The TRW Software Productivity System

Barry W. Boehm
James F. Elwell
Arthur B. Pyster
E. Donald Stuckle
Robert D. Williams

1. ABSTRACT/SUMMARY

This paper presents an overview of the TRW Software Productivity System (SPS), an integrated software support environment based on the Unix* operating system, a wide range of TRW software tools, and a wideband local network. Section 2 summarizes the quantitative and qualitative requirements analysis upon which the system is based. Section 3 describes the key architectural features and system components. Finally, section 4 discusses our conclusions and experience to date.

2. SPS REQUIREMENTS ANALYSIS

This section discusses the results of a software productivity study performed at TRW during 1980. This study analyzed the requirements for a TRW-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. Each analysis is summarized below, followed by the study's conclusions and recommendations.

2.1. Corporate Motivating Factors

As a competitive system and software house, TRW has continually strived to improve software productivity. Recently, however, several additional factors have motivated TRW toward a more substantial level of corporate investment for improving software productivity. Four of the primary factors are:

Increased Demand for Software

Each successive generation of a data processing system experiences a significant increase in demand for software functionality. For example, manned space-flight 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 (Boehm, 1981, Chapter 33).

Limited Supply of Software Engineers

Several sources (Business Week, 1980; NSF-DoE, 1980) have indicated that the current U.S. 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 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.

Reduced Hardware Costs

The cost and performance improvements of supermini mainframes, powerful personal microcomputers, and broadband communication systems permit significantly more powerful and cost-effective software support systems.

2.2. The 1980 Software Productivity Study

Given the motivating factors above, TRW embarked on an extensive study during 1980 of its software environment objectives, requirements, and alternatives, which led to recommended strategies for improving software productivity. This study included an internal assessment, an external assessment, a quantitative analysis, and a set of recommended actions, each of which is discussed in turn below.

2.2.1. Internal Assessment

TRW's internal assessment began with a series of interviews with representative higher-level and intermediate-level managers, and software performers. Each interviewee was asked, "If there were only two or three things you could get TRW to do to improve software productivity, what would they be?"

In general, the interviewees were highly enthusiastic, and provided a wide-ranging menu of attractive suggestions for improving productivity. Although there was a general consensus on the primary avenues for improving software productivity (in the four areas of management actions; work environment and compensation; education and training; and software tools), there were some significant differences.

For example, Figure 1 shows the relative importance of these four areas from the standpoint of three classes of TRW personnel: upper managers, middle managers, and performers.

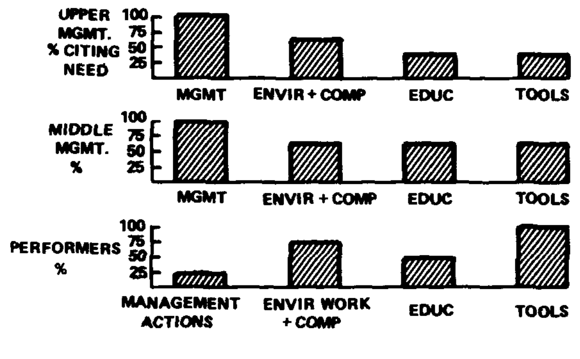

Figure 1. Software Productivity — Perceptions of Major Needs

*Unix is a trademark of Bell Laboratories

0270-5257/82/0000/0137$00.75 © 1982 IEEE

It is evident from Figure 1 that the upper managers' world-view conditions them to see management actions as the high-leverage items, while the performers' world-view conditions them to see 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. If we had proceeded with a big campaign to improve project planning, organization, reporting, etc., without providing the performers with improved tools, our resulting productivity gains would not have been anywhere near their potential.

2.2.1.1. Software Support Environment Requirements

Another portion of the internal assessment involved an analysis of TRW's software support environment requirements. Since the DoD ADA *Stoneman* requirements document (Buxton, 1980) had recently provided an excellent general definition of software support environment requirements for Ada, TRW used Stoneman as its baseline, and focused on identifying additional TRW-specific environment requirements not included in Stoneman. The primary additional requirements identified are summarized below.

1. *Support of Multiple Programming Languages*

   The internal assessment included a forecast of the evolution of TRW's government-systems business base in various dimensions, 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 for existing FORTRAN and JOVIAL systems. Thus, a pure Ada-based environment would not support all of TRW's needs even by the year 2000.

2. *Support of Mixed Target-Machine Complexes*

   A similar forecast of the hardware nature of TRW's future business base indicated a strong trend toward hierarchical mixed-vendor maxi-mini-micro target-computer complexes. Although the APSE concept may provide a unified virtual environment supported on each computer in such complexes, experience to date on such virtual environments as the National Software Works indicate that a number of outstanding problems need to be resolved before one can count on this solution.

3. *Support of Classified Projects*

   Among other things, this implies that a single corporate-wide network with shared data and programs would not be feasible for TRW's classified projects. Such projects have severe access constraints and require extensive precautions to enforce those restrictions. We were not prepared to tackle the problems of ensuring such security in a wide-area network which would necessarily use non-secure communications media. The 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. This is the approach we have adopted.

4. *Integration with TRW Management Programs and Data*

   TRW's tool requirements included a number of project and financial management tools not identified in Stoneman, plus the need to integrate these with existing TRW management support programs and data.

