SOCIAL PROCESSES AND PROOFS OF THEOREMS AND PROGRAMS[+]

Richard A. DeMillo*
Georgia Institute of Technology

Richard J. Lipton and Alan J. Perlis*
Yale University

> "I should like to ask the same question that Descartes asked. You are proposing to give a precise definition of logical correctness which is to be the same as my vague intuitive feeling for logical correctness. How do you intend to show that they are the same? . . .
>
> " . . .[the average mathematician] should not forget that his intuition is the final authority . . ."
>
> J. Barkley Rosser[++]

It is exactly those processes which mediate proofs of theorems in mathematics that *require* that program verification, as perceived by a large segment of the computer science community, is bound to fail in its primary purpose: to dramatically increase one's confidence in the correct functioning of a particular piece of software. This is a view that we have found to be shocking to many, but the key to our position rests on a relatively simple observation. A large measure of credit for the continued success and growth of *mathematics* belongs to the *social* mechanism of "proving" theorems; in *proving* programs, however, these social mechanisms are almost totally lacking.

It has been extensively argued that the art and science of programming should strive to become more like mathematics. In this paper we argue that this point of view is correct, but that the reasons usually given for it are wrong. In the first part of the paper we present our view that mathematics is, rather than a formal process, an ongoing social process and that the formalistic view of mathematics is misleading and destructive for proving software. In the second part of this paper we interpret these arguments as positive suggestions for producing reliable software and better programming tools, and we conclude that there is every reason to believe that there are social mechanisms that can make program verification more like "real" mathematics.

We have used 'proving' in two different senses in a previous paragraph. The doubly quoted sense of 'proof' refers to the common notion of informal proof used in everyday mathematics; the italicized sense of 'proof' refers to the formal notion of proof that appears in the literature on program verification.

By a *proof* we shall mean the sort of valid, step-by-step, syntactically checkable deduction as may be carried out within a consistent, formal, logical calculus such as Zermelo-Fraenkel set theory or Peano arithmetic. We have been conditioned to confuse *proofs* with "proofs." Let us call this viewpoint *monolithic*. The monolithic view is the one of Hilbert school of logic: mathematics proceeds from axioms (or postulates or hypotheses) to theorems by steps, each of which is easily justifiable from its predecessors by a fixed

---

[+]Many of the ideas reported in this paper were crystallized out of discussions held at the DOD Conference on Software Technology, July 12 & 13, 1976 in Durham, NC. We acknowledge in particular the help of J.R. Suttle who organized this conference and has been of continuing encouragement in our work.

This work was supported in part by the US Army Research Office, Grant No. DAHC04-74-G-0179, DAAG 29-76-G-0338 and by NSF Grant No. DCR74-12870.

*Authors' Addresses: R. DeMillo, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, GA 30332; R. Lipton, A. Perlis, Department of Computer Science, Yale University, New Haven, CT 06520.

[++]Logic for Mathematicians, McGraw-Hill, 1953.

allowable rule of transformation. We will begin by arguing that the views of the Hilbert school can be very misleading and that mathematics is (and must be) highly non-monolithic.

BELIEVING THEOREMS

> "Indeed, every mathematician knows that a proof has not been 'understood' if one has done nothing more than verify step by step the correctness of the deductions of which it is composed and has not tried to gain a clear insight into the ideas which have led to the construction of this particular chain of deductions in preference to every other one."
>
> N. Bourbaki[+]

Stanislaw Ulam [1] estimates that every year over 200,000 theorems are published by mathematicians. A fair number of these are subsequently contradicted or disallowed for other reasons, others are thrown into doubt, most are ignored, but a tiny fraction come to be believed by a sizeable portion of the mathematical community.

Theorems that get published are seldom the work of crackpot scientists, and those that are discredited are not always the work of incompetent mathematicians. In 1879, Kempe [2] published a "proof" of the four-color conjecture which stood for eleven years before a fatal flaw in the reasoning was uncovered by Heawood [3] (this fallacious "proof," by the way, has since been rediscovered by many well-known mathematicians who managed to convince their colleagues that they had at last solved this celebrated problem). The algebraist Britton published a monumental 300-page "proof," apparently settling an important problem in group theory [4]--unfortunately, the "proof" contains a flaw so subtle that it can only be detected by a very careful analysis of an induction involving many lemmas chained together.[++] Mathematical folklore is filled with stories of famous mathematicians whose best efforts were sometimes less than perfect--of, for instance, Legendre rising to address a mathematical congress only to pause, return his notes to his pocket, and quietly announce that the matter would require further thought.

