TOWARDS A DISCIPLINE FOR DEVELOPING VERIFIED SOFTWARE

William M. Farmer
Dale M. Johnson
F. Javier Thayer

The MITRE Corporation
Bedford, Massachusetts

Technical Report MTP-261

August 1986

Abstract

In this paper the formal verification of computer systems and software is viewed as an endeavor in applied mathematics. It is argued that a formal verification should consist of three separate but interacting processes: a modelling process, a theorem proving process, and a review and acceptance process. Suggestions are made for improving the development of these processes. Taken together, they outline a proposed discipline for the development of verified software. The ideas presented were principally, though not exclusively, motivated by the authors' work in reviewing the design verification of the Restricted Access Processor (RAP). Examples are drawn from the RAP verification to support our suggestions for improving formal verification.

This paper will be presented at the 9th National Computer Security Conference, September 15–18, 1986 at the National Bureau of Standards, Gaithersburg, Maryland.

Acknowledgments

We gratefully acknowledge the support of the Rome Air Development Center under contract number F19628-86-C-0001 during the preparation of this paper. We would also like to thank E. Bensley, J. Millen, P. Tasker, and J. Williams, who offered guidance in the formulation of the ideas presented in this paper.

1. INTRODUCTION

The main purpose of this paper is to propose a discipline for the development of verified software. Our comments in this paper are motivated in part by our recent experience reviewing the design verification of the Restricted Access Processor (RAP) (cf. [3], [7]). We also draw some examples from the RAP verification to support our views. The paper attempts to describe verification as an endeavor in applied mathematics. Though this viewpoint is not completely new (cf. [1], [10]) and might even be regarded as the obvious one to take, from our review experience we are led to believe that the exact consequences of taking this view are not fully and clearly understood. In particular, perceiving verification as applied mathematics requires a clear differentiation between the following two processes:

(1) *Establishing formal mathematical models of natural-language requirements or specifications.* In the case of design verification for secure systems, these models are usually called the formal security model and the (formal) top level specification (TLS). We refer to this as the modelling process, which in our view is perhaps the most critical part of the verification, yet it is apparently the least understood.

(2) *Using mathematical techniques to reason about the formal models obtained by the modelling process.* In the design verification of the RAP, this reduced to proving formally that the TLS satisfied the formal security model. We call this the theorem proving process.

We realized in our review of the RAP that the distinction between the modelling process and the theorem proving process is especially important from the reviewer's perspective, since the tasks involved in each of these processes are to be understood and judged in very different ways. Yet while these processes are distinct, it is inevitable (and definitely beneficial for the verification) that they will interact with one another.

In addition to these two processes we believe that it is useful, in analogy with similar validation processes occurring in the mathematical sciences, to include a third interacting process as well:

(3) *Reviewing and accepting the verification.* Generally, this means ascertaining that the verification satisfies requirements agreed upon by the customer and the verifier. Requirements may include, for example, the use of automated tools. Another relevant and more important requirement is soundness of the logical principles used in the verification. In

our view, part of the review process should allow for interaction between the reviewers and the verifiers.

2. THE MODELLING PROCESS

Mathematical modelling is a crucial process in the task of formal verification. For example, in the verification of secure systems, a formal security model for the security policy is constructed or, in some cases, provided (e.g., the Bell-LaPadula security model). In this section we discuss various aspects of the modelling needed in verification, using our findings from reviewing the design verification of the RAP to develop examples and special points.

In general, models provide a description of some real-world phenomenon. By "phenomenon" we mean something very general. A phenomenon may be a process or a system; even a natural language description of a system or process is a phenomenon. A fundamental aim for constructing a model is to allow the use of formal deductive techniques on the model to gain some new information or conclude something about the phenomenon. This aim requires that models be comprehensive in the sense that they contain all information necessary for applying these formal techniques. It should be emphasized that formal deduction is clearly distinguished from other forms of evidence, such as empirical evidence, so that the requirement of comprehensiveness is quite important.

Another highly significant aim of the modelling process is to make the phenomenon intelligible to others. In order to achieve this, the models have to be clear and thoroughly explained. Models that resemble computer code do not meet these goals.

