AD-A083 887 NAVAL RESEARCH LAB WASHINGTON DC F/G 9/2
SOFTWARE ENGINEERING TECHNIQUES APPLIED TO PROTOCOL SIMULATION.(U)
FEB 80 C E LANDWEHR
UNCLASSIFIED NRL-8385 SBIE -AD-E000 399 NL

END
DATE FILMED
6-80
DTIC

(12) LEVEL II

NRL Report 8385

Software Engineering Techniques Applied To Protocol Simulation

C. E. LANDWEHR

Communications Sciences Divison

February 19, 1980

NAVAL RESEARCH LABORATORY
Washington, D.C.

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER NRL Report 8385	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) SOFTWARE ENGINEERING TECHNIQUES APPLIED TO PROTOCOL SIMULATION		5. TYPE OF REPORT & PERIOD COVERED Interim report on a continuing NRL problem
		6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) C. E. Landwehr		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Research Laboratory Washington, D.C. 20375		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS NRL Problem 75-0113-0-0 61153N RR14-09-41
11. CONTROLLING OFFICE NAME AND ADDRESS		12. REPORT DATE February 19, 1980
		13. NUMBER OF PAGES 14
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		15. SECURITY CLASS. (of this report) UNCLASSIFIED
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release, distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

Simulation	Protocols	Satellite communication
Software engineering	Abstract types	
Data communications	CPODA	

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

The design, implementation, debugging, validation, use, and modification of a computer simulator for the study of satellite communication protocols are reported. An important objective of the effort was to apply state-of-the-art software engineering techniques in the design and construction of the simulator. Experience with seven specific techniques is described.

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

CONTENTS

SOFTWARE ENGINEERING TECHNIQUES APPLIED TO PROTOCOL SIMULATION

INTRODUCTION

Simulation remains an important tool for studying the performance of complex systems. Over the past few years, there has been significant progress in the application of statistical techniques to the analysis of simulation results [1-4]. Techniques used in the construction of simulation programs have received less attention, except in the sense that software engineering methods are supposed to apply to the construction of programs in general. This paper reports experiences gained in applying certain software engineering techniques to the construction of a simulator of satellite communication protocols.

Simulation provides a good test case for software engineering concepts in that simulations often require (a) the use of concurrent processes, (b) the modeling of a complex system through abstraction, (c) the validation of the program against vague requirements, (d) the modification of the program to model alternative systems or configurations, and (e) the achievement of reasonable CPU and storage efficiency to allow sufficient replications of experiments for statistically meaningful results. In this project, we applied the following software engineering techniques:

- Complete design prior to coding [5]
- Design review by knowledgeable outsiders [6]
- Use of the information hiding principle in design [7]
- Use of abstract types [8]
- Code reading by other than the coder prior to testing [9]
- Use of co-operating sequential processes [10]
- Use of pseudo-code [11]

The results observed from the use of these techniques were generally favorable, although some proved of greater benefit than others.

DESIGNING THE SIMULATOR

The first step in the design was to survey and document the requirements for the simulation and the environment to be simulated. The general goal for the project was to

Manuscript submitted December 10, 1979.

evaluate the performance of alternative channel management algorithms for broadcast satellite UHF channels. Thus, the basic requirement for the simulator was that it accurately model the relevant characteristics of this type of channel and provide a structure in which different communications protocols could be tested. Paragraphs were written describing the primary aspects of the real world that would affect the performance of protocols in the communications systems of interest. A secondary set of requirements that would allow modeling of more general systems (e.g., point-to-point networks, networks including voice links) was also developed. The primary list of requirements was viewed as mandatory, the secondary as desirable -- extensions of the initial simulator were to allow their inclusion, if possible. Brief descriptions of three protocols were included to provide examples of the operations that would have to be modelled by the simulator. Finally, the measurements that might be needed from the simulator were discussed.

Following the identification of requirements, a design was proposed. Since the characteristics of traffic to be transmitted over the channel (arrival rates, sources, destinations, priorities, message lengths) were unspecified in general, the design had to provide a flexible way to specify them. A goal for the design was that changes in these characteristics should only require parameter changes in the simulator, not coding changes. With respect to testing different protocols in the simulator, coding changes were to be expected; the goal was to limit them to the protocol module. Changes in measurements recorded were expected to require changes to code; a goal was that these changes should be minimized and localized with respect to each variable, and that code for the generation of reports should be unaffected by changes in the variables actually measured.

