An Introduction to the Programmer's Workbench

T. A. Dolotta
J. R. Mashey

Bell Laboratories
Piscataway, New Jersey 08854

Keywords: Program development facility, UNIX.

Abstract: The Programmer's Workbench (PWB) is a specialized computing facility dedicated to satisfying the needs of developers of computer programs. The PWB might well be called a "human-end" computer; like "front-end" and "back-end" computers, it improves productivity by efficient specialization. It provides a convenient working environment and a uniform set of programming tools to a diverse group of programming projects. These projects produce software for various "target" computers, including IBM System/370 and UNIVAC 1100 systems of much greater size than the PWB machines. The projects range in size from several people up to several hundred. The first PWB machine was installed in October, 1973; usage, acceptance, and interest have grown rapidly since that time. The PWB currently supports about 110 time-sharing terminals, utilizing a network of four DEC PDP-11 computers, all running the UNIX Time-Sharing System. The PWB adds tools to UNIX to support large projects. This paper gives an overview of the PWB and its development; further details appear in the five following companion papers [BIA76A, DOL76B, KNU76A, MAS76A, MAS76B].

1. INTRODUCTION

The Programmer's Workbench (PWB) is a specialized computing facility dedicated to supporting large software development projects. Although it performs many of the functions that have been suggested for projects such as the Program Support Library [LUP74A], the Development Support Library [BAK75A], the Automated Project Management Information System [BRA75B], or the Software Factory [BRA75A], some of its goals are quite different, and its implementation is even more so. Each of these systems may be categorized as a program development facility (PDF), i.e., a system or package of programs meant specifically to support software development.

The PWB is a production system that has been used for several years in the Business Information Systems area of Bell Laboratories. It supports a user community of about 700 people, and can handle 110 simultaneous on-line users. Although the PWB is still growing and evolving at a rapid rate, accumulated experience provides strong support for the viability, economy, and effectiveness of its approach.

This paper explains the rationale for the existence of the PWB and describes its history and current status. It also serves to supply the background for five companion papers, which, in turn, supply examples and evidence that support the assertions and opinions presented here.

2. THE PWB CONCEPT

The PWB concept embodies the following ideas:

- Program development and execution of the resulting programs are two radically different functions. Much can be gained by assigning each function to a computer best suited for it. Thus, as much of the development as possible should be done on a computer dedicated to that task, while execution should occur on another computer, called a "host" or "target" system.
- Although there may be several target systems, possibly supplied by different vendors, the PDF should present a single, uniform interface to its users. Existing PWB targets include IBM System/370 and UNIVAC 1100 computers; in some sense, the PWB is a target also, in that it is built and maintained with its own tools.
- A PDF should be implemented on several computers of moderate size, even when the target machines consist of very large systems. The rationale for this idea is given in sections 4.1 and 4.6. The PWB currently is implemented as a network of four large DEC PDP-11's (three 11/45's and one 11/70), all operating under the UNIX Time-Sharing System [RIT74A].

Although the PWB is a special-purpose system in the same sense as a "front-end" or "back-end" [CAN74A] computer, its goal is to be a "human-end" computer. As shown in Figure 1, it provides the primary interface between program developers and their target computer(s). Unlike a typical "front-end," the PWB supplies a separate, uniform environment in which people perform their work.

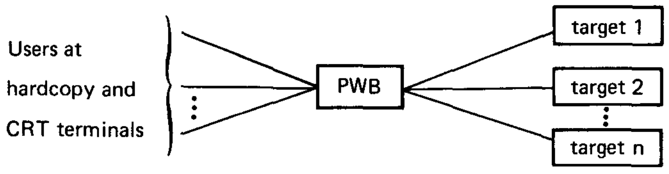

Figure 1. PWB Interface for Users

3. DISADVANTAGES—REAL AND IMAGINARY

The PWB approach of using small computers to build a PDF for use with much larger targets has both good and bad points. Fortunately, many of the potential disadvantages that caused concern at the outset of the PWB project have not materialized, and the actual disadvantages are far outweighed by the benefits [IVI75A]. The following points apply mainly to our implementation of the PWB; some may have little relevance for other PDFs.