Just increasing the number of mathematicians working on a given problem does not necessarily insure believable "proofs." Recently two independent groups of topologists (one American, the other Japanese) announced results concerning the same homotopy group (a type of object arising in homotopy theory; a branch of topology). The results turned out to be contradictory, and since both "proofs" were very complex and involved detailed symbolic and numerical calculation, it was not at all evident who had goofed. But the stakes were sufficiently high to press the issue, so the Japanese and American "proofs" were exchanged. Obviously each group was highly motivated to discover an error in the others' "proof," but neither the Japanese nor the American "proof" could be discredited; yet one was clearly incorrect. Subsequently, a third group of researchers obtained yet another "proof," this time supporting the result in favor of the Americans. The weight of evidence now being against their "proof," the Japanese retired to reconsider the issue. Notice that the *probable* truth of the theorem, at this point, is still very low compared to the far more comprehensible results we tend to regard as theorems. One of the American mathematicians was so unsettled by this experience that he has since given up mathematics for, presumably, a career in which the results of his labor can be judged more objectively.

A recent issue of *Science* [5] contains a provocative suggestion that the apparently secure notion of *mathematical truth* may be due for revision. The central issue here is not how "proofs" get believed, but rather what it is that is believed when one asserts his belief of a particular theorem. There are two relevant views that may be roughtly classified as *classical* and *probabilistic*.

The Classical View. When one believes a mathematical statement A, one believes that, in principle, there is a correct, formal *proof* of A in a suitable logical theory that (semantically) completely formalizes the Aristotelean notion of truth: "'A' is true if it says of what is that it is, and it says of which is not that it is not."

[+]The Architecture of Mathematics," American Math. Monthly, 57 (1950): 221-323.

[++]The authors are indebted to Prof. Y. Zalcstein for this observation.

Two points deserve special attention here. First, the classical view requires that we judge properties of mathematical objects in a strictly bivalent fashion: a theorem either ascribes or fails to ascribe a property to an object. Secondly, the classical view does *not* require that every informal "proof" be accompanied by its formal counterpart. In fact, the classical view does not even require that it be humanly possible to produce such a proof. There is a mathematically sound reason for allowing the gods to formalize some of our arguments. For even the most trivial mathematical theories, there are simple statements whose finite proofs are impossibly long. Albert Meyer's outstanding lecture on the history of such research [6] concludes with a striking physical interpretation of how hard it may be to prove comparatively simple mathematical statements. Suppose that we encode logical formulas as binary strings and set out to build a computer that will decide the truth of a simple set of formulas of length, say, at most 1000 bits. Then even allowing ourselves the luxury of a technology that will produce proton-sized electronic components, connecting them with infinitely thin wires, the computer we design must densely fill the entire observable universe. These precise observations concerning lengths of *proofs* coincide with our intuitions about the amount of detail embedded in "proofs." For example, we often use

> " . . . let us assume without loss of generality . . ."

and

> " . . . therefore, by renumbering if necessary . . ."

to replace enormous amounts of formal detail.

The Probabilistic View. Since long "proofs," in the classical view, can only be viewed as *probably correct*, it is perhaps reasonable to deliberately give *probabilistic demonstrations* of mathematical statements. The reasoning is that a probabilistically valid "proof" may have the dual advantage of being technically easier than a classical one and may allow mathematicians to isolate those critical ideas that give rise to uncertainty in classical "proofs"--hopefully, this process leads to a more plausible (classically) valid "proof." An illustration of this notion is Michael Rabin's algorithm for testing probable primality [7]. For very large integers N, all of the classical techniques for determining whether or not N is composite fail because of computational reasons--more time is required for the test than is left in the lifetime of the earth. Rabin's insight was that it is possible to guarantee that any such N is prime or not with vanishingly small probability of error, within a reasonable amount of time.