The initial design was specified informally (i.e., in English) as a set of functions grouped into modules based on the principle of information hiding [7], so that each module keeps one or more "secrets" from the other modules. The design also defined a structure for processes to model the activities occurring in a real communications system -- message generation, transmission, and reception. Processes were defined on the basis of sequentially predictable activities: actions that could occur simultaneously in the real world (e.g., creation of one message and transmission of another) were modeled by separate processes. Pseudo-code for some example processes was included in the design. This code consisted of function invocations embedded in ALGOL-like control structures.

The design document also included a table listing system characteristics likely to be changed in experiments (such as traffic distributions, communications equipment, statistics collected, channel error characteristics) cross referenced with the proposed simulation modules. At the intersection of each module (row) and experimental variable (column), an indication was given as to whether a change in the variable would require no change, a parameter change, or a coding change in the corresponding module. This table provided a convenient way to check that the modularization chosen was appropriate according to the information hiding principle.

The system requirements and the proposed design were both described in a 20-page technical memorandum [12] and circulated to three interested observers for a thorough review. Keeping the documents together allowed reviewers to judge the design in relation to the requirements. The results of these reviews were incorporated into the design memo and redistributed; the revised memorandum served as the basic document guiding the implementation. Differences between the initial and final designs derived principally from a

sharpening of the concepts of process, function, and module. The revised document included definitions of these terms and omitted one synchronization process that had been present in the initial design. This process was found not to represent a real world activity.

The final design consisted of two major parts: (a) the User Interface and Simulator Control, which controlled a given simulation run, providing check pointing and restarting facilities, parameter input/output, report generation, and initialization, and (b) the Communications System Simulator, which actually simulated the given system.

The Communications System Simulator included two types of processes: Message Generation Processes (MGPs) and Channel Access Processes (CAPs). Each node in a broadcast communications network would have one or more MGPs to generate traffic according to specified distributions for message lengths, arrival rates, etc., and one CAP to transmit messages across the channel (later versions split each CAP into two separate processes, one for transmitting messages and the other for receiving messages). These processes would invoke the functions contained in five modules. Listed below are the modules, with brief descriptions of their functions and secrets:

Message Generation — This module hides the details of message generation, such as the particular probability distributions in use for determining message inter-arrival time, message length, etc. It provided functions to generate a new message and to generate the time between messages.

Message Storage — The queues of messages that are awaiting transmission are hidden by this module, as are the priority queuing algorithms. Functions are provided to return the highest priority waiting message, to remove a message from the store, to enter a message into the store, etc.

Channel — This module hides the noise and delay characteristics of the communications channel. There are functions to transmit or receive data, and to obtain the propagation delay between a pair of nodes.

Protocol — The detailed behavior of the protocol for transmitting and receiving messages is concealed in this module. The specific functions vary according to the protocol, but there are generally functions to handle acknowledgments and retransmissions, to manage the storage of portions of messages, and so forth.

Statistics — This module provides a common set of functions for the collection and reporting of statistics recorded during a simulation run. It hides the algorithms and data structures used to generate the statistics and reports.

The user interface and simulator control design was specified in less detail, since the functions to be performed were typical of many other simulation programs. Three modules were identified, as follows:

Control — This module provided the user interface to top level simulation functions, such as reading parameters, initializing the simulator, initiating a simulator run, and generating reports. As implemented, this module reads user requests, checks them for validity, and invokes lower level functions to execute them. It requires no detailed knowledge of the lower level functions.

Parameter Input/Output — The user interface and functions for the acquisition, display, and saving of parameter values were hidden in this module. In order to prevent this module from requiring detailed knowledge of the parameters and input/output formats for all modules in the system, each module in the Communications System Simulator implemented functions to perform these operations for itself. The Parameter Input/Output module needed only to know the names of these functions so that they could be invoked when needed.

Report Generator — This module controlled the printing of simulation statistics. To prevent this module from knowing all the statistics variables in the system, a function was provided for each type of statistics variable to print a report. All instances of statistics variables were linked together, and the Report Generator required only to know the name of the list head in order to sequence through the set of variables, printing the required output.

