Future trends in software development for real-time industrial automation

by H. E. PIKE

General Electric Manufacturing & Process Automation Advanced Development Operation
Lynn, Massachusetts

INTRODUCTION

"The ______ is a desk-size, stored program computer. It has a medium-scale capacity and uses a single address system. The ______ has all the advantages of high component reliability, automatic operation, and ease of programming." ¹

These words are from the introduction of the operations manual for one of the first minicomputers—the "LGP 30 Royal Precision Electronic Digital Computer." The manual was written in 1959. In twelve years we have just barely begun to understand the impact of this size computer upon the industrial automation.

In this paper we will examine future trends for programming for process control and real-time applications of industrial automation. Although blind forecasting is exciting, we will take the more conservative approach of examining past history and our current environment for factors which will lead to future developments. A short definition of the scope of applications considered will be followed by examination of current technological and economic trends. A brief recap of the companion survey paper by Schoeffler ² will serve to establish the current state-of-the-art and a survey of current active development and/or standardization activities will be followed by predictions for the future. These predictions will be made from the fundamental viewpoint that future developments are determined by three factors: the current state-of-the-art, current problem areas, and new technological tools available for reduction to practical use. An underlying assumption is that there is or will be sufficient economic motivation to undertake the developments required, with timing being the only uncertainty.

RANGE OF APPLICATIONS

It is necessary to be more specific about the range of applications to be discussed. Future trends in real-time industrial automation will be discussed, rather than the more encompassing possibility of all real-time applications. Not included in our considerations are the applications typically called command and control, teleprocessing and interactive management information systems (certain portions of our discussions will, of course, apply to these areas).

Typical real-time industrial automation applications span a range from direct numerical control to load flow control and economic dispatch in the electric utility industry. Also included are direct digital control of chemical processes, supervisory control of chemical processes, sequencing control on assembly lines, control of batch chemical processes and computerized numerical control (see References 3-13).

In direct numerical control a small, fast, dedicated computer is used to control a machine tool directly. Typical required response times are on the order of milliseconds. Load flow control calculations on the other hand involve large data-handling requirements because of large matrix inversions, and involve response times of minutes.¹⁴

In direct digital control the computer directly manipulates the positions of control elements such as valves without an intermediate mechanical control device. Direct digital control requires response times on the order of seconds to disturbances in the controlled variable.¹⁵,¹⁶,¹⁷

In supervisory control the computer is used to provide setpoints which are then actually controlled by hard-wired analog equipment. In this case, the real-time response capability of the control system is on the order of minutes.

We could go on, but the point is clear: there exists a tremendous range of application requirements in this area which we have labeled industrial automation.

PRESENT STATE-OF-THE-ART

Schoeffler ² has assessed the current available tools for software development. To summarize, in practical
916 Spring Joint Computer Conference, 1972

AVERAGE PRICE PER FUNCTION (¢)


Figure 1—Digital integrated circuit cost trends

use today we have as wide a range of software approaches as there are applications,18,19 about which the following statements seem to hold.

- Most of the real-time programming is still done in assembly language.
- Various dialects of Fortran have been tried, with mixed results. They have been most successful where a high content of engineering calculations is included (References 20 through 27).
- Initial experiences with problem-oriented programming systems appear very promising, particularly when two conditions are met:
  —functional similarity between installations, allowing a reasonable match between programming system and control problem (References 28 through 38).
  —sufficient economic base to merit development of general purpose packages.
- Various attempts at general purpose real-time languages have shown the concept to be feasible, but have remained as academic exercises, in part because of the inherent restrictions due to programming architectures of current computers (References 39-47).

CURRENT TRENDS

Two current trends will combine to motivate future development of improved programming techniques: microelectronics, and recognition of the cost and complexity of the software production process.

Microelectronics

The dramatic downward trend in price/function in microelectronics is widely recognized. Figure 1 is typical of the current extrapolations of this trend. An easily customized MOS/LSI desk calculator chip has been recently announced at a reported volume price of less than $20.48. Our colleagues in the hardware design area confirm that this device is comparable in logic power (in terms of gate equivalents) to the LGP/30 of ten years ago discussed above.

If this isn't enough, a major manufacturer has announced a CPU on a chip, of perhaps five times the complexity of the LGP/30, and our R&D associates advise us that such devices with electrically alterable ROM’s are feasible, for the ultimate in flexibility.