In view of these difficulties, how is it that mathematics has survived and has been so successful as a description of nature? Obviously, important theorems get believed by mathematicians and most of these survive. Is it possible that this process is at an end, and that mathematics in its classical form will not survive? Almost certainly not. The mechanisms that cause theorems to be believed and understood operate exceedingly well in large communities of mathematicians. Let us sketch some of these mechanisms.

First, "proofs" are widely read. An author writes a "proof" down on paper and reads it for errors. In the common case that several mathematicians collaborate on the same "proof," it is usually read by them all, by their students, and by many others. This leads to a second mechanism. "Proofs" are refereed, published and reviewed. This is a triple filter. A referee must be convinced before publication. A large audience must be convinced in a printed article. Finally, after some of the smoke has cleared, a reviewing publication such as *Mathematical Reviews* takes a more leisurely look at the "proof." Third, mathematicians talk to each other. They give symposium and colloquium talks which attempt to convince doubting (sometimes hostile) audiences of their arguments, they burst into each others' offices with news of insights for current research, and they scribble on napkins in university cafeterias and expensive restaurants. All for the sake of convincing other mathematicians. They key is that other mathematicians are inclined to listen!

What happens to a "proof" when it is believed? The most immediate transformation is probably an internalization of the "proof." This leads usually to alternative "proofs" and several versions of the same theorem. An excellent source of examples

of the transformations that can take place by this process is the little monograph by Paul Erdös and Joel Spencer [8]. The combinatorial properties discussed in [8] are usually presented in a variety of guises, each of which aids the intuition in a slightly different way. After enough internalization and alternative "proof" construction, the community seems to decide that the central concepts have an ultimate *stability*. If the various proofs begin to "feel right" and the concepts are examined from enough angles, the classical truth of the theorem becomes established.

The next activity that sometimes takes place is that the theorem is generalized in some way. If, in the passage to the generalization, the social mechanisms lead to the belief of the generalized "proof," then the probable truth of the original statement is considered established.

Finally, important theorems are *used*. They may appear as lemmas in larger "proofs"; if they do not lead to contradictions, then the confidence in the supporting lemmas of a larger "proof" is increased. Theorems can be used in other ways. For example, engineers use theorems by "plugging in" values and relying on a particular physical interpretation of the conclusion. Planes that fly and bridges that stand are impressive evidence of belief in theorems.

Believable theorems sometimes make contact with other areas of mathematics--important ones invariably do--and the successful transferral of information between distinct branches of mathematics also increases confidence in theorems. The most famous example of this sort of "technology transfer" is Hadamard's "proof" [9] of the prime number theorem. This celebrated "proof" uses connections established by Riemann and Dirichlet between theorems in complex analysis and certain asymptotic properties of the sequence of prime numbers. A more recent example of the same phenomenon is the comparatively rapid digestion of the notion of *forcing* into mathematical logic. Paul Cohen's original "proof" [10] of the independence of the Axiom of Choice and the Generalized Continuum Hypothesis from the remaining axioms of set theory was so radical that it was believed (i.e., understood) by very few logicians. Dana Scott, Robert Solovay and J. Barkley Rosser internalized Cohen's central notion of *forcing* giving it an alternative algebraic characterization [11]. Abraham Robinson and others [12] connected forcing arguments with more familiar ideas in logic, generalized the concept, and found the generalization to be immensely useful. When Cohen announced his results to the National Academy of Sciences in 1964, very few logicians believed his "proofs." By 1976, forcing arguments are routinely studied by graduate students in logic and are used as standard tools in certain areas of logic (half of the papers examined in random selection from the Journal of Symbolic Logic make essential use of forcing).

Let us summarize the main argument of this section. Working mathematicians usually do not believe "proofs" because they believe that they can translate them to a formalized logical theory to act as the ultimate arbiter. Rather, mathematical theorems get believed because they are:

1. read
2. refereed, published, and reviewed
3. discussed
4. internalized and paraphrased
5. generalized
6. used, and
7. connected with other theorems.

