Computer Systems
G. Bell, D. Siewiorek, and S.H. Fuller

The Programmer's Workbench – A Machine for Software Development

Evan L. Ivie
Bell Telephone Laboratories, Murray Hill

On almost all software development projects the assumption is made that the program development function will be done on the same machine on which the eventual system will run. It is only when this production machine is unavailable or when its programming environment is totally inadequate that alternatives are considered. In this paper it is suggested that there are many other situations where it would be advantageous to separate the program development and maintenance function onto a specialized computer which is dedicated to that purpose. Such a computer is here called a Programmer's Workbench. The four basic sections of the paper introduce the subject, outline the general concept, discuss areas where such an approach may prove beneficial, and describe an operational system utilizing this concept.

Key Words and Phrases: computer configurations, computer networks, software development, software engineering, software maintenance, UNIX
CR Categories: 3.2, 3.5, 3.7, 3.8, 4.0

Author's address: Bell Telephone Laboratories, Inc., 600 Mountain Ave., Murray Hill, NJ 07974.

1. Introduction

Although the computer industry now has some 30 years of experience, the programming of computer-based systems persists in being a very difficult and costly job. This is particularly true of large and complex systems where schedule slips, cost overruns, high bug rates, insufficient throughput, maintenance difficulties, etc., all seem to be the rule instead of the exception. Part of the problem stems from the fact that programming is as yet very much a trial and error process. There are at this point only the beginnings of a methodology or discipline for designing, building, and testing software. The situation is further aggravated by the rapidly changing hardware industry and by the continuing evolution of operating systems which continues to nullify much of the progress that is made in the development of programming tools.

What can be done to move the programming industry toward a more professional and stable approach to software development? Certainly education (courses, books, conferences, etc.) will play a part in the long run [1]. And, of course, the development of new techniques in programming and program management will contribute as these techniques are accepted and put into use. More compatibility and standardization of hardware, operating systems, languages, and programming procedures will be of great value. Also, an increased investment in the development of programming tools and procedures must occur, but much of this will continue to be lost as computer hardware and operating systems evolve.

2. The Programmer's Workbench Concept

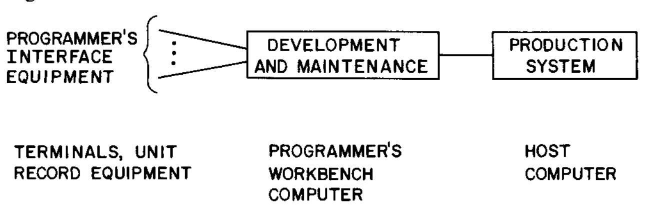
In this paper a very different approach to improving the development process is proposed. It is suggested that the programming community develop a program development "facility" (or facilities) much like those that have been developed for other professions (e.g. carpenter's workbench, dentist's office, engineer's laboratory).

Such an approach would help focus attention on the need for adequate tools and procedures; it would serve as a mechanism for integrating tools into a coordinated set; and it would tend to add stability to the programming environment by separating the tools from the product (the current approach is equivalent to carpenters leaving their tools in each house they build).

Figure 1 shows the separation between the Workbench, which performs the development and maintenance function, and the host or target computer on which the production system will run. The link shown between the two machines represents a physical connection which is used to transfer data, run tests, etc.

The idea of splitting off a well-defined and cohesive function onto a separate dedicated computer is certainly not a new idea. Front-end computers for message

Fig. 1. Division of functions between workbench and host.


PROGRAMMER'S INTERFACE EQUIPMENT
DEVELOPMENT AND MAINTENANCE
PRODUCTION SYSTEM
TERMINALS, UNIT RECORD EQUIPMENT
PROGRAMMER'S WORKBENCH COMPUTER
HOST COMPUTER

concentration and line discipline control are in wide use, having proved to be not only economical, but beneficial for other reasons also [4]. Back-end computers for database management are just beginning to impact the database management field [2, 3]. Specialized computers for the control of peripherals (disks, terminals, photocomposers, etc.) have become commonplace. Indeed the availability of inexpensive microprocessors is certainly going to increase the pressure to go to networks of interacting computers with each processor performing certain specialized functions. The proposal that there should be computers which have been designed and configured to perform just the program development function is merely a further step in this direction.