5. *Integration with Office Automation Capabilities*

   TRW studies indicate that about 2/3 of the effort on a large software project results in a document as its direct product, and only 1/3 results in code as its direct product (Boehm 1981, Chapter 31). This and related insights have caused us to emphasize the integration of traditional software tools with word-processing and other office automation capabilities as a top-priority requirement.

6. *Support of Non-Programmers*

   TRW software projects require the close cooperation, communication, and support of both programmers and non-programmers such as technicians, managers, secretaries, hardware system engineers, and business personnel. This implies the need for both tools and a user interface which will support all these classes of people.

2.2.1.2. Uncertainty Areas

In trying to determine the specific information support needs of the TRW software engineer, we encountered a wide variety of user opinions on such items as:

tool priorities (development, management, office support);

attribute priorities (efficiency, extensibility, ease of use by experts vs. novices);

degree of methodology enforcement (do tools assume requirements are written in TRW's Requirements Specification Language, etc.);

command language characteristics; (menu vs. command, terse vs verbose, etc.).

As a result, we concluded that it would be an extremely time-consuming, inefficient, and uncertain process to obtain universal concurrence on a requirements specification for a TRW software support environment before proceeding into design and code. Developing an experimental prototype system and using it on a TRW software project would be a more cost-effective approach.

2.2.2. External Assessment

The 1980 study included visits to a number of organizations with experience or active R&D programs in the software support environment area. The industrial organizations we 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:

Organizations investing in significant improvements in their software environments felt they were getting their money's worth. Some, such as IBM (Christensen, 1980) and Bell Labs (Dolotta et al, 1978) were able to at least partially quantify their resulting benefits.

Organizations achieving some integration of software development support capabilities and office automation capabilities considered this a high-payoff step.

Significant progress was being made toward providing very high-powered personal work station terminals (with high-resolution bit-mapped displays supporting window editors, integrated text and graphics, and well-integrated screen-pointing devices) at a reasonable cost.

No system we saw provided all the capabilities TRW required.

2.2.3. Quantitative Assessment

Our quantitative assessment of alternative avenues for improving software productivity was based primarily on TRW's Software Cost Estimation Program, or *SCEP* (Boehm-Wolverton, 1978). SCEP is similar in form to the *COCOMO* model described in detail in (Boehm, 1981). It estimates the cost of a software project as a function of program size in Delivered Source Instructions (DSI) and a number of other cost driver attributes summarized in Figure 2. Figure 2 shows the Productivity Range for each attribute: the relative productivity in DSI/man-month attributable to the given attribute, after normalizing for the effects of other attributes. 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 high level of tool support will require only 0.83 of the effort required for a project with a nominal level of tool support, while an equivalent project with a very low level of tool support would require 1.24 times the effort required for the nominal project, or 1.24/0.83 = 1.49 times the effort on the "very high" tools project. The "very high" and "very low" ratings correspond to specific levels on a COCOMO rating scale for tool support (Boehm, 1981).

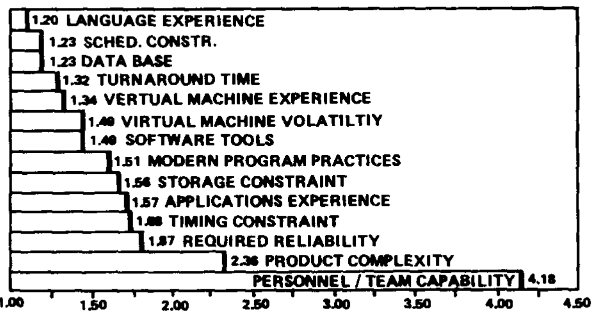

Figure 2. Comparative Software Productivity Ranges (Based on Analysis of 63 Software Projects)

Productivity Audit Results

The Software Tools rating scales and those of the other cost driver attributes were used to conduct a "productivity audit" of TRW projects, to determine the weighted-average productivity multipliers characteristic of the overall TRW distribution of software projects, both at present and for several future scenarios representing varying levels of TRW investment into productivity-improvement programs. 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 a factor of 7.8 by 1990. Besides providing an estimated productivity gain, this analysis provided insights for determining which items (e.g., tools) to emphasize as part of a TRW 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.

Activity Analysis

Concurrently, we performed a complementary analysis which assessed the likely reduction of software project effort devoted to each software activity during each software development phase, as a result of a software environment improvement program[**].

[*]More details underlying this analysis are given in (Boehm, 1981a).

[**]This analysis was performed by E. J. Hardy to support the U.S. Army Ballistic Missile Defense Advanced Technology Center's Distributed Computer System Design project. See (Alford et al, 1981).

Table 1. Evaluation of Overall Productivity Strategy

COCOMO ATTRIBUTE	WEIGHTED AVERAGE MULTIPLIER		
	1981	1985	1990
USE OF SOFTWARE TOOLS	1.05	0.94	0.88
MODERN PROG. PRACTICES	1.07	0.89	0.83
COMPUTER RESPONSE TIME	1.02	0.91	0.89
ANALYST CAPABILITY	1.00	0.88	0.80
PROGRAMMER CAPABILITY	1.05	0.90	0.80
VIRTUAL MACHINE VOLATILITY	1.06	0.95	0.90
REQUIREMENTS VOLATILITY	1.27	1.08	1.00
USE OF EXISTING SOFTWARE	0.90	0.70	0.50
CUMULATIVE MILTIPLIER	1.46	0.34	0.19
PRODUCTIVITY GAIN		3.4	7.8

This analysis was based on the COCOMO model's software effort distributions by phase and activity (Boehm, 1981; Chapter 7). The results are given in Table 2.