The message is clear—Software functions will disappear into hardware (or firmware) and modest investments in increased hardware capability (if made correctly) will reap outstanding software cost savings.

The software development process

Another important trend is increasing recognition of the difference between “programming” and “software development.” Examining the following breakdown of the software production process makes this distinction clear:

1. Problem analysis
2. System structure design
3. Logic design
4. Program coding
5. Language processing
6. Unit test and debug
7. System test and debug
8. Installation
9. Maintenance

Although we all agree that the cost of software development exceeds the cost of hardware, we cannot find any common agreement as to particular areas of
concentration of cost within this breakdown of the software development process.

CURRENT ACTIVE DEVELOPMENT AND STANDARDIZATION ACTIVITIES

A number of industry-wide and university related standardization and development activities are taking place which will influence trends for process control programming. Among these are the Purdue Workshop on Standardization of Industrial Computer Languages, the American National Standards Institute’s PL/I Standardization and Development effort, and several university programs in industrial automation.

Purdue workshop

The Purdue Workshop on Standardization of industrial computer languages has stated as its objectives:

1. To make procurement, programming, installation and maintenance of industrial computer systems more efficient and economical through the development and standardization of industrial computer languages.
2. To minimize educational and training requirements for the use of industrial computer languages.
3. To promote program interchangeability of application programs between different computer systems.

The standing committees of the workshop include:

1. Long Term Procedural Language Committee
2. Problem Oriented Language Committee
3. Glossary Committee
4. Fortran Committee
5. Steering Committee

The workshop met first in the spring of 1969 and approximately every six months thereafter.

AMERICAN NATIONAL STANDARDS INSTITUTE SUB-COMMITTEE X3J1

X3J1 is a sub-committee of Committee X3 of the American National Standards Institute whose scope is the proposal of a draft American National/ISO Standard Composite Programming Language based on PL/I. X3J1 is organized into four working subcommittees which are responsible for:

1. Standardization
2. Development
3. Subsets
4. Development for Industrial Computers

The scope of Committee X3J1.4, PL/I Development for Industrial Computers, is the development of a version of PL/I suitable for application to process control and manufacturing automation by defining and fulfilling the functional requirements of this application area. A basic rule governing this activity is that any string of characters which has validity in the language being developed and the PL/I that X3J1 proposes as a standard must have the same semantics.

X3J1.4 and the Long Term Procedural Language Committee of the Purdue Workshop are currently cooperating to determine if the production of a single result is feasible.

UNIVERSITY AND OTHER ACTIVITIES

Various university laboratories both within the United States and abroad are active in the development of real-time languages and/or programming techniques for industrial computers.

Some of the germinal early real-time language work in the United States was done at Case-Western Reserve University in their Systems Research Center. The Purdue Laboratory for Applied Industrial control continues as sponsor of the Purdue Workshop. Results of significant interest have appeared from the University of Toronto, and German, French and British working groups.

FUTURE TRENDS IN SOFTWARE DEVELOPMENT FOR REAL-TIME INDUSTRIAL AUTOMATION

We have noted above a number of forces which are currently at work to influence future trends:

1. Microelectronics—a capability to produce very powerful central processors at reduced cost.
3. A current proliferation of programming techniques which are more or less satisfactory, depending upon their match to the application for which they are used.
4. A number of currently active development and standardization activities working in areas which range from customized problem-oriented languages to development of real-time version of PL/I.

Consideration of these forces leads us to believe that the following trends will become evident in the future.
1. Computers with architectures more oriented to the use of special purpose real-time programming languages.
2. Hidden computers.
3. High-level real-time languages.
4. Problem-oriented languages.
5. Software factories combining sophisticated languages and debugging tools for the more effective and efficient production of software on host computers.

**New real-time computer architectures**

Because real-time computers are special purpose rather than "general purpose" machines, a trend will occur toward more customized architectures, particularly from a programming point of view. Characteristics of these new architectures are likely to include:

1. **Bit addressability**
   
   Arrangement of data in a real-time computer in quanta of 8 bits is not very meaningful in the real-time environment. Character oriented data is manipulated relatively seldom in a real-time environment. Convenient methods of accessing data fields of varying widths will be required for convenient implementation of data structures, for example.

2. **Direct implementation of real-time features**
   
   Real-time features such as semaphores, more powerful interrupt structures with less inertia, and direct hardware support of events will ultimately be implemented directly in the hardware.

