000021

BEYOND TSO:
PRODUCTIVITY TOOLS COME OF AGE

INPUT LIBRARY

FEBRUARY 1982

C00021

BEYOND TSO: PRODUCTIVITY TOOLS COME OF AGE

CONTENTS

		Page
I	INTRODUCTION	1
II	WHY TSO?	3
	A. Data Processing Productivity: Overview	3
	B. The Rise Of TSO	7
	C. TSO Problems	11
III	BEYOND TSO	13
	A. Early Attempts To Deal With TSO Problems	13
	B. Newer TSO Alternatives	14
	1. Programmer's Workbench (PWB)	15
	2. Maestro	17
	C. Current Choices	19
	D. Future Directions	21
IV	BEYOND THE TSO "MIND-SET"	23

©1982 by INPUT. Reproduction Prohibited.

BEYOND TSO: PRODUCTIVITY TOOLS COME OF AGE

EXHIBITS

			Page
II	-1	Hardware Cost/Performance	4
	-2	Hardware Versus Software Improvements, Cost/ Performance	6
	-3	Selected Interactive Development Systems	10
III	-1	Modes Of Programming Development	20
IV	-1	Cumulative Costs Over System Life	24

©1982 by INPUT. Reproduction Prohibited.

INPUT

C0C021

I INTRODUCTION

I INTRODUCTION

- For three years, INPUT has been tracking current efforts in the area of software productivity. Previous INPUT reports on the subject include:

  - Performance Improvement: User Techniques and Experiences (February 1979).

  - Software Directions: Languages, Development Aids, DBMS and DDP (July 1979).

  - Managing the System Development Process (December 1980).

- In addition, a major multiclient study was delivered in 1981, entitled Improving the Productivity of Systems and Software Implementation.

- The consistent refrain from managers of system development has been that interactive program development, using IBM's Time Sharing Option (TSO) or the equivalent, is the most visible productivity improvement tool in use today. However, this is not a new tool by any means, having been introduced to many of these organizations in the early 1970s.

- There is little question that the "bottleneck" of systems development can be relieved, but not eliminated, by turning an increasing percentage of routine development and enhancement effort back to the end users to accomplish directly via "user-friendly" tools.

©1982 by INPUT. Reproduction Prohibited.

- This report focuses on the applications which, because of their complexity, performance requirements, and/or criticality (to multiple units within the organization), will remain the responsibility of the information systems group.

©1982 by INPUT. Reproduction Prohibited.

II WHY TSO?

II WHY TSO?

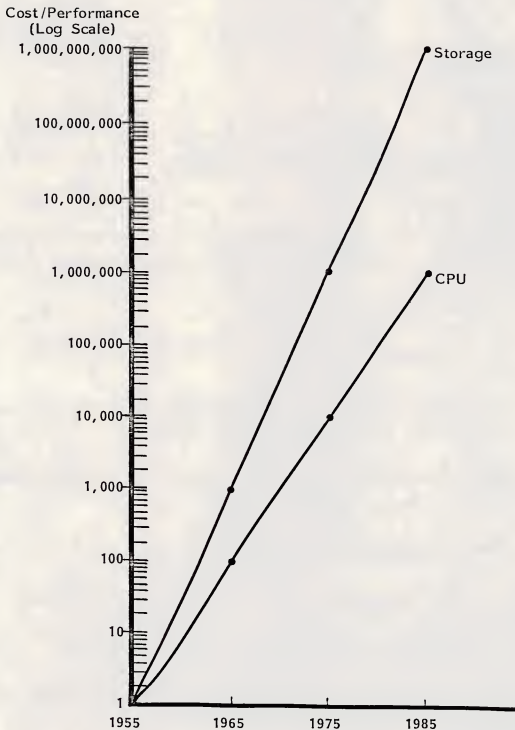
A. DATA PROCESSING PRODUCTIVITY: OVERVIEW

- It is a truism of data processing that advances in hardware have occurred at a steady rate which promises to continue for the foreseeable future.
  - Capabilities increase dramatically.
  - Reliability continues to improve.
  - Equally important, the cost/performance curve shows little sign of faltering, as shown in Exhibit II-1.
- Without these hardware improvements, computers would be laboratory curiosities with few practical applications.
- Software is a somewhat different story.
  - Capabilities (e.g., languages, utilities, DBMS, etc.) have increased.
  - However, reliability, while perhaps improving, is variable.
  - In the critical area of cost/performance, software development and maintenance are in the doldrums.

