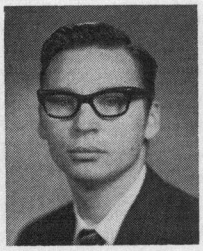


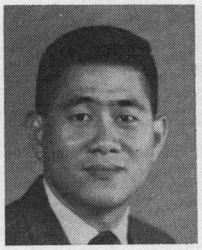
Pittsburgh, Pa., and the Ph.D. degree in electrical engineering from Syracuse University, Syracuse, N. Y., in 1957, 1963, and 1968, respectively.

From 1958 to 1959, on a leave of absence from Westinghouse Corporation, he served in the Signal Corps at Ft. Monmouth, N. J., and Ft. Huachuca, Ariz. After rejoining Westinghouse, he was involved in instrumentation and data logging for an extra-high voltage transmission-line project, and was later engaged in programming a process control computer. In 1963 he joined IBM Corporation, working on the logic design of small computers. During most of 1966 and 1967 he was an IBM Resident Study Fellow at Syracuse University. Returning to IBM in Endicott, N. Y., he joined the Digital Systems Science Department, where his interests include data flow techniques and reliable computer design.

Dr. Langdon is a member of the Association for Computing Machinery, Phi Eta Sigma, Sigma Tau, Tau Beta Pi, Phi Kappa Phi, and Sigma Xi.

❖

William G. Wee (S'66–M'67) was born in Iloilo, Philippines, on September 9, 1937. He received the B.S.E.E. from Mapua Institute of Technology, Manila, Philippines, in 1962, where he was awarded the "President Gold Medal" for attaining the highest grade average of the graduating class in electrical engineering. He received the M.S. and Ph.D. degrees in electrical engineering from Purdue University, Lafayette, Ind., in 1965 and 1967, respectively.



From 1964 to 1967 he was engaged in graduate teaching and research at Purdue University. At present he is a principal Research Engineer of the Systems and Research Division of Honeywell Inc., St. Paul, Minn. His areas of interest include pattern recognition and learning and adaptive systems.

Dr. Wee is a member of Eta Kappa Nu and Sigma Xi.

Reviews of Books and Papers in the Computer Field

DONALD L. EPLEY, *Reviews Editor*

D. W. FIFE, A. I. RUBIN, R. A. SHORT, H. S. STONE
Assistant Reviews Editors

Please address your comments and suggestions to the Reviews Editor: Donald L. Epley, Department of Electrical Engineering, University of Iowa, Iowa City, Iowa 52240.

A. ANALOG COMPUTATION

R68-51 The Design of an Automatic Patching System—D. A. Starr and J. J. Jonsson (*Simulation*, vol. 10, pp. 281–288, June 1968).

"Getting rid of the patchpanel" has long been the dream of analog computer users (and designers). Besides the aesthetic value of eliminating "all those messy wires," there are the more substantial advantages of shortened turnaround time, long-term low-cost problem storage and greatly simplified programming.

An analog computer with automatic patching, operating in conjunction with a digital computer and the appropriate software, opens up the possibility of an automatic programming system in which the user would merely type in his differential equations and let the system scale the problem, assign components, "patch" and check the problem, and display the solution. Such a system would be programmed much like a digital computer with a simulation language, but a typical run would take milliseconds rather than minutes. The operator's freedom to change parameters and see the results immediately would give the system a considerable advantage over conventional simulation languages.

Several attempts have been made to replace the patchpanel with some automatic or semiautomatic switching system (relays, crossbars, static card readers, etc.); but whatever switching scheme is used, the number of switches turns out to be prohibitive. For example, a typical medium-scale computer (150 to 200 amplifiers) has about 650 outputs and 1300 inputs. Any of the outputs may be patched to any of the inputs. To allow all these connections to be made directly through switches requires an array of 650×1300, or about 850 000 switches. Even with modern techniques of mass-production and miniaturization, a system with this many switches would be too bulky and expensive to be practical.

