CONSTRUCTING
VERIFIED AND RELIABLE
COMMUNICATIONS PROCESSING SYSTEMS

Donald I. Good

May, 1977

CERTIFIABLE MINICOMPUTER PROJECT
Institute for Computing Science and Computer Applications
The University of Texas at Austin
Austin, Texas  78712
ABSTRACT

A comprehensive methodology that has been developed for constructing verifiably reliable and secure computing systems is summarized. The methodology can be applied to many different kinds of systems, but is specifically oriented toward communications processing systems. The methodology is a system of methods for attaining total system reliability and is based on constructing verified software and highly reliable hardware. The methodology has been formulated by bringing a diversity of advanced research concepts to bear on the real problems of communications systems. This has led to the development and integration of

* program specification methods
* program proof methods
* program validation methods
* a program design language
* a program design system
* hardware designs to support verified software
* hardware reliability analysis and enhancement methods

into a coherent methodology for constructing verifiably reliable and secure systems. The methodology has been successfully applied to the experimental design of a secure message switching system structured as a packet-switched computer network.
# TABLE OF CONTENTS

1 Introduction .................................................. 1

2 The Methodology ................................................ 2
   2.1 Methodology Structure ....................................... 3
   2.2 Verified Software ........................................... 3
      2.2.1 Program Specifications .................................. 5
      2.2.2 Program Verification .................................... 5
      2.2.3 Supporting Tools ........................................ 7
      2.2.4 The Network Communications System ...................... 8
   2.3 Reliable Software .......................................... 10
      2.3.1 Hardware Architecture .................................. 10
      2.3.2 Physical Reliability .................................... 10

3 Conclusion ..................................................... 11

Bibliography ..................................................... 12
ACKNOWLEDGMENTS

This report provides only a brief introduction to a methodology that is the result of the cooperative efforts of many people. Various segments of the methodology have been developed by I. Agerwala, A. Ambler, J. Browne, W. Burger, R. Cohen, D. Hare, C. Hoch, J. Howard, M. Moriconi, M. Smith, S. Szygenda, E. Thompson, M. Tyson, and R. Wells, and their individual contributions are expressly identified in [Good, 77]. The strength of the methodology, however, lies primarily in the integration of these efforts, and, therefore, in the strong and stimulating interactions among all of those who have participated in CMP.

The direction and substance of CMP also has benefited from numerous discussions with L. Robinson, K. Levitt, and P. Neumann of the Secure Operating System Project at Stanford Research Institute; with K. London, D. Musser, M. Yonke, and D. Lynn of the Program Verification Project at USC-Information Sciences Institute; with W. Wulf of the Alphard project at Carnegie-Mellon University; with A. Marmor-Squires of the Department of Defense; and with W. Bledsoe of the Automatic Theorem Proving project at The University of Texas. The work of W. Bledsoe and M. Tyson was supported in part by National Science Foundation Grant DCR 74-12866, and the work of M. Moriconi was supported in part by the Defense Advanced Research Projects Agency under Contract No. DAHC15-72-C-0308, ARPA Order No. 2223 at USC-Information Sciences Institute.
CONSTRUCTING VERIFIED AND RELIABLE COMMUNICATIONS SYSTEMS

Donald L. Good

1. Introduction

"It's late." "It costs too much." "It doesn't work." All too often these are the Big Three of computer software development. This puts the customer in the unenviable position of waiting for an expensive computing system that is riddled with bugs. Good management techniques can help to alleviate each of these problems, but the basis for a solution to the reliability problem clearly rests in the domain of computer and programming technology. Once systematic methods for constructing reliable systems are established, they can be incorporated into good management techniques for attaining the goals of reasonable production schedules and costs. This report gives a brief introduction to a comprehensive methodology that has been defined by the Certifiable Minicomputer Project (CMP) at the University of Texas at Austin for constructing computing systems with a degree of reliability far exceeding that which can be attained using conventional methods. The methodology currently is being applied successfully to the construction of a hypothetical, secure message switching network resembling the IMP subnet of the ARPANET.

