Improving a working environment often results in improved products. This age-old notion was recently applied in the creation of several software development and measurement tools.

A Software Development Environment for Improving Productivity

Barry W. Boehm, Maria H. Penedo, E. Don Stuckle, and Robert D. Williams, TRW

A major effort at improving productivity at TRW led to the creation of the software productivity project, or SPP, in 1981. The major thrust of this project is the establishment of a software development environment to support project activities; this environment is called the software productivity system, or SPS. It involves a set of strategies, including the work environment; the evaluation and procurement of hardware equipment; the provision for immediate access to computing resources through local area networks; the building of an integrated set of tools to support the software development life cycle and all project personnel; and a user support function to transfer new technology. All of these strategies are being accomplished incrementally. The current architecture is Vax-based and uses the Unix operating system, a wideband local network, and a set of software tools.

This article describes the steps that led to the creation of the SPP, summarizes the requirements analyses on which the SPS is based, describes the components which make up the SPS, and presents our conclusions.

SPS requirements analysis

A software productivity study was performed at TRW during 1980. This study analyzed the requirements for a company-oriented software support environment, evaluated the technology base available for such a support environment and the likely trends in that base, and performed an economic analysis to determine whether a significant level of investment into software productivity aids would be justified. In the following paragraphs, each analysis is summarized, followed by the study's conclusions and recommendations.

Corporate motivating factors. Various factors have motivated a more substantial level of corporate investment for improving software productivity in recent years, including (1) increased demand for software, (2) limited supply of software engineers, (3) rising software engineer support expectations, and (4) reduced hardware costs.

Increased demand for software. Each successive generation of a data processing system experiences a significant increase in demand for software functionality. For example, manned spaceflight software support functions grew from 1.5 million object code instructions for the 1961 Mercury program to over 40 million object instructions for the 1980 space shuttle program.

Limited supply of software engineers. Several sources have indicated that the current US shortage of software personnel is between 50,000 and 100,000 people and that the suppliers (primarily university computer science departments) do not have sufficient resources to meet the future demand.

Rising software engineer support expectations. Good software engineers are, in general, no longer satisfied to work with inadequate tools and a poor work environment. Successful hiring and retention of good software engineers requires an effective corporate software support environment.

*An earlier version of this article entitled The TRW Software Productivity System was presented at the Sixth International Conference on Software Engineering.
Reduced hardware costs. The cost and performance improvements of supermini mainframes, powerful personal microcomputers, database machines, graphics devices, and broadband communication systems permit significantly more powerful and cost-effective software support systems.

The 1980 software productivity study. Given the preceding motivating factors, an extensive study of corporate objectives, requirements, and alternatives was done in 1980. The study included an internal assessment, an external assessment, and a quantitative analysis, which led to recommended strategies for improving software productivity.

Internal assessment. Our internal assessment began with a series of interviews with representative high-level and intermediate-level managers and software performers. Each person interviewed was asked, “If there were only two or three things we could do to improve your software productivity, what would they be?”

In general, those interviewed were highly enthusiastic and provided a wide range of attractive suggestions for improving productivity. There was a general consensus that the primary avenues for improving software productivity were in the areas of management actions, work environment and compensation, education and training, and software tools. For example, Figure 1 shows the relative importance of these four areas from the standpoint of three classes of personnel: upper managers, middle managers, and performers.

It is evident from Figure 1, though, that the consensus was not universal. For example, the upper managers’ worldview conditions them to see management actions as the high-leverage items, while the performers—that is, nonmanagers such as engineers, analysts, and programmers—worldview conditions them to see software tools as providing the most leverage. The important point is not which group is more correct, but that each group brings a valid set of perceptions to bear on the problem. Furthermore, since motivation is such a key factor in software productivity, people’s perceptions are an important consideration.

Software support environment requirements. Another portion of the internal assessment involved an analysis of our software support environment requirements. Since the DoD Ada Stoneman requirements document had recently provided an excellent general definition of software support environment requirements for Ada, we used Stoneman as a baseline and focused on identifying the following additional company-specific environment requirements not included in Stoneman.

1. Support of multiple-programming languages. The internal assessment included a forecast of the evolution of our government-systems business base, including its distribution by programming language. It showed that even though DoD is strongly committed to Ada for its new starts, there is likely to be a significant segment of software projects consisting of compatible developments and maintenance for existing Fortran and JOVIAL systems. Thus, a pure Ada-based environment would not support all of our needs even by the year 2000.

2. Support of mixed target-machine complexes. A similar forecast of the hardware nature of our future business base indicated a strong trend towards distributed target-machine complexes with a wide range of computer types (mics, minis, and mainframes from different vendors). Although the concept of the Ada program support environment may provide a unified environment supported on each computer in such complexes, experience to date on such environments as the National Software Works indicate that a number of outstanding problems need to be resolved before this approach can be counted on.

3. Support of classified projects. Much of our business involves classified projects for DoD. Such projects have severe access constraints and require extensive precautions to enforce those restrictions. Participation in a single corporate-wide network with shared data and programs would not be feasible for classified projects. The state of the art does not ensure such security in a wide-area network that would necessarily use insecure communications media. An alternative approach is to have a collection of local networks, each of which can individually impose tight security arrangements. A classified project could then use a single local network within a restricted area.

4. Integration with existing programs and data. The company has a large collection of partially integrated programs and databases to support project activities, particularly in the areas of cost and

Figure 1. Responses or perceptions of major needs in software productivity. Upper managers, middle managers, and performers were asked “If there were only two or three things you could get TRW to do to improve software productivity, what would they be?”
schedule management. It is not practical to discard all of these at one time because of the large financial investment required to replace them. New tools must be integrated with this existing base of programs and data.

(5) Support of nonprogramming activities. Studies indicate that about two thirds of the time spent on a large software project results in documentation as its direct product, and only one third results in code as its direct product (see Figure 2). For example, software requirements analysis and the preliminary design phases typically produce no code at all and may not even be performed by programmers. Even in the coding phase, peripheral activities—such as the generation of unit test plans, memos, and reports—consume a significant percentage of a programmer's time. Since one of our goals is to support all efforts of all project personnel, the development and integration of word processing, calendar management, spreadsheet, and other office automation capabilities into the software development environment are required.