3.1 Additional Hardware Costs

The first disadvantage is that of requiring more computer hardware and communications gear. This is a real disadvantage only if one has target systems with excess capacity. In our case, the reverse was true: overall hardware costs were probably decreased by using several minicomputers instead of obtaining additional large systems or upgrading those already installed. An associated (and real) problem is that of the overhead of separate contracts, maintenance, training, etc.

3.2 Increased Complexity of Total System

The second disadvantage is the increased number of nodes and links in the resulting network. There is a potential for disaster if failure of a few components can make the rest unusable. This can (and has sometimes been) a real problem, but is lessened, if not completely eliminated, by providing appropriate redundancy, and by configuring the overall system so that individual components can perform useful stand-alone work. As an example, when the PWB is up, users can edit programs and queue jobs for later transmittal to a target, even if that target or the communications links to it are down.

3.3 Data and Function Splitting

The third problem is that of splitting data and capabilities between machines. Communication links must be fast enough to avoid unacceptable delays. Fortunately, it turns out that many types of data need exist only in one place. For example, the PWB concentrates on the handling of text files containing source programs, control statements, and documentation. A target only keeps such things temporarily, but it does keep its own object programs, which are seldom, if ever, seen by the PWB.

3.4 Incompatibilities

Incompatible character sets and the resulting data conversion costs are the fourth problem. In practice, this has been at most a minor irritation. It may not be a problem at all if target and PDF computers are chosen to use the same character sets.

3.5 Load Balancing

The fifth disadvantage is the possible loss of flexibility in load balancing, because some resources are dedicated to specific tasks and cannot be used for others.

3.6 Limits on Size and Speed

Large systems are capable of many actions far outside the range of small ones. For example, as currently implemented, the PWB does not offer interactive debugging on the target systems.

3.7 Duplication of Software

The final problem is that of the duplication of software, not only between the target and PWB, but between separate PWB processors. For example, each PWB CPU has an independent copy of the system software. However, this software is relatively small, and consumes much less space than user files and programs.

Another aspect of the problem is the need to distribute and coordinate the installation of software on separate systems. One compensation is the ability to test new or modified programs (including the operating system) on one system before installing it on all the others.

4. ADVANTAGES

The PWB approach is no panacea for many of the problems facing the software industry today. However, it appears to be very effective in the presence of certain conditions, each making the PWB approach advantageous. It is hardly accidental that the PWB originated at an installation where *all* these conditions exist.

4.1 Gain by Effective Specialization

The computer requirements of software *developers* often diverge quite sharply from those of the *users* of that software. This observation seems relevant to many kinds of software projects, but is especially true in the production of large, interactive, database-oriented systems. In our opinion, the following are some of the primary needs of software developers:

- Convenient, inexpensive, and continually available interactive computing services.
- A file structure oriented to interactive use, as opposed to one grafted as an additional layer onto a batch-oriented structure. The overall structure and individual files must be quickly changeable, should avoid unnecessary involvement with hardware details, and should *never* require explicit allocation of storage. This observation derives from watching program developers waste much effort on managing disk space, rather than doing their work.
- Powerful, human-engineered, and surprise-free tools for scanning and editing text files. Much programming activity consists of manipulation of text.
- A flexible and easy-to-use command language. It should be a programming language in its own right, but still simple enough to allow quick learning.
- Extensive document preparation facilities.
- Facilities to help solve *small* data base management problems quickly and cheaply.
- Adaptability to rapid organizational and personnel changes.
- Guaranteed service during normal working hours. PWB users can survive more easily the temporary loss of a target machine or communication links than the loss of the PWB itself.

On the other hand, users of the end product may have any or all of the following needs:

- Hardware large and fast enough to run the end product, possibly under stringent real-time and deadline constraints.
- File structure and access methods that can be optimized to handle large amounts of data.
- Transaction-oriented teleprocessing facilities.
- The use of a specific type of computer and operating system, in order to meet any one of a number of possible, externally-imposed requirements.

Although, in principle, it is possible to conceive of a system that satisfies both of the above sets of requirements in a cost-effective way, we are not aware of the existence of such a system. If it is necessary to choose one set over the other, the developers often lose the battle because the end users are usually paying for the system. Given the fact that software costs already exceed hardware costs, and are expected to do so to an even greater extent in the future [BOE73A, DOL76C], this may be a very unwise approach in the long run.