©1982 by INPUT. Reproduction Prohibited.

EXHIBIT II-1

HARDWARE COST/PERFORMANCE



©1982 by INPUT. Reproduction Prohibited.

- There have been positive advances in programming, as far as languages are concerned.

  - Without the progression from machine language to assembly language to today's "high-level" languages (COBOL, FORTRAN, etc.), it is doubtful whether the great expansion in the use of data processing could have taken place.

  - But Exhibit II-2 reveals the relatively modest software improvement compared to hardware improvement.

  - Many of the efficiency gains in switching to high-level languages were immediately lost by the concomitant rise of ever more complex operating systems (needed to act as an interface between application programs and the machine).

    - No single invention has caused as much programmer suffering as IBM's operating systems, and especially the related job-control language (JCL).

    - While operating systems have become more stable in recent times, and the applications programmer is more shielded from their grim realities, "user-hostility" of the IBM OS (DOS, MVS, etc.) has not changed much in 15 years.

  - The remaining gains in language efficiency have largely also been lost to the increased complexity of the overall programming environment.

    - The programmer must now routinely contend with the challenges of teleprocessing and data base management systems.

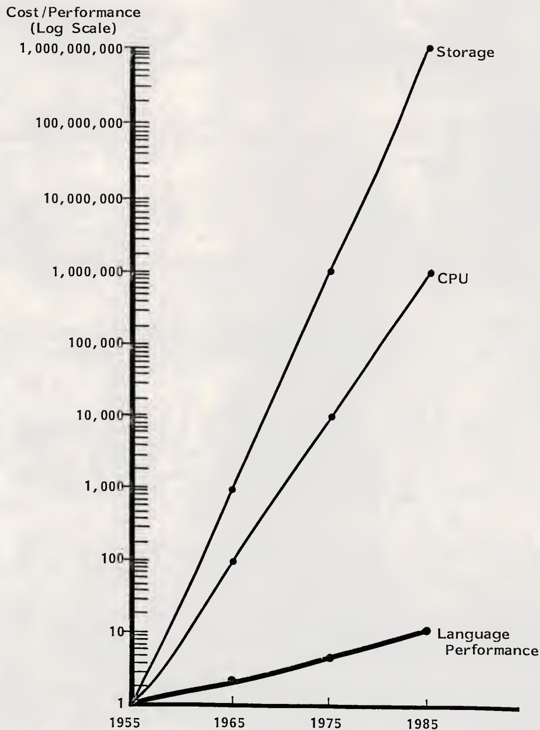
    - The "layering in" of distributed data processing adds significantly to the problems of programming.

©1982 by INPUT. Reproduction Prohibited.

INPUT

EXHIBIT II-2

HARDWARE VERSUS SOFTWARE IMPROVEMENTS, COST/PERFORMANCE



©1982 by INPUT. Reproduction Prohibited.

B. THE RISE OF TSO

- Many of the efforts to improve the programming activity have focused, quite justifiably, on the mechanics of programming; i.e., the process of coding and debugging.

- Until less than ten years ago, the mechanics of programming were tortuously slow, frustrating, and hardly categorized as productive.

  - Coding sheets, keypunched cards, and batch operating procedures combined to keep the execution of ideas far behind the conception.

    - Getting three turns at the computer each day was "programmer heaven." One turn a day was considered good at many installations.

  - Remote job entry (RJE) card readers and keypunch machines for the use of programmers were considered significant productivity tools.

- In this environment, the introduction of IBM's TSO in the early 1970s was a giant step forward.

- TSO is essentially a powerful text editor that eases communications with the IBM mainframe, operating system, and associated utilities.

  - Code can be entered and stored from a CRT.