Uncertainty areas. In trying to determine the specific needs of project personnel, we encountered a wide variety of user opinions on such items as:
- tool priorities (development, management, office support);
- attribute priorities (accuracy, extensibility, ease of use by experts versus novices);
- degree of methodology enforcement (for example, do tools assume requirements are written in a specific requirements specification language or are they methodology independent?); and
- command language characteristics (such as menu versus command and terse versus verbose).

As a result, we concluded that it would be an extremely time-consuming, inefficient, and uncertain process to obtain universal concurrence on requirements for our software support environment. Thus, we decided to build the environment incrementally, using prototypes and incorporating both our experimental and user feedback in subsequent increments.

External assessment. The 1980 study included visits to a number of organizations with experience in active R&D programs on software support environments. The industrial organizations visited included IBM-Santa Teresa, Xerox Palo Alto Research Center, Bell Laboratories, and Fujitsu; the universities included Stanford, MIT, Harvard, and Carnegie-Mellon. The primary conclusions resulting from these visits were:

1. Organizations investing in significant improvements in their software environments felt they were getting their money's worth. Some, such as IBM and Bell Labs, were able to at least partially quantify their resulting benefits.
2. Organizations achieving some integration of software development support capabilities and office automation capabilities considered this a high-payoff step.
3. Organizations providing office facilities oriented towards the needs of software personnel (privacy, quiet, comfortable and functional furniture) such as IBM-Santa Teresa, felt the investment was highly worthwhile.
4. Significant progress was being made toward providing very high power personal workstation terminals (with high-resolution, bit-mapped displays supporting window editors, integrated text, and graphics and well-integrated screen-pointing devices) at a reasonable cost.
5. No system we saw provided all the capabilities required.

Quantitative assessment. Our quantitative assessment of alternative avenues for improving software productivity was based primarily on TRW's software cost estimation program, or SCEP. SCEP is similar in form to the Cocomo model described in detail elsewhere. It estimates the cost of a software project as a function of program size in delivered source instructions, or DSU, and a number of other cost driver attributes summarized in Figure 3. Figure 3 shows the productivity range for each attribute. Thus, the 1.49 productivity range for the software tools attribute results from an analysis indicating that, all other factors being equal, a project with a very low level of tool support will require 1.49 times the effort required for a project with a very high level of tool support. The "very high" and "very low" ratings correspond to specific levels on a Cocomo rating scale for tool support.

The software tools' rating scales and those of the other cost driver attributes were used to conduct a "productivity audit" of our projects, to determine the weighted-average productivity multipliers characteristic of the overall distribution of software projects, both past and for several future scenarios representing varying levels of company investment into productivity-improvement pro-
grams. Table 1 summarizes a typical analysis of this nature. It shows that a productivity improvement program achieving several cost driver attribute improvements in parallel could improve productivity by a factor of 3.4 by 1985 and by a factor of 7.8 by 1990. Besides providing an estimated productivity gain, this analysis provided insights for determining which items (such as tools) to emphasize as part of a productivity improvement strategy. It also provides a valuable framework for tracking the actual progress of the productivity program and for determining whether its goals are actually being achieved.

Study conclusions. The 1980 productivity study reached some major conclusions. First, significant productivity gains require an integrated program of initiatives in several areas. These areas include improvements in tools, methodology, work environment, education, management, personal incentives, and software reuse. A fully effective software support environment requires integration of software tools and office automation capabilities. Second, an integrated software productivity improvement program can have an extremely large payoff. Productivity gains by factors of two in five years and factors of four in 10 years are generally achievable and are worth a good deal of planning and investment. Third, improving software productivity involves a long, sustained effort. The payoffs are large, but they require a long-range commitment. There are no easy, instant panaceas. Fourth, in the very long run the biggest productivity gains will come from increasing use of existing software. This requires building general tools and utilities suitable for reusability. Fifth and last, software support environments require integration of software tools and office automation capabilities.

Study recommendations. Based on these conclusions, the 1980 study made several recommendations. First, initiate a significant long-range effort to improve software productivity. The recommended effort included initiatives in all areas of tools, methodology, work environment, education, management, personal incentives, and software reuse, and established goals of improving software productivity by (1) a factor of two by 1985, and (2) a factor of four by 1990. Although these goals are conservative with respect to the estimated productivity gains cited in the quantitative assessment, they are clearly large enough to justify a significant investment into a productivity improvement program. Second, establish a software productivity project with responsibility for establishing a software development environment. A single focal point is needed in order to be able to effectively produce the SPS environment. Third, develop SPS incrementally. Given that the SPS requirements and some of the technology issues were not completely understood, incremental development was the most effective strategy for proceeding. Fourth, commit to using SPS on a large-production software project. This ensured that SPS would be realistic and that early feedback from its use could affect later increments. Fifth and last, measure the impact of SPS on its users' productivity. This helps determine whether the SPS is having the desired effect and helps shape requirements for future SPS increments.

SPS architecture and components

On the basis of our preliminary conclusions, SPP was established in January 1981 to implement the SPS. This internally funded project's mission is to develop a fully integrated TRW software development environment. We recognize this goal is very ambitious, but we believe significant progress toward that goal will generate the desired

![Figure 3. Comparative software productivity ranges. The overall impact of tools on software productivity can be determined, for example, in this way: If 1.24 = very low tool support and 0.83 = very high tool support, than the total impact is 1.24*0.83 = 1.49.](image-url)

Table 1. Evaluation of overall productivity strategy.