Given the prevailing costs and natures of large and small computers, it would appear that small ones are much more suited to the usual needs of program developers. Furthermore, assuming that a PDF should have duplicated systems in order to guarantee service, it seems logical to duplicate small, cheap computers instead of large, expensive ones.

4.2 Multi-Vendor Installations

It is desirable to have a uniform set of tools in order to ease training and to permit the transferral of personnel between various projects. The creation of such a set of tools is made difficult by the differences in file structures, command languages, and communications protocols of the various targets. These differences are very troublesome; they are qualitatively different from problems encountered in transferring, say, COBOL programs between various target systems. Nevertheless, they must be dealt with. As a result, a few PWB tools are target-specific, but, wherever possible, they possess target-independent user interfaces.

4.3 Changing Environments

Changes to hardware and software occur and cause problems even in single-vendor installations. Such changes may be disastrous if they affect both development and production environments at the same time. This problem is at least partially solved by using a separate program development system. The approach also helps programmers "get going" on a project before the production hardware becomes available.

As an example, over the last year, there have been major reconfigurations in both the hardware and software of all of our target systems, and in the geographic work locations of all of our users. Because of the "insulation" that PWB provides, most of our users were relatively unaffected by these changes.

4.4 Effective Testing of Terminal-Oriented Systems

It is difficult enough to test small batch programs; effective testing of large, interactive, data base management systems is far more difficult. It is especially difficult to perform load testing when the same computer is both generating the load and running the application program being tested. It is much simpler and more realistic to perform testing with the aid of a separate computer.

4.5 Availability of Better Software

Many time-sharing systems for large computers often retain significant vestiges of batch processing. Of necessity, some of them have evolved over a long period of time, and may contain design elements that would not exist if they were redone in the light of current knowledge. Separation of support functions onto an appropriate minicomputer may offer an easy way to gain access to more up-to-date software. For example, much of the stimulus for the PWB arose from the desire to make use of the simple, elegant, and powerful UNIX Time-Sharing System.

4.6 Human Orientation and Sizing

In many cases, operating systems, especially interactive ones, have become incredibly complex and incomprehensible. Small systems can be made more comprehensible, friendly, and adaptive, and an individual user can have more impact on their development. Many people welcome *any* movement in the direction of simplicity: "Small is Beautiful" applies to this area, as well as many others [SCH73B].

5. CURRENT STATUS

This section represents a snapshot of the status of the PWB as of June, 1976. Due to the continuing, rapid growth in its usage, this snapshot will soon to be obsolete.

5.1 Hardware Configuration

Figure 2 displays the current structure of the PWB and of its target computers.

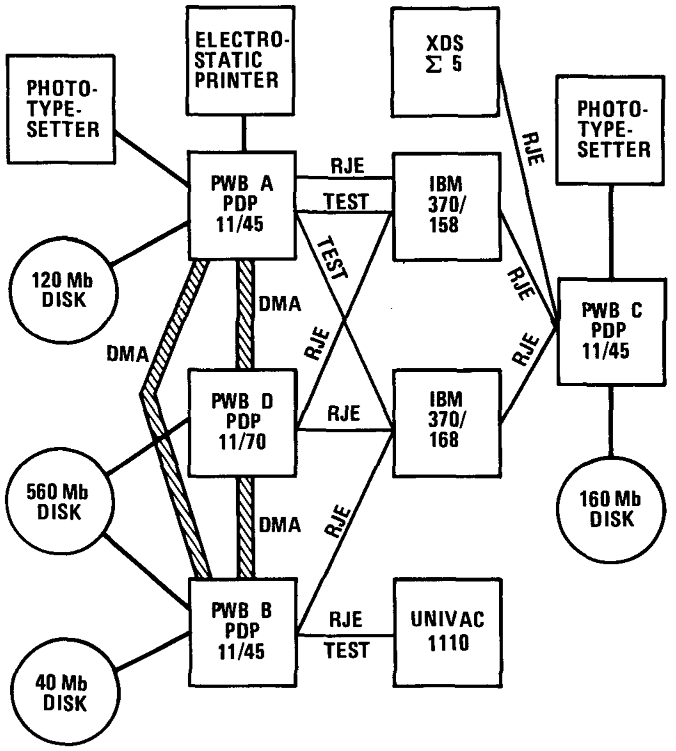

Figure 2. PWB Hardware Configuration

Several notes are in order regarding Figure 2:

- Systems A, B, and D are all interconnected via direct memory-to-memory (DMA) links.
- The communications links between PWB and targets consist of a number of 4.8kb, 9.6kb, and 48kb channels; these links are actually implemented as sub-channels of several multiplexed 230kb lines.
- Each PDP-11 has 124K words of memory (248K bytes). Systems A, B, and D are located in one building (Piscataway, N. J.); the IBM and UNIVAC systems are in another building a few miles away; System C is located about 20 miles away (at Murray Hill, N. J.), as is the XDS SIGMA 5.
- System A is used primarily by the people who develop and support the PWB, by users of some testing facilities [DOL76B], and by users of the phototypesetter.
- System B is primarily used by the developers of a very large, UNIVAC-based system, and also supports additional documentation activities.
- System C supports a large IBM-based project, as well as some members of a project using XDS and other computers.
- System D (a PDP 11/70) is used by developers of other IBM-based systems. Some additional text processing (typing pool) is also done here.
- Systems B and D share disk drives and each can easily be used as backup for the other.
- Temporarily, System C has no convenient backup. This situation will be rectified in the near future. In the next upgrade, Systems A and B will become PDP 11/70's, and a PDP 11/45 (System E) will be moved to join System C.
- Equivalent monthly lease & maintenance costs total approximately $26,000 for these four systems, exclusive of terminals and communications lines.

5.2 Sample Usage Data

Figure 3 displays data obtained at the end of June, 1976.

System	Login Names	Files	Directories	Dial Ports	Prime Connect Hours/Day
A	134	16266	1334	16	71
B	160	27924	1828	26	124
C	156	10430	872	20	121
D	224	32254	2016	48	312
Totals	674	86874	6050	110	628

Figure 3. PWB Usage

In many cases, the same login name is used by many people who work together. In a few cases, a single person possesses several such names. Some login names exist for training new users and for experimental use, and are used only occasionally.

The connect times are for 8-hour prime shifts only and represent averages computed from one month's totals. Of the 880 possible prime connect hours, 71% were actually consumed. It is difficult to raise this percentage significantly: during some parts of the day, *all* ports are in use. The prime shift accounts for approximately 75% of all connect hours. Usage during other shifts includes stand-alone and other testing, maintenance procedures, and execution of some extremely time-consuming user programs that need no human interaction.

6. CURRENT FACILITIES

6.1 UNIX

Much of the impetus for the development of the PWB arose from the desire to utilize the UNIX environment. Although the basic flavor of UNIX has been maintained as much as possible within the PWB, extensions have been necessary to make it more suitable as the operating system for the PWB. The PWB is currently the largest known UNIX installation, in terms of number of systems, disk space, and number of users: some changes were needed to support such an atypically large user community. In particular, much effort has gone into improving the robustness and efficiency of the operating system itself. It is a tribute to the clarity and adaptability of UNIX that it can be easily adjusted to a wide range of configurations.

Various UNIX commands were changed in significant ways. The editor was modified to better support users dealing with fixed-format source programs and data. Many additions and modifications were made to increase the performance of the command interpreter, improve it for use by multi-person teams, and upgrade its capabilities as a programming language [MAS76A].

6.2 Remote Job Entry (RJE) Facility

The Remote Job Entry (RJE) facility includes the following:

- A command used to generate job streams for target systems: it performs nested file inclusions, keyword substitutions, prompting, and character translation (e.g., ASCII to EBCDIC). It also includes a generalized interface to other PWB commands, so that all or parts of job streams can be generated dynamically from the output of other UNIX commands.
- Transmission subsystems used to handle all communications activity: these are target-specific, but are not visible to the users.
- Status reporting: users may inquire about the status of jobs on the target systems, and can elect to be notified in various ways (e.g., on-line or in absentia) of the occurrence of major events in the processing of these jobs.
- Output retrieval: users may route the target's output to remote printers, or may elect to have all or part of it returned into PWB files. These files can be examined by various scanners, or manipulated using many different commands.
- Supporting tools: some users need to create their own.job streams. Others want only to transfer files among systems or print PWB files on target system printers. Various utility programs exist to simplify these functions. Other utilities are used to aid in converting existing target source programs to formats more appropriate for the PWB. In particular, sequence numbers and trailing blanks are eliminated, and other strings of blanks are converted to tabs if possible.