Table 2. Activity – Oriented Estimate of Effort Reduction

PHASE	PLANS & REQUIREMENTS	PRODUCT DESIGN	PROGRAMMING	INTEGRATION & TEST	DEVEL-OPMENT TOTAL	MAINTENANCE
PHASE EFFORT Pi (%)	7.4	16.7	44.4	31.5		100.0
ACTIVITY	AE, AS, ES	AE, AS, ES	AE, AS, ES	AE, AS, ES	ES	AE, AS, ES
REQUIREMENTS ANALYSIS	0.42 0.25 0.8	0.10 0.50 0.8	0.03 0.50 0.67	0.02 0.50 0.3	2.6	0.05 0.40 2.0
PRODUCT DESIGN	0.16 0.15 0.2	0.42 0.30 2.1	0.06 0.50 1.3	0.04 0.50 0.6	4.2	0.11 0.30 3.3
PROGRAMMING	0.10 0.10 0.1	0.14 0.50 1.2	0.55 0.25 6.1	0.48 0.60 9.1	16.5	0.41 0.50 20.5
TEST PLANNING	0.06 0.25 0.1	0.06 0.35 0.4	0.08 0.30 1.1	0.05 0.25 0.4	2.0	0.06 0.35 2.1
V & V	0.10 0.20 0.1	0.10 0.40 0.7	0.12 0.50 2.7	0.30 0.35 3.5	7.0	0.14 0.55 7.7
PROJECT OFFICE	0.08 0.30 0.2	0.07 0.25 0.3	0.04 0.20 0.4	0.6 0.20 0.4	1.3	0.05 0.30 1.5
CM / QA	0.03 0.35 0.1	0.02 0.30 0.1	0.06 0.55 1.5	0.08 0.55 1.4	3.1	0.06 0.50 3.0
MANUALS	0.05 0.55 0.2	0.07 0.45 0.5	0.05 0.45 1.0	0.07 0.45 1.0	2.7	0.11 0.55 6.0
TOTAL	1.8	6.1	14.8	16.7	39.4	46.1

The Phase Effort (P_i) row shows the percent of the overall development effort consumed during phase *i*.

For each phase, the AE column shows the COCOMO effort distribution by activity, including an activity distribution for the maintenance phase. For example, Table 2 indicates that 7.4% of the development effort is devoted to the Plans and Requirements phase. Within this 7.4%, Requirements Analysis activities consume 42%, Product Design activities consume 16% of the effort, etc.

The AS (Activity Savings) column shows the results of a Delphi exercise to estimate the percentage of effort for each phase-activity combination that would be saved as a result of using the SPS. For example, 25% of the Requirements Analysis effort was estimated to be saved during the Plans and Requirements phase, etc.

The ES column shows the resulting Effort Savings for each activity *j* within phase i:

$$ES_{ij} = (P_i)(AE_{ij})(AS_{ij})$$

When summed over all phases and activities, the overall results show a development savings of 39%, and a maintenance savings of 46%, exclusive of any savings due to software reuse. These savings are not as great as those estimated by the cost-driver approach, but they are reasonably comparable and quite significant.

2.2.4. Study Conclusions

The 1980 productivity study reached the following major conclusions:

> *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.
>
> *An integrated software productivity improvement program can have an extremely large payoff.* Productivity gains by factors of 2 in four years and factors of 4 in nine years are generally achievable, and are worth a good deal of planning and investment.
>
> *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.
>
> *In the very long run, the biggest productivity gains will come from increasing use of existing software.*
>
> *Software support environment requirements are still too incompletely understood to specify precisely.* In this respect, software support environments fall into an extensive category of man-machine systems whose user requirements are not completely understood.

2.2.5. Study Recommendations

Based on these conclusions, the 1980 study made the following recommendations:

> *Initiate a significant long-range effort to improve software productivity.* The recommended effort included initiatives in all the areas above, and established goals of improving software productivity by:
>
> - a factor of 2 by 1985;
> - a factor of 4 by 1990.
>
> Although these goals are conservative with respect to the estimated productivity gains cited in the Study Conclusions, they are clearly large enough to justify a significant investment into a productivity improvement program.
>
> *Begin by developing a prototype system.* Given that the system requirements and some of the technology issues were incompletely understood, prototyping was the most effective strategy for proceeding.
>
> *Commit to using the prototype on a large production software project.* This ensured that the prototype would be realistic and that we would get early feedback from its use. A related recommendation was that a productivity improvement measurement and analysis activity be an integral part of the program.

3. SPS PROTOTYPE DEVELOPMENT

After an extensive review of the 1980 Software Productivity Study and its recommendations, TRW's management decided to invest a significant effort into developing a prototype SPS during 1981, and to use it on a large TRW software project anticipated to begin its major effort in late 1981. Thus, in early 1981, a two-pronged effort was initiated: one, to determine the long-range hardware-software architectural options for an eventual mid-1980's SPS-2 system (under the name *Advanced Productivity Project* -- APP); the other, to build a compatible SPS-1 prototype to be ready for use by the end of 1981 (under the name *Software Productivity Project* -- SPP). This section describes the overall architecture and concept of operation of SPS-1; discusses each major SPS-1 hardware and software component; summarizes some of the primary development lessons learned to date on SPS-1; and describes the ongoing efforts of SPP. There are no details offered on SPS-2 since it is still being shaped by our experiences with SPS-1.