<table>
<thead>
<tr>
<th>COCOMO ATTRIBUTE</th>
<th>1981</th>
<th>1985</th>
<th>1990</th>
</tr>
</thead>
<tbody>
<tr>
<td>Use of Software Tools</td>
<td>1.05</td>
<td>0.94</td>
<td>0.88</td>
</tr>
<tr>
<td>Modern Programming Practices</td>
<td>1.07</td>
<td>0.89</td>
<td>0.83</td>
</tr>
<tr>
<td>Computer Response Time</td>
<td>1.02</td>
<td>0.91</td>
<td>0.89</td>
</tr>
<tr>
<td>Analyst Capability</td>
<td>1.00</td>
<td>0.88</td>
<td>0.80</td>
</tr>
<tr>
<td>Programmer Capability</td>
<td>1.05</td>
<td>0.90</td>
<td>0.88</td>
</tr>
<tr>
<td>Virtual-Machine Volatility</td>
<td>1.06</td>
<td>0.95</td>
<td>0.90</td>
</tr>
<tr>
<td>Requirements Volatility</td>
<td>1.27</td>
<td>1.06</td>
<td>1.00</td>
</tr>
<tr>
<td>Use of Existing Software</td>
<td>0.90</td>
<td>0.70</td>
<td>0.50</td>
</tr>
<tr>
<td>Cumulative Multiplier</td>
<td>1.46</td>
<td>0.34</td>
<td>0.19</td>
</tr>
<tr>
<td>Productivity Gain</td>
<td>3.4</td>
<td>7.8</td>
<td></td>
</tr>
</tbody>
</table>
productivity gains. The average number of people working on SPP over the last three years is 14, excluding personnel supporting the computing facilities.

The four major areas of the SPS environment are (1) Work environment—define better work environments, which includes providing private offices and immediate access to computing facilities; (2) Hardware—evaluate and procure new hardware equipment, including computers, printers and local area networks; (3) Master project database—define and implement a master project database that contains all information relevant to project activities including budget, personnel, schedules, and other managerial data in addition to such technical information as software requirements, design, test procedures, and code; and (4) Software—provide an integrated tool set, supporting the entire software development life cycle with convenient access to the master project database.

Following the study recommendations, the work was to be done incrementally and a large production project was chosen to benefit from the improvements provided by SPS. These facts had an impact on the initial SPS requirements and implementation, since hardware decisions had to be made with current technology and availability and tool priorities were highly influenced by production project user needs. From 1981 to the present, progress has been made in all the preceding areas. The most success achieved has been in the work environment; the most work remains to be done in producing an integrated master project database and a fully integrated set of tools. The remainder of this section describes the current state of each component, indicates the short-range compromises, and explains envisioned improvements.

Work environment. SPS work in this area involved evaluating current office facilities and environmental conditions of project personnel, experimenting, and proposing a new environment. Typically, technical staff members work in relatively Spartan offices shared by two or sometimes three people. Large programming "bullpens" are a rarity; however, private offices are very rare and are provided on a space-available basis only. Standard functional business desks, filing cabinets, chairs, and bookcases adorn each office. Except in several leased buildings, there is typically no carpeting or wallpaper; however, each office usually has floor-to-ceiling walls with a closable door.

SPS proposed a productivity environment which includes (1) private offices of approximately 90 to 100 square feet with floor-to-ceiling walls, carpeting, soundproofing, an ergonomic chair, adequate work space, storage, and lighting; and (2) a terminal or personal workstation for each office, with a high-speed network connection to a number of computers, file servers, and printers. As part of the experiment, 39 productivity offices were constructed in one of our major facilities. A broadband coaxial cable was installed in this building and the office occupants had access to eight Vax computers through a local area network. These offices housed both SPP personnel and the supported project personnel. Two surveys of occupants conducted six months apart indicate that this productivity work environment had a real impact on their daily activities (see "Usage measurement" on p. 41).

With the successful completion of the work environment experiment, the responsibility of building productivity facilities and cabling other buildings has been transfer-
Hardware. The current SPS architecture is shown in Figure 4. It supports a broadband local area network that has been operational since January 1982. Currently, approximately 200 users share this network. As more buildings are cabled, that number is expected to rise sharply. The network is used primarily for high-speed terminal-to-computer communications (up to 19.2K baud).

SPS is based on the source/target concept of operation; that is, the source development machine hosts the integrated tool set and the master project database. The source development machine can be reproduced from project to project providing the same source environment independent of the type of target machine. Currently, the configuration is centralized and the source machine is a Vax 11/780, which runs the Unix operating system as illustrated in Figure 4. An IDM-500 database machine and laser printers are attached to the source machine. In 1984, two more Vax machines will be added to the existing configuration, linked together through an Ethernet.

The LAN is a general utility that can be used by projects not using either Unix or SPS software. For example, one major project has eight Vax computers, all of which are attached to the LAN. Seven run VMS; one runs Unix with the full complement of SPP-developed software. Other large non-Unix machines, such as a large IBM mainframe used for business applications, will likely be connected to the LAN within the next year.

Several company computers running Unix are connected by way of a UUCP mail service and belong to the Usenet Unix network. Mail is exchanged daily with many sites around the country. Additionally, a Unix emulator furnished by the Wollongong Group permits electronic mail and file transfer service between the Unix and VMS machines.

It was realized early in the project that, as the SPS user community expanded, a centralized configuration using several Vax machines would rapidly become saturated. Thus, as early as 1981, personal workstations started being considered. It was worked out that a single underlying database structure would be overly constrained to support a large software project. In the absence of a database management system to support this structure, we elected to use a multidatabase support structure for SPS' current MPD that includes

- A hierarchical file system for the software artifacts. This is provided by the Unix file system.
- An update-tracking system for representing the successive updates of each artifact. This is provided by the Unix source code control system, or SCCS.¹⁰
- A relational database management system for representing the relations between artifacts. This is currently

Master project database. As emphasized in the Ada Stoneman requirements document,¹⁴ the core of a software support environment is the project master database of software artifacts such as plans, specifications, standards, code, data, and manuals. Figure 5 shows a high-level view of this project database. On the left are the various products generated by the project: specifications, code, manuals, and reports. On the right are the various classes or resources required to develop the products: capital dollars, labor dollars, personnel, and computer resources. In the center are the various plans that link the expenditure of resources to the creation of products: development plans, schedules, work breakdown structures, and unit development folders. The upper half of Figure 5 indicates the primary entities in the MPD, generally stored as hierarchical text files with change or version tracking. The lower half of the figure indicates the various attributes of the entities, such as architectural relationships (in the product portion) or traceability relationships (in the plans portion).