[BIA76A] includes a brief description and examples of RJE usage as seen by the users.

6.3 Source Code Control System (SCCS)

The Source Code Control System permits unusually powerful control over changes to *modules* (i.e., files of text—source code, documentation, data, etc.). It records every change made to a module, can recreate a module as it existed at any point in time, controls and manages any number of concurrently existing versions of a module, and offers many useful audit and administrative features. SCCS is described in [ROC75A].

6.4 Modification Request Control System (MRCS)

Any extensive software project usually evolves mechanisms for requesting changes, reporting errors, and tracking the progress of modifications. Many projects have created systems for handling these items, but the systems are often specific to one project. MRCS is a generalized system intended for use by many projects, and it is described in [KNU76A].

6.5 Document Preparation

UNIX has traditionally provided good documentation tools. Additional commands have been added, and the existing tools put to use in various novel ways. The PWB currently supports a great deal of documentation work, for programmer and non-programmer alike. Many of the documentation efforts are difficult to distinguish from data base applications. Descriptions for the tools and techniques of document preparation are given in [MAS76B].

6.6 Test Drivers

The PWB is often used to run various kinds of tests of IBM and UNIVAC data base management systems and of data base applications implemented on these systems. The PWB contains two test drivers that can generate repeatable tests for very complex interactive systems; these drivers are used both to measure performance under well-controlled load and to help verify the initial and continuing correct operation of these systems and applications while they are being built and modified. One driver simulates a Teletype® cluster controller of up to four terminals, and is used to test programs running on UNIVAC 1100 series machines. The other driver (LEAP—[DOL76B]) simulates an IBM 3270 cluster controller managing up to eight terminals.

7. HISTORY AND DESIGN PHILOSOPHY

7.1 Brief History

The PWB concept was suggested in April, 1973, and the first PDP 11/45 was installed in October, 1973. This machine was used at first by the development department for its own education and experimentation, while the original RJE, SCCS, and LEAP components were constructed. Additional systems were installed in October, 1974 and in May, 1975. The PDP-11/70 arrived in October, 1975; further upgrades are planned for 1976.

At first, the PWB was an experimental project that faced considerable indifference from a user community heavily oriented to large computer systems, working under difficult schedules, and a bit wary of what then seemed like a radical idea. However, as word spread about the system, demand for service began to outrun our ability to supply it. When forced to make six-month usage forecasts, users consistently underestimated the extent of their usage, because they kept discovering unexpected applications for PWB facilities. New hardware was (and is) continually being acquired to meet the demand.

Several large projects are awaiting the arrival of additional hardware before they can completely convert to the PWB. We expect the growth of PWB facilities at Bell Laboratories to continue in the foreseeable future.

7.2 Design Philosophy

Early in the PWB development cycle, many spirited discussions were held regarding the nature of the overall design approach to be taken. One proposed approach was that of first designing the PWB as a completely integrated facility, then implementing it, and then obtaining users for it. A much different approach has actually been followed. Its elements are:

- Get users on the system quickly, and let their needs and problems drive the design.
- Build software quickly, and expect to throw much of it away.
- Build many small, independent programs, rather than large, interrelated ones. This fits well with the nature of UNIX.
- Use as few different file formats as possible, and keep them simple. This avoids the need for a plethora of "utilities" needed to deal with a wide variety of formats, and maximizes the use of existing commands.
- Monitor user problems and rearrange functions or add new ones as needed.
- Design each new feature to be as consistent with existing ones as possible, in order to maintain an environment that is simple, coherent, and conducive to productive use.
- When incompatible changes are necessary, let them derive from user demands, or make sure they offer so much benefit that users will be glad to have them.

This approach may appear chaotic, but evidence to support its desirability can be found in [BRO75A, p. 116] and [NAU69A, pp. 19, 22, 32, 40, 41, 47, 73, 95], for example. In practice, it seems to work better than designing "perfect" systems that turn out to be obsolete or unusable by the time they are implemented. Of course, we are lucky in being able to utilize an operating system that both permits and encourages this kind of approach.