The process of building useful models is one of the most difficult in all of science. The model-builder has first to select carefully the tools and techniques from mathematics that seem the most appropriate for presenting the model. Most critically, he must decide how to represent elements of the phenomenon with mathematical constructs.

The model should be a clear portrayal of the phenomenon, so that it can be accepted. Acceptance of a model is based on the collective experience of the researchers doing modelling and also on subjective factors, such as mathematical taste. A precept that is universally true is that models are meant to be understood. Questions of style and format are not to be brushed aside as technically irrelevant. Moreover, specific sciences have developed special methodologies for validating models. These methodologies generally rely on experimentation and statistical sampling; even some form of disciplined introspection may be used.

Unfortunately, no modelling methodology has, to our knowledge, been successfully developed for the young science of verification. The lack of a methodology makes modelling even more difficult.

We must emphasize that by the term "modelling" we do not refer exclusively to the construction of the formal security model, though this construction is a significant part of the modelling process in some verifications such as the RAP. We must also include the writing of the formal top level specification (TLS) as a part of the modelling of the system.

The modelling required for the design verification of the RAP is typical of that needed in verification. We can identify the parts of the modelling process in general as follows:

(1) Selection of a methodology for the verification, such as the Hierarchical Development Methodology (HDM) [6]. This selection has significant implications for the verification. HDM was used for the RAP verification. (Other possible methodologies are Gypsy and Formal Development Methodology. Also, an Enhanced HDM has recently been released.)

(2) Construction of a formal security model derived from a security policy. The purpose of this model is to formalize natural language requirements concerning security. As part of the modelling process, the functioning and adequacy of the model should be explained. In the case of the RAP the formal security model was derived from an Air Force security policy. Some explanation of the functioning of the RAP accompanied the formal model.

(3) Characterization of the design by writing a formal top level specification. The TLS was a large and significant part of the modelling for the RAP verification.

(4) Generation of conjectures during the modelling process, which then need to be investigated during the theorem proving process. In the case of the RAP verification we found that it was necessary to make the exact nature of these conjectures as clear as possible.

(5) Justification of the decisions taken in steps (1)–(4), in order to advance the (implicit or explicit) claim that the modelling is adequate. Unfortunately, it is often the case that this aspect of the modelling is not adequately carried out.

We have prepared some suggestions for improving the modelling process in verification. These were in part prompted by our examination of the modelling done for the RAP verification. In looking at the modelling in the RAP verification we were particularly concerned with the need for adequacy, comprehensiveness, intelligibility, and simplicity. These are highly desirable features that should be considered in the modelling done in verification. Our suggestions are intended to help verifiers make these features a part of their verifications.

(1) Explain and carefully justify fundamental decisions about the modelling.

Throughout a verification project, but more especially near the beginning, the verifiers should attentively think about the modelling needed or being done. Decisions about the modelling should be carefully documented and justified. At the outset of the RAP verification, certain modelling ideas had to be established, i.e., decisions had to be taken about how to portray the actual RAP (the reality in this case) as a mathematical model. The RAP is a processor guarding the data link between the Network Control Center (NCC) and the NASA Communications Message Switching System (MSS). Its purpose is to prevent uncleared users from accessing classified information or facilities available through the NCC. A security policy had been provided by the Air Force and the architecture of the system hardware had been developed. The basic modelling problem was to find mathematical constructs that reflected the chosen architecture of the hardware and the intended security of the system. The verifiers decided to model the operation of the RAP conceptually as sequences of events that passed over a (conceptual) security perimeter. The selection of a particular security perimeter and a particular way to portray the flow of events is a fundamental modelling decision. Verifiers must not only understand the nature of this basic decision about modelling, but be able to justify it as well.

(2) Give broad explanations of the models and, if possible, key information about the process by which they were derived.

Broad explanations of entire models are extremely helpful to a reader or reviewer. Moreover, information about the genesis of the models can illuminate the models themselves. During the construction of a formal model various modelling decisions are made. These are reflected in the final constructed model, but often in obscure ways. The key information about the construction of the models should be preserved in an abbreviated form in the documentation.

In the case of the RAP we found that the documentation, though substantial, could have contained more information about the ideas behind the actual construction of the two main models, the formal security model and the formal top level specification. To take a simple example, we found that one very large definition in the formal security model could be reduced to a pair of tables. Once these tables were constructed, the formal definition became much easier to understand.