The TRW productivity work environment—which incorporated a master project database and a fully integrated set of software tools—reportedly improved office production.

This master database must support efficient query and update of software artifacts; representation of relations between artifacts (such as requirements traceability), and effective configuration management (version control, change control, problem report tracking, and library management) of the various versions and updates of the software artifacts.

The definition of the contents of this MPS is an ongoing activity. The high-level structure defined is illustrated in Figure 5; the definition of the detailed entities, attributes, and relationships is being addressed as research projects. Some portions have been worked out in enough detail to support the development of key SPS tools such as the requirements traceability tool. Others, such as the entities, attributes, and relationships involved in software project planning and control, have been developed in some detail (see Software Engineering Economics,¹ chapter 32) but their implementation is still awaiting the complex job of integrating the SPS master project database with the corporate financial database.

On the basis of an analysis of our previous support environments, proposed Ada programming support environments, or APSEs, and others such as Unix/Programmer's Work Bench and Gandalf,¹⁰ we determine that a single underlying database structure would be overly constrained to support a large software project. In the absence of a database management system to support this structure, we elected to use a multidatabase support structure for SPS' current MPD that includes

- A hierarchical file system for the software artifacts. This is provided by the Unix file system.
- An update-tracking system for representing the successive updates of each artifact. This is provided by the Unix source code control system, or SCCS.¹⁰
- A relational database management system for representing the relations between artifacts. This is currently

June 1984

35
pro, provided by the Ingres relational DBMS and the IDM-500 database machine.

Integration of the various elements of this structure has been partially achieved; we envision doing much more work in this area in the near future. The MPD as currently organized has provided a solid, workable base for software configuration management for SPS. All baselined source code, manual pages, user's manuals, and other software artifacts are controlled through SCCS in what is called an electronic maintenance folder, or EMF. All software produced by SPP is stored in a single Unix directory. Under this directory, there is a separate subdirectory for each package—that package's EMF. For example, there is a subdirectory calendar, a subdirectory fillin, and a subdirectory menu for three packages developed by SPP. Each EMF directory is further subdivided into separate subdirectories for source code, documentation, manual pages, test information, requirements, and design. Since the configuration for storing artifacts is uniform for all SPP software, it is a straightforward task to locate the artifacts relating to any version of SPP-developed code.

Using SCCS to guard the artifacts guarantees that no one except the SPP configuration manager may change any controlled document, that all changes are recorded, and that there is full opportunity to recover earlier versions. Through control procedures established by SPP and supported by SCCS, developers and managers have access to and can update a copy of a document without affecting the official baselined copy. In addition to using SCCS for documents controlled at the project level, many SPP members apply SCCS to other documents for subproject or personal use.

Until recently, almost no one outside SPP used SCCS for configuration control. That has been changing over the last few months as people in other projects have become aware of the power that SCCS offers. For example, one large project now baselines system engineering specifications using SCCS. It is expected that usage of SCCS will grow dramatically in the future.

Software. Internal projects develop software for a diverse array of target machines. The hardware requirements for these projects usually determine which operating system is used, and therefore, which software development tools are available. An unfortunate consequence of this diversity is the need to retrain developers when they change projects. One of the major focuses of SPS is to provide a portable software development environment that will operate on many hardware environments. The goal is to make it possible for a developer to change projects yet have a relatively constant software development environment. This would mitigate what is currently a large retraining problem.

Projects usually build a collection of software tools to support their software development. Unfortunately, the need to satisfy its own pressing project commitments

Figure 5. A high-level view of the project master database including (a) products—that is, what the project produces; (b) plans and results—that is, how resources are expended for such activities as product development, testing, maintenance, change control, and resource control; and (c) resources—that is, what the project consumes in developing products.
An extensive study of candidate operating systems to support the SPS was performed in 1981. Because of the strong requirement for portability, the need to provide a sound basis for developing a powerful software development environment, and the need for immediate tool availability, SPP decided to use Unix to host its software development environment. SPP began with the large collection of tools offered in the UC Berkeley version of Unix. It has expanded that software base in three ways: building, buying, and porting.

In 1981, a group of software developers and managers identified a small set of software tools to be incorporated in the SPS to support our software development methodology. Since then, on a yearly basis, a new plan of expansion is prepared and coordinated with company management and existing or potential SPS users.

Whereas, the long-term goal is to build a fully integrated development environment, short-term needs and limited resources influenced our decision of providing integration only at the level of the tools we build. Thus, tools that were bought and ported are not necessarily integrated with the existing ones. Examples of purchased tools are the Viewcomp spreadsheet calculator from Unicorp and the Ada compiler from the Irvine Computer Science Corporation. Examples of available tools that were obtained free or for a nominal fee from universities and research institutes are the Rand message handler, Wang Institute's Wicomo, Donald Knuth's text formatting language Tex, and Purdue University's revision control system. Occasionally a tool, such as the TRW Fortran 77 analyzer—which was written for another operating system in fairly portable Fortran 77 code—has been rehosted onto Unix by SPP.

The SPS tools built by the project are grouped in three general categories: (1) general utilities, (2) office automation and project support, and (3) software development.