IMPLEMENTING AND DEBUGGING THE SIMULATOR

SIMULA [13,14,15] was chosen as the implementation language, based on its support for abstract types and its availability on a local PDP-10. Although this choice was made prior to completion of the design, it is fair to say that the language choice had no influence on the design, since the author had not more than a nodding acquaintance with the language prior to this project. No SIMULA code was written prior to the completion of the design. Consequently, the implementation began with the development of several small SIMULA programs to resolve questions on how certain language features worked.

The pseudo-code examples for processes in the design had employed the concept of events — a process would wait for an event to occur, service an event passed to it, and wait for the next event. The SIMULA test programs revealed that SIMULA did not support this concept directly; either events and a scheduling mechanism for them had to be introduced, or the design for processes had to be modified. Since this problem affected the design, a memorandum was written posing the alternatives and was circulated to the reviewers of the earlier design document. The decision made was to avoid the use of events and to split the CAP into two separate processes, one to receive and one to send traffic over the channel.

Following this decision, the mapping of the design into SIMULA was reasonably straightforward. The CLASS structure of the language provides a good mechanism for the implementation of type abstractions. Of the abstract types used in the implementation, the one that has proven most useful in later developments is the abstract type for statistics collection. The development and use of this abstract type has been thoroughly documented elsewhere [16], but we will review it briefly here as an example of how type mechanisms were used to hide information.

The purpose of this family of types is to encapsulate the functions and storage required for recording simulator measurements such as message delays and backlogs, calculating statistics (such as the mean and variance) from these measurements, generating printed reports of these statistics, and, optionally, histograms of the measurements. Rather than a single type, a family of types is required because some measures (such as delay) are for inherently continuous (floating point) quantities, while others (such as the number of items

queued) are for inherently discrete (integer) quantities. Further, if histograms are to be collected, more storage is required for collecting the statistics and more parameters (number of bins, bin size) must be specified. Each statistics variable is specified as either real or integer and histogram or no histogram. The same operations are provided for all of the types: initialization (to generate a new instance of the type), update (to record an observation), and report (to print a summary of the observations). For histogram type variables, one additional operation is supplied to print the histogram.

Defining the types as SIMULA CLASSes satisfied the need for encapsulation of the storage and operations, but an additional goal was to hide from the simulator report generation mechanism the identity and number of statistics variables (instances of these types) that were to be printed. If this information were successfully concealed, changes in the number and type of statistics collected would cause no changes in the report generator. To accomplish this goal, all statistics variables were contained in a single linked list. (SIMULA provides a built-in type for such lists.) Each time a new instance of one of the statistics types was created, it was linked to the end of this list. The report generation routine then was required only to know the name of the list head and the SIMULA list operators. After printing a general heading, it merely invoked the report operation provided with each list element. Since it is possible to interrogate the type of a list element, the report generator could also detect whether a particular variable was of a histogram type and, if so, invoke the type-specific histogram generation routine.

Coding and debugging were carried out in parallel. The lowest levels of the system (in the sense of the "uses" hierarchy [17]) were generally coded earliest and were compiled as soon as they were coded, in order to detect errors as early as possible and to make the code more readily available for reading by an interested observer. Thus, the abstractions for the definition of probability distributions and the generation of random numbers were the first coded, followed by the abstractions defining messages, message stores and message generator processes. A skeleton control module was implemented next, together with procedures to save and restore parameters, in order to allow execution of the completed portions of the code. The control functions and utilities were then gradually expanded as the coding of the abstractions for slots, blocks, channels, and finally, the test protocol module for slotted ALOHA was completed. The mechanisms for defining the recording statistics to be collected were the last to be added.

The procedures just described facilitated the integration of new sections of code with existing code, since the existing code had already been compiled, read by an observer, and, generally, executed in some fashion before the new code was written. Attention could then be focused on the relatively small new section added rather than dispersed over the entire simulator. Initially, all code produced was read by a reviewer, but this requirement was dropped after several weeks because of competing demands for the reviewer's time.

