SOFTWARE PRODUCTIVITY TOOLS

A SURVEY FOR TOSHIBA

INPUT

SOFTWARE PRODUCTIVITY TOOLS

A SURVEY FOR TOSHIBA

A Proprietary Study

Prepared by

INPUT

AUTHOR
TIM TYLER

TITLE SOFTWARE PRODUCTIVITY TOOLS: A SURVEY FOR TOSHIBA

DATE LOANED	BORROWER'S NAME

Software Development Tools

I. Introduction

- The scope of this study probably deserves more resource and time to gain in-depth insight. However, INPUTs extensive research in productivity improvement in systems and software implementation will permit us to place the specific research associated with this study into a broader framework which will make it more meaningful.

- One observation of importance should be made at the beginning: it is INPUTs opinion that the tools required for the development of business systems, and those for the development of embedded and/or process control systems, are becoming indistinguishable as computer/communications networks permit (and often force) levels of integration where the flow of business data and information becomes an essential process to the enterprise. It is also true that productivity tools must assist in the development of applications systems which will be spread over the processing hierarchy (mainframes, minicomputers, and microrocessors) and be dynamic in the allocation of these resources. The fact that adequate tools do not currently exist to address these problems is primarily the result of inadequate problem definition rather than technical in nature.

- The research approach INPUT has taken for this study is to rely heavily upon past detailed research, and use telephone interviews with recognized experts to update the current status of both specific case studies involving the use of productivity tools and aids, and to refine general patterns of acceptance and usage of tools and aids, and to verify our analysis of the software development productivity problem.

- The major productivity research studies which have been conducted by INPUT will be referenced in this report. They are as follows:

- The seminal productivity study was sponsored on a multiclient basis by some of the largest corporations in America and consisted of the following:

  . In-depth, on-site interviews were conducted at multiple staff levels (programmer/analyst through MIS Director) in 60 firms to determine their approaches to the improvement of software development and maintenance.

  . An additional 32 organizations which had enmployed specific tools, approaches and methodologies were interviewed over the telephone to complement the on-site interviews.

  . Approximately 1,300 mail surveys were conducted to provide a statistical base for eveluating the results of the in-depth interviews.

  . Extensive desk research was conducted on published information.

  . When warranted, authors and experts who have made specific contributions to productivity improvement were interviewed either in person or on the telephone.

  . The results of this study was presented to clients in a series of seminars, and a 400 page report of findings was published by INPUT in 1980. (Improving the Productivity of Systems and Software Implementation)

- In 1982, INPUT conducted a carefully selective updating and extension of Improving the Productivity of Systems and Software

Implementation for the Computer Development Laboratories (CDL). This study concentrated on the following:

. On-site interviews with knowledgable experts in seven broad industry segments.

. On-site interviews with five vendors advocating the following approaches to productivity improvement: 4GL/DBMS, applications packages, information management systems(data management, statistical analysis, and information(graphics) display systems), custom systems consulting and implementation services, and data base machines.

. In addition, special attention was given to the work being done in Software Engineering Economics (Barry W.Boehm), Information Engineering (James Martin and Clive Finklestein), and Queuing Networks (Dr. Ralph L. Disney).

. The results of this study was presented to CDL in a 200 page report - Software Development Productivity; INPUT, December 1982.

- In 1984, as part of its continuing subscription programs for computer users and vendors INPUT published two companion reports on the use of productivity tools, techniques and methodologies.

. New Opportunities for Software Productivity Improvement was addressed to users and focused on the the emerging systems development environment for business systems and potential problems associated with that environment.

. Market Impact of New Software Productivity Techniques contained specific recommendations for vendors for new productivity tools required to address the anticipated problems of the emerging systems development environment.

- In early 1985, INPUT had an opportunity to address the primary issue which was isolated in the 1984 reports - quality assurance. This study was conducted for JIPDEC and included specific case studies studies of what was being done in selected organizations. Both on site and telephone interviews were conducted and the title of the report was Software Quality Assurance.