  - The TSO command language provides immediate access to data (including source programs, job control, etc.) stored in the TSO libraries, or other data sets catalogued on the system.

    - Program code and other accessed data sets can be modified in an on-line environment.

©1982 by INPUT. Reproduction Prohibited.

- Narrative text (for documentation or other purposes) can be stored, manipulated, and retrieved.

  - The TSO software permits RJE submission of the completed program for compilation, testing, etc.

    - To the user, this mode often appears to be on-line compilation and testing, although it really is not.

    - Some versions of TSO and similar productivity tools do permit interactive compilation and debugging.

  - However, TSO has inherent limitations caused by the batch orientation of IBM architecture. At the least, this causes a significant amount of resources to be consumed.

    - This is in contrast to most minicomputers, which are designed to be interactive machines.

- TSO was developed for the MVS (OS/VS2) operating system. For virtual machine (VM) operating systems, the Conversational Monitor System (CMS) provides similar capabilities, including:

  - Immediate access to programs and data.

  - Advanced text-editing features.

  - Ability to run programs interactively for testing and debugging.

- Independent software vendors have developed functionally similar software packages running under their own or IBM's telecommunications monitors.

- Other mainframe vendors offer development aids which provide many of the same functions as TSO.

©1982 by INPUT. Reproduction Prohibited.

- Exhibit II-3 lists some of the packages similar in purpose to TSO. ("TSO" is used collectively to describe TSO and its "look-alikes").

- From one standpoint, TSO has been a smashing success. It is no exaggeration to say that any moderate to large installation that is not using TSO is probably planning or wishing to use it.

  - Many larger installations have hundreds of TSO terminals.

  - Frequently, the goal is to have a separate terminal for each programmer (the typical ratio now is three to four programmers sharing a terminal).

- Productivity, measured in lines of code (LOC) per programmer day, has increased, according to the fragmentary and noncomparable data available.

  - While not a comprehensive measure of productivity, LOC may be used by managers who feel the need to track something other than milestones as programmer output.

- Eventually, however, most installations with large TSO networks have experienced performance difficulties caused by insufficient CPU or memory capacity.

  - TSO jobs themselves are delayed.

    - Even when partially disabled, TSO turnaround is far superior to the older batch/RJE; but it then no longer meets heightened expectations of its users (who are programmers, for the most part).

  - Production jobs may also experience worsened response time.

©1982 by INPUT. Reproduction Prohibited.

EXHIBIT II-3

SELECTED INTERACTIVE DEVELOPMENT SYSTEMS

NAME	VENDOR
ROSCOE and VOLLIE	Applied Data Research
WYLBUR	On-Line Business Systems, Inc.
TONE	Tone Software Corp.
DOS/VS Entry Time Sharing System (ETSS)	IBM
O-W-L (On-Line Without Limits)	National Computing Industries
OTS-On-Line Terminal System for Program Development	JBL Systems
ICCF-Interactive Computing and Control Facility	IBM
ACEP	Software Module Marketing

©1982 by INPUT. Reproduction Prohibited.

C. TSO PROBLEMS

- Why has TSO not been the panacea it was expected to be? The main reason is that TSO simply consumes a great deal of resources, for the following reasons:

  - Text editing. Pre-TSO, a lot of text editing was done off-line (keypunching, replacing cards in decks). The trade-off from time-consuming manual methods to electronics is good. However, using 3033 CPU cycles for primitive word processing is very expensive, especially if done during prime business hours.

  - Increased prime time use. Batch-era compilation and testing gave computer operations groups more direct control over the scheduling of programmer jobs. More importantly, programmers acquiesced in this and expected that much of their work would be performed during off-hours.

    - During the initial installation of TSO there are usually sufficient computer resources for very impressive turnaround performance (otherwise the TSO installation would - or should - be delayed). This quick turnaround gives rise to new programmer expectations and behavior that can rarely be changed.

  - Software overhead. IBM-developed software products tend to require significant amounts of hardware resources in relationship to work done. TSO is not an exception.