(3) Choose a methodology that is adequate to formalize the notions that need to be modelled.

This choice is a very difficult matter. One wants to choose an adequate methodology for a verification, but at present there are only a few from which to choose. A fundamental modelling decision taken for the RAP verification was to adopt the Hierarchical Development Methodology (HDM) [6]. This decision had many implications for the modelling process. Most notably it implied the adoption of the sequential state machine model, a basic part of HDM, in the modelling. For this general model concurrency is not so easily taken into account. Hence, it is at least questionable whether this sequential model is adequate to deal with the reality of the RAP. Arguments ought to be given for the (implicit) claim that the chosen methodology is adequate.

(4) Try to develop a formal model that has a direct and clearly understood relation to the English policy statement or English requirements specification.

The formal security model was a very significant part of the modelling for the RAP [2]. The purpose of this formal model was to capture the Air Force security policy in a succinct and correct way. The model was based on event histories and was written in the specification language SYSPECIAL, a variant of the SPECIAL of HDM. The heart of the model consists of a hierarchy of definitions of predicates on event histories, with a single predicate (MBPS_OK) at the top of the hierarchy representing the desired security invariant.

In order to facilitate the construction of the formal model a shortened form of the Air Force security policy was developed, called the "derived security policy". This was undoubtedly a great help in constructing the formal security model, in particular, in seeing how the Air Force policy should be related to the model. The derived security policy is a terse English-language statement of the security requirements that is closely related to the formal model; in many instances there are direct (one-to-one) correspondences between words of the derived policy and functions or predicates of the model. The model would have been even better if it could have been a simpler formalization of the policy with more direct correspondence between policy and model. However, formalization is a very difficult art.

In general, formal models should be made as simple as possible and the relation to English-language requirements specifications should be made as clear as possible through informal explanations in the documentation and perhaps through the construction of derived policy statements. One can see the advantages of a derived tersely-worded security policy statement in the case of the RAP. Generally an English statement or specification should be as simple as possible.

(5) Definitions in a model should have a hierarchical structure and this structure should be presented fully.

The definitions of the formal security model for the RAP were arranged in a hierarchy. This arrangement is certainly a good one. It is one that can be used to good effect in modelling in verification. However, it is useful to have as much information as possible about the hierarchy. The hierarchy effectively indicates a “flow” from the most general to the least general, revealing a great deal about the structure of the model. The verifiers of the RAP might have given more information about their hierarchy. Their diagram of dependencies in the hierarchy was reduced to a brief summary in the documents. A general explanation of a hierarchy of definitions can be very helpful as a supplement to the explanations of the individual definitions found in the hierarchy.

(6) Use nonprocedural forms of expression.

In our attempt to understand the formal security model for the RAP we were led to develop our own intermediate mathematical model and explanation. We found that it was very helpful to remove the recursions from the basic definitions in the formal model and state these with the aid of quantifiers and logic. The formal model as given effectively had a mix of procedural description (the recursions) and strict logical description. This mix was not always conducive to providing a direct and clear exposition of the model.

It was only by constructing our own intermediate mathematical model for the given formal security model that we could begin to see the relation between the English security policy and the given formal model. This intermediate model allowed us eventually to decide that for the most part the policy was correctly reflected in the given formal model.

The modelling done in constructing the TLS for the RAP [8] had some special problems, partly associated with adoption of HDM. We found the TLS at times quite difficult to understand. We have a number of suggestions for improvement in writing these kinds of specifications. In the following we assume an understanding of the terminology of HDM.

(7) Use homogeneous data types, whenever possible.

To avoid confusion, use homogeneous data types. If for some reason the use of homogeneous data types is impossible, pending data types should be considered.

(8) Describe state transitions as simply as possible.

The effects of O-functions on individual V-functions should be easily understood. Ideally, the effects of an O-function should be of the simple form:

$$V = F(W),$$

where V, W are V-functions and F is some simple function. The functionality of O-functions of this sort is manifestly clear to a reader.

(9) Provide an information flow diagram.