The primary reviewing and debugging techniques used during the initial coding phase included the code-reading already mentioned and the use of compiler features for compile-time syntax checking, run-time error detection, and interactive debugging. Although the code-reading uncovered some bugs, it was primarily helpful in ensuring that the code that was produced was well commented and consistent with the design documents. As successive modules were coded and compiled, many typographical errors resulted in faulty block structure and were detected thereby. The errors reported by the compiler in such instances are, unfortunately, usually unrelated to the true source of the problem. The structure

provided by PDP-10 SIMULA for separately compiled procedures proved to be more hindrance than help, and separate compilation finally was abandoned as a development tool in favor of a single large compilation for the entire simulator. The problem centered on the requirement that PDP-10 file names (limited to a length of six characters) and the names for the separately compilable CLASSes they contain be consistent. The run-time debugging package was very helpful (in fact, its functions are used by the control module to allow display and alteration of parameters). The ability to display values of variables after an execution error was particularly valuable during debugging; its benefits would be even greater if values returned by procedure calls could be made available.

An informal error log maintained during the development revealed that the most numerous errors were clerical, and, of these, the most bothersome were errors resulting in mismatched BEGIN-END pairs. Other errors included improper I/O function usage (caused partly by a lack of details in the SIMULA documentation), problems in attempting to use separate compilation, improper parameter passing, default initialization (SIMULA sets variable values to zero initially, masking some references that should be errors), and improper assumptions about the order of evaluation of Boolean expressions. The most difficult errors to find were problems that only revealed themselves in unusual statistical measurements from simulation runs. In one case, a few messages seemed to have inordinately long processing delays; the bug found was the improper resetting of an index on an infrequently exercised processing path. This error caused parts of a message to remain queued when they could have been transmitted. Debugging the simulator in this respect appeared quite similar to debugging an actual implementation of the protocol. The design, construction, and validation (described below) of the simulator were completed within 6 man-months. The length of the program was roughly 2450 lines of SIMULA source (this and subsequent measures of code length include blank lines and comments). About 200 of these lines were code specific to the ALOHA protocol.

VALIDATION

The initial validation of the simulation was performed by implementing the required processes and models for the slotted ALOHA protocol and comparing simulation results with analytic and simulated results from Lam [18]. This process uncovered some bugs in the implementation and led to a deeper understanding of both the details of the protocol and the assumptions on which Lam's analysis was based. Ultimately, satisfactory agreement between the simulation and Lam's work was obtained, and the details of the implementation and validation were described in a technical memorandum [19].

MODIFYING THE SIMULATOR FOR OTHER PROTOCOLS

Following validation, the simulator was used to study three different protocols relating to two particular Navy problems. These studies provided useful results and are documented elsewhere [20,21], Their relevance here is primarily in the changes that were made to the simulator to accommodate the new protocols. The design and development techniques used in the project were intended to reduce the need for changes and to limit them to particular areas of the program. Although we did not succeed in limiting changes solely to the protocol

module and processes, the software engineering approaches used in the design and construction of the simulator did simplify the few changes that were required outside of the protocol module. In this section we review the protocols that were modeled and the changes they required.

Navy Tactical Protocol

The first additional protocol implemented (since the original slotted ALOHA) was a design for a proposed Navy system for transmitting tactical data among ships and shore stations. This protocol was based on polling, with a central controller broadcasting the polling list. Unlike the slotted ALOHA system, explicit acknowledgments were required for received data. Consequently, a functioning CAP-receive process was required to receive data and generate the acknowledgment. Because of the delay present in the channel, it was possible that parts of two transmissions for the same receiver might be outstanding at the same time. The process synchronization primitives in SIMULA, however, do not support the concept of multiple events pending for a single process. The solution applied was to introduce a channel process (in addition to the channel module). This process was responsible for receiving all transmissions, queuing them for the appropriate delay period, and then awakening the appropriate receiving processes. Beyond the addition of this process, only the transmission format blocks and protocol processes were rewritten (as planned), and even these were able to use many of the same utility functions implemented in the protocol module for slotted ALOHA. The Message Generation, Message Storage, Statistics, Control, Parameter Input/Output, and Report Generator modules were unchanged. The effort to make the changes required for this protocol and to debug them was about 2 man-months. The length of the program grew to approximately 3100 lines, of which about 700 were devoted to protocol-dependent routines.