The methodology can be applied to many different kinds of systems, but is specifically directed toward the problems of reliability and security of communications processing systems implemented on minicomputers. Communications processing refers to systems whose primary purpose is data transfer. This type of system normally is characterized by relatively large amounts of input and output processing and small amounts of computation. Typical examples of communications systems are front-end processors in which communications terminals are interfaced to a host processor, remote data concentrators where data from many low speed lines are multiplexed onto one or more high speed lines, terminal multiplexing in which the system multiplexes and controls a cluster of directly connected terminals to share the terminal control hardware and provide a common interface to a communications network, and message switchers in which messages are received from various sources, temporarily stored, analyzed, and then routed to one or more destinations. If the communications system processes privileged information, system security becomes one of the important dimensions of reliability. The system should prevent either the accidental or deliberate compromise of any privileged information that flows through the system. Such compromises would include any introduction of unauthorized information into the system, and any unauthorized disclosure, acquisition, modification or loss of information.

The CMP methodology which is described in detail in [Good, 77] is a system of methods for attaining total system reliability by constructing verified software and highly reliable hardware. The development of the software methodology has included
* the definition of a unified specification methodology and extension based on the integration and extension of several existing formal program specification methods,

* the integration of rigorous methods of program proof and execution-time validation into a powerful and practical approach to program verification,

* the definition of automatable proof methods for data abstractions based on access control,

* the definition of automatable proof methods for systems of concurrent processes,

* the structuring of the specification and verification methods so that program specification, implementation, and verification can proceed incrementally and in parallel,

* the realization of the specification and verification methods in the definition of a program design language called Gypsy that is both a formal program specification language and a verifiable, high-level language for systems programming, and

* the implementation of a system that supports the application of the software methodology by providing automatic and interactive facilities for the incremental design and verification of Gypsy programs.

In addition to the software methodology, the hardware methodology provides

* the definition of a systematic set of procedures for evaluating and enhancing the physical reliability of contemporary minicomputer systems, and

* the definition of new hardware architectures that directly support the software concepts of modularity and access control which are used to create reliable software systems.

The integration of program proof and execution-time validation, the definition of effective proof methods for systems of concurrent processes, the merging of a formal specification language and a verifiable programming language into a common program design language, and the implementation of a system that supports the incremental design and verification of large programs are major conceptual advances in solving the problem of constructing verifiable systems.

2. The Methodology

The CMP methodology includes methods for attaining both software and hardware reliability. **Reliability** is the adherence of the system to what is expected of it by the user. A system is reliable if it always performs as the user expects it to. **Security** is the dimension of reliability that deals with the flow of privileged information
through the system. In a reliable system, privileged information should be accessed only in certain privileged ways. The methodology is directed at the broad problem of system reliability, and therefore, also at the more narrow problem of system security. (This also is the view taken by Linden [7b].)

The methodology defines methods for designing the system software and for verifying, with mathematical precision, that the software always conforms to its specifications. It is important to draw a careful distinction between "reliability" and "verifiability." Reliability refers to a desired conformity with user expectations, verifiability refers to a demonstrated conformity with formal specifications. Expectations are in the mind of the user, specifications are an expression of those expectations in precise language. As pointed out in Liskov [75] and Good [75], the extent to which the specifications capture the expectations determines the ultimate degree of reliability of rigorously verified software. Verification will rigorously demonstrate that the program conforms to its specifications, and if the specifications completely capture the user expectations, then verifiability gives reliability. Also, limited experience suggests that the exercise of stating formal specifications usually forces a deep analysis and understanding of user expectations and therefore verifiability does lead to reliability.

2.1 Methodology Structure

The methodology for constructing verifiably reliable and secure communications systems is based on constructing verified software and highly reliable hardware. The major components of the methodology are related as shown in Figure 1 and are summarized in the following sections.

2.2 Verified Software.

The software methodology is based on the principles outlined by Good [75] for verifying programs of significant size and complexity.

* **State precise specifications.** This is an obvious prerequisite for rigorous verification.

* **Simultaneous design and verification of the program.** Verification is one of the design goals of the program, and can be carried out in parallel with the program development.

* **Program factorization.** Programs can be structured so that their complete verification can be factored into independent verifications of a large number of small units.

* **Verifiability. Then efficiency.** A program should be designed first for verifiability, and then measured and optimized for efficiency. In the end, we should be able to attain both verifiability and efficiency.