THE ROLE OF SIMPLICITY

We acknowledge that very few theorems are subject to *all* of the filters 1 through 7 listed above. Indeed, the degree to which a theorem is believed is largely governed by its importance, and this seems to be highly correlated with its simplicity of statement. As a general rule, the most important mathematical problems are "clean" and simple to state. An important theorem is much more likely to take the form

Theorem. Every ------ is a ------,

rather than the form

Theorem. If ---- and ---- and ---- and ----
and ----- except for special cases
i. ----------
:
xx.----------,
then unless
i. ----------, or
:
xiv.----------
every ------ that satisfies --- is a ----.

The problems that have most occupied mathematicians since Thales have been simple to state. In fact, mathematicians probably use simplicity as a *first guage* of a problem's importance, so that the decision to consider a "proof" in detail is often influenced by some slightly irrational concern--"how does the problem *feel?*" This is Occam's razor for mathematics; Einstein held that the maturity of a scientific theory could be judged by how well it could be explained to the man on the street [13]. A similar criterion applies to mathematical theories.

Of course, a judgement of simplicity must be tempered by the underlying theory. The four-color conjecture rests on such slender foundations that it can be stated with complete precision to a child. The Riemann hypothesis asks about location of the zeroes of the complex function

$$\zeta(s) = \sum_{n=1}^{\infty} n^{-s} ,$$

which may not be clear to a non-scientist, but certainly requires only the most terse introduction to mathematics. The deeper independence questions of set theory are similarly basic; can, for example, the existence of certain large objects be *proved* from the remaining axioms of set theory? Even though it requires some effort to be precise in stating such a problem, the informal statements serve the intuition very well. A simple statement of category theory may look very complicated as a formal statement in axiomatic set theory; but we are generally willing to allow for this sort of translational phenomena *because of* the social processes. Category theorists have internalized the fundamental concepts of their field in ways which are foreign to, say, set theorists. But the important concepts are simple in their domains.

The point, again, is that simple, unspecialized theorems--those that are easy to state and motivate in a specific domain--are far more likely to be read, published, and used, than idiosynchratic, narrow theorems that apply to such a paltry class of structures that no mathematician will ever again consider the class. Yet it is exactly this kind of theorem that arises in program *proving*. For *real* programs deal with *real* human activity and are thus detailed and messy, reflecting the ad-hocery and defaults which are imposed upon them by their human designers. Real programs are not simple in the sense we mean here, and the theorems that arise in trying to *prove* real programs are not simple.

BELIEVING SOFTWARE

> "The program itself is the only complete description of what the program will do."
>
> P. J. Davis[+]

The mechanisms that save "proofs" of mathematical theorems are the ones that doom *proofs* of software. Let us concentrate on program verification by the method of inductive assertions.

The kinds of specifications that usually accompany software tend to be long and complex. That is, the verification conditions that result from common input-output assertions (these are the theorems to be *proved*) seldom satisfy the simplicity criteria outlined above. Workers in the area of program correctness counter this argument by claiming that the long complex theorems to be *proved* are seldom deep; that they are, in fact, usually nothing more than extremely long chains of substitutions to be checked with the aid of simple algebraic identities. One group we know of has taken this concept to its extreme by proposing that verification conditions be *proved* by teams of "grunt mathematicians." Presumably, these are to be teams of low-level mathematical "technicians" who will--savant-style--check the verification conditions. Besides offending the sensibilities, the "grunt mathematician" concept is exactly why verification must fail.

What program *provers* have failed to recognize is that the *proofs* will be unbelievable, not because the theorems that are being *proved* are deep, but rather because the social mechanisms that we have discussed here simply will not apply to the kinds of theorems that they are *proving*. The theorems are neither simple nor basic, and the *proofs* of even very simple programs run into dozens of printed pages. Thus, the incentive for

[+] "Fidelity in Mathematical Discourse: Is One and One Really Two?" American Math. Monthly, 79 (1971):252-263.

a community to access and assimilate the "proof" of an individual piece of software is no longer present; *proofs* simply will not be read, refereed, discussed in meetings or over dinner, generalized, or used. They are too specialized for any of those activities. The sad fact is that the kind of mathematics that goes on in *proofs* of correctness is not very good.