**General utilities.** Reuse of existing software is being accepted as the most powerful way to improve productivity. Unix already provides a number of library packages that can be readily incorporated into new software—for such mundane tasks as mathematical computation to more interesting operations such as screen management. All software written by SPP takes as much advantage of these packages as possible. SPP has also spent much effort in building general tools and utilities that provide for reusability within the SPS environment. Written in the C programming language, these packages include

- Menus, a menu user interface tool, available both at the command level and subroutine level, that (1) provides a simple menu language to allow format and contents of menus to be easily built, (2) supports convenient selection of menu choices, and (3) allows for prompts and help definition on each selection. The package is template-driven, and it handles all the user interaction with the menus. In this manner, any tool that provides menus uses this package, and user interface consistency is achieved across tools. A sample of a menu display is illustrated in Figure 6; this menu is the interface of a front end to the Rand mail system.
- Software productivity editor, or Sped, a collection of subroutines that implement a simple screen-oriented editor, compatible with the Unix editor vi. It performs simple editing operations such as character insertion, deletion, replacement, and cursor movement operations. Tools can make use of this capability to permit the editing of windows without losing the context. Sped makes it possible for SPP-developed tools that require simple editors to have a uniform interface.
- Fillin, a fill-in-the-blank user interface tool, available both at the command level and subroutine level. It provides a simple form language that allows format and contents of forms to be easily built. The package is template driven, and it handles all the user interaction with the forms. It displays a form on the screen for editing or read-only access, and supports sophisticated editing within fields. Any tool that provides fill-in-the-blank interfaces makes use of this package, and thus user interface consistency is achieved across tools.
- Date/time, a set of subroutines that does relatively sophisticated analysis of dates and times; expressions such as “three weeks from next Tuesday at noon” can be parsed, normalized, and displayed in a variety of output formats. Arithmetic operations on times and dates are defined.
- Report writer, a set of subroutines to aid in report writing. It takes files containing data, header information, and formatting directions, and produces a new

![Figure 6. The menu for mail messages.](image-url)
file suitable for processing by tbl and troff, two standard Unix programs for manipulating tables and performing text formatting. It supports the flexible generation and formatting of reports.

Every time a major application is developed by SPP, an analysis is performed to determine if a reusable library component can be constructed as a by-product of the implementation of that application. Further analysis determines whether already existing libraries can be used to minimize development effort.

Office automation and project support. A software environment must support all project personnel including secretaries, business managers, project managers, as well as technical staff. As illustrated in Figure 2, two thirds of the time spent on a large software project results in documentation (such as activity reports, memoranda, documents, and plans). Thus, SPS includes a large set of tools for office automation and project support, which constitute the automated office. Most of these tools provide both command-oriented and menu-driven access to a number of basic Unix tools relevant to office functions as well as to new tools developed by SPP. A user can choose which interface to use; thus, when a beginner becomes very familiar with a particular tool, going through the menu interface can be avoided. Because of their relative importance, word processing, forms management, electronic mail, and calendar management have received the greatest emphasis.

Large software projects must cope with a wide variety of forms such as software problem reports, change requests, expense reports, purchase requests, travel forms, and data entry forms. To accommodate these in an efficient and unified way, SPP has developed a forms management system, or FMS. With FMS a user can create, edit, summarize, copy, query, and perform a number of other operations on forms both individually and grouped into folders. For example, programmers can maintain a separate folder for each software unit they are developing; each folder containing the problem reports for that unit. Anyone with access to that folder can query on such attributes as whether a problem report is still open or when the problem is to be resolved. Summary reports can be produced on the form instances in the folder. About a third of the FMS consists of software components developed earlier by SPP. For example, the FMS uses the fillin and report writer packages.

The SPS automated office differs from most of its commercial counterparts in that it is not a self-contained system complete with a menu-driven interface. The primary reason for this difference is to capitalize on the flexibility provided by Unix. One of the major strengths of Unix is the ready availability of large numbers of simple commands for a user to invoke at any time. Encased systems normally do not have full access to basic Unix services, such as pattern matching for file names. The user therefore loses much of the advantage of Unix or the tool builder is forced into duplicating services already on Unix but otherwise inaccessible to his system's users. All office functions in SPS are available as ordinary programs directly accessible from the Unix command interpreter.

In several instances, menus or on-line help capabilities have been added as front ends for what are otherwise independent programs. For example, a novice user can invoke a menu interface to the Rand message handler, the standard SPP mail system, as illustrated in Figure 6. Most experienced users prefer the raw command interpreter interface, but novices find the menus quite helpful.

Even though SPP has used most of the Unix tools as standards for tool integration, tools of similar functionality are made available upon user demand. For example, troff is the standard text formatter used by the project, but other formatters such as Scribe and Tex are also available.

The major automated office/project support components, other than those provided with the standard UC Berkeley Unix, are

- SPS tool catalog, a catalog of the tools that are part of the SPS. This tool makes use of the menu utilities and the forms management system, where summaries of existing tools are built automatically. In this manner, the maintenance of this catalog as SPS grows is trivial.
- Rand message handler, or MMH, an electronic mail system from Rand.
- Scribe, a text-formating and document-preparation system that complements troff.
- Tex, another text-formating and document-preparation system that complements troff.
- Author, a word processor built by SPP that closely approximates 'what you see is what you get' text entry; actually a front end for the standard Unix text processor troff. The user is shielded from troff commands by seeing an approximation of their effects on the screen as he enters data. Author makes extensive use of keypad function keys; its back end produces a file compatible with troff so that the full power of that formatting program is retained.

- Interoffice correspondence package, or ICP, a collection of commands created by SPP to manipulate TRW standard interoffice correspondences. ICP uses the fillin package.
- Menu message handler, or MMH, a menu-driven variant of the Rand message handler that was created by SPP. MMH uses the SPP menu driver to provide menus complete with on-line help to aid the novice user of electronic mail.
- Calendar management system, or CMS, a collection of programs created by SPP to manipulate personal calendars, including scheduling, summarizing, rescheduling, and canceling of appointments, travel, and daily notes.
- Forms management system, or FMS, a system created by SPP that allows easy creation of new forms, a uniform way to manipulate electronic forms, management of forms in folders, easy report writing capability, and a version of query by example; and
- Viewcomp, an electronic spreadsheet calculator.

Because the forms management system permits end users to define their own form formats and reports, SPS users have built additional tools as forms applications, including an inventory control system, a design problem report tracker, and the SPP automated library. New applications of FMS are being created by innovative users.
regularly. Of particular interest is that secretaries and administrative support personnel have created many of these applications, rather than technical staff.