The flow of information between V-functions should be clear. Ideally, one should be able to represent the flow induced by an O-function as a directed graph. The nodes of this graph correspond to V-functions and the edges correspond to assignment statements. The graph provides a clear understanding of the general architecture of the TLS.

(10) Give adequate explanations of the relations among the models.

This suggestion brings up the issue of comprehensiveness of the models. One of the goals of the RAP verification was to prove that the TLS satisfied the security requirement or predicate formalized in the formal security model. This formalized security requirement is essentially a predicate on finite sequences of "events". Since sequences of events do not constitute part of the state of the TLS state machine, it is not clear how to interpret the assertion that the TLS satisfies the security predicate.

An interpretation can be made by associating event histories with certain sequences of O-functions or OV-functions. These sequences are the possible execution sequences of the TLS state machine. The assertion that the TLS satisfies the model then means essentially that for every execution sequence, its associated event history satisfies the formal security predicate. However, this correspondence is not a part of either the formal security model or the TLS. How one associates an event history to an execution sequence is a problem of modelling. In the case of the RAP, however, event histories were introduced as part of the theorem proving stage in a manner which seemed to suggest that one could prove that the association chosen was the correct one. Nevertheless, this association was especially problematical, since crucial assumptions about concurrency were implicitly made. In general, the omission of the relation between models means that the modelling process is not as comprehensive as it should be.

3. THE THEOREM PROVING PROCESS

The second process of formal verification is the theorem proving process. In this process mathematical proofs of the conjectures formulated during the modelling process are constructed and analyzed. These proofs serve two functions:

(1) To determine whether the conjectures formulated during the modelling process are true.

(2) To clarify the meaning of these conjectures.

The first function is well understood. No doubt it is the part of formal verification that has received the most attention. The second function is often ignored. It is, however, essential for identifying inappropriate or incorrectly formulated conjectures, and thus spotting apparent errors in the modelling process.

Unlike the modelling process, the theorem proving process is a completely mathematical endeavor. Proofs of theorems are developed in a well-defined mathematical theory (created by the modelling process), in which there is no direct mention of the real-world application.

As part of a verification, mathematical proofs can provide a level of assurance for the correctness of a conjecture that is not obtainable by traditional means of software testing. Nevertheless, mathematical proofs are not infallible. Their validity must be ultimately grounded in some kind of critical process.

In mathematics, this critical process occurs within the community of research workers.

Because proofs used in formal verification tend to be long and complicated, verifiers usually try to construct them with the aid of machines (theorem provers, proof checkers, simplifiers, etc.). This approach is certainly good and probably necessary. However, without care it can become an obstruction to the theorem proving process, leading to results such as the following:

(1) Theorems are proved without being clearly understood.

(2) Opaque calculation is given instead of careful argument.

(3) Proof analysis is given less emphasis than proof construction.

(4) Conceptual simplification is overlooked.

(5) Errors in the formulation of conjectures are not discovered.

(6) The fallibility of proofs is forgotten.

If verifiers are to construct good proofs with the assistance of machines, they need to have a clear understanding of what a verification proof should be. We feel that there are six basic goals that a verification proof should attempt to achieve:

(1) A verification proof should clearly state the theorem it purports to prove.

To any mathematician this goal is so obvious that it hardly needs stating. Nevertheless, achieving this goal is essential to any good proof. A proof's value is diminished in proportion to the lack of clarity in the statement of the theorem.

(2) A verification proof should increase one's confidence in the truth of the theorem.

This is clearly the major goal of any proof. It is important to remember that proofs can never give an absolute guarantee of correctness.

(3) A verification proof should be rigorous.

The one thing that separates formal verification from traditional ways of testing software and computer systems is that formal verification attempts to show something is correct with a rigorous proof. A rigorous proof strives to use only well-defined concepts and to have no loose ends. Nothing is ignored or left to chance.

(4) A verification proof should clarify the meaning of the theorem.

