Editorial Process Verification

Fetzer's article "Program Verification: The Very Idea" in the September 1988 issue of *Communications* (pp. 1048–1063) is not a serious scientific analysis of the nature of verification. The article distorts the practice and goals of program verification and reflects a gross misunderstanding on the part of the author about the nature of program verification. This article does not meet minimal levels of serious scholarship and its publication highlights some important problems with *Communications* in general.

First, Fetzer's argument misrepresents the goals of program verification. He erroneously asserts that the purpose of program verification is to provide an absolute guarantee of correctness with respect to the execution of a program on computer hardware. The author provides no citations to published claims of "conclusive absolute verification" as the goal of program verification research; indeed, the article is amazingly devoid of references to the program verification literature at all. Moreover, there are many uncited technical papers (some references are given below) in which the limits of proofs and verification are carefully drawn. Despite the author's claim to the contrary, he has set up a "straw man" definition of verification in which he misrepresents the work of large numbers of computer scientists.

Furthermore, the article demonstrates a gross misunderstanding of program verification and the role of mathematics in any engineering endeavor. Statements are made that are simply untrue, e.g., that assembly language programs do not lend themselves to verification. The author appears totally unaware of the large body of work that demonstrates the application of formal verification methods to compilers, operating systems, and computer hardware. In fact, the argument in the article seems to be based on the premise that verification can be applied only to abstract programs written in high level languages, which is patently false and should have been known by anyone having an acquaintance with the program verification literature.

The conclusion of the article is that since program verification provides no certainty, it is useless and possibly harmful. However, the same arguments used in the article apply to any research in computer science that involves the static analysis of programs, including most of theoretical computer science. We are amazed that the editors failed to recognize the potential implications of such a far-reaching and controversial argument and that they did not take greater care to ensure the accuracy of the statements in the article—as indicated by the considerable amount of misinformation contained in it.

ACM claims to be a professional society, responsible to the public at large for establishing and maintaining professional standards of conduct among its members, and for advancing and promulgating responsible codes of practice. Formal approaches to system construction and analysis are directly motivated by the concern to construct trustworthy systems—systems whose developers will be willing to stand before the public and take responsibility for the consequences of their deployment. As yet, this goal is unrealized, but work in formal verification is a serious contribution toward that goal, and is both socially and scientifically responsible. As such, *Communications*' responsibility is surely to educate the members of the ACM about this field: its philosophical and technical basis, its achievements, its prospects, and its limitations. Serious and well-informed discussion of the limitations to formal verification, and of its strengths and defects relative to other responsible proposals for the construction of trustworthy systems, is entirely appropriate and would, we are sure, be welcomed by the practitioners of formal verification as well as by the general membership of ACM. Notice that *Communications* has several categories of contributions other than articles (subject to review and authentication) which are appropriate for expressions of opinions.

However, by publishing the ill-informed, irresponsible, and dangerous article by Fetzer, the editors of *Communications* have abrogated their responsibility, to both the ACM membership and to the public at large, to engage in serious enquiry into techniques that may justify the practice of computer science as a socially responsible engineering endeavor. The article is ill-informed and irresponsible because it attacks a parody of both the intent and the practice of formal verification. It is dangerous because its pretentious and ponderous style may lead the uninformed to take it seriously.

REFERENCES

Good starters are several books published by Prentice-Hall, see especially the postscript to Cliff Jones' *Systematic Software Development using VDM* and Don Good's article "Mechanical Proofs about Computer Programs" in *Mathematical Logic and Programming Languages*, edited by Hoare and Shepardson. A current technical reference is Avra Cohn's Cambridge University Technical Report Number 134 "Correctness Properties of the Viper Block Model: The Second Level." Overviews and pointers to verifica-

tion work appear in the Verification Workshop series [*Software Engineering Notes*, July 1980, July 1981, August 1985].