  - Increased productivity. If a programmer can get ten TSO turns at the computer a day versus three using RJE, this is called increased productivity - and in some cases, actually is.

  - Increased sloppiness. It is reasonable, though, to inquire as to the character of the ten turns mentioned above. Are five of these turns

©1982 by INPUT. Reproduction Prohibited.

devoted to correcting errors (including errors in keystroking, syntax, and logic)? No one knows, but there is a widespread belief that TSO has further encouraged the already pronounced characteristic of many programmers to code rather than to think.

- In most installations this performance squeeze is aggravated by a capacity planning process that is not able to:

  - Forecast hardware requirements sufficiently ahead of the need.

  - Justify increased hardware resources for no obvious compensating reduction in system development time.

- Capacity planning for TSO-related needs may sometimes be more difficult than for production work. Some types of software development place sudden burdens on the system.

  - However, in principle, development-related needs should be as predictable as any other resource demand.

  - These issues will be discussed at length in the INPUT Report, Performance Measurement and Capacity Planning (September, 1982).

- Switching from TSO to CMS, or some other more efficient program development aid, offers little more than to delay the inevitable shortage of resources.

  - Of course, a strong case can be made for starting out with a more efficient alternative, if the commitment to TSO has not already been made, or if the organization is willing to acquire or develop the "bridge" software (for example, from VM to VS files).

©1982 by INPUT. Reproduction Prohibited.

III BEYOND TSO

III BEYOND TSO

A. EARLY ATTEMPTS TO DEAL WITH TSO PROBLEMS

- Increasingly, installations which can afford it are attempting to deal with these capacity problems by giving programmers the sole use of a complete system (e.g., perhaps an "outgrown" 370/158 now that production is run on a 303X machine).

  - This isolates the conflicting and somewhat unpredictable programmer demands from production demands for machine time.

  - A side benefit is that it becomes easier to ensure that development, test, and production versions of the same software remain separated.

- Many installations, however, do not have an "outgrown" 370 lying around to give to programmers. Other installations are looking for development tools that are easier than TSO to use and that support efficient programming approaches.

©1982 by INPUT. Reproduction Prohibited.

B. NEWER TSO ALTERNATIVES

- An innovative approach to on-line program development has surfaced in two commercial products, the Programmer's Workbench (PWB) and Maestro, based on the concept that program source coding is largely a text-editing function and is radically different from the execution of object code.

  - Pursuing this concept to its logical conclusion, there is no compelling reason why on-line source program development has to be done on the same system on which the related object program will execute: historically, it just happened that way because additional memory was less expensive than redundant CPUs.

  - Accordingly, Maestro and, to a lesser degree, PWB have effected a physical separation of the on-line program development function from the "target" system on which the program being developed will execute.

  - Both PWB and Maestro are implemented on small computers - Digital Equipment PDP-11s and Four-Phase systems, respectively - and besides the demonstrably lower hardware costs involved, both systems were intended from their inception to be on-line systems rather than timesharing adjuncts to existing batch operating systems.

  - Both systems provide extensive source library maintenance facilities and macro-like capabilities that expand programmer-defined "short-hand" coding into full source-language constructs acceptable to the target compilers. Additional features unique to each system are described below.

- Up to this point, aside from off-loading the mainframe, there is little difference between PWB or Maestro and the mainframe-based text editors (TSO, CMS, ROSCOE, etc.).

©1982 by INPUT. Reproduction Prohibited.

- Nevertheless, several customers of the mini-based products justified their procurement solely on these cost and availability trade-offs.

- Others have utilized satellite 4300s to perform development functions, using the same kind of justification arguments.

I. PROGRAMMER'S WORKBENCH (PWB)

- PWB runs under the proprietary UNIX operating system developed by Bell Laboratories. PWB relies heavily on the unique characteristics of UNIX, and is therefore logically inseparable from UNIX.

- PWB was originally developed for use by engineering departments, hence its support of FORTRAN and C compiler languages. (UNIX itself is written in C.)

- Guided by the perception that the key to encouraging productive use of the system was the provision of tools with which users are familiar (and which they believe are useful), the developers of PWB/UNIX incorporated an extensive set of primitive functions into UNIX that enables each user to build his own program development tools.