It is a rare luxury to begin proving a conjecture that is correctly formulated. This is true in general mathematics as well as in formal verification. Thus it is very desirable that the process of proving a conjecture helps to correct the statement of the conjecture itself. The ideal process goes like this: a (partial) proof of the conjecture is constructed, it is analyzed, the conjecture is modified, and the process is begun again. The process ends when one is satisfied that a complete proof of an appropriate and correct conjecture has been obtained. (Cf. [5] for further discussion of this "dialectical" process.) In order for this process to be successful, it is necessary that verification proofs elucidate the meaning of the theorems they prove. This cannot be done by proofs consisting merely of a long series of opaque logical calculations.

(5) A verification proof should be maintainable.

Software and computer systems need to be modified virtually on a continuous basis. Hence, verification proofs should be modified whenever the things they verify are modified. In other words, verification proofs should be maintainable just as computer systems should be maintainable.

(6) A verification proof should be machine checkable.

Verification proofs tend to be long and complicated. One cannot expect to check them by hand without making mistakes. It is reasonable to expect that many of these mistakes would not occur when a proof is machine checked. Although it is desirable that a verification be machine checkable, it is not necessary that a verification proof be machine generated. (Of course, it is often useful to construct parts of a verification proof with the aid of a machine.)

The state of the art of verification proofs falls significantly short of these six goals. We believe that this is due in part to an inadequate understanding by verifiers of what proofs should be and what role machines should play in proof construction.

To help illustrate how real verification proofs satisfy (and fail to satisfy) the goals we have stated above, we shall briefly examine the verification proof for the design of the RAP. The RAP verification proof is a good example to consider because, although it certainly is one of the best large-scale verification proofs produced to date, it exhibits some of the deficiencies that commonly plague verification proofs.

The principal theorem of the design verification of the RAP can be stated as follows:

Theorem. Every implementation of the Top Level Specification (TLS) of the RAP satisfies the requirements formulated in the formal security model.

This theorem is only stated informally in the RAP Verification Results Report [9] and its mathematical meaning is not explicated at all.

The proof of the theorem breaks up into two parts:

Part A. The proof that the theorem holds if the assertions of an Augmented TLS (ATLS) are invariants of the ATLS.

Part B. The proof that the assertions of the ATLS are invariants of the ATLS.

These two parts are handled very differently. Part A of the proof is an informal mathematical argument, which is given very little attention relative to Part B. Moreover, the argument is flawed because part of the modelling process (the construction of the ATLS from the TLS) is mixed up with it.

Part B is of a completely different nature from Part A. It is essentially a series of 36 very detailed formal deductions. The formal deductions are not actually given in the Verification Results Report [9]; instead logs are given of the theorem prover commands used to construct the deductions.

Part B does a reasonably good job of satisfying the goals of rigor (3) and machine checkability (6). Its success results from being a formal proof constructed (and checked) with the use of a machine. Since the logs are modifiable and reusable, part B also contributes to the goal of maintainability (5). The logs, however, are opaque. They do not help one to understand the subtheorems they prove, nor do they communicate the mathematical meaning of the deductions.

The lack of perspicuity in the formal deductions means that one's confidence in the claim that the ATLS assertions are invariants of the ATLS is almost purely a matter of faith in the MUSE system, the theorem proving system used to construct the formal deductions. The MUSE system [4] was developed by Sytek, Inc. It is a competent and, in many ways, admirable theorem proving system. However, although the MUSE system appears to work correctly, it has not been formally verified (like all such systems) and it is relatively untested, having thus far been used on only one large project. As long as the MUSE system itself is not verified, genuine confidence in it can only come after it has been used by several different parties on several different projects.

In summary, the RAP design verification proof is composed of an informal part and a part constructed with the aid of a machine. The first part received only cursory attention. The second part was carried out in great detail, but with too much reliance on automated reasoning tools. Although the RAP verification proof is successful in several ways, it illustrates some common shortcomings of verification proofs:

(1) Insufficient attention is paid to the informal parts of the proof.

(2) Justification for modelling decisions is presented as part of the verification proof.

(3) Formal deductions are not presented in an understandable form.

(4) Too much reliance is placed on unverified theorem proving tools.

We finish our discussion of the theorem proving process by giving five suggestions for constructing good verification proofs.

(1) Use hierarchical construction.