3. **Control structures**
   
   Because sequencing problems represent a high percentage of real-time calculations, the instruction repertoire of real-time computers will become more oriented to the easy implementation of more sophisticated control structures.

4. **More sophisticated addressing techniques**
   
   Although recursion is not high on the list of requirements for real-time applications, the ability to handle reentrancy in a concise manner is important. This and other requirements such as the manipulation of inhomogeneous data aggregates will dictate more sophisticated addressing modes.

When these more powerful architectures will occur is not entirely evident at this time. It seems that an orderly approach would be to pursue a more consistent definition and understanding of the true real-time language requirements and then let these architectures be designed to support such languages (Refs. 50 through 60).

**Hidden computers**

The capability to build special purpose hardware will, in the future, increase the trend toward distributed digital systems. The computational capability of real-time process control systems will not be concentrated in one central processor as is typical now but will appear both in the central processor and a number of remote dedicated processors which function to gather intelligence about the process under control, execute direct control functions and communicate to the central processor for supervisory control functions. The major force which will cause this to occur is what is sometimes called "the cost of copper." This expression refers to the fact that in many installations it is possible to spend as much installing the wiring to carry signals from sensors and control devices back to the computer as on the computer itself. The impact of this trend upon programming techniques is that these devices will, to all intents and purposes, be hidden from the user. They will perform their functions in a manner analogous to hardwired hardware devices in current use, and the programming problems associated with direct control will frequently be segregated from the rest of the system. As an example of the importance of this, it has been found that in the development of large systems for both supervisory and direct digital control that such major differences in operating systems are required to support these two control techniques as to make organization of a system with a combination of them rather difficult and costly. The use of hidden computers will reduce this problem significantly.

**Problem oriented languages**

For those users who have applications of some generality, problem oriented languages and systems seem to be unquestionably the best long term solution to their software development problems. There is no question but that a truly general package incurs a penalty in overhead and excess hardware requirements, but with the cost trends in hardware which we have discussed, it will become more and more apparent that this is the most economical solution. As an example, our experiences with certain problem oriented packages have been that they can reduce the programming requirements by a factor of 10 and allow personnel whose major area of specialty is the application rather than software to do the programming. This usually results in a much better control system. Thus, on the belief...
that what is best for the user will eventually ensue, we predict that for many of them, their problems will be solved by problem oriented systems which will very much minimize their requirements for a professional programming staff.

Real-time procedural languages

As we have seen, the major real-time languages in current use are variants of Fortran. Table I shows some of the typical extensions which have been made in Fortran for this application area. It is clear from examining the length and breadth of these extensions that if an all-encompassing procedural language based on Fortran were to be implemented, the base language would disappear. For this reason, current development activities are now turning in a different direction.

The general goals of real-time procedural language development activities under way today are to develop a language which leads to:

1. Lucidity—the clear expression of what is to occur when a program is executed, providing a higher level of self documentation.
2. Freedom from side effects—in a real-time environment a predictable system behavior is extremely important and real-time programming languages must be free from side effects.
3. Enforcement of programming discipline—in the business of producing software, it is clear that programming discipline is increasingly important. A well designed language will help maintain this discipline.
4. Natural modularity—the language must be constructed so as to make easy increased modularization of real-time systems.
5. Ease of learning and use—it must not require a major investment in training to put a new language into use, if possible.

Also of particular interest to the designer of a real-time procedural language is the maintenance of a proper balance between run time efficiency, integrity