3.1. SPS-1 Architecture and Concept of Operation

The SPS-1 architecture was developed in response to four primary guiding principles:

1. *Methodology Support*

   SPS-1 should reinforce TRW's life-cycle software development methodology.

2. *Master Database*

   The SPS-1 master database should be both hierarchical and relational.

3. *Local Network*

   SPS-1 should support interactive access to its shared resources via a local network.

4. *Source-Target Concept of Operation*

   SPS-1 should reside in a single type of source machine or source operating system, rather than being rehosted on each of the many target machines for which TRW develops software.

Each of these guiding principles and its impact on SPS-1 are discussed below.

3.1.1. Methodology Support

The essence of TRW's software development methodology is summarized in Chapter 4 of (Boehm, 1981) on the "waterfall" model of the software life-cycle and its refinements (prototyping, incremental development, and advancemanship). Within TRW, the methodology is elaborated in a set of 18 software development policies (Goldberg, 1978) and an underlying set of software product standards. SPS-1 reflects the methodology's strong emphasis on requirements determination and validation via such tools as the *Software Requirements Engineering Methodology* (SREM) (Alford, 1977; Bell et al, 1977) and a *Requirements Traceability Tool* for relating software requirements to design, code, and test cases. SPS-1 reflects the methodology's strong emphasis on management visibility and control via such tools as the *Unit Development Folder* (UDF) (Ingrassia, 1978).

3.1.2. Master Database

As emphasized in the Ada Stoneman requirements document (Buxton, 1980), a crucial element of an effective software support environment is the structure of a master database of software artifacts: plans, specifications, standards, code, data, manuals, etc. This master database must support efficient query and update of software artifacts; representation of relations between artifacts (e.g., requirements traceability) and effective configuration management (version control, change control, problem report tracking, library management) of the various versions and updates of the software artifacts.

Based on an analysis of previous TRW software support environments, proposed Ada Programming Support Environments (APSE's) and others such as Unix/Programmer's Work Bench (Ivie, 1977; Kernighan-Mashey, 1981), PDS (Cheatham, 1981), and Gandalf (Habermann, 1979), we determined that a single underlying database structure would be overly constrained to support a large software project. Therefore, we elected to experiment with a multi-database support structure for the SPS-1 prototype:

A hierarchical file system for the software artifacts;

An update-tracking system for representing the successive updates of each artifact;

A relational database management system (DBMS) for representing the relations between artifacts.

One major advantage of this approach is that each artifact is stored only once and updated in only one place. The fact that it is part of several larger software artifacts is handled by the relational DBMS. (For example, a routine's design can simultaneously be part of the system design spec, a requirements - traceability report, and are individual programmer's Unit Development Folder). More details on the structure of the master database are given later.

3.1.3. Local Network

A key productivity determinant in the 1980 study (Table 1) was interactive software development. To provide flexible, quick-response interactive support for such capabilities as screen editors, window editors, and rapid file transfer, a local network was determined to be most appropriate. Furthermore, to accommodate TRW classified projects, the overall SPS was configured as a federation of local networks with optional connection-gateway processors. Based on growth patterns in similar information networks, we decided that growth potential was the most important feature in selecting the local network's bandwidth. Thus, a TRW-developed multichannel broadband (300 MHZ) cable-TV network was preferable to single channel baseband (10 MHZ) networks such as Ethernet.

3.1.4. Source-Target Concept of Operation

Given TRW's need to develop software for a wide variety of customer-determined target computers, we had two primary options for developing SPS software:

An extremely portable tool system which could be rehosted onto any target computer or operating system;

A source-target configuration, in which the tool system would be available for software development on a single class of source machines or operating systems, and the developed software communicated to the target computer configuration for system integration and test.

Overall, the source-target approach offered more attractive features for TRW's needs; however, such considerations as field maintenance of the support system placed a strong premium on portability as well. Such portability considerations were a major factor in the choice of Unix as the host operating system for SPS-1.

3.2. SPS-1 Components

To achieve our ambitious productivity goals, the concept of an integrated TRW software development environment was defined. Drawing on the best of currently available environments and modified to meet cost constraints and the unique requirements of TRW, the environment has four primary components:

1. *Software.* An integrated tool set, supporting the entire software development life-cycle, and well-engineered to be friendly, portable, extensible, flexible, and robust.

2. *Hardware.* Low cost, medium power, personal computer with identical keyboards; very high quality printing available centrally; moderately high quality printing available locally.

3. *Communications.* High capacity bus which connects terminals and computers in a local area network;

4. *Facilities.* Private offices of approximately 80-100 square feet with floor to ceiling walls, carpeting, soundproofing, adequate work space, storage, and lighting; each office contains a computer terminal with a network connection.

3.2.1. Software

3.2.1.1. Support and General Software

One of our primary goals is to present the same software environment to each user independent of the actual machine he is currently using. All tools should be available on all machines (except where size, speed, or security makes this impossible), and a user should be able to invoke the tool using the same keystrokes. Whenever a tool is revised or a new tool implemented, it should be made available on all machines at approximately the same time. To achieve this degree of uniformity and portability requires that all development computers use the same operating system, regardless of machine differences.