Thus, progress is needed on two fronts: low-cost schemes for packaging the switches economically, and designing the switching system to minimize the number of switches required. This paper, which is a summary of Mr. Starr's M.S. thesis, prepared under the guidance of Mr. Jonsson, discusses the first problem briefly and the second one at greater length.

As far as the hardware problem is concerned, the authors make two useful observations. First, they suggest the use of latching, rather than momentary relays, which eliminates the need for flip-flop storage and also reduces the noise and power-supply problems, since no current need flow through the relay coils to keep them in the "energized" state. Second, they suggest arranging the relay coils in a rectangular array and addressing them by a scheme similar to a

2D core memory. Voltage pulses are generated which put half the necessary energizing voltage on each coil in the same row or column as the relay to be selected. Only the selected relay itself (at the intersection of the desired row and column) receives the full energizing voltage. This system allows much of the selection to be performed in the relay coil itself, thus drastically reducing the need for decoding logic and relay drivers. A matrix of N^2 relays thus requires only $2N$ driving lines, instead of one per relay.

These ideas appear sound, but they are hardly original (as the authors readily admit). Their main concern in this paper is the second half of the problem: reducing the number of switches to a reasonable value. This the authors propose to do by means of two techniques: *modularization* and *concentration*.

The technique of modularization consists of dividing the computer components into small groups, called modules. Each component has only a limited and indirect access to components in different modules, but much more free and direct access to components in the same module. This is the way the telephone company designs its switching system, on the valid assumption that most calls are local calls. (An analogous statement would be "most patched connections use short patchcords.")

Even with modularization, the number of switches is excessive if all components are in use at once. To reduce the number of switches further, the authors introduce the concept of *concentration* based on the assumption that "all customers will not be using the telephone at the same time."

For example, the typical module in the proposed system (about one-twentieth of the machine) contains 58 component outputs and 101 component inputs. Direct connection of each input to each output requires 5858 switches (58×101). This is clearly excessive, since there are twenty such modules in the computer.

To keep the number of switches within reasonable bounds, the authors assume that at most 20 percent of the input and output terminals are likely to be in use at any one time. They divide the 58 outputs into two equal groups (of 29 each) and the 101 inputs into four nearly equal groups (three groups of 25 and one group of 26). Each of these groups has associated with it a small "concentrator" matrix of switches which allow up to 20 percent of the terminals in any group to be connected to trunklines. Additional swtches interconnect the trunklines to complete the connections between inputs and outputs. The entire system uses only 1242 switches, and appears adequate to handle the majority of problems which satisfy the authors' basic assumption (at most 20 percent of the terminals are in use at any one time).

A natural question arises at this point: What is the optimum grouping for the inputs and outputs? The authors have chosen to divide the inputs into two groups and the outputs into four. Would a different grouping produce an adequate system with fewer switches? This type of problem has been analyzed by Clos[1] in connection with telephone network design. Subsequently, this reviewer extended Clos's work and applied it to the automatic patching problem in a study under a contract with the National Aeronautics and Space Administration.[2]

The general line of attack and the results of these studies may be summarized as follows. First, conditions must be found to determine which switch configurations are adequate to handle the proposed traffic; this is a problem in set theory and combinatorial analysis. Second, once the adequate systems are characterized mathematically, one must select from the set of all adequate systems the one that uses the fewest switches. This is the optimal grouping problem described above. It turns out that a few large groups are inefficient and so are many small groups. Between the two extremes lies an optimum which may be found by the usual methods of calculus.

Using the techniques developed in the NASA report and the module size chosen by Starr and Jonsson (58 component outputs, 101 component inputs), a switching scheme can be designed which is adequate for 20-percent traffic density using only 518 switches, instead of the 1242 used by the authors. In fact, a system which is adequate for 50-percent traffic uses only 1145 switches.

Since the NASA report was not available at the time this paper was written, and since Clos's paper was available only in a 14-year old journal, the authors can hardly be blamed for being unfamiliar with the technique. What *is* surprising is that the authors apparently failed to recognize that an optimization problem exists at all. Faced with 58 component outputs and 101 component inputs per module, they arbitrarily propose a particular scheme for interconnecting them. If they ever considered any alternative schemes, this fact is not mentioned in the paper.