2.1. Workbench Capabilities

The term "Programmer," in Programmer's Workbench, should be taken in its most general sense and should not be restricted to the coding function. To emphasize this, the steps which go into the development and maintenance of a computer-based system will be briefly outlined:

Step 1. Define what the application system is to do (e.g. system specification, functional description).

Step 2. Design the system. This is normally done at numerous levels of detail (e.g. system, subsystem, program, subroutine) and for various components of the system (e.g. the software, the hardware configuration, the user interface, the operating procedures, etc.).

Step 3. Implement the system through installation of the hardware, coding of the software, writing of users' manuals, etc.

Step 4. Test and evaluate the system. This occurs at numerous levels and for various components as in step 2 and includes the integration of all of the pieces of the system into a working whole.

Step 5. Convert to the system and cut it over to live operation. This is generally a much neglected but significant part of the total job.

Step 6. Operate, support, and maintain the live system. This involves machine operation, user consultation, software bug fixing and enhancement, hardware maintenance and change, etc.

Some of the capabilities that the ultimate Programmer's Workbench might contain to perform these steps are as follows:

(1) Generation, modification, and production of specifications, manuals, catalogs, reports, and documents in general.
(2) Creation, editing, and control of programs and test data.
(3) Compilation, execution, and debugging of programs (either directly or through the host computer).
(4) System generation, integration, and installation.
(5) Regression testing and load testing of subsystems and of the total system.
(6) Analysis and reduction of test results.
(7) Tracking of changes to the system (e.g. trouble reports, enhancement requests).
(8) Evaluation and monitoring of system performance. Also, system modeling and simulation.
(9) Conversion of data files and the loading of the database into the host for live operation.
(10) Production of lists, reports, and statistics for use by management in the control of each phase in the development and maintenance process.

The implementation of the Workbench concept described in Section 4 currently includes only capabilities (1), (2), (3), (5), and (7). The full list is presented here to convey the possible scope of the Workbench concept.

3. Areas of Applicability of Workbench Concept

Four situations will now be described where a dedicated machine for program development would be of particular merit. Some arguments relating to why the Workbench might or might not be a good approach in general will then be presented.

3.1. Multi-Vendor Installations

Many companies operate a variety of different computers. This diversity might occur across several manufacturers or it might be across several different models of a given manufacturer. For example, in the Business Informations Systems Programs (BISP) area of Bell Laboratories, where the Workbench is in use, there is an IBM 370/158 and 168, a UNIVAC 1108 and 1110, two Xerox sigma 5's, plus a number of minicomputers.

Initial efforts in BISP concentrated on development of a programming environment for each vendor line (IBM, UNIVAC, Xerox). Programming tools were implemented on two and sometimes three machines so that the tools would be available to all of the projects. An attempt was also made to keep these tools fully compatible across machines, but this met with only partial success.

Part of the motivation which led to the implementation of the Workbench was based on the realization that a much better set of tools could be provided for less money by concentrating efforts on building and enhancing a single set of development tools. As a specific example, Figure 2 compares the approximate costs for developing and maintaining a specific programming tool on the Workbench versus doing it directly on two host machines. Development and maintenance costs for the Workbench version actually turned out to be less than half that of the dual

Fig. 2. Comparison of development and maintenance costs for a specific programming tool (includes both staff and computer time).

	WORKBENCH IMPLEMENTATION	IMPLEMENTATIONS ON IBM AND UNIVAC
DEVELOPMENT (NOT INCLUDING DESIGN)	$50,000	$ 120,000
MAINTENANCE (PER YEAR)	$ 22,000	$ 56,000

implementation, owing perhaps to the better programming environment. The savings would be even more significant if more than two hosts were included, or if more than one tool were considered.

A second benefit of the Workbench approach is the more nearly uniform programming environment that is possible, even across projects which run on different computers and under different operating systems. Such a standard environment offers the following advantages:

(1) *Training*—It has been found that many of the same training courses on programming tools can be offered to programmers on, for example, both IBM and UNIVAC projects.