We are not prepared to rehost operating systems ourselves onto new machines as they become available in the market place because of expense and the rapidity with which new microcomputers are appearing. Hence, we wanted an operating system which was likely to be hosted onto new processors by commercial vendors. The only operating system which currently is being implemented on nearly every processor of interest is Unix.

Because the environment must support the general office worker such as a secretary, and because software developers spend the majority of their time performing mundane office tasks (writing activity reports, editing documents, etc.), SPS-1 includes an *Automated Office.* This system provides 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. Because of their relative importance, word processing, forms management, electronic mail, and calendar management have received the greatest stress in the Office. For example, the Office now supports a complete personal calendar management system and will by year's end support multiple calendar scheduling.

The Automated Office is somewhat unique compared to most office systems currently being marketed because it is not encased as a completely separate package. Office functions are Unix shell commands. Menus are optionally placed in front of them using a separate utility. Hence, a secretary who desires a rigidly structured system at the expense of flexibility can use the menu front-end to obtain Office services, while a programmer who can comfortably switch contexts between compiling and reading mail is not hindered.

In addition to the general menu utility, SPP has also developed a *forms management package* which allows the user to enter and manipulate data in a structured manner on the full video screen. This package has far ranging application and has been adopted for most SPS-1 tools, including calendar management, inventory control, problem reporting, action item assignment, and interoffice correspondences. Use of the menu and forms management utilities eliminates one of the major criticisms voiced about Unix -- its terseness.

Another SPP effort is in *front-end help*. A user can display the structure of SPS-1 tools (tools groupings exist for file manipulation, programming, editing, sorting and searching, etc.) and obtain more detailed and better structured help on locally developed components than is available from standard Unix. There are plans to enhance this capability considerably over the next year to permit sophisticated querying about SPS tools, and about libraries of reusable software components.

Reuse of existing software is the most potent way to improve productivity. Unix already provides a number of libraries which can readily be incorporated into new software. All software written for SPS-l takes advantage of these libraries as appropriate. Taking advantage of the *terminal capabilities* and *cursor control* libraries permit software to be written which will work appropriately on nearly all full-duplex asynchronous terminals. The menu-driver and forms management packages, using those libraries, work on a large variety of terminal types from a TI Silent 700 (in a very degraded mode) up through a DEC VT100. In addition, we are currently developing libraries for relatively simple applications such as date analysis (e.g., to allow tomorrow's date to be specified as either "tomorrow", "Saturday", 5 June 1982, 6/5/82, etc.), for relatively sophisticated applications such as the next version of the forms management package, and will soon be developed for graphics applications.

3.2.1.2. Master Database Structure

As discussed earlier, we determined that a multi-database support structure provided the strongest mix of performance and range of functional database capabilities required to support software development and evolution. The particular packages selected for the multi-database components were:

> The Unix hierarchical file system for storing software artifacts.
>
> The *Source Code Control System* (SCCS) (Rochkind, 1976) for representing successive updates of each artifact.
>
> The *Ingres* relational DBMS (Stonebraker, 1976) for representing relations between artifacts.

The Unix file system is the most heavily used database component. Program text, user's manuals, test cases, calendar entries, etc. all reside as ordinary text files. Unix has excellent facilities to organize and manipulate such files and has in effect become the *standard* hierarchcial file system against which other such systems are measured.

All baselined SPS-1 source code, manual pages, user's manuals, and other software components are controlled through SCCS. This 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 all earlier versions. Through control procedures supported by SCCS, developers and managers have access to and can update a version of a document without affecting the official baselined copy. In addition to using SCCS for documents controlled at the project level, many SPS-1 users apply SCCS to other documents for sub-project or personal use.

The most powerful component of the SPS database is Ingres. SPP has designed a single *Unified Database* (UD) which encompasses many aspects of project activities (although for administrative reasons the database is divided into several physical databases). As other development activities become supported by SPS-1 tools, additional relations, relevant to those activities, will be added to the UD.

Figure 3 shows some of the relations in the UD. These components support the specification of relationships between various software artifacts -- requirements, specifications, design, code, and test cases -- which themselves normally reside in the Unix hierarchical file system. For example, a test case may *verify* that a specification has been properly implemented, a relationship which can be captured in the database. There are numerous such interesting relationships between artifacts. Once captured in this way, it becomes possible to automatically determine such discrepancies as when a requirement has no test case to verify its proper implementation. This is of incalculable value in a large software project where there are tens of thousands of requirements, specifications, test cases, etc. One of the SPS-1 tools, the *Requirements Traceability Tool* (RTT) performs such automated analysis on the UD. It supports easy entry of relationships, performs consistency and completeness checks, produces formatted reports suitable for inclusion in larger documents, and supports interactive querying.

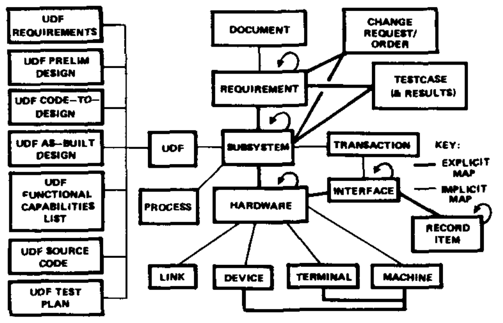

Figure 3. SPP Master Database

Our operational experience with UC Berkeley's implementation of Ingres convinced us that it was functionally satisfactory but unacceptably slow. We have attacked this problem on two fronts. First, SPP is investigating a faster version of Ingres from Relational Technology Incorporated. This will offer a strong performance improvement with no additional hardware overhead. Second, SPP has obtained a functionally compatible but highly efficient Britton-Lee IDM-500 database machine (Britton-Lee 1980). The IDM-500 is not yet operational because of problems in the early software releases from Britton-Lee. However, our preliminary benchmarks indicate that when the IDM-500 becomes fully operational, it will perform an order of magnitude or more faster than Berkeley's Ingres.