In fact, almost every aspect of the proposed design has an arbitrary character. Critical design decisions are made by appeals to intuition. Thus, the authors admit that they selected the number of intermodule trunks "intuitively." Similarly, the assumption of 20-percent usage of terminals was based on an investigation of some "typical analog computer problems" (unspecified by the authors) "mixed with a little intuition." Other decisions, such as the abovementioned grouping of terminals, are made without even appealing to intuition (simply by ignoring the possibility of alternatives).

Of course, intuition is a legitimate and useful tool in designing any new system. The proof of the soundness of a proposed system lies in how well it works, not in how it was obtained. The next step, therefore, is to evaluate the proposed system by programming some actual problems on it. Note that this does *not* require actually building the system. Once the proposed system is specified, that is, once the number of components and their method of interconnection are defined, it is possible to analyze a given problem and search through the proposed switching matrix to see if the required connections can be made. This should be done for a number of problems, including at least one or two that use most of the major computing components in the machine.

Although the authors do not include any such evaluation in their paper, a cursory examination of the structure of the system indicates that it almost certainly is *not* adequate. The main trouble seems to lie in the authors' assumption of 5:1 concentration (i.e., 20-percent traffic density). This assumption means that only 12 of the 58 components in any module can be used in any problem, which drastically limits the usable equipment complement.

For example, the typical module contains twelve pots (two handset, and ten servo-set), two integrators, two inverters, one multiplier, one summer (or high-gain amplifier), and an assortment of comparators, loose resistors and diodes, etc. Consider a program that uses all twelve pots. These twelve pots are presumably connected to amplifiers (e.g., summers and integrators), but it is hard to see what role these amplifiers can play in the simulation since their outputs cannot be connected to anything (the twelve pots use up the entire intramodule trunking system). I conclude that no meaningful problem can possibly use all twelve pots in any module. If this is the case, why not reduce the number of pots, thus saving both switches and analog hardware?

To see what *can* be done within one module of the proposed system, consider a loop that contains the two integrators and the multiplier. Such a loop represents a second-order system with a variable spring stiffness, for example. The system has a total of fifteen outputs (six pots, two integrators, one multiplier, one high-gain amplifier, two inverters, two incoming trunks, and reference voltage for an initial condition). Since there are only twelve trunks available, even this simple system cannot be programmed.

To summarize, the proposed system appears to be neither adequate nor optimal. The main weakness of the paper, however, lies

[1] C. Clos, "A study of non-blocking switching networks," *Bell Sys. Tech. J.*, vol. 32, pp. 406–423, March 1953.
[2] G. Hannauer, "Stored program concept fo: analog computers," Final Rept., NASA project NASA-21228.

not in its results, but in its methodology. The system is designed on the basis of intuition, rather than analysis, and the authors do not even pay lip service to the ideal of evaluating the system by programming real problems on it.

It is rather disappointing that the authors' only "evaluation" of their proposed system consists of counting the number of switches (about 27 000) and pronouncing it "reasonable."

Since the paper under review is only a seven-page summary of an M.S. thesis, one might expect that the detailed analysis and evaluation were merely omitted from the summary for the sake of brevity. This is not the case. The original thesis suffers from the same limitations. It appears that neither the paper nor the thesis itself makes a substantial advance in the state of knowledge about this problem.

GEORGE HANNAUER
Electronic Associates, Inc.
Princeton, N. J.

B. MULTIPROGRAMMING

R68-52 The Structure of the "THE"-Multiprogramming System —E. W. Dijkstra (*Commun. ACM*, vol. 11, pp. 341–346, May 1968). (See also *Computing Rev.*, vol. 14, p. 979.)

This paper is worthy of attention. Written with the facility one has grown to expect from Prof. Dijkstra, this is an announcement of a new operating system designed and constructed at the Technological University at Eindhoven for the EL X8.