Mark Ardis
Software Engineering Institute
Victor Basili
U. Maryland
Susan Gerhart
MCC
Donald Good
Computational Logic Inc.
David Gries
Cornell University
Richard Kemmerer
U.C. Santa Barbara
Nancy Leveson
U.C. Irvine
David Musser
R.P.I.
Peter Neumann
SRI
Friedrich von Henke
SRI

Response from the author:

The ancient practice of killing the messenger when you do not like the message receives its latest incarnation in the unfortunate letter from Ardis, Basili, et al. ("The Gang of Ten"). The authors allege (a) that I have misrepresented the goal of program verification (thereby attacking a "straw man"); (b) that I have misunderstood the role of mathematics in any engineering endeavor (especially within computer science); and (c) that my conclusion, if it were true, would undermine research in vast areas of computer science (including most theoretical work). They claim my paper contains "considerable misinformation", contending (i) that there are no published claims to "conclusive absolute verification" as the objective of program verification; (ii) that assembly language programs *are* amenable to verification procedures; and, most significantly, (iii) that "the author appears totally unaware of the large body of work that demonstrates the applicability of formal verification methods to compilers, operating systems, and computer hardware". The magazine has therefore published an "ill-informed, irresponsible and dangerous paper".

(a) No doubt, there are various interpretations of what program verification is all about. Since I take pains to define what I have in mind within the context of my paper, it should be evident to those who read it carefully that the merits of alternative conceptions—especially those that dispense with the quest for certainty by embracing inductive procedures—are not at issue. Insofar as I am attacking Hoare's account of program verification, moreover, I am attacking a "straw man" only if Hoare's position is that of a "straw man".

(b) The role of mathematics in engineering is not unambiguous. When it is employed to describe abstract structures for which there are not supposed to be physical counterparts, it tends to qualify as *pure* mathematics. When it is employed to describe abstract structures for which there are supposed to be physical counterparts, it tends to qualify as *applied* mathematics. The position that I presented on this matter would be mistaken only if there were no difference between pure and applied mathematics, which is not the case.

(c) The authors misdescribe my conclusion, since it is not my thesis that program verification is useless and possibly harmful because it provides no certainty. As I emphasize here ["Technical Correspondence", this issue], my point is that, since program verification cannot guarantee the performance of any program, it should not be pursued in the false belief that it can—which, indeed, might be entertained in turn as the "ill-informed, irresponsible and dangerous dogma" that my paper was intended to expose.

With respect to allegations of misinformation, (i) the authors trade upon an equivocation in asserting that there are no published claims to "conclusive absolute verification" as the objective of program verification. For while the phrase "conclusive absolute verification" may not appear in just those words, the conception that those words represent is certainly to be found in publications by Hoare and Dijkstra that I cite. Moreover, (ii) while the sentence that I used in describing assembly language programming as a type that does not lend itself to program verification may be literally false, the thought that assembly language programs are no more amenable to verification procedures than are those of programs in other higher-level languages is significant and true.

Indeed, the specific avionics example that I was discussing (involving the transmission of realtime streams of data from sensors to processors) reflects a special type of programming that can be found in cruise missiles and other sophisticated systems with the capacity to reprogram themselves en route to their targets. The only technique that would permit the verification of these programs as they are generated in flight would be procedures permitting the correctness of these programs to be established as they are constructed. Perhaps the authors of this letter could volunteer to accompany these missiles on future flights in order to demonstrate that this is a type of programming that actually does lend itself to the construction of program verifications, after all.

When these authors maintain (iii) that my paper ignores a large body of work which "demonstrates the application of formal verification methods to compilers, operating systems, and computer hardware", alas, they betray a complete failure to understand the issues with which I was concerned. The compilers, operating systems and computer hardware they have in mind, no doubt, are physical entities, i.e., complex causal systems whose properties in principle cannot be ascertained independently of experience. Hence, there is a crucial ambiguity in their use of the phrase "formal verification methods", by which might be meant those of *absolute* or those of *relative* verification.

This difference, of course, is critical, since the methods of absolute verification are conclusive but cannot be applied to causal systems, while those of relative verification can be applied to causal systems but are not conclusive. This letter, I am

afraid, displays no comprehension at all of the fundamental distinctions between validity and soundness, programs and algorithms, pure and applied mathematics, abstract entities and physical systems and the like upon which the extremely important issues at stake here ultimately depend.