  - Thus, each user can devise his or her own "shorthand" for writing source statements in abbreviated form, under control of user-devised tools. PWB/UNIX then expands abbreviations into syntactically correct compiler language statements.

  - A user can therefore start with the easy-to-use command language system (called the "Shell"), and as the user becomes comfortable with the PWB/UNIX environment, can add new program development tools for either personal or systemwide use. This also helps to spread the capital investment needed to tailor a program production system that meets a specific organization's requirements.

©1982 by INPUT. Reproduction Prohibited.

INPUT

- In addition to this macro-like facility, a powerful text processing system enables the system's users to update narrative program documentation conjointly with source statement revisions. In fact, many users are familiar only with the powerful text processing features.

- The text processing feature interfaces directly with a typesetting system of great flexibility.

- The Source Code Control System (SCCS) is a PWB/UNIX feature that automatically maintains each generation of source code and documentation by date or version number. Thus, reconstruction of any version of a program or document is available on request.

• Since PWB was originally developed in scientific and engineering environments, it supports the C language (and the "Shell") better than other languages.

  - The lack of the same level of support for COBOL would be a hindrance in the typical business environment, although some users have employed it successfully for COBOL maintenance tasks.

• Another problem is in the level of support for target machines. Target hardware includes (in decreasing order of ease of transferring compiled programs to such hardware environments):

  - The UNIX system itself (e.g., the DEC hardware on which development is undertaken).

  - Other UNIX-based mini- and microcomputers.

  - Mainframes which accept the UNIX system underneath the mainframe's operating system (e.g., the Amdahl mainframes).

©1982 by INPUT. Reproduction Prohibited.

- Perhaps the biggest drawback to the PWB is the wide variety of choices and adaptability open to the user.

  - New users may not know how to proceed.

  - Early versions of UNIX needed expert programmers. Later versions are more user-friendly, but still require a very knowledgeable person to make the system perform.

- PWB/UNIX is licensed by Western Electric, AT&T's manufacturing subsidiary, on an "as-is" basis, with no support and minimal documentation.

  - Several firms, such as Interactive Systems Corp. of Los Angeles, California, are licensed by Western Electric to resell PWB/UNIX, and such resellers typically add features, documentation, and support.

  - PWB-UNIX users cite poor documentation (even from resellers) as a major shortcoming of the system. Other shortcomings are the lack of on-line prompting and the absence of support from Western Electric.

    - This problem is partially inherent in its flexible design: it is not possible to document all the things that <u>might</u> be done with a very flexible tool.

2. MAESTRO

- Maestro was developed by Softlab GmbH of Munich, West Germany, and is marketed in the United States by Softlab Systems, of San Francisco, California. Over 100 Maestro systems have been installed in Europe, and about 40 in the U.S.

- Maestro's program preparation tools include:

©1982 by INPUT. Reproduction Prohibited.

INPUT

- Interactive procedures that assist in the production on-line of Nassi-Schneiderman design charts ("Structograms," in Maestro's product literature) by systems analysts and end users through on-line prompting.

- Support of HIPO charts and decision tables.

- Syntax prompting and limited editing for COBOL, FORTRAN, and PL/I, reflecting Maestro's business data processing orientation.

- Like PWB/UNIX, Maestro offers extensive source library maintenance facilities and shorthand coding capabilities.

  - Skeletal programs, programmer notes, system documentation, test results, work in progress, previous versions, etc., can all be tied together via programmer-defined "electronic bookmarks" to permit immediate screen reference back and forth among up to 12 separate files of information.

  - Support and extensive documentation are available from the vendor.

- Unlike PWB, Maestro does <u>not</u> produce compiled programs (or permit direct testing). It will produce code which can be transmitted to the host for compilation and later testing.

  - Obviously, the designers of Maestro realized the problem of producing compiled code for different computers and decided to avoid the problem entirely.

  - This has an advantage in that, unlike users of the PWB, Maestro users do not have to concern themselves over target machine questions.