- Also in 1985, INPUT reviewed the current status of knowledge-based systems (expert systems), as a potential tool for productivity improvement. The resulting report - Artificial Intelligence and Expert Systems was published as part of INPUTs subscription program for computer users.

o Analysis of individual software development tools has not been attempted except to classify them based upon their stated purpose and how they have been employed. INPUT urges caution in accepting published information concerning productivity improvement based on either vendor claims or as the result of case studies which have not been thoroughly researched. There are two reasons for this: 1) valid metrics are seldom employed to support such claims, and 2) the classic "Hawthorne effect" can be observed when new tools are introduced into an organization on a trial basis.

- No attempt to summarize the information contained in this study will be made since the entire report can be considered a summary of extensive past research. It is strongly recommended that a more targeted research project be undertaken if the purpose of this study is to be used for either applications development tool selection or marketing.

II. Review of Past INPUT Research

- The primary conclusions reached in Improving the Productivity of Systems and Software Implementation (INPUT, 1980) were as follows:

  - Then current "solutions" did not solve productivity problems. The problems were manifested by the following:

    . Systems were becoming more complex and expensive, and time and cost overruns were becoming even more of a problem than they had once been.

    . Software maintenance was consuming ever increasing amounts of the software dollar, and users were dissatisfied with quality and reliability.