<table>
<thead>
<tr>
<th>TABLE I—Typical FORTRAN Extensions</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 Program &amp; Data Structure</td>
</tr>
<tr>
<td>(a) Data types</td>
</tr>
<tr>
<td>Fixed point</td>
</tr>
<tr>
<td>Bit</td>
</tr>
<tr>
<td>Status</td>
</tr>
<tr>
<td>Byte</td>
</tr>
<tr>
<td>Binary</td>
</tr>
<tr>
<td>Boolean (vector)</td>
</tr>
<tr>
<td>Octal</td>
</tr>
<tr>
<td>Hexadecimal</td>
</tr>
<tr>
<td>Alphanumeric</td>
</tr>
<tr>
<td>(b) Data manipulation</td>
</tr>
<tr>
<td>Shifts</td>
</tr>
<tr>
<td>Logical operators</td>
</tr>
<tr>
<td>Bit and byte replacement</td>
</tr>
<tr>
<td>Character manipulation</td>
</tr>
<tr>
<td>String manipulation</td>
</tr>
<tr>
<td>Bit testing</td>
</tr>
<tr>
<td>2 Communications</td>
</tr>
<tr>
<td>(a) Between programs</td>
</tr>
<tr>
<td>Global common</td>
</tr>
<tr>
<td>External data</td>
</tr>
<tr>
<td>Static file system</td>
</tr>
<tr>
<td>Dynamic file system</td>
</tr>
<tr>
<td>(b) Input/Output</td>
</tr>
<tr>
<td>Unformatted input/output</td>
</tr>
<tr>
<td>Read write bulk</td>
</tr>
<tr>
<td>3 Program control</td>
</tr>
<tr>
<td>Schedule another program</td>
</tr>
<tr>
<td>Link with interrupt</td>
</tr>
<tr>
<td>Delay</td>
</tr>
<tr>
<td>Look at clock</td>
</tr>
<tr>
<td>Terminate execution</td>
</tr>
<tr>
<td>Decision tables</td>
</tr>
<tr>
<td>4 Compiler features</td>
</tr>
<tr>
<td>Diagnostic trace</td>
</tr>
<tr>
<td>Conditional compilation</td>
</tr>
<tr>
<td>Reserved words</td>
</tr>
<tr>
<td>Code optimization directives</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>TABLE II—Some Functional Requirements for a Real-Time Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>I. INTERRUPT HANDLING &amp; SYNCHRONIZATION</td>
</tr>
<tr>
<td>A. Interrupts</td>
</tr>
<tr>
<td>1. Ability to indicate that some program segment is to be executed upon the occurrence of some particular interrupt.</td>
</tr>
<tr>
<td>B. Events</td>
</tr>
<tr>
<td>1. Ability to retain the flow of control at a point in a computational process until some event has occurred.</td>
</tr>
<tr>
<td>2. Ability to cause the occurrence of an event at some point in a computational process.</td>
</tr>
<tr>
<td>C. Synchronization</td>
</tr>
<tr>
<td>1. Ability to synchronize parallel computational processes with security.</td>
</tr>
<tr>
<td>II. TASKING</td>
</tr>
<tr>
<td>A. By a task on itself</td>
</tr>
<tr>
<td>1. Schedule Execution</td>
</tr>
<tr>
<td>2. Find Status</td>
</tr>
<tr>
<td>3. Exit</td>
</tr>
<tr>
<td>4. Kill</td>
</tr>
<tr>
<td>5. Delay</td>
</tr>
<tr>
<td>6. Wait</td>
</tr>
<tr>
<td>B. On another task</td>
</tr>
<tr>
<td>1. Execute</td>
</tr>
<tr>
<td>2. Schedule</td>
</tr>
<tr>
<td>3. Suspend</td>
</tr>
<tr>
<td>4. Delay</td>
</tr>
<tr>
<td>5. Terminate</td>
</tr>
<tr>
<td>6. Set Priority</td>
</tr>
<tr>
<td>7. Find Status</td>
</tr>
</tbody>
</table>
of the system and ease of programming. A general set of functional requirements for a real-time language is shown in Table II. Whether these languages will be general purpose or dedicated to particular architectures remains to be seen.

**Software factories**

Just as the ultimate elimination of the requirements do a great deal of custom programming by the use of special purpose application packages is the ultimate answer in reducing software costs for the users of real-time automation systems, the development of a unified, well-designed software will provide the mechanisms for more economical development of such systems. Figure 2 shows the general nature of such a software factory. It consists of four major elements; a compiler for a high-level language for systems software development, a utility library of building block software modules, a well organized approach to debugging software produced in the software factory, and a powerful host computer upon which the system is run.

The general-purpose real-time language described above will be used with the debugging system to develop the utility library and various application packages.

**Debugging system**

One of the major motivations for using a large host computer for software production is the more powerful facilities which it makes available for software debugging. The well designed debugging system for the software factory will attack three problem areas:

1. **Observability**—to make more visible the actions of the program being tested.
2. **Controllability**—to allow the programmer to control the flow of control within his program to suspect control paths.
3. **Repeatability**—to insure that the same response occurs from given stimuli, aiding problem diagnosis.

Table III lists the features generally used in such a system to achieve these goals.