(2) *Documentation*—Only one set of users' manuals is needed for describing the common programming tools.

(3) *Standards*—The development of standard policies, procedures, and methods relating to program development and maintenance are greatly simplified because one no longer has to take account of a multitude of environments every time a standards decision is made. The enforcement of standards is also simplified.

(4) *Programmer mobility and retraining*—Programmers can become productive much more quickly when they are transferred from one project to another.

It should be quickly noted that while the Workbench system currently in operation (Section 4) has in general achieved host machine independence for those tools that have been implemented, there is still much about the host machine that the programmer must be painfully aware of. Take, for example, the job control language. Eventually it may be possible to develop a universal job control language (jcl) for use of the Workbench which can be translated into the appropriate jcl for each host. Until then the best the Workbench can do is to provide various facilities for generating, concatenating, and modifying host-specific jcl.

In addition to the advantages offered to existing projects, the Workbench approach also offers substantial benefits to new projects. If a particular Workbench system has been accepted as the standard approach at an installation and if all the programmers there are trained in its use, then a new project can bypass much of that lengthy "getting started" period during which tools and standards are developed, adopted, and learned. This should save not only money but also provide a quicker start, and thereby shorten schedules. It would, of course, be necessary to develop a Workbench interface to the new host and to expand any host-dependent tools to handle the new machine. (However, the only parts of the current implementation that are host dependent are certain aspects of the job submission module.)

3.2. Installations in Transition

Even companies that make it a policy to use only one type of hardware find it necessary to periodically upgrade their equipment or to change operating systems. The transition to the new equipment or to the new operating system is usually a very painful period. If the installation were using the Workbench approach, it could make the transition a less painful process in several ways.

First, a change in the program development environment is no longer inextricably tied to a change in the production equipment. That is, changes in the two machines can be scheduled independently. Indeed, one would probably want to avoid a change in the Workbench at the time the production machine is changing so that all of the programmers' attention could be concentrated on the new equipment and so that they would not have to simultaneously worry about changes in their tools. This should shorten the transition period in bringing up the new equipment significantly. It should also eliminate that difficult "uncovered" period on a new machine when adequate tools are not available, and one has to make do with whatever happens to be provided by the vendor. Then, after the new production equipment is installed and operational, one could consider upgrading the Workbench equipment and/or software.

In addition to being able to stagger changes in the production equipment with upheavals in the program development environment, the Workbench approach also allows one to independently decide upon the frequency with which changes in these two areas will occur. Indeed, if one has a good development environment on a Workbench, it might be well to decide to adopt a much less frequent change cycle, thereby providing more stability to standards, less retraining, smaller tool development costs, etc.

One also has more flexibility in upgrading the host (production) machine when it becomes obvious that such a move is advantageous. This is true since such a decision is no longer clouded by considerations of what it will do to the programmer's ability to develop a code.

3.3. Projects Where Needs of Developer and User Conflict

One of the most critical decisions in the development of a computer-based application is the selection of the computer on which it is to run. In many cases a formal selection procedure is not undertaken because the decision is based on machine availability or other compelling factors. In those cases where an actual choice can be made, there is generally a conflict between the needs of the developer and the user.

If the selection is based strictly on the ability of the computer to perform the eventual application, then software developers may have to survive in an environment that is poorly suited to their needs. If the selection is based entirely on the needs of the developer then the target system may perform its functions expensively if at all. The third possibility is that both the developer and the user compromise their needs in the machine selection process. This may, of course, leave neither very happy.

The development programmer is looking for a machine that has a powerful command language, sophisticated editing tools, flexible and easy to use file structure, terminal access (time-shared), quality document production facilities, and good human engineering in general. The end user is, on the other hand, more concerned with sufficient throughput and size, appropriate peripherals and equipment to support the application, hardware, and software options to optimize certain features (e.g. access methods, block sizes, physical placement), special needs in the areas of availability and reliability, and quality maintenance.