3.2.1.3. Software Development Tools

TRW has completed several iterations on a complete methodology for software development including the implementation of tools to automate and reinforce much of that methodology (Lanzano, 1970; Brown, et al, 1972; Boehm, et al 1975; Alford, 1977; Ingrassia, 1978). The initial TRW tools selected for SPS-1 were chosen primarily on the basis of their degree of support for that methodology, the need to support a uniform easy-to-use human interface, their compatibility with an interactive Unix-based system, and their value to the contract software project they were to support. Table 3 summarizes the tools developed for our initial release and those under development for the second release. Short descriptions for some of the tools are given below:

Automated Unit Development Folder (AUDF)

-- TRW keeps a "Unit Development Folder" for each software unit developed. The UDF contains such information as the unit's requirements, design, code, test plans and test results, responsible personnel and milestone dates (Ingrassia, 1978). It has been kept manually in the past. This tool automates the UDF process by using the UD for storing information about each unit.

Table 3. SPS Tools by Category

CATEGORY	1981 RELEASE	1982 RELEASE
REQUIREMENTS AND DESIGN	• S/W REQ ENG METHODOLOGY (SREM) • REQ TRACEABILITY TOOL (RTT) • NETWORK DEFINITION TOOL (NDT) • PDL 72	• REFINEMENTS TO SREM, RTT AND NDT • • PDL 81
DEVELOPMENT	• UNIX TOOLS • SOURCE CODE CONTROL SYS • EDITOR • USER FILE SUPPORT	• UNIT DEVELOPMENT FOLDER (UDF) • UC BERKELEY FORTRAN 77 COMPILER
TEST	• FORTRAN 77 ANALYZER (F77A)	• REFINEMENTS TO F77A
MANAGEMENT		• PLANNING AND CONTROL • INVENTORY CONTROL • PROBLEM REPORT TRACKING
GENERAL USE	• AUTOMATED OFFICE (AO) • ELECTRONIC MAIL • WORD PROCESSING • CALENDAR MANAGEMENT • FRONT END (FE)	• BIBLIOGRAPHY DOC • ACTION ITEM TRACKING • SOFTWARE LIBRARY • PLANS AND PROPOSALS TEXT • REFINEMENTS TO FE
SUPPORT	• MENU, HELP, ERROR UTILITIES • DATA BASE UTILITIES	• GRAPHICS UTILITIES • UNIX-VMS NETWORKING

Program Design Language (PDL)

-- a well-known design tool (Caine-Farber-Gordon, 1977) which is available on Unix. It was purchased from Caine, Farber and Gordon, Inc.

FORTRAN 77 Analyzer

-- as its name implies, it performs static and dynamic analysis of ANS FORTRAN 77 programs. It is useful as a static code analyzer, test effectiveness measurer, and general software development aid.

Software Requirements Engineering Methodology (SREM)

-- supports the definition and analysis of software requirements (Alford, 1977; Bell et al, 1977).

Requirements Traceability Tool (RTT)

-- allows the user to trace requirements through software design and test; generates several reports including a test evaluation matrix and exception reports; references the UD.

Forms Management Package (FMP)

-- provides a uniform way to manipulate electronic forms, including summarization and a version of query by example; applied to such diverse applications as inventory control, problem reports, and calendar management.

AUTHOR

-- a word-processor which supports "what-you-see-is-what-you-get" text entry. The user is shielded from *troff* commands by seeing 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 the Unix formatting programs is retained.

3.2.2. Hardware

3.2.2.1. Processors

SPS-1 has a local network which currently includes one DEC VAX 11/780 Unix source machine, four VAX 11/780 VMS target machines, and 5 LSI-11/23 based semi-personal microcomputers. The acquisition of the latter represents a compromise between cost and purely personal terminals based on equipment which supported Unix available during Spring 1981.

Most SPS-1 users still communicate via a dumb terminal connected to a VAX. Our real goal, however, is to offload functions from the relatively expensive VAX onto low-cost individual Unix/microprocessor based personal computer terminals. We will begin to phase in personal computers over the next year as the marketplace brings down their cost. These will give each user complete control over his local environment with no contention for resources. A small number of these will be purchased in late 1982 or early 1983 for experimentation.

3.2.2.2. Terminals

The terminals currently used for SPS-1 are character oriented. Graphics devices have not yet been incorporated into SPS-1, although graphics devices will be integrated into SPS-1 by early 1983. We are currently exploring both monochrome bit-mapped as well as color graphics stand-alone terminals. In addition, it seems likely that all personal computers purchased by SPP in the future will offer some graphics support.

3.2.3. Communications

There are two different computers which will be available to each user -- his personal microcomputer and the VAX supermini. Personal microcomputers offload the supermini. We do not yet know what mix of operations will be performed locally and which will be performed on the supermini. This will not be determined until the personal computers are available in number.

If SPP fully incorporates the IDM-500 into its network, the SPS-1 database will be distributed across machines. Text files will be kept in the hierarchical Unix file system on a VAX or personal computer, while relational database information will be kept centralized on the IDM-500. Hence, all relational database queries will require distributed processing.