There are, of course, algorithms that *do* get "proved." But these tend to be algorithms like FIND [14], which while generally useful, have nothing to do with software--the kind of program that is written for specialized uses and which must operate in a production programming environment! The fast pattern matching algorithm of Knuth, Morris and Pratt [15] was first implemented and *proved* by Jim Morris as part of Berkeley's text editing system. Subsequently, a system programmer who did not understand the new algorithm, pulled it from the text editor and replaced it with a much slower pattern matching routine. Presumably the system programmer "undersood" the new algorithm. The point is simple: the kinds of algorithms that get "proved" correct have nothing to do with software; give a choice between a very good algorithm with a *proof* of correctness, but which may be hard to understand, and a straightforward, unproven algorithm which an implementor believes he understands, the complex algorithm invariably loses. And, it is the complex algorithms that are most interesting and have the most chance of being subjected to the sociology of "proof."

There have been several apparent trap doors left in our argument thus far. We will now seal them off.

What about machine-aided *proof* systems? Well, totally automated *proof* systems are out of the question for a variety of reasons. In the first place, the lower bounds [16] on the length of computer proofs for mathematical theories paints a very dim picture of automated *proof* Secondly, researchers in the area do not even take seriously the possibility of totally automated proof. Ralph London [17] characterizes the "out-to-lunch" program verification system as one that can be left in an unsupervised mode to grind out the *proof* of a program. Even if an out-to-lunch system could be built (and London doubts that such a system *can* be built) to work with reasonable reliability, imagine the reaction of a programmer who inserts his 300-line input-output package and returns several hours later to find the message "VERIFIED," followed by a 20,000 line proof. We seem to be left with some kind of man-machine system that uses machine checks on complex *proofs*. This type of system, of course, does not help at all. The human *prover* still makes mistakes that will never be subjected to the social processes of "proof."

Another apparent way out is to use very high level languages to raise the intellectual content of the theorems being proven. The strategy here is to use the fact that very high level languages can deal directly with a broader range of mathematical objects to insure that the complex theorems resulting in a set of verification conditions will be truly interesting and therefore subject to the social processes we have been discussing. Unfortunately, the situation in *proofs* of very high level programs is not much changed over their lower-level counterparts. For example, the following verification condition arises in the *proof* of a Fast Fourier algorithm written in Madcap, a very high level language similar to SETL [18].

Theorem (?). If N is a power of 2, F is a complex vector and r is an integer, and if $S \in \{1,-1\}$ and $b = 2^{2\pi Si/N}$, and $C = \{2j : 0 \le j \le N/4\}$, then

i. $\langle b^j : 0 \le j \le N/4 \rangle = \langle a_r : a_r = b^{r \bmod n/2}$ and $0 < r < N/2 \rangle$

ii. $\langle j : 0 \le j \le N/2 \rangle = \langle j : j \bmod N < N/2$ and $0 \le j < N \rangle$

iii. if $F = \langle f_r = x_r = x_r$ and $0 \le r < N \rangle$ then

$$\langle \triangleright(F_A + F_{A*}) \triangleright a(F_A - F_{A*}) \rangle =$$

$$\langle f_r : f_r = \sum_{k_1 \in R_r} b^{(k_1 | r\, 2^{r-1}|) \bmod N} \times k_1$$

and $R_r = \{j : (j-r) \equiv 0 \bmod N/2\} \rangle$

At some level, the transition between specification and program *must* be left unformalized; this follows by two compelling observations. First, the data sets on which programs operate are, in general, enormously complex objects. In computing the payroll for the French National Railroad, for example, we find that each employee has more than 3,000 pay rates, incorporating, for instance, the time his train was traveling uphill versus downhill. Similarly, the input assertions for many numerical algorithms are not even formulatable--certainly they cannot be formalized [19]. The second observation seems to us to be the most telling, however. The *purpose* of a program is an informal, often unstated, criterion and the transition from informal to formal objects must forever be unformalized, lest we be caught in the paradox of assuming the formalization of an object we know only informally.