  - However, it also means that Maestro users have not cut themselves off from <u>all</u> the problems of having to depend on the mainframe for development: compiling and testing must still be done on the host.

©1982 by INPUT. Reproduction Prohibited.

C. CURRENT CHOICES

- PWB/UNIX and Maestro are indicative of the positive efforts currently underway in the area of computer-aided design and computer-aided programming, and clearly suggest the trends which will be influencing the systems development cycle in the future. However, each has defects under certain circumstances, as shown in Exhibit III-1.

- PWB/UNIX is a very powerful tool, but its power makes it most suitable for the professional programmer or engineer.

  - In addition, its orientation is still primarily toward the scientific programmer in terms of both:

    - Primary language.

    - Target machine.

- Maestro's drawbacks are almost the mirror image of PWB/UNIX:

  - It is still tied to the host machine; consequently its throughput during compilation and testing will be affected by host capacity "crunches".

  - Its ready-made promptings and guidance are good for the journeyman programmer, but much less supportive for the expert programmers (which all installations should be striving to attract and keep).

- Within these constraints, either of these systems can be a desirable alternative to TSO.

    - However, installations with an obsolescent (i.e., cheap) 370 and an adequate TSO-type system (to serve as a standalone development machine) may have an attractive interim alternative.

©1982 by INPUT. Reproduction Prohibited.

EXHIBIT III-1

MODES OF PROGRAMMING DEVELOPMENT

CHARACTERISTICS	BATCH	RJE	TSO-TYPE SYSTEM SHARED WITH PRODUCTION	TSO-TYPE SYSTEM DEDICATED TO DEVELOPMENT	MAESTRO	PWB/UNIX
• Fast Response Time?	No	No*	Varies**	Yes	Yes	Yes
• Easy to Use?	No	No	No	No	Yes	Not Initially
• Editing Separate from Compilation?	No	No	Varies***	Varies***	Yes	No
• Prompted Coding?	No	No	Varies***	Varies***	Yes	Possible
• Compiled on Development System?	Yes	Yes	Yes	Yes	No	Yes
• Testing on Development System?	Yes	Yes	Yes	Yes	No	Usually
• Integrated Design Tools?	No	No	Some	Some	Many	Some
• "Documentation-Friendly"?	No	No	Somewhat	Somewhat	Yes	Somewhat
• Integral Libraries, Reusable Modules?	No	No	Varies***	Varies***	Limited	Yes
• Cost						
– Start Up	Low	Low/Med	Medium	High	Med/High	Medium
– Mainframe Overhead	Medium	Medium	High	High	Low/Med	Low/Med
• Flexibility	Very Low	Low	Medium	Medium	Medium	High

* BUT BETTER THAN BATCH
** DEPENDS ON WORKLOAD
*** DEPENDS ON PACKAGE AND IMPLEMENTATION

©1982 by INPUT. Reproduction Prohibited.

D. FUTURE DIRECTIONS

- The drawbacks to current tools may be alleviated by tools that are currently in the design and experimental stage.

  - One possibility is to use powerful microprocessors as the basis for small shared systems.

  - Another even more exciting possibility is the "personal development machine" that would function as a self-contained unit for each programmer.

- The latter is beginning to be discussed under the term "software engineering environment."

- In either case, the software implementation would require several key attributes:

  - The environment would have to be highly structured for the great majority of programmers.

    - Most programmers could be trained to view this as being user-friendly, in the sense of being helpful and supportive.

    - From management's standpoint this would be a useful way of enforcing rules and encouraging sound programming practice.

  - On the other hand, the superior programmer needs to be able to create a unique individual environment, such as with PWB/UNIX.

    - There will have to be a (conceptual) "lock" on the machine which allows only the elite to use these features.

©1982 by INPUT. Reproduction Prohibited.

- The machine should have its own compilers to allow code to be generated for a variety of target machines.

- The individual personal development machines should have the capability of linking together to share files, program modules, etc. so that the development network can keep itself isolated from hosts.

©1982 by INPUT. Reproduction Prohibited.

