical Constants", *Physics Today*, 1961, pp. 32–43; and W. J. Youden, "Enduring Values", *Technometrics*, 14(1), 1972, pp. 1–11.

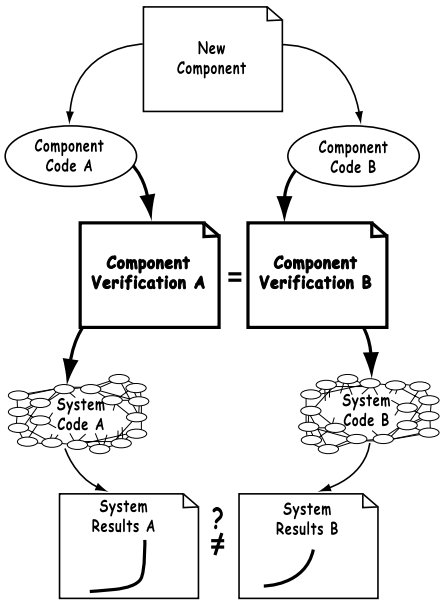


Figure 4: Proposed method of translating the "paper" model to numerical results: Publish component tests so that developer "B" can verify the numerical implementation of the mathematical model or algorithm in isolation before inserting it into her simulation system.

5 *Examples*

Suppose a new flux function for a finite-volume solver were created. As usual, it would be documented in mathematical terms, but it would also be accompanied by analytical test cases that document its behavior for known interface states such as supersonic flow to the right and left, vacuum expansion, and so forth.

Numerical results would also be provided that not only showed the flux resulting from left and right input states covering the typical regimes but cases which explored the limits or transition points of the scheme would be provided as well.

Consider the CIR scheme[11] for the linear wave equation, $u_t + a\,u_x = 0$,

$$F_{i+\frac{1}{2}} = \frac{a}{2}\,(u_i + u_{i-1}) - \frac{1}{2}|a|\,(u_i - u_{i-1})$$

[11] R. Courant, et al., "On the solution of non-linear hyperbolic differential equations", *Communications on Pure & Applied Mathematics*, 1952, pp. 243–255.

The corresponding tests are shown in table 3. While this

Table 3: Example of CIR tests.

(a) Mathematical

inputs			$F_{i-\frac{1}{2}}$
a	u_{i-1}	u_i	
> 0	$\forall$	$\forall$	$a\,u_{i-1}$
< 0	$\forall$	$\forall$	$a\,u_i$
0	$\forall$	$\forall$	0

(b) Numerical

inputs			$F_{i-\frac{1}{2}}$
a	u_{i-1}	u_i	
2	-1	5	-2
-1	0	2	-2
0	2	3	0

example is trivial, it serves to give a flavor of the proposed component tests.

An example that demonstrates a sensitivity to numerical implementation is the Van Albada symmetric averaging function commonly used to limit slope reconstructions,

$$M(a,b) = \frac{(a\,b + \epsilon^2)(a+b)}{a^2 + b^2 + 2\,\epsilon^2}$$

where a and b are slopes and ϵ is proportional to the local mesh spacing.[12] Example component tests are shown in table 4.

If this averaging function was implemented by using the limiter form of Sweby[13] that uses ratios and drops ϵ,

$$\psi\left(\frac{a}{b}\right) = \frac{\frac{a}{b}(\frac{a}{b}+1)}{(\frac{a}{b})^2 + 1}$$

one would not obtain correct behavior when slope b approached zero and when both slopes approached zero. (See the last column of table 4b.)

[12] G. D. van Albada, et al., "A Comparative Study of Computational Methods in Cosmic Gas Dynamics", *Astronomy and Astrophysics*, 108, 1982, pp. 76–84.

[13] Peter K. Sweby, "High Resolution TVD Schemes Using Flux Limiters", in *Large-Scale Computations in Fluid Mechanics*, edited by Bjorn E. Engquist, et al., American Mathematical Society, vol. 22 of *Lectures in Applied Mathematics*, 1985, pp. 289–309.

Table 4: Example of Van Albada tests.

(a) Mathematical

inputs			M
a	b	ϵ^2	
$\forall$	a	0	a
$\forall$	$-a$	0	0
$\forall$	0	$\forall$	$\frac{a\epsilon^2}{a^2+2\,\epsilon^2}$

(b) Numerical

inputs			M	$b\,\psi(\frac{a}{b})$
a	b	ϵ^2		
2	2	0	2	2
2	-2	0	0	0
1	0	2	0.4	NaN
0	1	1/2	1/4	0
-1	-2	70000	-1.5	-1.2

More extensive examples of component-based testing are available for an advection-diffusion solver[14] that was written during an exploration of Extreme Programming for scientific research.[15] The code was written in Ruby[16] and is available from the authors.

6 Concluding Remarks

The Scientific Method is used as a backdrop against which current computational simulation development practices are compared. The argument is presented that the community has strayed from the Scientific Method by failing to publish component-level verification tests when introducing a new component algorithm. These tests should contain specific inputs and numerical outputs. A plea is made for realigning with the Scientific Method by publishing such tests to facilitate verifications by others who also implement the component.