8. CONCLUSIONS

We have given a brief summary of the rationale for the PWB, its advantages and disadvantages, its current status, and the philosophy behind the approach taken. No claim is made that the PWB can solve everyone's programming problems. We do claim that the PWB approach is quite appropriate for many problems existing today, that the directions of change in the computer industry [DOL76C] may make it even more applicable in the future, and that it does handle many real problems, as evidenced by considerable live experience.

ACKNOWLEDGEMENTS

The PWB concept was first suggested by E. L. Ivie [IVI75A]. The authors of the five companion papers, as well as many of our other colleagues, have contributed to the design, implementation, and continuing improvement of the PWB. UNIX itself, without which PWB could not have been built, was developed by members of the Bell Laboratories' Computing Science Research Center. Finally, much of the success of the PWB can be attributed to a user population willing to try new things and to give us the feedback necessary to make these things useful.

REFERENCES

All references for this and the five companion papers [BIA76A, DOL76B, KNU76A, MAS76A, MAS76B] are given below.

ADR73A Applied Data Research. The LIBRARIAN: User Reference Manual. Report P112L, Applied Data Research, Inc., Princeton, N. J., 1973.

BAK75A Baker, F. T. Structured Programming in a Production Programming Environment. *Proc. Int. Conf. on Reliable Software,* April 21-23, 1975, 172-85.

BIA76A Bianchi, M. H., and Wood, J. L. A User's Viewpoint on the Programmer's Workbench. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

BOE73A Boehm, B. W. Software and its Impact: A Quantitative Assessment. *Datamation 19,* 5 (May 1973), 48-59.

BRA75A Bratman, H., and Court, T. The Software Factory. *Computer 8,* 5 (May 1975), 28-37.

BRA75B Bratman, H. Automated Techniques for Project Management and Control. In *Practical Strategies for Developing Large Software Systems,* ed. E. Horowitz, Addison-Wesley, Reading, Mass., 1975, 193-211.

BRO75A Brooks, F. P., Jr. *The Mythical Man-Month.* Addison-Wesley, Reading, Mass., 1975.

BRU76A Brunt, R. F., and Tuffs, D. E. A User-Oriented Approach to Control Languages. *Software—Practice and Experience 6,* 1 (Jan. 1976), 93-108.

CAN74A Canaday, R. H., Harrison, R. D., Ivie, E. L., Ryder, J. L., and Wehr, L. A. A Back-End Computer for Data Base Management. *Comm. ACM 17,* 10 (Oct. 1974), 575-82.

COL76A Colijn, A. W. Experiments with the KRONOS Control Language. *Software—Practice and Experience 6,* 1 (Jan. 1976), 133-36.

COM76A COMTEN. HyperFASTER Concepts and Facilities Manual. COMTEN, Inc., 2 Research Court, Rockville, Md.

COW75A Cowan, R. M. Burroughs B6700/B7700 Work Flow Language. In *Command Languages—Proc. IFIP Working Conference on Command Languages,* ed. C. Unger, North Holland, Amsterdam, 1975, 153-66.

DEC74A Digital Equipment Corp. Option Description, DQS11-A/B Communications Controller. Form CSS-MO-F-32-3A, Digital Equipment Corp., Maynard, Mass., 1974.

DOL69A Dolotta, T. A., and Irvine, C. A. Proposal for a Time Sharing Command Structure. In *Information Processing 68—Proc. IFIP Congress 1968, vol. 1,* North Holland, Amsterdam, 1969, 493-98.

DOL76A Dolotta, T. A., and Mashey, J. R. An Introduction to the Programmer's Workbench. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

DOL76B Dolotta, T. A., Licwinko, J. S., Menninger, R. E., and Roome, W. D. The LEAP Load and Test Driver. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

DOL76C Dolotta, T. A., et al. *Data Processing in 1980-1985.* Wiley-Interscience, New York, 1976.

GRE69A Greenbaum, H. J. *A Simulator of Multiple Interactive Users to Drive a Time-Shared Computer System.* M.I.T., Cambridge, Mass., 1969 (avail. from NTIS; AD 686 988).

IBM72A IBM. OS/VS Utilities. Form GC35-0005, IBM Corp., White Plains, N. Y., 1972.