Software development tools. TRW has iterated several times on a complete methodology for software development including the implementation of tools to automate and reinforce much of that methodology. Most of the currently available tools to support software development have been selected in part on their degree of support for that methodology.

The major software development components, other than those provided with standard UC Berkeley Unix, are:

- Requirements traceability tool, or RTT, a bookkeeping capability that allows the user to trace requirements through software design and test; it generates several reports including a test evaluation matrix and exception reports. The actual text of requirements is not stored as part of this tool, but it allows a user to specify the name of the file where it resides. This tool was built by SPP, and a user may choose to use either the Ingres database management system or the IDM-500 database machine.

- Program design language 81, or PDL-81, a well-known design tool that is available on Unix. It was purchased from Caine, Farber and Gordon, Inc.

- Ada program design language, or Ada PDL, a tool supporting a program design language based on Ada. It supports Ada concepts such as packages, as well as providing such standard PDL capabilities as pretty-printing, and cross-referencing.

- Fortran 77 analyzer, a code analyzer for ANSI Fortran 77 programs, originally developed by TRW for the National Bureau of Standards. It is useful as a static-code analyzer, test-effectiveness measurer, and general software-development aid. This tool is widely used throughout the company, and it was ported to the SPS system.

- Software requirements engineering methodology, or SREM, a methodology and a collection of tools, developed by TRW for the US Army BMD-ATC, supporting the definition and analysis of software requirements. A preliminary version of SREM was ported in 1982; an enhanced version including integration with the IDM-500 database machine is planned for 1984.

SPS support scenarios. All of the four major SPS components—hardware, software, office facilities, and master project database—are separable, that is, a project can use one or more of these components without the others (although effective use of the database without the accompanying software tools would be somewhat strained). Thus, projects using a non-Unix environment (such as the Sel MPX operating system, or DEC's VMS) have benefited significantly from having a LAN and the specially furnished private offices. Conversely, projects whose current operating conditions have not permitted use of private offices have still benefited by using SPS software and Unix to support their software development activities.

The simplest implementation scenario, of course, is for a project to have all SPS components, and to deliver software that is also SPS-based. However, it is often necessary to deliver software for target systems for which SPS-based software is not appropriate; for example, in many cases the target hardware is a network of small microcomputers with severe memory limitations intended for real-time applications. The source-target concept of operation offers an attractive approach that retains the advantages of a SPS-based software development environment, while satisfying customer requirements to deliver software on a non-SPS target system. If there is a large degree of support for source-target operations, including high-bandwidth communications and tools such as cross-compilers and remote testers, then a project can work almost exclusively from the SPS-based development system throughout the entire life cycle.

In the event that support for source-target operations is incomplete, an SPS-based system can still be used very effectively for the earlier life cycle when systems engineering, management planning, and design activities dominate, and in the later life cycle for such activities as documentation, configuration management, and project management, which do not rely on the particular target environment. This actually reflects the operating scenario for a major project which SPP is supporting. Development and maintenance of most documentation, design, and other activities that do not require actual use of the non-Unix-based target environment are done under Unix. Coding and testing are performed on the target machine.

Project experience

SPP has learned many lessons during the last three years, including problems and benefits due to the incremental nature of the SPS, the user support program, and some of the measurements to date.

Evolutionary development. When developing a software environment, prototyping and evolutionary development is preferable to paper analysis and detailed requirements specifications. This has been especially true for SPP. The project has changed direction several times during the last two and one-half years as real user needs became better understood. For example, the earliest project plans did not call for the development of office automation software. However, the large production software project serving as the representative SPS user indicated that several office automation capabilities would be more valuable than some of the originally-planned SPS tools.

Two aspects of our evolutionary development have already returned benefits: the building of tools to provide for user interface consistency and the building of tools for reusability.

From the beginning of the project, special emphasis has been given to the uniformity of the user interface. Since most SPS tools are interactive, SPP developed a set of user interface standards that include syntax standards, a help language, interfacing, and documentation. Very soon we realized that reliance on tools to provide much of the user interface fosters more effective standardization than does the issuance of standards documents. Two examples of such tools are the menu and fillin subroutine libraries, the
latter having been the most widely used SPP-developed tool. Thus, the commands to deal with menus or to fill forms on the screen are the same, independent of the tool being used. SPP needed to provide a friendly, uniform way for users to interact with the SPS, and these two libraries have been remarkably effective in doing just that.

Another advantage of these tools is the ease with which user interface protocols can be partially standardized for non-SPP-developed software. SPP recognizes that one role for SPS is to be a repository for useful programs such as the Rand message handler. Modifying the internals of that package so that it conforms to SPP user interface standards would be impractical because of the package's large size. However, placing menus or forms in front of package components to collect much of the information required to operate the package is relatively easy. Such efforts have the effect of providing high-level integration of otherwise independent tools.

In the early days of the project, we built a few tools that had hardwired information or hardwired attributes; these tools would suit one type of user, but not all. One example was the automated unit development folder, built in early 1982, which supports keeping a unit development folder for each software unit being developed. Every new user wanted one more attribute or one more report, which meant recoding parts of the tool. We realized that building general and flexible tools—that users could tailor to their own needs—was the only way to satisfy users within and across projects. This is not a trivial task, but we have been successful in building tools for that purpose; an example is the forms management system, where users can easily define the contents of forms and reports to be used by the system.

The building of reusable tools and libraries, some of which were previously described, have also increased SPP's productivity.

Training. SPP recognized early in the project that its best technical efforts could be thwarted by a lack of support for a large user community who would initially be unfamiliar with Unix and SPS. We have paid special attention to the beginner and the casual user. To ensure user satisfaction, SPP took a three-fold approach to meeting the general users' needs.

Documentation. User manuals are written and on-line help is provided for each locally developed tool. In addition, supplements to existing Unix documentation were written explaining, for example, the most commonly used system commands. Sections of existing Unix documents which were found lacking were rewritten; for example, SPP wrote a tutorial introduction to the screen-editor vi more suitable for computer novices than the one distributed from UC Berkeley. This tutorial has proven instrumental in gaining acceptance for that editor.