* **Automatic assistance.** Automatic support for the verification
Figure 1

STRUCTURE OF THE METHODOLOGY

- Verified and Reliable Communications Systems
  - Verified Software
    - Methods
      - Specification
    - Verification
      - Program Proof
      - Program Validation
  - Tools
    - Program Design Language
    - Program Design System
  - Reliable Hardware
    - Hardware Design
    - Physical Reliability
process is used where practical. This amplifies human verification capability and reduces the probability of human error.

The methodology follows these principles by defining methods for program specification and verification, by defining a program design language for expressing both programs and their specifications with a high degree of modularity, and by implementing an interactive system that supports the design and verification of a program through its development lifetime.

2.2.1 Program Specifications

The program specifications are precise statements about the desired behavior of the program, and about the units from which the program is built. Specifications can be expressed as any of ten different kinds of invariant properties about the program state. Each of these specifications has a precisely defined accessibility and scope. The accessibility of a specification is either internal or external. External specifications are visible outside the unit they specify; internal ones are not. The scope of a specification is either instantaneous, closed, or open. Instantaneous invariants hold at particular points in the program execution. Closed invariants hold throughout a specified segment involving several statements. Open invariants hold everywhere except throughout certain specified segments.

2.2.2 Program Verification

The CMP methodology provides three complementary methods for verifying that the system software meets specifications: pre-execution proofs, in-execution validation, and post-execution testing. Post-execution analysis of trace records and program output permits testing and debugging of programs by conventional methods. The basic flaw in testing, however, is that the inputs used in testing are not the same as the inputs encountered in practice. Testing shows that the program meets specifications for the test cases, but does not show that it will in all cases. The program validation methods test the program as it runs. If its specifications are violated, an error is signalled and the program can attempt to recover. Run-time validation solves the problem of testing on the wrong inputs because the tests are made as the program runs on its actual data. Testing, however, either after or during execution, can only detect errors after they occur. Therefore, a major part of the methodology is directed at rigorously proving, prior to execution, that a program always will execute according to its specifications. These methods reduce the program and its specifications to a set of logical verification conditions which are sufficient to show that the program always conforms to its specifications. These verification conditions are proved using the high degree of rigor and precision of traditional mathematics, and these proofs can be certified by an independent review.
It is important to understand that these proofs offer no absolutes. Mathematical proofs are created by humans and subject to human errors. Historically, however, the integrity of proofs in mathematics has far exceeded the reliability of systems software. This is the avenue by which the CWP methodology offers a much higher degree of software reliability than can be attained by conventional software development methods.

The integration of in-execution validation and pre-execution proofs is a significant new approach to the problem of rigorous program verification. Any specification can be designated to be verified by proof or by validation at run-time or both. Specifications that are proved need not be validated during execution. Conversely, specifications that are validated at run-time need not be proved, but can be assumed at the appropriate points in proofs. Therefore, proof can significantly reduce the computational expense of run-time validation, and run-time validation can significantly reduce the size and complexity of a proof. It is left to the program designer to establish a balance that is best for a given program. The possibility of both proving and validating a specification provides a means of checking the validity of the proof and the integrity of the execution environment. If a proved specification fails a run-time test, then either the proof is invalid or the supporting system has malfunctioned. This malfunction might have been caused by any one of a number of problems ranging from a faulty program compilation to a hardware malfunction. Run-time validation not only is desirable, but also sometimes is necessary to perform explicit tests about the integrity of the execution environment. For example, one cannot prove that data transmission errors will not occur, but these errors can be detected explicitly, and recovery procedures can be defined. These properties establish a very effective working relationship between verification by proof and by run-time validation.

The proof methods are based on extensions of the inductive assertion method of proof. The Gypsy language provides extensive facilities for both operational and data abstraction. These facilities permit the program to be broken into many small independently provable units. The proof methods are uniformly structured so that if a primary unit A uses a secondary unit B, only the external specifications of B are required in the proof of A. This permits a program proof to be carried out in parallel with a top down design strategy. A top down design strategy is desirable because a successful proof of a primary unit assures the adequacy of the specifications of its secondary units. The proof methods, however, can be used in the context of any reasonable program design strategy. The success of the proof will depend largely on the degree to which abstraction is used to break the program into logically manageable units.