CPODA (Contention-Based, Priority Oriented, Demand Access)

Next, a simulation of CPODA, a packet protocol with distributed control [22,23] was implemented. In this protocol there is a fixed length frame, which is divided into reservation and data transmission subframes. These subframes are of varying length, and the reservation portion is accessed in a contention mode. Each node transmits reservations prior to transmitting data, and each node keeps track of the current queue of reservations by listening to all of the traffic on the channel. Two types of timeouts are performed by nodes operating under CPODA. First, each time a node transmits a reservation or data packet, it listens for the echo of that packet from the satellite. If the echo does not occur, the reservation packet is requeued for transmission later. Second, if the echo of a data packet is received, the node waits a specified length of time for the positive response from the intended recipient, and, if none is received, queues a reservation packet to allow later retransmission of the data packet. In the Navy tactical protocol, there was only one type of timeout, and this was tied to the polling frame structure, so the channel process queuing mechanism was sufficient. The CPODA case was sufficiently complex to require a reexamination of this choice.

This reconsideration led to the introduction of an event mechanism in the simulation. An implementation was devised wherein each event was an instance of a SIMULA PROCESS that would, on awakening, activate the process intended to receive the event and pass it a

parameter defining the event type. This approach avoids the requirement of a duplicate scheduler running on top of the SIMULA scheduler. (It was later discovered that Franta [15] proposes a similar implementation in his book (p. 176).)

Again, the changes to the simulator were principally in the protocol module, as planned, but the event mechanism made possible the elimination of the channel process added in the previous protocol simulation, so there were some changes in the channel module as well. The definition of the event mechanism was made outside of the protocol module, so that it would be available for use in future simulations. The effort required to make all of these changes was approximately 3 man-months. Length of the entire simulator grew to approximately 3500 lines of SIMULA, of which almost 1000 were devoted to CPODA-dependent code.

Existing Navy Polling Protocol

A third protocol simulated was a Navy protocol currently in use for the exchange of messages over satellite channels. This protocol is again a polling scheme with a central controller; it has a simpler structure than the Navy Tactical Protocol, since its delay requirements are less stringent. It is used principally for the transmission of text messages from ships to a shore station over a UHF satellite link.

Some additional constraints placed on this study allowed the model to be simplified greatly: acknowledgments and channel noise did not have to be modeled, and specified distributions were to be used for message generation. These restrictions eliminated the need for some lengthy routines that had defined an abstract type for probability distributions, the need for the channel module, and for any processes to receive messages or queue and process acknowledgments. Consequently, instead of building relatively trivial routines within the general framework already implemented, it was decided to construct a separate, simple simulator for this protocol by borrowing only the appropriate modules from the original simulator.

The statistics and report generator modules were borrowed intact, while the message generation module was simplified to use probability distributions instead of the more general distribution descriptors from the other simulator. The abstraction implemented to model messages was simplified, since no specifications of destinations or priorities were required, and the message store module became a simple FIFO queue, using the predefined SIMULA operators. The protocol module and a very simple simulation control and parameter input routine were written from scratch.

This approach was quite successful within the limited goals of the study. The simulator was constructed quickly and debugged easily. Other than the elimination of unneeded functions, virtually no changes were required in the borrowed modules. Total length of this simulator was about 700 lines, of which about half were borrowed. The simulator was constructed and debugged in less than a month.

The need for a "minimal" simulator lends further credence to the design concepts of program families and minimal subsets [24,25]. It is interesting to note that, although those concepts were not explicitly employed in this project, the use of abstract types and information hiding modules led to a result consistent with them.

EXPERIENCE GAINED IN USING THE SIMULATOR

In addition to the lessons to be learned from the successive modifications to the simulator software, the effectiveness of the simulator as a tool for studying communications protocol performance is of interest. As mentioned above, the simulator was applied to two particular Navy problems. One of these involved the use of the distributed CPODA protocol to control communications among one to twenty ships and a shore station. The traffic in this case was primarily transactions with remote, land-based computer systems. The other problem involved evaluating the use of CPODA and the other protocols as vehicles for merging traffic from two presently independent Navy communication systems over a single channel. Useful results were obtained in both studies; CPODA appeared to provide superior performance in nearly all of the cases examined in the second study, and it seemed likely to provide adequate (though in some cases marginal) responsiveness in the first study.

In the course of performing these studies, a number of discoveries were made about the simulator itself. The simulator was designed to be run interactively, and the commands provided by the user interface were helpful in general. The PDP-10 run time SIMULA support aided the implementation of some of these — the debug package (DDT) provides facilities to display and alter values of program variables, set breakpoints, and to interrupt the simulator during execution. These facilities were helpful both during debugging and, occasionally, to alter parameter values during production runs. An unfortunate side effect of the run-time support was that it was impossible (without coding a special assembly language routine) to save a complete core image in a file so that it could be restarted later. The uncertainty of the effort involved in coding this routine and the limited project resources prevented its implementation.