    . Backlogs continued to increase while software related expenses were becoming an increasing portion of total systems budgets.

  - Managements perception of the problem was limited to a narrow focus on schedule overruns and the increasing backlog. Increasing costs and cost overruns were considered "normal" (or secondary issues), and software quality was rarely cited as a contributing problem. The result was undue emphasis upon tools (both hardware and software) to improve programmer productivity, and failure to develop a consistant strategy for productive systems development.

  - The major INPUT conclusion was that an overall strategy was required and that, unless certain priorities were established, environmental changes and "investing" in improved tools and would not be effective. This conclusion lead to the "Productivity Pyramid" which

has been published in numerous INPUT reports. Exhibit II.1. Essentially, this relatively simple diagram emphasizes the need to build an effective strategy from the bottom up, and it warrants repetition in this study.

- Commitment to quality software may sound simplistic, but there has been a tendency to concentrate on the latest hardware technology and assume that necessary software can always be improved later - the result has been a lot of "quick and dirty software" at all levels in the processing hierarchy.

- User involvement throughout the development process is absolutely essential if a quality system is to be implemented.

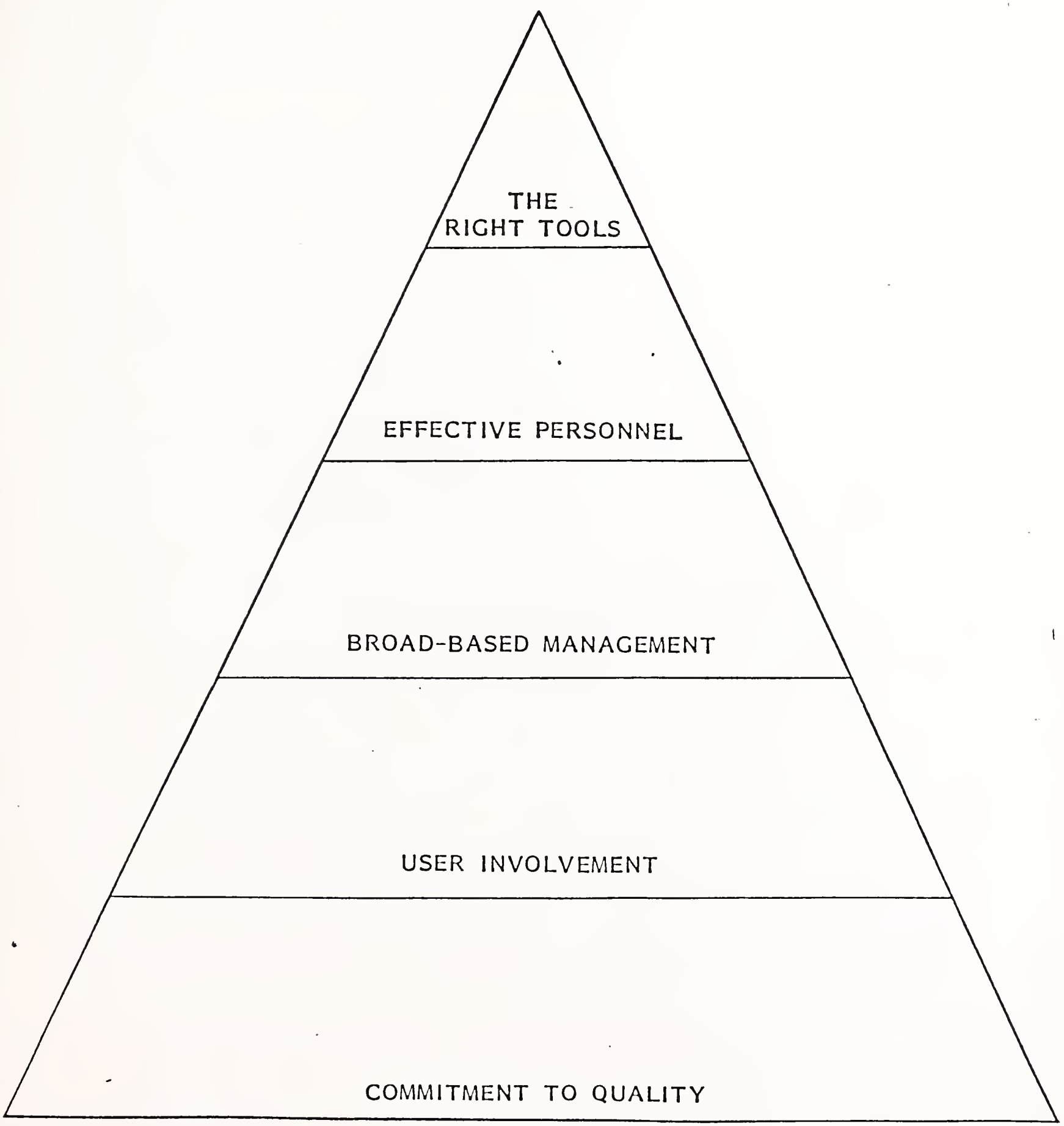
- Broadbased management stresses the necessity for all levels of management (for example, corporate executives and engineering management) to recognize the importance of quality software development over the systems life cycle, and to remain involved in the establishment of priorities and the allocation of resources.