This partitioning of effort among up to three different machine types has certain hardware implications, the most important one being the requirement for a high capacity bus to support rapid file transfer between them, and accompanying software to support such distributed processing.

3.2.3.1. SPNET

The *Software Productivity Network* (SPNET) currently supports 9600 baud UUCP batch file and mail transfer between Unix/VMS machines using the toolbag available from Lawrence Berkeley Laboratories. It also supports 9600 baud interprocess communication between the Britton-Lee IDM-500 and the Unix VAX. During the next year mature software which supports full interprocess communication and terminal processing protocols between Unix/Unix and Unix/VMS will become available (early versions of Unix/Unix support for IPC/TP protocols are already being marketed) and will appropriately be incorporated into SPNET.

Computer/terminal communications is supported by a local network. Up to 9600 baud communication between any computer on the network and a terminal is currently supported on a coaxial cable by Sytek Corporation's System 20. The Sytek equipment will soon be replaced by *Bus Interface Units*, a product of TRW research, which will support higher performance than that currently offered by Sytek.

Work is currently proceeding to upgrade the computer/computer communications. TRW is also developing a *High Speed Expansion Interface* which will support a 30 megabaud bandwidth. Early versions of these devices are expected this year.

3.2.4. Facilities

The final component of SPS-1 is the office facilities in which the software developer works. After surveying existing facilities in industry and universities, the basic goals for the office facilities evolved. Currently there are 37 prototype offices co-located in the Space Park complex of TRW. Each office houses a single occupant, has a closable door, floor-to-ceiling walls, carpeting, sound-proofing on the wall, and furniture tailored to software developers' usage patterns. Each is connected to the network and has sufficient power, lighting, and air conditioning to support current and planned hardware configurations.

3.3. Development Experience

3.3.1. Rapid Prototyping

As discussed earlier, when developing a software environment, prototyping and evolutionary design is preferable to paper analysis and detailed requirements specifications. Classically, however, the development of realistic-scale prototypes solely for the purpose of better understanding what is really needed has been far too expensive to actually do for systems of any size and sophistication.

With the proper software development environment, however, rapid prototyping becomes feasible. As a demonstration of this fact, many of the major components of SPS-1 which were developed at TRW specifically for this project (i.e., not part of native Unix and not ported from other systems) were developed using rapid prototyping techniques. We did find, however, that prototyping required some revisions to our usual development methodology (Pyster and Boehm, 1982). For example, we found it valuable to develop and iterate a rough requirements spec for the system, but not to follow it rigorously or to put it under configuration control. We also found the need for added-standards, even for the prototype, in the user-interface area.

3.3.2. User-Interface Standards

As mentioned before, special emphasis has been given to the uniformity of the user interface. Since most SPS-1 tools are interactive, we developed a set of user interface standards which include: syntax standards, a help language, interfacing, and documentation. These are constantly evolving as our user community relates their experiences with SPS-1 to us.

3.4. Training

We recognized early in the project that our 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-1. To ensure user satisfaction, we took a four-pronged approach:

3.4.1. Documentation

User manuals are written 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; e.g., we wrote a tutorial introduction to the screen-editor *Vi* more suitable for computer novices than the one distributed from UC Berkeley.

3.4.2. Consulting

A regular consulting service was established so that users from outside SPP can obtain expert help on all aspects of SPS-1.

3.4.3. Courses

Several in-house courses were developed and are offered on a regular basis. Besides an introduction to SPS-1, we offer courses on such diverse topics as word processing, C programming, and advanced system utilities. One has been video-taped, and greater use of video-taping in the future is planned. In addition, commercial software houses now sell CAI courses on Unix. These are being examined for possible use by SPP.

3.4.4. On-Line Help

Facilities are being developed to permit a user to browse through the system and to quickly find a tool he needs. These will be built on the simple but useful utilities such as *whatis* already offered in Unix (You can ask *whatis X* for any system command *X* and Unix will present a one line description of that command.).

3.5. User Acceptance

There was concern when SPP began that the user community outside SPP itself would resist the different way of approaching software development which SPS-1 and its accompanying methodology support. This skepticism was anticipated for several reasons:

Unix is different 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;

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;

Unix is not really "supported" by either Bell Labs or by UC Berkeley in the sense that commercial vendors support their operating systems, causing concern over operating system maintenance.

These concerns motivated us to pay extra attention to ensure that our software worked well, that users were consulted on requirements, that training was adequate, that user manuals were well-written, and that the tools placed into SPS-1 would offer valuable services not easily found elsewhere.

This strategy is paying off. Initial skepticism was indeed encountered, but acceptance of SPS-1 has been steadily increasing. There has been keen interest in obtaining SPP support by several projects based on what they have seen and heard, and SPS-1 has been written into proposals for future development projects. We expect that by the end of 1984 the technology which SPP is pioneering within TRW will have spread throughout much of the company.

4. CONCLUSIONS

The primary conclusions from the Software Productivity requirements analysis are:

1. **Significant productivity gains require an integrated program of initiatives in several areas.**

2. **An integrated software productivity improvement program can have an extremely large payoff (a factor of 4 by 1990).**

3. **Improving software productivity involves 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 requirements are still too incompletely understood to specify precisely.**

The primary conclusions from the SPS-1 development experience to date are:

6. **No single software support system architecture will be optimal for all organizations.** For example, the source-target concept of operation most appropriate to TRW is unnecessary for organizations with a single type of target computer.