IBM73A IBM. Introduction to Programming the IBM 3270. Form GC27-6999, IBM Corp., White Plains, N. Y., 1973.

IBM73B IBM. IBM 3270 Information Display System Component Description. Form GA27-2749, IBM Corp., White Plains, N. Y., 1973.

IBM74A IBM. IMS/VS System Operator's Reference Manual. Form SH20-9028, IBM Corp., White Plains, N. Y., 1974.

IBM75A IBM. IMS/VS System Programming Reference Manual. Form SH20-9027, IBM Corp., White Plains, N. Y., 1975.

IBM75B IBM. IMS/VS General Information Manual. Form GH20-1260, IBM Corp., White Plains, N. Y., 1975.

IBM75C IBM. DB/DC Driver System General Information Manual. Form GH20-1639, IBM Corp., White Plains, N. Y., 1975.

IBM76A IBM. Teleprocessing Network Simulator (TPNS) Program Reference Manual. Form SH20-1823, IBM Corp., White Plains, N. Y., 1976.

IVI75A Ivie, E. L. The Programmer's Workbench—A Machine for Software Development. Unpublished Report, Bell Laboratories, May 19, 1975.

JAM75A James, D. L., and Lambert, D. W. *Remote-Terminal Emulator (Design Verification Model)—Introduction and Summary.* Mitre Corp., Bedford, Mass., 1975 (avail. from NTIS; AD A007 827).

KER75A Kernighan, B. W., and Plauger, P. J. Software Tools. *Proc. First National Conference on Software Engineering,* Sept 11-12, 1975, 8-13.

KER75B Kernighan, B. W., and Cherry, L. L. A System for Typesetting Mathematics. *Comm. ACM 18,* 3 (Mar. 1975), 151-56.

KER76A Kernighan, B. W., and Plauger, P. J. *Software Tools.* Addison-Wesley, Reading, Mass., 1976.

KNU76A Knudsen, D. B., Barofsky, A., and Satz, L. R. A Modification Request Control System. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

LUP74A Luppino, F. M., and Smith, R. L. *Programming Support Library (PSL) Functional Requirements.* Structured Programming Series, vol. 5, IBM Federal Systems Div., 1974 (avail. from NTIS; AD A003 339).

MAS76A Mashey, J. R. Using a Command Language as a High-Level Programming Language. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

MAS76B Mashey, J. R., and Smith, D. W. Documentation Tools and Techniques. *Proc. Second Int. Conf. on Software Engineering,* Oct. 13-15, 1976.

NAU69A Naur, P., and Randell, B., eds. *Software Engineering.* Scientific Affairs Division, NATO, Brussels 39, Belgium, 1969.

OSS74B Ossanna, J. F. NROFF User's Manual. 2nd ed. Unpublished Report, Bell Laboratories, September 11, 1974.

PUL68A Pullen, E. W., and Shuttee, D. F. MUSE: A Tool for Testing and Debugging a Multi-Terminal Programming System. *Proc. AFIPS Spring Joint Computer Conference, vol. 32,* 1968, 491-502.

RIT74A Ritchie, D. M., and Thompson, K. The UNIX Time-Sharing System. *Comm. ACM 17,* 7 (July 1974), 365-75.

RIT75A Ritchie, D. M. C Reference Manual. Unpublished Report, Bell Laboratories, February 1, 1975.

ROC75A Rochkind, M. J. The Source Code Control System. *IEEE Trans. on Software Engineering SE-1,* 4 (Dec. 1975), 364-70.

SCH73B Schumacher, E. F. *Small is Beautiful.* Harper & Row, New York, 1973.

SIM74A Simpson, D., ed. *Job Control Languages—Past, Present, and Future.* National Computing Centre Ltd., Quay House, Quay St., Manchester, England, 1974.

THO75A Thompson, K. The UNIX Command Language. In *Structured Programming—International Computer State of the Art Report,* Infotech Information Ltd., Maidenhead, Berkshire, England, 1975, 375-84.

THO75B Thompson, K., and Ritchie, D. M. UNIX Programmer's Manual. 6th ed. Unpublished Report, Bell Laboratories, May, 1975.

UNG75A Unger, C., ed. *Command Languages—Proc. IFIP Working Conference on Command Languages.* North-Holland, Amsterdam, 1975.