IV BEYOND THE TSO "MIND-SET"

IV BEYOND THE TSO "MIND-SET"

- Even the best replacement for TSO will still be focusing on mechanics; i.e., how to generate code.

  - Code generation and debugging consume less than one-fifth of the classic system development cycle's costs, and less than 10% of total life cycle costs, as shown in Exhibit IV-1. Therefore, the scope for true (as opposed to apparent) cost savings is low.

  - In fact, by making coding progressively easier, there may be continually less thought given to basic design issues.

- If coding were where analysis and design are now, it would still be in the darkest batch days (undoubtedly using machine language).

  - Most installations would do far better to devote themselves to finding better ways to analyze problems and translate these results into program design.

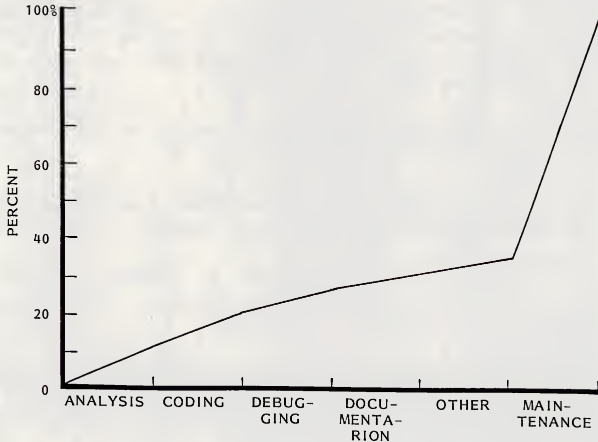
    - This is what will produce long-lasting reliability and cost savings.

- Another approach, even further away from the coding tool concept, is to try to do away with coding itself as much as possible.

©1982 by INPUT. Reproduction Prohibited.

EXHIBIT IV-1

CUMULATIVE COSTS OVER SYSTEM LIFE



©1982 by INPUT. Reproduction Prohibited.

- This would mean following the "bread boarding" approach by means of the so-called fourth generation languages (FOCUS, RAMIS, etc.).

- These concepts are still largely experimental. Translating from the prototype to the production environment requires skill, and a considerable amount of programmer intervention.

- However, the concept holds great promise from the standpoint of speed and accuracy in meeting user needs.

  . Questions concerning efficiency are reminiscent of the age-old COBOL-Assembler debate, considering that over half of the programs in existence today cost more to develop than to operate for their lifetime.

• The "software engineering environment" approach which is an integral part of the Ada language, plus the experimental work being done by the Japanese and by a few specialized vendors in the U.S., will deliver by mid-decade a substantial advance to the technique of software development.

• It is not particularly difficult to automate portions of the development process, and even the entire process if batch-oriented and not too complex.

  - Screen-generators, file-management systems, and skeletal programs (in the form of macroprocessors) have been in existence for some time.

  - Rigorous requirements languages permit computers to assist in finding conflicts or "holes" in systems definitions.

  - Specialized test analyzers, while expensive to run, have proven their worth in applications of high criticality.

• Many microcomputers boast of existing, or soon-to-be released, application generators for routine kinds of processing applications.

©1982 by INPUT. Reproduction Prohibited.

- But the truth is that the more complex kinds of real-time applications (the integration of existing applications or the state-of-the-art, first-time-ever applications) are still beyond the scope of these powerful tools.

- An even more serious drawback is that many organizations have not even made the managerial commitment to choose the strategic solution over the tactical alternative, and thus continue to add to their problems rather than solve them.

  - For these organizations, even the software engineering environment will not be a solution.

- The five-year plan for improved software productivity must include:

  - Initial research and experimentation with "foundation" tools, including a study of the changes these tools cause (and permit) in basic thinking about systems.

  - Greater understanding of data base requirements, including cost/benefit analysis of the appropriate media and level of detail for data storage. Not everything can - or needs to be - on-line!

  - Preliminary efforts to integrate "tool fragments" for specific, rather than general, purposes.

  - Development of a laboratory or testbed for reconfiguring system development tools under controlled conditions.

©1982 by INPUT. Reproduction Prohibited.