In addition to these potential functional conflicts, opposing needs can also manifest themselves because of a desire to utilize existing equipment or because of the experience and background of the programmers, the operators, or the terminal users.

All of these conflicts are based on the assumption that the development of the software for a project will be done on the same machine as the one on which the project will finally run. The Workbench approach helps to eliminate these built-in conflicts by providing an independent choice of the computer for the developer and for the user. No compromises are necessary and each can choose the machine best suited to their needs and experience. Here again not only the initial choice but the frequency with which a change is to be made is decoupled. This also eliminates possible downstream conflicts and compromises.

3.4. Terminal-Oriented Systems

One of the most time consuming and critical parts of software development is testing, especially total system testing. If the application being developed services terminals, the testing is additionally complicated. It is very unsatisfactory to perform such testing by stationing people at terminals and having them type in data and examine output. Aside from the cost and frustration, it also is a nonrepeatable and error-prone approach. A reasonable solution to this dilemma is to provide a canned scenario of user interactions that can be fed into the system in a timed manner. However, if the insertion of the input messages and the capturing of the output messages is done in the internal queues, then the total system is not really being tested (e.g. terminals, lines, controllers, line servicing software, etc.). To help circumvent this problem, various "loop back" devices can be developed which allow one program operating within the computer to send out data which is returned as though it came from a terminal.

Such an approach provides more complete testing but still suffers from certain side effects. For example, it may be difficult to isolate whether a failure occurred because of an error in the test driver or in the application being tested. This is because of the various possible interactions that can occur between two systems operating simultaneously within the same computer. Also if the application operates on a dedicated computer, it is impossible to effectively load test it to determine its total capacity since the test driver is consuming a portion of the resources. Thus there are special reasons why a system test facility needs to be on a separate computer such as a Workbench.

3.5. General Advantages

It's probably safe to assume that every installation will periodically upgrade to a new computer. It is also fair to say that user and developer needs always conflict to some extent. Thus, to a greater or lesser extent, the advantages ascribed to the Workbench approach in the preceding paragraphs apply to all installations.

Additionally there are potential economies which may accrue to the Workbench because of specialization. Applications computers are typically large general purpose computers with complex operating systems having many options and features. The Workbench consists of a specialized set of functions running on a dedicated system. The hardware configuration for the Workbench should be much simpler, the operating system should be less cumbersome, and the actual tools should therefore be smaller and faster. Front-end and back-end computers have been found to be economically attractive for the same reasons.

Another general advantage to the Workbench is the fact that it encourages the development of machine independent programming tools. Each tool must now function for programmers developing code for a number of different vendor machines. There is no room for shortcuts which are dependent on the idiosyncrasies of a given machine (or a given project). One is thus forced into a more stable and generalized software development approach which should be more applicable to new machines.

It has already been noted that a separate program development facility will help focus attention on the importance of the programming environment in the software development process. It should also provide some stimulus for the integration of the programming tools operating on the Workbench into a coordinated set of interconnected functions.

3.6. Potential Disadvantages

Thus far a number of arguments favoring the Workbench concept have been presented. The other side of the ledger will now be examined. One disadvantage to the Workbench approach is that another machine is needed which costs money and which is one more link in the system that can fail. The cost may be counterbalanced by the fact that the host computer(s) may not have to be available quite as early in the development cycle, they may not have to be as big, and fewer may be needed. Having an extra link that can fail is a problem if the components cannot do useful work on a stand-alone basis (e.g. program editing and document production on the Workbench when the host is down) or if there is insufficient redundancy (e.g. other Workbenches to shift the load to when one fails).

In addition to the question of the actual purchase or rental cost, there may be other "costs" in having a second (Workbench) machine. For example, the Workbench machine may be manufactured by a vendor which is different from the host machine vendor. This will duplicate all the problems associated with contracts, maintenance procedures, operator training, system programming support, etc. It will also force the programmer to be aware of two machines and not just one. However, if the Workbench has a good user interface, it may actually be easier for the programmer to keep track of it and the operational aspects of the host than to use the host tools.