There is still some disagreement between the use of source language debugging through compilation to host machine executable programs, and direct instruction simulation of the target computer. Each point of view has merit. The source language execution approach allows speedier execution and provides for quick elimination of gross structural errors in the program. Simulation of the target computer at the instruction level is generally more expensive because of increased execution requirements on the host computer, but allows one to locate more subtle errors, particularly those with respect to behavior of the target machine hardware in unusual timing circumstances. (The latter requires, of course, an extremely sophisticated simulation of the target computer system.) In the future both techniques will play an important role (References 63-68). (It is interesting to note, however, that the ultimate distinction may be moot as target machines come closer and closer to executing high level languages directly.)

**Utility library development**

Perhaps the most significant aspect of this overall software factory will be the availability of software modules for use in building real-time systems. The

---

**TABLE III—Features Required in Debugging Systems**

<table>
<thead>
<tr>
<th>Feature</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Breakpoint removal and insertion</td>
</tr>
<tr>
<td>2. Online variable revision</td>
</tr>
<tr>
<td>3. Insertion and removal of snapshot dumps</td>
</tr>
<tr>
<td>4. Execution initiation or transfer of control to any point</td>
</tr>
<tr>
<td>5. Fault trapping</td>
</tr>
<tr>
<td>6. Incremental addition of new instruction</td>
</tr>
<tr>
<td>7. Variable searches by value</td>
</tr>
<tr>
<td>8. Subroutine call and operating system linkage tracing</td>
</tr>
<tr>
<td>9. Input/output and interrupt simulation</td>
</tr>
<tr>
<td>10. Control of internal timing</td>
</tr>
</tbody>
</table>
modules required may be grouped in four categories:

1. System building modules
   These modules will form the components of various language processing systems desired by the user. With the use of these modules, the development of special purpose language processors for various applications areas will be made much more economical.

2. Application modules
   Modules of on-line application software will be available for combination into application-oriented control software packages. Examples of these modules will be software for direct digital control, supervisory control, and real-time data base management.

3. Systems operating modules
   Operating systems as we know them today may very well disappear in the future. Instead, the various operating system components like interrupt handlers, generalized input/output systems, priority schedulers and executives will be available for combination as required.

4. User interface modules
   Customized user interface modules will be available for each application area. These modules will actually include the linkages necessary to combine the library modules and any specially developed software into a functioning control system.

SUMMARY AND DISCUSSION

We have examined a number of interesting trends in real-time software development. Of particular interest is the three-way interaction foreseen between the development of new, more powerful architectures for real-time computers, more powerful high-level programming languages, and the integration of these programming languages into a more efficient software production system.

The most interesting technical challenge in this situation is the opportunity to develop languages and architectures in a unified manner. In the past the designer of computer architectures has been far removed from consideration of the application requirements, and his products have been extremely difficult to apply. The designer of real-time programming languages now has the opportunity to act as a coordinator in this overall process while insuring that the language fulfills the functional requirements of the application area hand, he will coordinate with the compiler developer and computer architecture designer to insure that the resultant combination of languages, software factory, and computer form an integrated system for the efficient solution of real-time automation problems.

Exact timing of these trends is very hard to forecast. Working prototypes of most of the items discussed exist today. When the economics of the marketplace are satisfactory, these prototypes will be turned into smoothly functioning systems for general use. Development of languages and architectures are already under way. As this work matures the software factory concept will provide the unifying factor which knits this all together.

REFERENCES

1. LAP-30 Royal Precision Electronic digital computer operations manual
   Royal McBee Corp 1959
2. J C SCHOEFFLER
   The development of process control software
   Proceedings of the 1972 Spring Joint Computer Conference
3. Proceedings of the IEEE-special issue on computers in industrial process control
   Vol 58 No 1 January 1970
4. C L SMITH
   Digital control of industrial processes computing survey
   Vol 2 No 3 September 1970
5. A S BROWER
   Digital control computers for the metals industry
   ISA J Vol 12 pp 51-63 February 1965
6. B O JENDER
   Problems in computer control of bleach plants
   Paper Trade J pp 48-50 May 26 1969
7. D S HOAG
   Computer control of a Kamyr digester
   Presented at the 2nd Ann Workshop on the Use of Digital Computers in Process Control Louisiana State University Baton Rouge La March 1967
8. M R HURLBUT et al
   Applications of digital computer control to the cement manufacturing process
   International Seminar on Autom Control in Lime Cement and Connected Industries Brussels Belgium September 1968
9. T H LEE G E ADAMS W M GAINES
   Computer process control: Modeling and optimization
   Wiley New York 1968
10. T M STOUT
    Where are process computers justified?
    Presented at the International Symposium on Pulp and Paper Process Control Vancouver Canada April 1969
11. T M STOUT
    Process control
    Datamation 12 2 Feb 1966 pp 22-27
12. T J WILLIAMS
    Economics and the future of process control
    Automatica 3 1 October 1965 pp 1-13
13. T J WILLIAMS
    Computer systems for industrial process control, A review of progress, needs and expected developments
    Proc 1968 IFIP Congress
On-line computer applications in the electric power industry
Proceedings of the IEEE Jan 1970 Vol 58 No 1 pp 78-87