Concurrency is one of the distinctive characteristics of communications processing software. The system normally consists of a number of processes operating concurrently, because of the emphasis on communications processing, message buffers were selected as the mechanism for process synchronization, and Gypsy has full facilities for programming systems of concurrent processes. The methodology
defines rigorous methods for specifying and verifying systems of concurrent processes, and the operational and data abstraction facilities of Gypsy are fully applicable to processes and buffers.

2.2.3 Supporting Tools

Rigorous verification is difficult, and sometimes impossible, to support with conventional languages and programming systems. This has led to the development of a program design language called Gypsy (Ambler, 76) and an interactive program design system to support the design and verification of Gypsy programs.

Gypsy is the unifying element of the methods for constructing verified software. It was developed from Pascal (Jensen, 74) and incorporates many of the ideas of Hoare [72]. The language is unique in that it contains extensive facilities for expressing both a program and its specifications. This integration of specification and programs into a common language provides the basis for the powerful verification relationship mentioned previously between proof and run-time validation, and is a major conceptual advance in the design of languages for reliable software. Gypsy also contains full facilities for operational and data abstraction. This permits the program to be expressed in Gypsy throughout all stages of its design, from initial conception through specification, implementation, verification, and successive evolution and growth. Thus, the verification methods can be brought into the program development process at the earliest stages when they can be of maximal benefit. All of the specification and programming features of Gypsy are fully verifiable. These are summarized in [Ambler, 77a] and include Pascal-like features for general programming as well as features for expressing concurrency, access control, error handling, and some limited real-time dependencies. A comparison of the access control mechanisms of Pascal (Jensen, 74), Concurrent Pascal (Birch, Hansen, 75), Euclid (Lampson, 76),CLU (Liskov, 73), and Gypsy appears in [Ambler, 77b].

The program design system supports the growth and evolution of a Gypsy program throughout its development cycle and automates much of the process of pre-execution proofs. This system amplifies human design and verification capabilities and reduces the probability of human error. The system is a 200K word Lisp program that runs under either TOPS-10 or TENEX on a PDP-10. It contains an interactive verification system with the usual capabilities. Verification conditions are constructed and simplified automatically, and can be proved interactively. The system further provides an integrated approach to program design and verification by maintaining a data base of all Gypsy program units and full information about the development status of the complete program and its proof. The system uses this information to maintain the validity of the proof in the presence of program growth and modification. The implementation of this system is a major step toward rigorously verifying programs of significant size and complexity.
The software methodology has been applied successfully to the design of a network communication system that resembles the IMP subnet of the ARPA/NET [Melis, 76]. The logical structure of NCS is shown in Figure 2. The complete network consists of a message system (MsgSys), a circuit system (CirSys), and a packet system (PacSys). MsgSys is a process that delivers messages among a set of user processes. A user can direct a message to any other user; however, there is a security function which determines whether user i is permitted to send a message to user j. The basic specification of MsgSys is that it is to deliver messages properly among all pairs of users according to the constraints of the security function. MsgSys also informs the user of the disposition of messages. The user may be informed that a message was delivered to its destination or that it was dropped. There is no guarantee or assumption that a user will receive a disposition for every message. MsgSys is implemented by a Connector process attached to each user and by the circuit system CirSys. The Connector receives and sends messages to and from the user (through message buffers), enforces security, and generates message dispositions. CirSys transports messages and message disposition notices among Connectors through a virtual circuit. This circuit is created through a request-to-send/ready-to-receive protocol between the Connectors and CirSys. CirSys is implemented by an Assembler/Disassembler process (A/D) that is attached to each Connector and by the packet switching system. The A/Ds implement the Connector/CirSys protocol and also perform disassembly of messages into packets and reassembly of packets into messages. The actual packet delivery is done by the packet switch(es) (PS) in PacSys using a fixed routing strategy.

The concentric structure of NCS as shown in Figure 2 is its logical structure. Each of the 20 circles is a process and the rectangles denote the connecting message buffers. All of these processes run concurrently, and the logical network structure can be mapped onto any physical structure that preserves this concurrency. To continue the parallel with the ARPA/NET, a user process corresponds to a host, a Connector, its A/D, and its adjoining packet switch correspond to an IMP that supports a host; and the packet switch in the center that is not attached to an A/D corresponds to an IMP with no host. Under this correspondence, the message buffers connecting the packet switches as well as those that connect the MsgSys to the user processes would be implemented in terms of physical communications lines, whereas the other buffers are implemented by software.