A second general problem relates to the fact that data and functions are now split between two machines. This may manifest itself in slower response to some requests for data and cpu processing because of transmission and queueing delays. It may also result in some duplication of data to avoid these delays. The best solution to this problem is a judicious choice of what data is stored on each machine and what processing is done on each machine. Also, the use of a high speed link will minimize the delays.

A third problem is the fact that machines use different character sets, number representations, etc. Thus there is a conversion cost in shipping data back and forth between machines. (However, in the current implementation this amounts to a very small fraction of the total Workbench cpu load).

A final problem may occur because a fixed part of one's computational power is dedicated to a given function (i.e. program development). Some flexibility is thereby lost in being able to balance the computational load. Potential imbalances can occur in any of the available resources: cpu, disk, printers, etc. Imbalances may also develop at certain points in the development cycle. For example, one may find that the host is being underutilized during the initial phases of the project. How serious this problem is depends a great deal on the relative costs and sizes of Workbench and host machines. If the Workbench machines are small and constitute a small fraction of the total computational budget, then adding or deleting a Workbench machine to match the total Workbench load may be adequate. Also, because of the link, some functions such as printing and disk storage can be shifted from one machine to the other to help balance the load.

4. Description of Current Workbench Implementation

When the potential benefits of, and the possible difficulties with, the Workbench were first considered, a number of questions arose. For example, were there unforeseen problems in the implementation or in the use of a Workbench system which would outweigh all of the projected advantages? Could such a system be implemented in a reasonable amount of time and with a modest expenditure of resources? Would programmers be willing to try something new and give the approach an honest trial? Could such a system be assimilated into an ongoing software development organization? How many of the benefits would turn out to be real? How serious would the potential problems be?

The idea of the Workbench was first conceived in April 1973. The Business Information Systems Programs (BISP) area in Bell Laboratories appeared to be an ideal environment in which to try the idea because all of the conditions described in Section 3, for which the Workbench approach would be beneficial, were present. Thus the decision was made to try out the approach on an experimental basis. The first Workbench machine was installed in October 1973. Three additional machines have since been installed with two more due to be installed in 1976.

The machines currently being used as the Workbench machine are the Digital Equipment Corporation PDP 11's. Initially 11/45's were used; more recently 11/70's have been used. The decision to use PDP 11's was based mainly on the fact that the UNIX [6] time-sharing system operated on the PDP 11. UNIX was developed at Bell Laboratories by Ken Thompson and Dennis Ritchie. It is an outgrowth of the MULTICS [5] system, but much simplified and streamlined. It has offered an ideal base on which to build program development tools which have been developed thus far.

One of the current Workbench configurations is shown in Figure 3. Monthly rental for such a system is in the neighborhood of \$6,000. Such a system can provide good response to 24-30 users simultaneously logged in. If one assumes that the average user is logged in about two hours a day, then one PDP 11/45 could handle a project of about 100 people. A PDP 11/70 can handle 45-50 users simultaneously or about double the 11/45 load.

Before discussing the components of the Workbench that are in operation, a brief comment on implementation philosophy is in order. The idea of designing and building a complete and fully integrated Workbench system was rejected for a number of reasons, not the least of which is the fact that no one in the programming field knows what that system should look like at this point in time. Instead, every effort was made to identify some of the immediate needs of potential users and to develop pieces of the Workbench that could satisfy those needs quickly. This approach provided the Workbench designers with much valuable

Fig. 3. Configuration of one of the workbench computers.