A protocol is proposed for the introduction of new methods and physical models that would provide the community with a credible history of documented, repeatable verification experiments that would enable independent replication. The community can then begin tracking uncertainties at the component level and begin systematic error analysis.

[14] William A. Wood and William L. Kleb, "Multi-Stage Runge-Kutta Circular Advection Solver, Release 3.0", *NASA Tech Briefs*, 2002, p. 43.

[15] William A. Wood and William L. Kleb, "Exploring XP for Scientific Research", *IEEE Software*, 20(3), 2003, pp. 30–36.

[16] David Thomas and Andrew Hunt, *Programming Ruby: The Pragmatic Programmer's Guide*, Addison-Wesley, 2001.

Acknowledgments

Both authors are indebted to Bill Oberkampf of Sandia National Laboratories, Chris Roy of Auburn University, and Pat Roache of Ecodynamics Research Associates for outstanding verification and validation AIAA short courses; to James Quirk of Los Alamos National Laboratories for planting the seed so many years ago that all was not right in the world of computational simulation and for demonstrating the courage and integrity to publish it.[17]

The authors would like to thank the following people for providing feedback during the course of this work: Steve Allmaras of Boeing; Dimitri Mavriplis of the University of Wyoming; Bill Oberkampf of Sandia National Laboratories, Natalia Alexandrov, Steve Alter, Bob Biedron, Peter Gnoffo, Mike Hemsch, Brian Hollis, Benjamin Kirk, Beth Lee-Rausch, Charles Miller, Joe Morrison, Eric Nielsen, Russ Rausch, Chris Rumsey, and Jim Thomas of NASA; participants in the 2nd AIAA Drag Prediction Workshop; Aldo Bonfiglioli of the University of Basilicata; Leonardo Scalabrin of the University of Michigan; James Quirk of Los Alamos National Laboratory; and Chris Roy of Auburn University.

The authors would also like to thank Eric Nielsen of NASA for providing an extensive list of FUN3D code components used to generate figure 1 on page 3; Richard Wheless of NCI Information Systems for translating the first author's lame sketches for figures 3 and 4 on page 5 into more presentable form; and Susan Hurd of NCI Information Systems for technical editing the final draft.

Bil Kleb would like to thank Mika LaVaque-Manty of the University of Michigan and James Grenning of Object Mentor for discussions regarding the origins and nature of the scientific method.

[17] James J. Quirk, *A Contribution to the Great Riemann Solver Debate*, ICASE Report 92–64, 1992.

About the Authors

Bil Kleb completed his BS in Aeronautical and Astronautical Engineering (AAE) from Purdue University in 1988. Afterward, Bil spent time at NASA's Langley Research Center developing a time-accurate, local-time-stepping algorithm for computational unsteady aerodynamics and was awarded his MS in AAE from Purdue University in 1990. For the past 14 years, Bil has worked in the area of computational aerothermodynamics at NASA's Langley Research Center. During this time, he pioneered the first full-vehicle reentry simulation of the Shuttle Orbiter, earned an MBA from the College of William and Mary, and earned a PhD of Aerospace Engineering from the University of Michigan.

Since 1999, Bil has been an active member in what is now called the agile software development community[18] and has given several invited lectures on team software development for scientific software. For the past two years, Bil has been the steward of the High Energy Flow Solver Synthesis team[19] and has been serving on the XP/Agile Universe conference committee since 2002.

Email: Bil.Kleb@NASA.Gov

[18] For agile software development's succinct, but extremely powerful manifesto, see agilemanifesto.org.

[19] hefss.larc.nasa.gov

Bill Wood has worked in the field of CFD in the Aerothermodynamics Branch at NASA's Langley Research Center since 1991, earning a Ph.D. in Aerospace Engineering from Virginia Tech in 2001. He has served on the program committee for the software development conference XP/Agile Universe from 2002–2004.

Email: Bill.Wood@NASA.Gov

Colophon

This document was typeset in Computer Modern font with the free, cross-platform LaTeX typesetting system using the `handout` option of the AIAA package,[20] version 3.7, which simulates the layout style espoused by visual design expert, Edward Tufte.[21] Also employed were the `color`, `subfigure`, `booktabs`, `multirow`, `threeparttable`, `varioref`, `wrapfig`, `hyperref`, and `nohyperref` packages.

Table 2 on page 3 was created using Ruby[22] and Figure 1 on page 3 was created using Ruby and Graphviz's `neato` program.[23]

[20] www.ctan.org

[21] Edward R. Tufte, *The Visual Display of Quantitative Information*, Graphics Press, 1983; Edward R. Tufte, *Envisioning Information*, Graphics Press, 1990; and Edward R. Tufte, *Visual Explanations: Images and Quantities, Evidence and Narrative*, Graphics Press, 1997.

[22] David Thomas and Andrew Hunt, *Programming Ruby: The Pragmatic Programmer's Guide*, Addison-Wesley, 2001.

[23] www.graphviz.org