That there are those in the verification community who appreciate these points goes without saying. An apt illustration, I believe, is the forthcoming paper by Avra Cohn entitled, "The Notion of Proof in Hardware Verification", [*Journal of Automated Reasoning* 5, 2 (1989)], which displays a great deal of sensitivity to the basic issues at stake in my article. [No one should conclude that Cohn therefore endorses the position that I defend, however, since—to my surprise—she is one of the three Cambridge scholars who reject my analysis on peripheral grounds elsewhere in this issue of this magazine.]

The pathetic quality of thought embodied in this letter is further manifest by its authors' failure to recognize that the truth of my position does indeed undermine research results in vast areas of computer science, but only to the extent to which the significance of those findings is not properly understood. I find it difficult to believe that the authors want to seriously maintain that computer science (or any other intellectual discipline) could possibly benefit from misunderstanding or from misrepresenting the certainty of its findings. This is yet another crucial issue that these people are unable to comprehend.

If the intellectual deficiencies of the position they represent are appalling, the unwarranted abuse to which they subject the editors of this magazine is both vicious and vindictive. As an author of more than forty-five articles or reviews and the author or editor of six books, I have nothing but praise for the efforts expended on my behalf by the editors and staff of this magazine. The work of Rob Kling, in particular, in providing sympathetic criticism and in enforcing high standards was superlative. In its inexcusable intolerance and insufferable self-righteousness, this letter exemplifies the attitudes and behavior ordinarily expected from religious zealots and ideological fanatics, whose degrees of conviction invariably exceed the strength of the evidence.

James H. Fetzer
Philosophy and Humanities
University of Minnesota
Duluth, MN 55812

Reply from the Editor in Chief:
The editors welcome controversy and strive to make *Communications* an organ in which well-developed controversial ideas can be expressed. Some readers will react to such ideas with anger, others with pleasure. It is better to publish all the sides rather than to suppress issues that are uncomfortable to some. Full discussion of controversial ideas is essential to the advancement of our science.

We received many letters about Fetzer's paper. Most of the commenters brought up technical and philosophical points in the controversy. We have gathered them together in the Technical Correspondence section of this issue, (pp. 374–381) together with a response from Fetzer. The letter printed above, however, goes beyond the content of the article and questions the safety of its words and the judgment of the editors. I will respond to these criticisms here.

Upon receiving the submitted manuscript, the executive editor (Jim Maurer) could have assigned it either to the dependable computing department (John Rushby, editor) or to the social aspects department (Rob Kling, editor). Because the article claimed to be a philosophical treatment of the interaction between program verification and social processes, with emphasis on the latter, Maurer requested Kling to handle the review process. Kling accepted the assignment and obtained reviews from four well-known computer scientists. He and the reviewers assisted the author through four rounds of revision before he recommended acceptance of the manuscript. I am satisfied that the editors acted responsibly, that the article was subjected to a review more rigorous than is required by ACM policy, and that the review process was fair and professionally sound. I stand fully behind my editors.

At the bottom of the complaint of the above writers is a fundamental disagreement with Kling's decision. I take full responsibility—there is no abrogation. I am dismayed by the *ad hominem* attacks sprinkled through the letter, which detract from the thoughtful criticisms and worthwhile suggestions for more discussion. The ACM does not condone such attacks. I am publishing their letter in full as an opportunity to remind all readers of our stand.

The writers call attention to ACM's commitment to educate the public about the field through serious and well-informed discussions. The editors of *Communications* are conscientiously doing precisely that. The researchers in program verification are convinced that significant scientific progress has been made and is not widely appreciated. At the same time, many others openly question whether program verification is up to the task of contributing significantly to dependable computing systems. It is time for the two sides to communicate. It is time for us all to engage in a serious discussion to understand the strengths and weaknesses of current approaches to program verification. It is time to reveal and address the sources of people's uncertainty about the efficacy of the methods. *Communications* is available as a medium for the ongoing inquiry and will thereby contribute to the education of the community. I am pleased to report that John Rushby is organizing a special section on program verification that will set forth the best work of the field in a way that all readers can understand. This will serve as the best answer to date for the critics of program verification, and an opening for the rest of us to join the in-

quiry. Fetzer's article stimulated this!