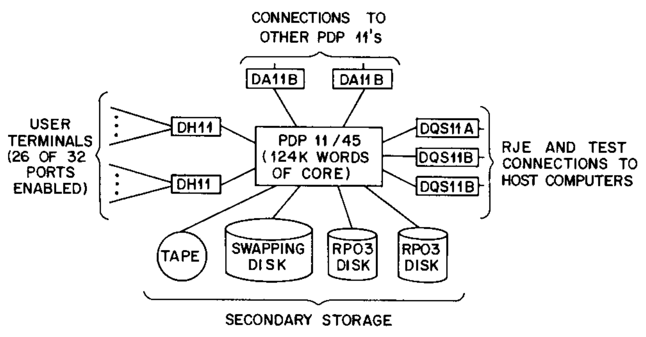
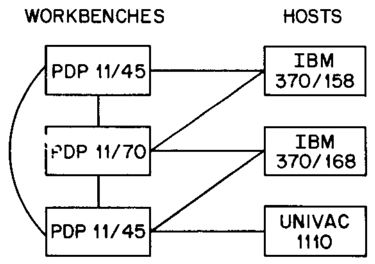

CONNECTIONS TO OTHER PDP 11's
DA11B
DA11B
USER TERMINALS (26 OF 32 PORTS ENABLED)
DH11
DH11
PDP 11/45 (124K WORDS OF CORE)
DQS11A
DQS11B
DQS11B
RJE AND TEST CONNECTIONS TO HOST COMPUTERS
TAPE
SWAPPING DISK
RPO3 DISK
RPO3 DISK
SECONDARY STORAGE

Fig. 4. Example of a workbench job submission network.



user feedback quickly, and it allowed the projects to start picking up pieces of the Workbench to satisfy their most critical needs immediately and thus to start a phased transition to the complete Workbench.

4.1. Basic Components

It is not the purpose of this paper to fully describe the current Workbench implementation. Other papers are planned which will describe in detail each of the Workbench subsystems. However, a brief description of each of the five Workbench components currently in operation will be presented here to establish the fact that the basic Workbench concept has in fact been given a live shakedown with a useful subset of functions. The five basic components of the Workbench which were selected for initial implementation are job submission, module control, change management, document production, and test drivers.

1. **Job submission.** The whole Workbench concept depends on having the capability to transfer data easily and quickly between the workbench and the host machines. Each Bell Labs Workbench system currently provides this capability by operating as a remote job entry (RJE) station to one or more host computers. Take, for example, the three Workbench machines currently located in Piscataway, New Jersey. Figure 4 shows the job submission network for these three machines. Inter-Workbench links provide access to a host not directly connected to a given Workbench. There are also links for test drivers which are not shown in Figure 4.

The RJE facility can be broken into four components:

(1) *Job preparation* — this step performs file concatenation (e.g. combining the job control language (jcl) file with the data file(s)), character conversion (ASCII to EBCDIC), and queueing.

(2) *Transmission* — this component empties the transmission queue, monitors the status of the communication line, and receives results from the hosts.

(3) *Status reporting* — facilities are provided so that Workbench users can determine the overall load on the host machine and the progress of each individual job. Each user can select what status is to be sent automatically. The user can also initiate requests for status information.

(4) *Post processing* — the output from each run can be returned to the Workbench or selectively routed to a host printer or elsewhere. That which is returned to the Workbench is placed in the appropriate file for examination. Various scanning programs have been developed to help determine such things as the success of the run, etc. The RJE link does not, of course, provide direct interaction with a program executing on the host. This capability is part of the test driver.

2. **Module control.** In the development of a software system, particularly a large system, each program goes through a number of revisions (releases, versions). In fact, at any one point in time there will probably be several revisions of a program in use simultaneously. The revision in use in the field may be different from the one in trial, which in turn may be different from the one in system test, which finally may be different from the one the programmer is working on. Also, there may be differences caused by different operating systems, data management systems, and user needs. Keeping track of all of these revisions is a major task. An even larger task is to make sure that the right modifications are applied to the right revisions and not to the others. The module control system developed for the Workbench provides:

(1) Creation of any revision of a program from any previous point in time.
(2) Protection against accidental tampering and change.
(3) Selective propagation of each change to a module to each of its revisions which should contain that change.
(4) Identification of object and source (revision number, date created, etc.).

In addition to programs, all sorts of other documentation on a project also go through many revisions. The Workbench module control system was generalized to handle not only source but any type of text. In fact, it is currently being used heavily for keeping track of the evolution of user manuals, test plans, error lists, etc. The module control system is also called the Source Code Control System (SCCS) and is described in [7].