A library has been established in a central location where a user can go to find a document about SPS. A reference copy of large manuals is available, as well as copies of smaller documents, which a user can take. Access to information in this paper library has also been automated as an application of the forms management system.

Consulting. A regular consulting service was established so that users outside SPP could obtain expert help on all aspects of SPS. Consultants are available daily.

Courses. Several in-house courses were developed and are offered regularly. Besides courses on introduction to Unix and specific Unix tools, specialty courses such as the forms management system, Author, and the requirements traceability tool are offered.

Experience supporting other projects. When SPP was formed, there was concern that the user community outside SPP would resist the different way of approaching software development that SPS and its accompanying methodology support. This skepticism was anticipated for several reasons. First, SPP proposed to automate many activities that had previously been done manually. Automation requires significant managerial and social adjustment. Second, Unix is different from what most TRW software developers are familiar with, and it takes a lot of work to learn another operating system and collection of tools; the users must be persuaded there is a large payoff in order to warrant such effort. Third, in some corners, Unix has the reputation of being too academic and therefore might not be appropriate for supporting large-scale real-time software development. Fourth and last, until the beginning of this year, Unix has not really been supported by AT&T or the major computer manufacturers in the sense that DEC supports VMS. This lack of visible support caused concern over operating system maintenance. These concerns prompted SPP to pay extra attention to make sure that users were consulted on requirements, software worked well when released, users were trained on how to use it, user manuals were well written, the tools placed into SPS would offer valuable services not easily found elsewhere, and the released software baseline was placed under rigid software control. This strategy is paying off. Acceptance of Unix and SPS has been steadily increasing. SPP began supporting one project in 1982. It is now supporting several contract projects, several research
and development projects, and non-project organizations. Use of Unix independent of SPP is also growing, providing an opportunity for SPS software to be disseminated. Some proposals are now being written specifically including the use of SPS for their software development.

Further enhancing the credibility of Unix are the many announcements of vendor support for Unix. Among these companies are AT&T, DEC, IBM, Gould, Hewlett-Packard, Intel, Motorola, Amdahl, Data General, and National Semiconductor. All these vendors have released or soon will release supported versions of Unix on their hardware. (In addition to providing support on their own computer systems, AT&T has entered into agreements with many vendors to support non-AT&T computers.)

Usage measurement. To determine the effect of SPS on user productivity, and to identify areas most needing improvement, SPP is using a four-fold measurement and evaluation approach: (1) automatic instrumentation of system usage; (2) encouraging informal feedback from users; (3) questionnaires to assess system impact on software effort distribution, and to acquire subjective user inputs on SPS and its impact on productivity; and (4) long-range measurement of project size, attributes, and development effort to support a continuing productivity audit along the lines of the Cocomo model discussed earlier.

SPP has started a system usage measurement activity, collecting data on tool usage and CPU utilization per user group. Tools have been categorized by their functionality and system users have been grouped by the type of job they perform (such as management, clerical, development, and engineering). Data is automatically collected and reports are generated weekly. Data is analyzed periodically, and some of our observations include: (1) word processing is the heaviest CPU-using application; (2) mail and network communications are not major CPU drains even though they are used heavily; and (3) the 80-percent-to-20-percent rule of program optimization seems to apply to computer usage; that is, 20 percent of the individual users consumed 76 percent of the CPU. The benefits of this measurement activity include a better understanding of which types of tools need to be added to the system, to pinpoint programs for which optimization would have a large payoff, to determine which user groups need better hardware support (for example, dumb terminals versus workstations).

Unfortunately, it is still too early to be able to state productivity improvements for large software development projects in terms of lines of code per man-month. The initial project that SPP has been supporting is still early in its overall project life cycle. However, there are three significant measurements that can now be reported, two surveys and SPP's own productivity in developing software.

The surveys were conducted to obtain subjective feedback on the impact of being in a productivity office with access to the LAN through a personal terminal. The first survey of 37 people included 10 SPP personnel and 27 people from the project that SPP was supporting. That survey asked performers who had occupied the offices for approximately six months to estimate any changes in their productivity. The response, shown in Figure 7, indicates that on the average their estimated gain was 30 percent. In a second survey, people who had occupied a productivity office and subsequently moved to a new location were asked how their productivity was affected by that move. These people had the added perspective of moving out of the productivity environment into traditional TRW surroundings. Of the 23 possible respondents, 20 indicated that they felt their productivity in the SPP environment was 47 percent higher. Figure 8 shows how much each respondent felt productivity was affected by occupying an SPP office. Figure 9 indicates how much each of five major factors contributed to the 47-percent productivity gain. The dominant features were the availability of software tools to support software development and office functions (15.6 percent), a personal terminal with high-speed access to the computers (14.2 percent) and a private office with modern office furniture (7.7 percent).

SPP recently completed the development of its own medium-size software package, the forms management system. This software, written in the C programming language, involved the writing of 11,995 lines of new code (excluding comments) and approximately 500 pages of documentation (including requirements, user interface
specification, design, test information, and user manuals),
plus the reuse of 4974 lines of existing C code. The total ef­
fort for this development—where DSI is delivered source
instructions—was 40 man-months, or MM, for a produc­
tivity of approximately

\[(11,995 + 4974)\text{DSI}/40\text{MM} = 16,969 \text{DSI}/40\text{MM}
= 424 \text{DSI/MM}\]

This productivity was analyzed using the Cocomo model
of software cost estimation.1 Cocomo was supplied with
parameters that represent the development environment
that SPP would have had as a typical company project,
without the benefit of the SPP environment and the SPP­
developed software tool libraries. Under those conditions,
Cocomo predicted an implementation effort of 83.7 MM,
for a nominal productivity of 203 DSI/MM. SPP’s actual
productivity was 109 percent higher than that predicted for
a typical company project. The dramatic difference be­
tween actual and estimated productivity is largely ac­
counted for by the fact that 29.3 percent of the FMS is
reused code. It justifies the direction that SPP has taken in
building reusable components as its foundation for con­
structing application packages. Even without counting the
reused code, the productivity gain on development of the
11,995 lines of new code was 42 percent over what
Cocomo predicts a classical TRW project would have re­
quired.