The simulator itself provided facilities to save and restore parameter sets in files. This proved to be a crucial function, since the volume of information required to specify a given experiment was substantial. In fact, the interactive dialog for parameter specification was burdensome enough that it became easier to edit a saved parameter file with a text editor than to go through the entire dialog if only a random number seed or one or two parameters were to be changed.

The primary problems encountered in actual simulator runs were storage limitations. Although limits on the number of nodes in a simulated communications network were not specified in the initial design, the number of nodes expected in the first application had been planned as less than 15. Storage requirements were not a major consideration in the design or implementation, consequently, and test parameter sets were generated with networks of more than 50 nodes with the slotted ALOHA protocol. The actual studies, however, often specified multiple priorities of traffic, a variety of message generation processes, extensive statistics collection, and other factors that multiplied the storage requirements per node simulated. Further, if a system were tested under conditions near saturation, message queues might generate sizable, though transient, backlogs that would exhaust the SIMULA freespace pool. All of these factors made simulations involving more than 30 nodes impractical. The second study described above was made with a 30 node network and three priorities of traffic, after removing all statistics collection variables that were not of central interest. Project sponsors felt the 30 node network was sufficiently realistic for the case at hand, but would have liked the ability to investigate larger networks as well. The restricted version of the simulator of the existing Navy Polling Protocol had substantially lower storage requirements, but, naturally, it restricted functionality as well.

CONCLUSIONS

Seven software engineering techniques applied in the design and construction of the simulator were listed in the opening section of this paper. Though all of these techniques proved beneficial to some degree and some of them overlap, the impact of each is reviewed separately below:

Complete design prior to coding — This requirement prevented premature freezing of the design, but was less important than the design reviews.

Design review by knowledgeable outsiders — This technique required that the design be sufficiently documented that others could comprehend it and suggest revisions. Although the reviews that were provided did cause design changes, the exercise of creating the documentation for the design was in itself beneficial. The requirements description that was generated to go along with design was also useful. Perhaps more time should have been devoted to the requirements statement; in retrospect, establishing a better definition of the environment to be modeled (e.g., with respect to channel noise characteristics) and delineating performance requirements might have revealed problems that were not apparent until later in the project.

Use of the information hiding principle in design — The modularization of the simulator based on the information hiding principle was beneficial in several respects: it made the design easier to describe, it forced consideration of future changes to the system early in the design process, it simplified the making of changes, and it led to a structure that provided useful parts to other simulators.

Use of abstract types — This concept, together with the information hiding principle and use of a language supporting abstract types (SIMULA) aided in the organization of the implementation and led to the creation of at least one abstract type (for statistics collection) that is now being used in new simulators under development by other projects.

Code reading by other than the programmer prior to testing — This procedure was only carried out for the first few weeks of the coding. A few bugs were found, but its primary benefit was to force the inclusion of good comments and clean coding practices.

Co-operating sequential processes — Although the use of processes is hardly novel, it was central to this design. If processes had not been available in the programming language chosen for the implementation, it would have been worth the effort to implement them.

Use of pseudo-code — This widely used technique was used to provide initial sketches of code throughout the project. It helped significantly in communicating the design to readers not familiar with SIMULA.

Often, software engineering techniques are argued to be desirable, but time consuming. Although this project did not have firm deadlines, the initial simulator was completed and validated with the slotted ALOHA protocol in less than 6 man-months. The documents and programs produced have continued to be useful over a period of more

than 2 years. Currently, a simulator for HF radio networks is under development by a staff member not involved in the earlier project. Given only the simulator listing and the relevant documents, he has been able to incorporate significant parts of the original simulator into his work.

There is, of course, room for improvement. In particular, if the project were repeated, more time should be devoted to the performance implications (CPU time and, especially, storage requirements) of the design, to the level of detail actually required in the channel model, and to the user interface for parameter specification. Nevertheless, the project succeeded in applying software engineering techniques and in producing a tool for performance analysis applicable to a variety of protocols and able to be used and modified by other persons.