CONSTRUCTING BELIEVABLE SOFTWARE

> "One of the chief duties of the mathematician in acting as an adviser to scientists . . . is to discourage them from expecting too much of mathematics."
>
> N. Wiener[+]

We can combine the arguments of previous sections with the intuition gathered from a century (or so) of experience in dealing with mathematics in a symbolic, formalistic way. Since "symbols" can be written and moved about with negligible expenditure of energy, it is tempting to leap to the conclusion that *anything* is possible in the symbolic realm. This is the lesson of computability theory (viz., solvable problems vs. unsolvable problems), and also the lesson of complexity theory (viz., solvable problems vs. feasibly solvable problems): physics does not suddenly break down at this level of human activity. It is no more possible to construct *symbolic* structures without using energy than it is possible to construct *material* structures for free. But if symbols and material objects are to be identified in this way, then we should perhaps pay special attention to the way material artifacts are *engineered*, since we might expect that, in principle, the same limitations apply. Before we return to our theme of relating *proofs* to believable software, let us briefly digress to consider the role of engineering principles.

It appears to us that the engineering professions have been remarkably adept at reconciling the normal cultural desires for the creation of reliable structures with two human failings:

(1) human beings cannot create perfect mechanisms,

(2) human beings tend to plunge into activities before they are understood.

Surprisingly, this reconciliation is accomplished by virtue of the same social mechanisms we have been extolling. First, in mature engineering disciplines, *"reliable" never means "perfect"* in the monolithic, classical notion of "perfection." There are simply no *proofs* (or "proofs") that bridges stand, that airplanes fly, or that power systems deliver electricity. Rather, engineers set probable limits of failure, relying on other design criteria to place these limits well above the conditions likely to be encountered in practice. It is perhaps "symbol chauvinism" which suggests to computer scientists that (a) our structures are so much more important than material structures that they should be *perfect*, (b) our available energy is, in practice, sufficient to certify programs as perfect. We now see connections with mathematics: the probabalistic view of mathematical truth is closely allied to the engineering notion of reliable structure. This suggests that we should make more serious attempts to treat software reliability in a setting distinct from program proving, since both merely establish expected limits of confidence.

Second, engineers tend to counteract the human tendencies to try "untested" designs by limiting the amount and kind of innovation in designs. This suggests that engineers tend to "recycle" designs and to thus subject them to some of the same social processes as are present in mathematics.

Is the situation as bleak as we suggest? There are almost certainly mechanisms which make software more like mathematics in the *real* sense--i.e., which elevate software to the level at which the social processes described above operate. Many of these mechanisms are just applications of

[+] I Am a Mathematician, The Later Life of a Prodigy, MIT Press, 1964.

unselfconscious engineering design principles (e.g., seek reliability within economic limits, channel innovation into constructive paths by reusing previously successful designs), while many are straightforward extensions of our thoughts on what causes mathematics to be, in the main, successful.

The *desire* to "prove" programs correct is constructive and valuable; the monolithic view of "proof" has succeeded in hiding the primary benefits to accrue from program proof. What we have done is to point out a basic difficulty with any large-scale attempt to *prove* programs correct. However, "proofs" of software may be coupled with engineering techniques and may yet succeed. Indeed there is considerable evidence that the verification of software can be subjected to the key social processes that allow us to understand and believe mathematical "proofs"; i.e., there are mechanisms with which to make software more like real mathematics.

The first mechanism which may be exploited is the device of creating *general* structures, the various *instances* of which become more reliable due to the believability of the general structures. In particular, the Alphard notion of *form* appears to be exactly the sort of mechanism that encourages this sort of activity: " . . . a library of useful abstractions will develop, and . . . programmers will simply not have to program as much to get a new system . . ." [20]. This notion has appeared in recent years in several incarnations: Donald Knuth's insistence [21] in the creation and understanding of generally useful algorithms represents such a trend; the team programming methodology of [22] is another example of attempts to *export* software explicitly for the purpose of exposing it to social processes. Reusability of effective designs causes a wider community to examine the programming tools to be most commonly used.

A second mechanism is the notion of high-level "proof." We have basically argued that a *proof* of correctness will not be correct. However, if "proofs" are correct, they must be generally *useful* in order that information may be extracted from them. The metaphor here is that algorithms yield ideas about other algorithms and are thus exportable, while low-level *proofs* may not be exported. The languages of "proof" must be carried out in high level structures--certainly at higher levels than programming is done. Consider that theorems which are subjected to social mechanisms are exported to a larger community, and if they are created in high-level languages then instantiations to programs become feasible and consistent with our other mechanisms.