The primary conclusions from the software produc­
tivity requirements analysis are (1) significant productivity
gains require an integrated program of initiatives in several
areas; (2) an integrated software productivity improve­
ment program can have an extremely large payoff (a factor
of four by 1990); (3) improving software productivity in­
volves a long, sustained effort; (4) in the very long run, the
biggest productivity gains will come from increased use of
existing software; (5) software support environment re­
quircments are still too incompletely understood to specify
precisely.

The SPP development experience, to date, conforms the
study conclusions. Additional conclusions are

Immediate access to a good set of software tools has the
highest payoff. The key element noted has been the
gradual change of daily habits, from preparing paper
products manually to the automated way. This change
is only made effective by providing immediate and easy
access to computing facilities containing a good and inte­
grated set of tools.

Office automation and project support capabilities are
required for all project personnel. These are among the
most often used programs on the system, crossing proj­
et assignments. However, in order to be fully effective,
every member of the project must have a terminal on
their desk.

There is a high payoff in placing all software develop­
ment artifacts on-line and providing tools to support
easy access to them. Developing a comprehensive
master project database is extremely worthwhile, but
 costly. Developing the tools to take full advantage of the
availability of the data is even more expensive, but can
dramatically improve productivity.

User interface standards are essential for preserving the
conceptual integrity of an evolving support system. An
excellent way to implement such standards is to embed
them into a family of toolbuilders’ utilities supporting
such functions as screen formatting, error processing,
help messages, and data access.

User acceptance of novel development environments is
a gradual process that requires careful nurturing by the
sponsoring organization. Involvement of the user com­
unity in planning the growth and direction of the en­
vironment will help ensure their acceptance of it. Train­
ing and documentation writing efforts must be
established early and be given strong support.

Local area networks strongly support distributed work­
environments. A LAN coupled with electronic mail
allows a physically scattered group of individuals to
work effectively as a team with excellent communications
between project members.

Private offices improve productivity. Both surveys con­
ducted by SPP indicated that privacy was a factor in
improving productivity. Organizations should plan for
enough space to ensure that each technical staff member
has a private office.

The integrated approach produces a high payoff. The
survey results of 39-percent and 47-percent productivity
increases, and the 109-percent improvement on the
forms management package indicate that the produc­
tivity results clearly justify the investment in the system.

The software productivity project is planning to con­
tinue its activities for many years. As mentioned, improv­
ing productivity involves a long, iterated, sustained effort.
Main activities planned for 1984 include evaluation of
workstations as part of the environment, enhancement of
existing tools, evaluation of graphics hardware and soft­
ware for use by tools with graphical applications, further
database machine experimentation and expansion of the
user community.

Acknowledgments

We thank all members of the software productivity
project for their dedication, hard work, creative ideas, and
enthusiastic support—which made this work possible.

References

1. B. W. Boehm, Software Engineering Economics, Prentice­


3. "Science and Engineering Education in the 1980's and
Beyond," US National Science Foundation and Depart­

4. J. Buxton, Requirements for Ada Programming Support
Environments: Stoneman. US Department of Defense,

5. K. Christensen. "Programming Productivity and the
Development Process," IBM Santa Teresa Laboratory,
tech. report 03.083, Jan. 1980.


Additional Reading


Maria H. Penedo is a staff engineer in the System Engineering Laboratory at the TRW Defense Systems Group and is currently involved in building a software development environment for TRW projects. She is also the principal investigator on the Integrated Software Master Database IR&D project. Penedo has worked on TRW's Ada technology activities. Previous positions include research in computer-aided instruction and in software methodologies and software development environments with the UCLA System Architect's Apprentice project.

Penedo received her BS in mathematics at Santa Ursula University in Brazil, her MS in computer science at the Pontificia Universidade Catolica in Brazil, and her PhD in computer science from UCLA in 1980. She is a member of ACM and a distinguished visitor of the IEEE Computer Society.

E. Don Stuckle is manager of the TRW Defense Systems Group Software Productivity Project. Since joining TRW as a computer programmer in 1958, he has been actively involved in the development and implementation of software standards and procedures. He has been a key participant in the formation of the TRW life-cycle software development methodology and accompanying software development policies and procedures. He has also been involved in a variety of software development activities including large, real-time data processing systems, scientific simulations, database management systems, and business data processing applications at TRW.

Stuckle received a BS degree in physics and mathematics from Jamestown College in 1956.

Robert D. Williams is currently the vice president and general manager of the TRW Software and Information Systems Division. Williams is also program manager for Software Development of the US Army's Site Defense Program; he has served as assistant general manager in the Systems Engineering and Integration Division and as chief of the software productivity team, which formulated advanced management tools and concepts for modern software developed for computer networks.

Williams received a BSEE from Texas A&M University in 1954. He is a member of the Operations Research Society, the American Institute of Aeronautics and Astronautics, and the Association of the US Army.

Arthur B. Pyster is director of productivity, quality, and resources for Digital Sound Corporation in Santa Barbara, California, and is responsible for development methodology, system integration and test, training, and computer facilities. Previously, he was an engineer for TRW Defense Systems Group in Redondo Beach, California, where he was responsible for systems engineering on TRW's software productivity project. His research interests include software development productivity and environments, software engineering project management, and programming language design and implementation.

Prior to joining TRW in 1981, Pyster was an assistant professor of computer science at the University of California, Santa Barbara. He is a member of the IEEE Computer Society and the ACM. He has a BS degree in mathematics and computer science from the University of Illinois and an MS and PhD in computer and information science from the Ohio State University.

Questions about this article can be addressed to E. Don Stuckle, TRW Systems, Inc., One Space Park, Redondo Beach, CA 90278.