3. **Change management.** When a software system goes into production, it becomes necessary to formalize

the way changes are made to it. For a large software system, some type of formal change control is necessary fairly early in the development cycle and increases in importance as time progresses. The mechanism for change control is usually some type of trouble reporting form describing the reason why a change (in programs, hardware, or documents) is needed. To this form is added information as to who will make the change, what it consists of, and when it will be made. The Bell Labs Workbench currently provides a facility for entering trouble reports into a project database and of subsequently editing and updating them. Facilities for generating summary, status, and other reports have also been developed.

4. **Documentation production.** Having accurate, up to date, and understandable documentation (e.g. design specs, test plans, user manuals) is vital to the success of a project. On a large software project it is generally a larger and more difficult job to produce such documentation than it is to produce the source programs.

A wide variety of document production tools have been built for the UNIX time sharing system on which the Workbench operates: text editors, formatters, typographical error finders, etc. [7]. A phototypesetter which provides multiple character fonts and point sizes has also been connected to the system. A document writer has the option of having text printed on a terminal, routed to a host machine for high speed printing, or sent to the phototypesetter for high quality output.

Over the past year a number of "macro packages" have been developed for the UNIX documentation tools. These packages automate the production of many of the standard document formats used by the BISP projects by producing headings, footings, labels, tables of contents, etc. Users can thus produce highly structured documents with minimal input effort. Typing pools, document production centers, programmers, and managers are all heavy users of this UNIX/Workbench facility.

5. **Test drivers.** Two test drivers have been implemented on the Workbench thus far. The IBM test driver simulates IBM 3271 cluster controllers, each with several 3277 terminals. It serves as a driver to projects which use the data management systems operating on IBM/360 and IBM/370 computers. The IBM test driver is used for both load testing and regression testing. The UNIVAC test driver simulates a teletype cluster controller with up to four terminals and is used for regression testing.

4.2. Possible Extensions

All of the above components are under continuing revision and improvement. In addition, efforts are now underway to integrate the current components into a more closely cooperating set of tools. For example, each change made to a module should be related to the trouble report(s) that initiated that change. This will allow one to supply a list of all the trouble reports that are to be resolved by a given release of a system and to have the Workbench automatically select which revisions of each program are needed in that release. Besides integration, several of the other functions noted in Section 2.1 are in various stages of planning. The system generation and configuration control area is one example. Also, consideration is now being given to connecting the Workbench to other types of host computers.

In addition to these short term efforts, some long term objectives are being studied. For example, an attempt may be made to develop a uniform job control language (jcl) which can be translated into each of the host jcl's. Also, a standard programming language might be designed which has code generators for each of the host machines. The value of this would be enhanced considerably if a set of Workbench system calls were devised that could be mapped into system calls to each of the host operating systems. These steps toward software portability are obviously very difficult but might well be eventually achieved in an environment like that offered on the Workbench.

4.3. Workbench Usage

As of February 1976 there were three Workbench machines in operation at the Bell Labs facility in Piscataway, N.J. and one in use at Murray Hill, N.J. Workbench facilities are also beginning to be used at two other Bell Labs locations. The four New Jersey machines were providing about 2600 hours of connect time per week to some 300 users, and they were maintaining disk files containing about 250 million characters. Both of these figures are expected to double by mid 1977 (with the addition of two more 11/70's). All of the Business Information Systems Projects in Bell Labs are using at least some parts of the system.

4.4. Example of Use

Perhaps the most effective way to describe the current operation of the Workbench is to give a very simple example of how it might be used. Assume a trouble report has been written describing a bug in the "sum" program. After the programmer responsible for the maintenance of the program dials in, the following interaction might take place. (The characters typed by the programmer are in bold type. The UNIX prompt symbol is the "%.")

```
login Jones
You have mail.
```

The programmer enters the appropriate identification code and is told that there is mail.

```
% mail
From smith Fri Mar 14 12:48
1975
tr number: a-75-83-3
originated by: R.H. Johnson
description:   The    value
  printed by the sum pro-
  gram is incorrect.
```

The programmer uses the UNIX mail command to print the mail and finds that a trouble report has been assigned.