- Effective personnel can be defined, selected and retained only with full understanding of systems complexity and associated priorties.

- Only after the first four levels of the productivity pyramid are assured can the right tools be selected; and it must be understood that there is no one right set of tools for all projects and all personnel.

EXHIBIT II-1

THE PRODUCTIVITY PYRAMID



- While the importance of the right tools was placed into proper perspective in the study, this should not be regarded as a lack of concern about the availability of appropriate tools and INPUT emphasized that available tools usually addressed only portions of the systems life cycle. Exhibit II.2.

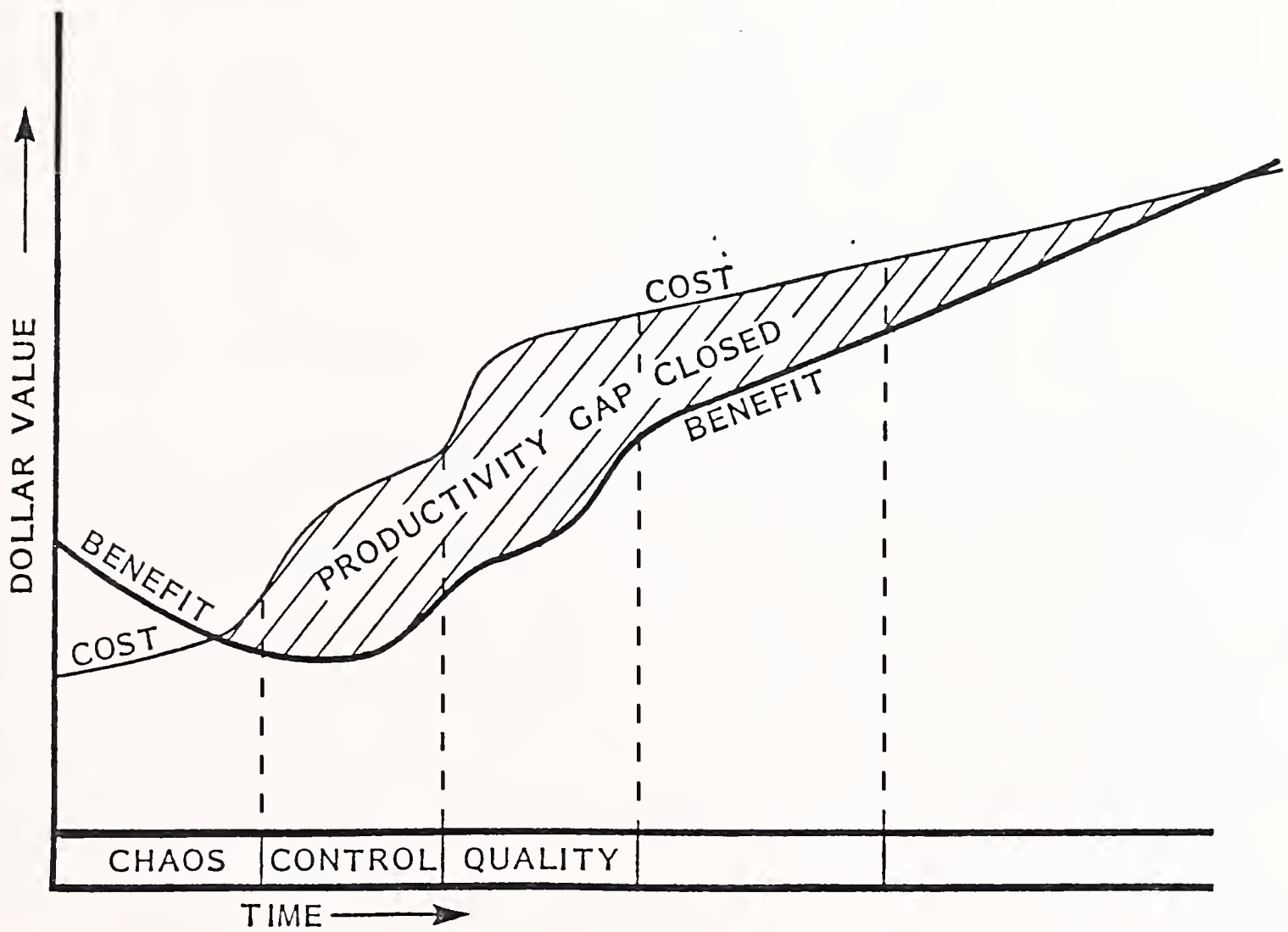
o In view of the general statement of requirements for the current study, it should be pointed out that sponsors of the multiclient study included several aerospace companies where software development for embedded, on-line, real-time computer systems have long been of primary concern; and where the most comprehensive work in attempting to improve software quality and productivity have occurred. An aerospace company which represented one of the detailed interview sites for the 1980 research(not a sponsor of the study), will be used as a case study later in this report.

o The 1982 custom study Software Development Productivity identified major trend towards getting end users involved in the systems development process, and speculated that this was the direct result of the proliferation of personal computers in the corporate environment. Therefore, information systems managers were using this involvement because it was expedient (as opposed to any commitment to quality), and INPUT foresaw certain problems which might attend this approach to productivity improvement. Specifically, it was stated that:

- User-developed systems, including prototypes, will remain installed and require maintenance and integration with other systems.

EXHIBIT II-2

STAGES OF PRODUCTIVITY IMPROVEMENT:
COST/BENEFIT RELATIONSHIPS, STAGE 2



——— BENEFIT LINE, WHERE CONTROL AND QUALITY HAVE BEEN INSTITUTED.

——— COST LINE, WHERE CONTROL AND QUALITY HAVE BEEN INSTITUTED.

- Data and information bases will not be manageable and will proliferate beyond control.

- Data processing personnel will become so involved with the development of end user directed systems that they will never be able to consider quality, much less make a commitment to it. Getting something done quickly, regardless of quality, will be the order of the day.

- The expense of systems development will increase substantially without really improving productivity.

o In addition, the 1982 study pointed to three areas which seemed promising to the overall productivity problem:

- The work of Barry W. Boehm as presented Software Engineering Economics was mentioned as holding promise for solving the continuing problems of software productivity measurement and metrics - provided this work was used intelligently. (It had been earlier determined that lack of accepted metrics was a significant impediment to progress in productivity improvement.)