Guidelines and General Information on User Requirements Concerning Direct Digital Control
Control Instrument Society of America Pittsburgh 1969 pp 251-258 and 259-278

Direct digital control—A survey
23rd Annual ISA Conf and Exhibit New York October 1966 Preprint 68-840

Progress in direct digital control
Instrument Society of America Pittsburgh Pa 1969

Process control software
Proceedings of the IEEE Jan 1970

Real-time software for industrial control

CDC 1700 Fortran for process control

FORTRAN for process control
Instr Technol April 1969

Protran, a fortran based computer language for process control
Automatica 6 No 4 p 555 1970

Extended fortran for process control
IEEE Trans Indus Electronics and Control Instrm IECI-15 No 2 p 75 1968

Extended FORTRAN for process control
Presented at the Joint Automatic Control Conf 1968

An integrated hardware/software approach for process control
Presented at the ISA Ann Conf New York NY 1968

Programming industrial control systems in fortran
IFAP/IFIP Symposium Toronto 1968

FORTRAN IV in a process control environment
Presented at the Joint Automatic Control Conf 1968

Special issue on computer languages for process control

Fill-in-the-form Programming
Control Engineering May 1968

Digital Equipment Corporation Indac 8 DEC Maynard Mass 1969

New process language uses English terms
Control Eng 15 No 10 p 118 1968
Workshop on standardization of industrial computer languages
Purdue University Lafayette Indiana Feb Sept 1969
March-Nov 1970 May 1971

50 L CONSTANTINE
Integral hardware/software design
A multi-part series that appeared in Modern Data from April 68 through February 69

51 F S KEELER et al
Computer architecture study
SAMSO report TR-70-420 available from NTIS as AD720798

52 C C CHURCH
Computer instruction repertoire—Time for a change
Proceedings of the 1970 Spring Joint Computer Conference

53 H WEBER
A microprogrammed implementation of Euler on the IBM 360/50
CACM 10 1967 549-558

54 B6500 information processing system reference manual
Burroughs Corporation Manual #1043676

55 R RICE W R SMITH
SYMBOL—A major departure from classic software dominated von Neumann computing system
Proceedings of the SJCC 1971 3, 8 AFIPS Press pp 575-587

56 C MCFARLAND
A language-oriented computer design

57 P S ABRAMS
An APL machine

58 T F SIGNISKI
Design of an algol machine
Computer Science Center Report 70-131 University of Maryland Sept 1970

59 G W PATTERSON et al
Interactions of computer language and machine design
Moore School of Electrical Engineering Report No 62-09
University of Pennsylvania Oct 1962

60 R W DORAN
Machine organization for algorithmic languages
Proceedings of the International Computing Symposium
Bonn Germany May 1970 pp 364-370

61 F J CORBATO
PL/1 as a tool for system programming
Datamation 15 1969 pp 68-76

62 N WIRTH
PL/360—A programming language for the 360 computers
Journal of the Association for Computing Machinery
Vol 15 #1 Jan 1968

63 D T ROSS
The AED approach to generalized computer-aided design
Proceedings of the 1967 ACM National Meeting pp 367-385

64 D T ROSS
Fourth generation software: A building-block science replaces hand-crafted art
Computer Decisions April 1970

65 R M SUPNIK
Debugging under simulation
Applied Data Research Inc Princeton NJ

66 MIMIC user’s guide—PDP-11
Manual CSD 03-70411M-00 Applied Data Research Inc Princeton NJ

67 M L GREENBERG
DDT: Interactive machine language debugging system reference manual
NTIS Report AD707406

68 W KOCHER
A survey of current debugging concepts
NASA report CR-1397 available from NTIS as N69-35613

From the collection of the Computer History Museum (www.computerhistory.org)