% **get sum −r2 −e**	*The programmer uses the get command of the module control system to extract the "sum" program at release 2 (r2) for editing (e).*
2.4 129 lines	*The get command responds with the release and level of the module (2.4) and its size.*
% **ed sum**	*The programmer invokes the UNIX editor to make the modification.*
8463	*The editor responds with the number of characters in the file.*
/put data/p put data (subtotal);	*The programmer locates with the editor search command the line containing "put data" and prints it. Note that within the editor there is no prompting. (It is assumed at this point that the programmer realizes that the output variable should be "total.")*
s/subtotal/total/p put data (total);	*The programmer uses the editor substitute command to change the variable name from "subtotal" to "total."*
w 8460 **q**	*The programmer writes the editor buffer and exits from the editor.*
% **send jcl sum**	*The programmer uses the send command of the job submission module to send the sum program to the host machine for a compile and go run to test the fix. (jcl is a file containing the necessary job control statements and test data.)*
153 cards queued as /u3/hasp/xmit07	*The send command responds with the total lines (cards) sent and the position of the job in the queue.*
15:40 /u3/jones/rje/prnt1 996. 1 dgo ready	*The job submission module notifies the programmer that the listing from the run has been returned to the Workbench.*
% **ed prnt1** 26845 **/TOTAL/p** TOTAL = 93,467 **q**	*The programmer scans the output listing for the total line and verifies the value.*
% **delta sum** history? **a-75-83-3 The output variable should be total, not subtotal.**	*The programmer uses the delta command of the module control system to make the change a permanent part of the module's second release.*
128 unchanged 1 inserted 1 deleted	*The delta program summarizes the amount of the change to the module.*

5. Conclusions

The programming profession has yet to produce a software development methodology that is sufficiently general so that it can be transferred from one project to another and from one machine to another. The development of a Programmer's Workbench, a machine dedicated to the software development and maintenance function, can serve as a vehicle for the development of such a methodology.

By achieving application and machine independence, the Workbench approach can offer significant economic and other benefits to companies with several different machines and to companies that are in the process of installing a new computer. Since the Workbench concept allows an independent selection of the development and the production machines, the capabilities and environment available to both the development programmer and to the end user can be optimized. By running on a separate machine, the Workbench provides additional advantages to the testing function.

The Workbench concept has been partially implemented at Bell Laboratories. In open competition it has met with enthusiastic user acceptance. Both IBM and UNIVAC programmers now take advantage of the UNIX time sharing system and the same set of Workbench development tools. It is suggested that this initial success has, in fact, provided some evidence of the utility of the basic Workbench concept. The extent to which this approach can be cost-effective at other installations and under other conditions is still an open question.

Acknowledgment. The concept of the Programmer's Workbench, its implementation, and its acceptance within Bell Laboratories are due to the efforts of the members of the Support Products and Systems Department. Much credit is also due to the members of the Computing Science Research Center who developed the UNIX time sharing system and the document production tools. Finally, special thanks go to the many users of the Programmer's Workbench who were willing to try something new and who have provided invaluable feedback and support.

Received January 1975; revised July 1976

References

1. Brooks, F.P. Jr. *The Mythical Man-Month.* Addison-Wesley, Reading, Mass., 1975.
2. 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), 575–582.
3. Ivie, E.L. A back-end computer for data base management. Presentation at Data Base Management Session of 1973 NCC, New York, June 1973.
4. Feinroth, Y., Franceschini, E., and Goldstein, M. Telecommunications using a front-end minicomputer. *Comm. ACM 16,* 3 (March 1973), 153–160.
5. Organic, E.I. *The MULTICS System: An Examination of Its Structure.* M.I.T. Press, Cambridge, Mass., 1972.
6. Ritchie, D.M., and Thompson, K.L. The Unix time-sharing system. *Comm. ACM 17,* 7 (July 1974), 365–375.
7. Rochkind, M.J. The source code control system. *IEEE Trans. Software Eng. SE-1,* 4 (Dec. 1975), 364–369.