The concept of verifiably correct software has been with us too long (as a goal) for it to be easily displaced; in fact, as a goal it offers considerable impetus for research in computer science, providing for example a ladder to even more "exportable" programming tools and methodologies. For the *practice* of programming, however, the notion of program correctness has overshadowed that of program believability. Like good scientists we should not confuse our mathematical models with reality--and correctness (like *proof*) is nothing but a model of "believability."

Relatively little philosophical discussion (see e.g. [23]) has gone into determining alternative methodologies, developing technical tools, and creating research paradigms for other styles of reliability issues.

We feel that the growth of complex software systems has not yet been discussed using metaphors that are sufficiently precise for development of alternative views of software believability. When another view of "reliable design" arises that more fully exploits the social mechanisms which have proved useful in the development of mathematics--and for which we have argued in this paper--we expect to see technical developments follow in this alternative track.

References

1. S. Ulam, *Adventures of a Mathematician*, Scribner, 1975.

2. A. B. Kempe, "On the Geographical Problem of the Four Colors," *Amer. J. Mathematics*, V. 2, (1879), pp. 193-200.

3. P. J. Heawood, "Map Coloring Theorems," *Quarterly J. Math. Oxford Ser.*, V. 24, (1890), pp. 322-339

4. J. L. Britton, "The Existence of Infinite Burnside Groups," in W. W. Boone, F. B. Cannonito and R. C. Lynden (eds.), *World Problems*, North-Holland, (1973), pp. 67-348.

5. G. Bari Kolata, "Mathematical Proof: The Genesis of a Reasonable Doubt," *Science*, Vol. 192, (1976), pp. 989-990.

6. A. Meyer, "The Inherent Computational Complexity of Theories of Ordered Sets: A Brief Survey," *Proc. International Congress of Mathematicians*, August 1974.

7. M. O. Rabin, private communication.

8. P. Erdös and J. Spencer, *Probabilistic Methods in Combinatorics*, Academic Press (1974).

9. J. Hadamard, "Sur la Distribution des Zeros de la Fonction $\zeta(s)$ et ses Consequences Arithmetiques," *Bull. Soc. Math de France*, V. 24, (1896), pp. 199-220.

10. P. J. Cohen, "The Independence of the Continuum Hypothesis," *Proc. Nat. Acad. Sci., USA*, Part I, V. 50, pp. 1143-1148 (1963), Part II, V. 51, pp. 105-110, (1964).

11. J. B. Rosser, *Simplified Independence Proofs*, Academic Press, (1971).

12. A Robinson, "Infinite Forcing in Model Theory," in J. E. Fenstad (ed.) *Proc. Second Scandinavian Logic Symposium*, North-Holland, (1971), pp. 317-340.

13. R. W. Clark, *Einstein: The Life and Times*, World (1971).

14. C. A. R. Hoare, "Algorithm 65: FIND," *C. A. C. M.*, V. 4(7), (1961), pp. 321+.

15. D. Knuth, J. Morris, V. Pratt, "Fast Pattern Matching in Strings," to appear in *SIAM J. Computing*.

16. L. Stockmeyer, "The Complexity of Decision Problems in Automata Theory and Logic," MIT Thesis, (1974).

17. R. London, private communication.

18. J. Schwartz, "On Programming," NYU Courant Report, 1973. (See also "A Comparison of MADCAP and SETL" by J. B. Morris, Los Alamos Sci. Lab, University of California, Los Alamos, NM, 1973.)

19. J. R. Rice, private communication.

20. W. A. Wulf, R. L. London, and M. Shaw, "Abstraction and Verification in Alphard," Carnegie-Mellon University, Computer Science Department Research Report, July 1976.

21. D. E. Knuth, *The Art of Computer Programming*, Vol. I (1969), Vol. II (1969), Vol III (1975), Addison-Wesley.

22. F. T. Baker, "Chief Programmer Team Management of Production Programming," *IBM Systems Journal*, Vol. 11, No. 1, (1972), pp. 56-73.

23. S. L. Gerhart and L. Yelowitz, "Observations of Fallibility in Applications of Modern Programming Methodologies," *IEEE Transactions in Software Engineering*, Vol. SE-2, No. 3 (September 1976), pp. 195-207.