Peter J. Denning
Editor in Chief
Communications of the ACM

Externally Speaking

Alan M. Davis describes and evaluates Real-Time Structured Analysis (SA/RT) among other methods in his article "A Comparison of Techniques for the Specification of External System Behavior" in the September 1988 issue (pp. 1098–1115). Although I agree with many of Davis's evaluations, I must take exception to his poor rating of SA/RT on "external view, not interval view." He states in defense of his judgement that "Due to its geneology (i.e., structured analysis, which is one of the worst offenders in this regard), SA/RT must also score low."

The primary notation of conventional structured analysis (the data flow diagram) is inherently neither external nor internal; it simply describes the transformation of inputs into outputs, possibly conditioned by the history of previous inputs. It is true that, as taught and written about in the late 1970s and early 1980s, structured analysis paired the notation with a top-down functional decomposition strategy for identifying model components. Not surprisingly, when applied by people who had written code, this strategy resulted in models with a software-like (or "internal," in Davis's terms) organization.

Although there are still some adherents of this early model-building style, considerable work has since been done on more effective strategies. In particular, McMenamin and Palmer [4] describe model-building guidelines based squarely on stimulus-response analysis and on the identification of external entities. Davis cites my 1986 article on SA/RT [6], which restricts itself almost entirely to notation, as a source. Had he also read the second volume of my 1985 book coauthored with Stephen J. Mellor [5], he would have discovered that we extend McMenamin and Palmer's strategy into the real-time area, and that we espouse a model building method based on responses to external stimuli and on external entity identification.

I would also like to point out that Davis's score of zero for SA/RT on "prototype generation" has been superseded by recent events. Several CASE products now permit execution of SA/RT models.

Paul T. Ward
Software Development Concepts
424 West End Ave., Apt. 11E
New York, NY 10024

I have a number of agreements and disagreements with the contents of Paul Ward's November 7 letter to the ACM Forum concerning my September 1988 *Communications* paper [1] on "A Comparision of Techniques for the Specification of External System Behavior."

I agree wholeheartedly with Ward's comment that the data flow diagram (DFD) notation itself is not inherently external or internal. However, DFD's were first used by software designers to design their software into hierarchies of software components, i.e., a pure internal view with respect to the overall system being analyzed, specified or designed. Ward claims that structured analysis was taught that way "in the late 1970s and early 1980s" and still has "some adherents." I contend that this is still a widespread problem today. Much, if not most, of the literature, tools, and courseware today on "structured analysis" continues to proliferate the problem by prescribing a methodology that says (1) perform structured analysis as a hierarchy of DFD's with associated information; (2) convert the static hierarchy of processes defined in step 1 into a dynamic hierarchy of the same processes (i.e., who calls whom). Many consider this dynamic calling hierarchy as the design and the static hierarchy to be requirements. I strongly believe that both are design; i.e., both are clearly specifications of the internal structure of the software system. What is needed is (1) to do an analysis of the problem domain (in the same vein as DeMarco's current physical and logical DFD's [3] where possible; then (2) analyze the external needs of the system's environment including the users; then (3) define an external system behavior to satisfy those needs (in the same vein as Ward's definitions of system context, external events, and system behavior [5]); and then (4) define the static hierarchy of software components and continue the design. I have captured some of these ideas in another paper. Based on Ward's letter and on reading his book, I suspect that he agrees as well. Our only disagreement here appears to be in the degree to which the problem exists today of structured analysis being used in lieu of design.

I have now read the Ward/Mellor book to which he referred in his letter. I agree with him that this work (unlike his later paper [6]) presents a methodology using DFD's which clearly maintains the analyst in the requirements analysis and specification domain, i.e., it creates a specification of the system from an external viewpoint, not an internal one. The hierarchy of DFD's that results from his methodology is created at the detailed level first. This is done by analyzing each interface and/or event independently and then merging to create more abstract views of the entire system. This results in a hierarchy in which the lowest, most detailed, level is still an external view of the system, while the highest levels are simply more abstract specifications of the same external view. In view of this, I believe that the rating of SA/RT for "external view, not internal view" on page 1113 of my paper should read "8—By following Ward's manual methodology, a pure external view can be easily maintained."