ACKNOWLEDGMENTS

Thanks are due to several members of the Information Systems Staff at NRL for their assistance in this work. Constance Heitmeyer and John Shore participated in many helpful discussions throughout the project. They and David Parnas reviewed the initial design; John Shore reviewed some of the code as well. He, David Weiss, and Honey Elovitz provided helpful reviews of drafts of this paper.

REFERENCES

1. M.A. Crane and D.L. Iglehart, "Simulating Stable Stochastic Systems: III Regenerative Processes and Discrete-Event Simulations," Operations Research 23, 1, 33-45 (1975).

2. G.S. Fishman, "Achieving Specific Accuracy in Simulation Output Analysis," Comm ACM 20, 5, 310-315 (May 1977).

3. H. Kobayashi, *Modeling and Analysis: An Introduction to System Performance Evaluation Methodology*, Addison Wesley, Reading, Mass. (1978).

4. C.H. Sauer, "Confidence Intervals for Queuing Simulations," Performance Evaluation Review 8, 1-2, 36-44 (spring-summer 1979).

5. H.D. Mills, "Software Development," IEEE Trans on Software Engineering SE-2, 4, 265-273 (Dec 1976).

6. D.P. Friedman and G.M. Weinberg, *Ethnotechnical Review Handbook*, Second Ed., Ethnotech Inc., Lincoln, Nebraska (1979).

7. D.L. Parnas, "On the Criteria to be Used in Decomposing a System into Modules," Comm ACM 15, 12, 1053-1058 (Dec 1972).

8. B. Liskov and S. Zilles, "Programming with Abstract Types," *SIGPLAN Notices 9*, 50-59 (April 1974).

9. F.T. Baker, "Chief Programmer Team Management of Production Programming," IBM Systems Journal 11, 1 (1972).

10. E.W. Dijkstra, "Co-operating Sequential Processes," in *Programming Languages*, F. Genuys, ed., Academic Press, New York, 43-112 (1968).

11. B.W. Kernighan and P.J. Plauger, *The Elements of Programming Style*, McGraw Hill, New York (1974).

12. C.E. Landwehr, "On the Design of a Simulator for Satellite Communications," NRL Tech Memo 5403-85:CL:la, March 1977.

13. G. Birtwistle, O.J. Dahl, B. Myrhaug and K. Nygaard, *SIMULA Begin*, Auerbach, Philadelphia (1973).

14. G. Birtwistle and J. Palme, DECSYSTEM 10 SIMULA Language Handbook, Part I. Available at NTIS PB-243 064, (September 1974).

15. W.R. Franta, *A Process View of Simulation*. Elsevier, North Holland, N. Y. (1977).

16. C.E. Landwehr, "An Abstract Type for Statistics Collection," NRL Report 8373, Winter 1980 (submitted for journal publication).

17. D.L. Parnas, "Some Hypotheses About the Uses Hierarchy for Operating Systems, Tech. Rep. Technische Hochschule Darmstadt, 1976.

18. S.S. Lam, "Packet Switching in a Multi-Access Broadcast Channel with Application to Satellite Communications in a Computer Network," Ph.D. Dissertation, Dept. of Computer Science, UCLA, 1974.

19. C.E. Landwehr, "Construction and Validation of the Satellite Communication Simulator for the Slotted ALOHA Protocol," NRL Tecn Memo 5403-259:CL:gls, June 1977.

20. C.E. Landwehr, "Performance Studies of the Distributed CPODA Protocol in the Mobile Access Terminal Network," NRL Memorandum Report 4804, Sept., 1979.

21. M. Melich, C.E. Landwehr and P. Crepeau, "Alternative Satellite Channel Management Strategies," NRL Report to appear, fall 1979.

22. I.M. Jacobs, R. Binder and E.V. Hoversten, "General Purpose Packet Satellite Networks," Proc IEEE 66, 11, 1448 (Nov 1978).

23. N.T. Hsu and L.N. Lee, "Channel Scheduling Synchronization for the PODA Protocol." 1978 Int Conf on Comm Conf Record Vol. 3, 42.3.1-42.3.5.

24. D.L. Parnas, "On the Design and Development of Program Families," *IEEE Trans on Software Engineering SE-2*, 1, 1-9 (March 1976).

25. D.L. Parnas, "Designing Software for Ease of Extension and Contraction," IEEE Transactions on Software Engineering SE-5, 2, 128-138 (March 1979).