To be readable and understandable, long proofs must be constructed in a hierarchical manner. This is eminently true for verification proofs, which tend to be oppressively long and full of minute details. The components of a hierarchical proof should be subproofs of the form

$$\text{by the argument } A,\ C \text{ follows from } H_1,\ldots,H_n.$$

At the top level C is the conjecture that is proved, and at the bottom level the H_j's are the hypotheses which are being assumed or are trivially true.

The crucial parts of the proof — the "idea" of the proof — should be in the arguments at the top of the hierarchy, and the tedious details of the proof should be at the bottom. One can read a proof of this form part way down and be confident that the basic idea of the proof and, hence, the basic idea of the theorem are correct, even though there could be minor problems with the details of the proof and the theorem.

(2) Use modular construction.

Mathematicians have been using modular construction in proofs for centuries, and modular construction is considered part of good programming. It is well understood why modular construction is desirable, even necessary, in mathematical proofs and computer programs. To a large extent the whole enterprise of formal verification rests on one's ability to develop methods of modularity. In conjunction with hierarchical construction, modular construction is an excellent way to satisfy the goal of maintainability.

By making use of proof parts that have been used many times (such as fundamental lemmas), one's doubt in a verification proof can be directed to a few specific aspects of the proof. This helps to increase the reviewer's confidence in the proof by allowing him to concentrate only on what is new. There is some use of modularity in the the RAP verification proof with the use of the theorem prover command logs.

(3) Identify all premises of the proof.

A good proof of any kind should clearly identify all the premises used and assumed in it. Incorrect proofs often result from the use of hidden assumptions that are not valid. Following this suggestion should help in attaining all but the last verification goal. A clear statement of the theorem includes the premises that are assumed (goal 1). In a well-constructed proof, the aspects of the proof which might be questionable are concentrated in the premises of the proof (goal 2). A precise list of premises is a requirement of a rigorous proof (goal 3). Knowing the premises of a proof increases one's understanding of the theorem (goal 4). The premises of a proof identify the conditions under which the proof is valid and can be used again (goal 5).

(4) Use calculations carefully.

Formal verification has been rejected by some [1] as a means of testing the reliability of software. One of the principal reasons for this is that all too often verification proofs contain massive amounts of opaque logical calculations. Calculation is certainly a very valuable aspect of mathematical reasoning. However, if one is going to use calculations in a fundamental way in a verification proof, one needs assurances that the calculations are performed correctly. Until one has these assurances, it is very dangerous to accept the results of calculation without closely examining what was done.

Calculations can be incorrect and, when calculations are opaque (such as arithmetic is in a digital computer), there is no way of easily detecting errors. Complicated calculations should be used in a verification proof only when there is a very high assurance that they will be correct. For example, it is appropriate to use arithmetic calculations performed by a good digital computer or simplifications performed by a well-tested and very reliable simplifier. As prominently mentioned by DeMillo et al. in [1], virtually all formal verifications to date suffer in some degree from excessive reliance on formal logical calculation.

(5) Use an expressive high level language.

It is clear that verifiers can greatly benefit from the use of machines to assist in the development of verification proofs. Using machines necessitates working with formal languages. It is exceedingly hard to get a machine to handle formal languages that have the expressibility of the informal languages used by mathematicians. Consequently, verifiers have usually been tempted into using very simple formal languages. This approach keeps the proof development at such a low level that the verifier and reviewer become lost in a heap of trivia.

Relief can only come with the use of expressive high level languages. Languages of this type are difficult to develop and difficult to program a machine to use, but they allow human beings to think naturally and to make use of the richness of modern mathematics.

4. THE REVIEW AND ACCEPTANCE PROCESS

In mathematics the validation or acceptance of a new result is the outcome of a complex interactive process involving the author, the interested community, and to a lesser extent a technical reviewer appointed by the editor of a journal. Based on our experience of reviewing the RAP, we now discuss how the process of validation ought to occur in the verification of design or program correctness. We believe that many useful analogies between the validation processes in mathematics and in verification can be made. However, while these analogies exist, we still think that the two processes have important differences.