Ward's last point concerns the availability of "CASE" tools to permit execution of SA/RT models. I am quite familiar with about two

(*continued on p. 293*)

Laureate in Physics, has suggested that a Nobel Prize be established in the area of mathematics, computing, and information science. I believe this step is entirely appropriate. Since the beginning, mathematics has been the language of science. Since its invention, the computer has become the universal tool of science. Computers are used to collect and analyze data in every area of science. Computer simulation is used to validate and refine theoretical models, saving both time and expense in achieving new insights. The power of computing recently opened a whole new field of scientific endeavor—the theory of chaos. Computers provide the power to analyze the randomness in experimental data; data which traditionally has been thrown away or smoothed away. Computers chart the interactions of subatomic particles in atom smashers and chart the vastness of the universe in radio telescopes. Without computers, scientific progress would slow dramatically.

With the reservation that the title be broad enough to cover the full scope of the award (not unintentionally seeming to exclude an area by implication), the Executive Committee and the Executive Director enthusiastically support this suggestion. We sent the letter reproduced here to the Chairman of the Nobel Foundation.

Forum (*continued from p. 290*)

dozen SA and/or SA/RT-related tools. My impression is that Cadre's Teamwork, ICONIX's PRISM, Index Technology's Excelerator, IDE's Software Through Pictures, McDonnell Douglas's Prokit Workbench, Meta System's Structured Architect, Nastek's Design Aid, ProMod's ProMod, Ready Systems' CARDTools, StarSys's MacBubbles, Tektronix's TEK, and Yourdon's A/D Toolkit all support the SA/RT notation. Of these, only Software Through Pictures and CARDTools provide prototype generation. In these two cases, the prototype generation is of the user interface screens, menus, commands, etc., and is independent of the SA/RT specification. Similarly, Learmonth and Burchett's Automate Plus, Menlo Business Systems' FOUNDATION and TI's IEF provide the same type of prototyping but do not appear to support SA/RT at all. i-Logix's STATEMATE provides simulation and prototyping of the actual product's complete external behavior (i.e., beyond "just" screens) but uses Harel's hierarchical statecharts as its basis, not SA/RT. Therefore, unless Ward can supply me with specific examples, I must continue to give SA/RT a zero for the generation of prototypes from SA/RT specifications.

Alan M. Davis
BTG, Inc.
1945 Old Gallows Rd.
Vienna, VA 22180

REFERENCES

1. Davis, A. A Comparison of Techniques for the Specification of External System Behavior. *Communications of the ACM* 31, 9 (Sept. 1988), pp. 1098–1115.
2. Davis, A. A Taxonomy for the Early Stages of the Software Development Life Cycle. *Journal of Systems and Software* 8, 4 (Sept. 1988), pp. 297–311.
3. DeMarco, T. *Structured Analysis and Design Specification.* Prentice Hall, Englewood Cliffs, N.J., 1979.
4. McMenamin S., and Palmer, J. *Essential Systems Analysis.* Prentice-Hall, N.J., 1984.
5. Ward, P., and Mellor, S. *Structured Development for Real-Time Systems; Volume 2: Essential Modeling Techniques.* Prentice-Hall, Englewood Cliffs, N.J., 1985.
6. Ward, P. The Transformation Schema: An Extension of the Data Flow Diagram to Represent Control and Timing. *IEEE Transactions Software Engineering 12,* 2 (Feb. 1986), pp. 198–210.

Pearls Before . . .

I have seen a disturbing trend recently in the *Communications of the ACM.* Jon Bentley's column "Programming Pearls" hasn't appeared very recently. I really appreciated this column and is, I confess, the first thing I look for when I get a new issue of the *Communications.*

Please do what you can to encourage Bentley to continue writing the column or perhaps at least oversee and act as editor to others who would write "guest" columns. I would guess that it would be easy to find any number of readers who would agree with my views. I plan to write Bentley with similar encouragement.

Thanks for listening—hopefully the column will appear in full force soon!

Joseph W. Kmoch
Washington High School
2525 North Sherman Blvd.
Milwaukee, WI 53210-2999