7. **The multiple relational-hierarchical database concept simplifies many software support functions.** It allows the support system to capitalize on the strengths of each type of database while largely avoiding their weaknesses.

8. **A rapid-prototyping capability is essential to the evolutionary development of a software support environment.** Unix has provided an excellent rapid-prototyping capability.

9. **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 error processing, help messages, master database access, forms management, etc.

10. **User acceptance of novel development environments is a gradual process which requires careful nurturing by the sponsoring organization.** Involvement of the user community in planning the growth and direction of the environment will help ensure their acceptance of it.

Bibliography

(1) (Alford, 1977). M. W. Alford, "Requirements Engineering Methodology for Real-Time Processing Requirements," *IEEE Trans. Software Engr.*, January 1977, pp. 60-68.

(2) (Bell et al, 1977). T. E. Bell, D. C. Bixler, and M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering," *IEEE Trans. Software Engr.*, January 1977, pp. 49-59.

(3) (Boehm et. al, 1975). B. W. Boehm, "Structured Programming: A Quantitative Assessment Computer," June 1975, pp. 38-54.

(4) (Boehm-Wolverton, 1978). B. W. Boehm, and R. W. Wolverton, "Software Cost Modeling: Some Lessons Learned," *Proceedings, Second Software Life-Cycle Management Workshop*, U.S. Army Computer Systems Command, Atlanta, August 1978. Also in *Journal of Systems and Software*, 1, 3, 1980, pp. 195-201.

(5) (Boehm, 1981). B. W. Boehm, *Software Engineering Economics*, Prentice Hall, Inc., Englewood Cliffs, NJ, 1981.

(6) (Boehm, 1981a). B. W. Boehm, "Improving Software Productivity," *Proceeding, IEEE COMPCOM 1981 Fall, September, 1981.*

(7) (Boehm-Pyster, 1982). B. W. Boehm and A. B. Pyster, "The Impact of Rapid Prototyping on Software Development Standards -- A Position Paper" *ACM SIGSOFT Second Software Engineering Symposium*, Columbia, MD., April 1982.

(8) (Brown et al, 1972). J. R. Brown, "Automated Software Quality Assurance: A Case Study of Three Systems," *TRW Software Series TRW-SS-72-05*, TRW Inc., Redondo Beach, CA 1972

(9) (Business Week, 1980). "Missing Computer Software," *Business Week, September 1, 1980, pp. 46-53.*

(10) (Buxton, 1980). J. Buxton, "Requirements for Ada Programming Support Environments: 'Stoneman'," U.S. Department of Defense, OSD/R&E, Washington, DC, February 1980.

(11) (Cheatham, 1981). T. E. Cheatham, "An Overview of the Harvard Program Development System", in (Hünke, 1981)., pp. 253-266.

(12) (Christensen, 1980). K. Christensen, "Programming Productivity and the Development Process," IBM Santa Teresa Laboratory, TR 03.083, January 1980.

(13) (Caine-Farber-Gordon, 1977). "PDL/74 Program Design Language Reference Guide (Processor Version 3)," Caine Farber Gordon Inc., 1977

(14) (Dolotta et al, 1978). T. A. Dolotta, R. C. Haight, and J. R. Mashey, "The Programms' Workbench," *Bell System Technical Journal*, July-August 1978, pp. 2177-2200.

(15) (Epstein-Hawthorn, 1980). Epstein, R., and P. Hawthorn "Aid in the 80's Datamation," pp. 154-158, February 1980.

(16) (Goldberg, 1978). E. A. Goldberg, "Applying Corporate Software Development Policies," *Proceedings, AIAA Third Software Life-Cycle Management* Conference, 1978."

(17) (Habermann, 1979). A. N. Habermann, "An Overview of the Bandolf Project," *Carnegie-Mellon University Computer Science Research Review* 1978-79, 1979."

(18) (Held-Kreps-Stonebraker-Wong, 1976). G. Held, P. Kreps, M. Stonebraker and E. Wong, "The Design and Implementation of Ingres," *ACM Transactions on Database Systems 1:3*, pp. 189-222, March 1976.

(19) (Hünke, 1981). H. Hünke, ed., *Software Engineering Environments*, North Holland, Amsterdam, 1981.

(20) (Ingrassia, 1978). F. S. Ingrassia, "Combating the 90% Syndrome," *Datamation*, January 1978, pp. 171-176.

(21) (Ivie, 1977). E. L. Ivie, "The Programmer's Workbench: A Machine for Software Development," *Comm. ACM*, October 1977, pp. 746-753.

(22) (Kernighan-Mashey, 1981). B. W. Kernighan and J. R. Mashey, "The Unix Programming Environment," *Computer*, April 1981, pp. 12-24.

(23) (Lanzano, 1970). B. C. Lanzano, "Program Automated Documentation Methods," *TRW Software Series TRW-SS-70-04*, TRW Inc., Redondo Beach, CA 1970

(24) (NFE-Doe, 1980). U.S. National Science Foundation and Department of Education; "Science and Engineering Education in the 1980's and Beyond," October, 1980.

(25) (Rochkind, 1975). M. J. Rochkind, "The Source Code Control System" *IEEE Transactions on Software Engineering*, Volume SE-1,4, December 1975, pp. 364-369.