While the operational characteristics of the new system are not unusual, the claims for the confidence level of both the design and implementation of the system certainly are. "We have found that it is possible to design a refined multiprogramming system in such a way that its logical soundness can be proved a priori and its implementation can admit exhaustive testing." For one of Dijkstra's stature in the community to make such a statement is to demand the consideration of all of us.

The author's abstract and his "key words and phrases" provide a terse summary of his announcement:

> A multiprogramming system is described in which all activities are divided over a number of sequential processes. These sequential processes are placed at various hierarchical levels, in each of which one or more independent abstractions have been implemented. The hierarchical structure proved to be vital for the verification of the logical soundness of the design and the correctness of its implementation.
>
> Key words and phrases: operating system, system hierarchy, system structure, real-time debugging, program verification, synchronizing primitives, cooperating sequential processes, system levels, input–output buffering, multiprogramming, processor sharing, multiprocessing.

Professor Dijkstra is quite precise in his abstract. Some amplification is offered below.

Two unusual claims are made for the THE system.

1) Its logical soundness was proved a priori (i.e., before implementation).
2) Its implementation admitted exhaustive testing.

Consider them in turn.

A Priori Proof of Logical Soundness: In the THE system a set of abstractions are realized in an hierarchy of "levels." These abstractions concern parallel processes, virtual memory, input–output streams, and other useful facilities. Though each abstraction is important in that it realizes some facility in the context of resource restrictions, the mere concept of *hierarchy*, in conjunction with parallel processes and the means for synchronizing their operation is fundamental to the objective of *proof of design* for the system.

Designed well within Habermann's[1] universe, the THE system enjoys the benefits of "harmonious cooperation." One aspect of such cooperation is the avoidance of "deadlock," or "the deadly embrace," one of the banes of designers of parallel-task systems.

But just how does one achieve a priori proof of design? Granted commitment to a Habermann hierarchy (so as to enjoy the benefits), just how does one *prove* that a particular society composed of mutually synchronized processes does indeed satisfy all requirements in its time behavior? Dijkstra is not explicit here. He presents only the announcement that he and his compatriots have "learned the art of reasoning" by which this was possible.

Admission of Exhaustive Testing: Given a correct design, how does one achieve a valid (correct) realization of it? Dijkstra's technique is that of dividing and conquering. One relies on the relative simplicity of functions on a given level, plus the ability to maneuver a subsidiary level, using cooperating sequential processes, to search out, establish, and demonstrate the correct operation of the system in each of its "relevant states." In effect, one attempts to design such that the combinatorial aspects of the operations are additive, rather than multiplicative.

The author's argument for exhaustive testing relies on our confidence in the perception of the designers on one hand, and of the testers on the other. The designers must keep the relevant states on each level to a manageable number. The testers must identify all relevant states, and devise tests (also to be verified) that 1) maneuver the system into each relevant state, and then 2) demonstrate the correct operation in that state. Detailed arguments are not provided in Dijkstra's paper.

Bonuses: Two elegant new primitives (the "P" and "V" functions) for the coordination of cooperating sequential processes are briefly described in an appendix to the paper.

A brief sketch of the proof of "harmonious cooperation" leads us to Habermann's work: apparently significant, and hopefully available in the general literature soon.

CODIE WELLS
The MITRE Corp.
Bedford, Mass.

[1] N. A. Habermann, On the harmonious co-operation of abstract machines, Ph.D. dissertation. The Netherlands: Technological University of Eindhoven, 1967.

C. INTERACTIVE GRAPHICAL PROGRAMMING

R68-53 A System for Interactive Graphical Programming—W. M. Newman (*1968 Fall Joint Computer Conf., AFIPS Proc.*, vol. 32. Washington, D. C.: Thompson, 1968, pp. 47–52).

This paper describes a scheme for organizing and creating the control portions of an interactive graphical program. Since the response of such a program to some console input may depend on the history of previous inputs, the author structures his control programs as finite-state automatons. Thus state diagrams are used as the notation for defining the behavior of an interactive graphical program. Such notation does indeed add clarity and order to the description of this information.

The control concepts presented have been implemented in a working system. A written description of the state and transition