In mathematics (or in other areas such as mathematical economics or mathematical physics), workers in each area of research try to build on or improve published results. They are strongly motivated to understand these results and make sure that they are correct. Mathematical papers are normally structured in a way that permits understanding at many different levels. For example, a specialist reader can take a cursory look at a good mathematical paper and still get some notion of the paper's contents. There are also aesthetic reasons for reading mathematical papers. Researchers read technical papers, not only because they can use the results in their own work, but also because reading them is a pleasurable experience. Generally, papers that are not well written are not immediately received, and the results they claim take longer to be accepted by the community of research workers. Insuring that their papers are read is a powerful reason for authors to write clear as well as interesting papers.

In contrast to the situation in the other mathematical sciences, very few people read verifications of large programs or systems. Most of the reasons that exist for reading research papers do not exist for reading verification proofs. Verification proofs are tedious and rarely provide a basis for further research in the same way as mathematical proofs. Cursory readings of verification proofs provide absolutely no insight. In short, verification proofs are written with no intention of attracting readers. Generally, new software has been accepted exclusively on the claims of developers. In the best of circumstances (as was the case for the RAP) the design or code undergoes some sort of independent review process. Even in the case of the RAP, the review process for the design verification was not explicitly discussed in the original verification plan, despite the fact that a review process for code development was carefully established.

As we have argued above, the existence of proofs, even automated ones, is in itself no guarantee of correctness. Proofs have to be submitted to a thorough review process in much the same way as in mathematics. Since it is unlikely that verifications will attract interested and critical readers, only a formal review process by appointed referees seems to be feasible. This review process should be considered an integral part of the verification effort.

It is our belief that the reviewers should be regarded as the main audience for a verification proof. If the purpose of such a proof is to persuade any potential doubter, then at least the reviewers must be convinced of the correctness of the verification proof. A verification effort which fails to satisfy this condition cannot be considered a proof in any reasonable sense. In order to achieve these goals, the following two conditions at least should be met:

(1) Any formal models used in the verification must be understandable without undue effort. Specific guidelines for clarity of specifications should exist to aid specification writers. These guidelines should be agreed upon before any formal models are written.

(2) Even if automated tools are used, the structure of the formal proof must be clearly stated. Presenting the structure clearly makes the automated proof more credible as well as making the verification much more maintainable.

5. SUMMARY

In this paper we have proposed a discipline for verifying software. We think that a verification should consist of three interactive processes: a modelling process, a theorem proving process, and a review and acceptance process. The modelling process should develop formal mathematical models of all requirements, specifications, processes, and systems that are relevant to the verification, and should generate conjectures in the mathematical framework established by the models. The models and conjectures should be clear and their appropriateness justified. In the theorem proving process, proofs of the conjectures generated during the modelling process should be constructed and analyzed. Unlike the modelling process, the theorem proving process should be a purely mathematical endeavor. The review and acceptance process should provide a means of communication, so that the verifiers can convince the reviewers — and ultimately the customer — of the adequacy of the verification.

REFERENCES

1. DeMillo, R. A., Lipton, R. J., and Perlis, A. J., "Social Processes and Proofs of Theorems and Programs," *Communications of the ACM* **22** (1979), 271–280.

2. DiVito, B., and Proctor, N., "Restricted Access Processor Formal Security Model," Technical Report TR-82041, Sytek (July 1985).

3. DiVito, B., and Sullivan, E., "Restricted Access Processor System Verification Plan," Technical Report TR-82046, Sytek (October 1983).

4. Halpern, D., and Owre, S., "MUSE: The Sytek Proof Processing System", Technical Report TR-85007, Sytek (July 1985).

5. Lakatos, I., *Proofs and Refutations*, Cambridge University Press, Cambridge, 1976.

6. Levitt, K., Robinson, N. L., and Silverberg, B. A., "The HDM Handbook," SRI International, Menlo Park, California, (June 1979).

7. Proctor, N., "The Restricted Access Processor: An Example of Formal Verification," *Proceedings of the 1985 IEEE Symposium on Security and Privacy*, Oakland, California, 1985.

8. Proctor, N., "Restricted Access Processor Message Block Processing System Formal Top-Level Specification," Technical Report TR-83002, Sytek (July 1985).

9. Proctor, N., “Restricted Access Processor Verification Results Report,” Technical Report TR-84002, Sytek (July 1985).

10. Scherlis, W. L., and Scott, D., “First Steps towards Inferential Programming”, *Proceedings of the IFIP Congress*, Paris, 1983.