NCS has been completely specified and coded in Gypsy. The complete text of the system is about 2500 lines of Gypsy, of which approximately 60% is specification and 40% is executable code. These 2500 lines are composed of about 360 separate program units ranging in size from one to 50 lines with an average of about seven lines per unit. The most difficult parts of the system have been proved, including all parts of the system that involve concurrency. The proofs were done using the packet system topology shown in Figure 2, but the proof was constructed so that it applies to any number of packet switches in any topology. These proofs were done manually because key parts of the verification system had not yet been
Figure 2
STRUCTURE OF RCS
implemented. It should be pointed out that NCS is not an operational system because Gypsy has not yet been implemented.

2.3 Reliable Hardware

Total system reliability requires more than just verified software. The software must be translated onto actual hardware that operates reliably. The translators that map the software onto hardware are themselves programs and, in principle, can be verified. A reliable translation, however, can be made easier to attain and to verify if the translation can be made simpler. The methodology defines several innovative hardware designs that directly support verified software and make this translation simpler. If the software is to be faithfully executed, the hardware must conform to the assumptions made about it by the software or report any nonconformity that may arise. The methodology, therefore, also defines methods for evaluating and enhancing the physical reliability of minicomputer systems. Minicomputers were chosen because of the emphasis on communications processing. Methods for enhancing minicomputer reliability through hardware redundancy are defined. Even though the hardware reliability can be enhanced, absolute reliability of physical components is not possible. Therefore, the software does not assume a perfect execution environment, but instead acknowledges the possibility of run-time errors. Gypsy provides explicit facilities for monitoring the occurrence of errors and exception conditions and for recovery where feasible.

2.3.1 Hardware Architecture

The conceptual gap between the abstract machines defined by modern software features, such as those available in Gypsy, and the machines provided by most contemporary architectures is disconcertingly large. High level, problem-oriented concepts must be implemented by complex compilers and operating systems. The difficulty of bridging this gap reliably can be reduced if the gap can be made smaller. The methodology addresses this problem by recommending improved hardware designs that directly support modularity, abstract data objects, concurrency, and reliability. The bases for these designs are a capability-based architecture and a tagged, stack memory. The capability-based architecture provides a uniform mechanism for process management, for unifying software and hardware routine invocations, for error detection, and for type checking. Typed data objects can be supported further by a tagged memory, and a stack memory can provide direct support for error recovery. These architectural features appear to be worthy of careful consideration in the design of systems of the future. (Hocn, 77) describes how this kind of architecture can be adapted to a PDP 11/45.

2.3.2 Physical Reliability

The CMP methodology also defines methods of analyzing and enhancing the physical reliability of contemporary hardware. Methods for systematically characterizing the architecture of minicomputer systems are defined. These methods can be used either to analyze the architectural features of an existing system or to configure a new
one. Primitive reliability measures have been defined for each possible feature in the architectural characterization. These specify what basic reliability can be expected from the various architectural features and what can be done to increase the reliability. Cost measures are defined for the various reliability features for each architectural feature. These are stated as percentage increases in hardware necessary to achieve increases in reliability rather than in absolute dollar figures. Taken together these measures provide an effective way of analyzing and enhancing physical reliability of contemporary communication system hardware.

3. Conclusion

The methodology had been characterized throughout by an example-driven development. It has been formulated by bringing a diversity of advanced research concepts to bear on the real problems of communications systems. The most significant example is the development of the Network Communication System. This approach has led to significant new research which has been integrated with

* program specification methods
* program proof methods
* program validation methods
* a program design language
* a program design system
* hardware designs to support verified software
* hardware reliability analysis and enhancement methods

into a promising methodology for constructing verifiably reliable and secure communications processing systems.

The definition of the initial methodology is complete. Work to date indicates that the methodology has a promising potential for actual applications. It is expected that this potential will be realized by allowing the methodology to mature through refinements that are made through additional trial applications to an increasingly realistic sequence of communications systems.
BIBLIOGRAPHY