- Work in Information Engineering as espoused by James Martin and Clive Finklestein was also mentioned as exhibiting some promise in the critical area of data and information quality. (It should be pointed out that information quality is an essential ingredient of productivity improvement whether one is considering the quality and precision of internal systems interfaces or the communications among/between project teams and/or individual analysts or programmers.)

- Work on Queuing Networks as it has evolved in operations research was deemed to be of particular importance to not only the problems of internal performance of complex systems but also to the problems which INPUT felt were inherent in project team communications; and an on-site interview with Dr. Ralph L. Disney of Virginia Polytechnic Institute was included as part of the study.

- All three of these major subjects have been briefly updated as part of this study.

o The two productivity studies conducted by INPUT in 1984 (New Opportunities for Software Productivity Improvement and Market Impact of New Software Productivity Techniques) confirmed some of the anticipated problems identified in the 1982 study. Essentially, systems were being prototyped using applications development tools (primarily 4GLs and DBMSs), and applications developed in information centers and on personal computers were being evolved into production systems (or becoming integrated with established systems). INPUT refers to this environment as Distributed Systems Development (DSD), and the problems specifically associated with this environment have been clearly identified:

- The synchronization and integrity of data bases in the DSD environment is a technical problem which does not have an accepted solution.

- Security and protection of distributed data bases adds a level of complexity to a problem which does not currently have accepted, economical solution.

- DSD leads to conflicting reports to management - frequently without qualification as to the source.

- The integration of systems (applications) developed in the DSD environment lead to excessive and unanticipated demands being made upon host mainframes.

- In addition, distributed systems development can actually be counterproductive even while demonstrating surface productivity improvement by getting systems developed rapidly and/or on schedule. This counterproductivity is demonstrated in the following ways:

  . Deterioration of data/information quality.

  . Unanticipated expense in terms of operational inefficienies, maintenance required, etc. which results in substantially higher life cycle costs.

  . Unworkable solutions (systems which can never go into production because of poor quality or excess operational expense).

  . Systems which continue to evolve in endless cycles of superficial requirements definition, systems analysis, and quick and dirty implementations designed to show quick "results" without ever really solving the problem.

o Among the tools and aids recommended by INPUT to assure quality in the DSD environment are the following:

- Expanded dictionaries, directories, encyclopedias, and glossaries of data and information are required to assure the quality of communication among systems developers and users.

- A meta language(s) is necessary to describe the internal interfaces between and among hardware/software/human components of both the development and target computer/communications networks. (In the study report, it was suggested that APL might serve this purpose, and subsequent research tends to confirm the potential of this language.)

- Processing and data-flow monitors for predicting and controlling network performance are required, and it remains INPUTs opinion that queuing networks provide a possible foundation for such monitors. (While there has been a substantial communications problem among operations research analysts, computer scientists, and mathematicians on queuing networks, progress is currently being made.)

- An integrated document storage and control system providing features for data base and document certification is necessary. This implies not only data base integrity, but information flow control for purposes of security and protection, and inherent in such a system is certification of the processing algorithms creating information.

- There is also a need for tools to analyse productivity tools. The proliferation of "solutions" to the productivity problem have in themselves become part of the productivity problem - there is

substantial residual cost associated with many approaches to software development productivity improvement. Until measurement of productivity in the systems development process is better understood, evaluation of specific tools will remain difficult - therefore, the first step must be in that direction, and meaningful metrics must be established.

- The 1985 Software Quality Assurance study was directed towards major government contractors who have had the most extensive experience in rigorous quality control problems. It was found that some progress has been made since INPUT identified commitment to quality as the foundation of any productivity improvement program in its 1980 report. Specifically, it was found that:

  - The companies interviewed had established quality assurance programs which focused attention upon quality as a separate and important issue over the system life cycle.

  - The individuals responsible for the operational quality assurance programs seemed Knowledgeable of what could and could not be accomplished, and were dedicated to improving the quality assurance programs within their organizations.

- However, there seemed to be little concensus among the various individuals as to what constitutes an effective quality assurance program except for management support (broadbased management in the INPUT productivity pyramid). A number of other conclusions were reached which support this lack of consensus.

- The diversity of tools and approaches being recommended and used leads to substantial confusion in even discussing individual programs. For example:

  . Reviews can be extremely detailed or quite superficial, and "walkthroughs" can range from actual code inspections to simply signing off on the paperwork.

  . Test beds established by the quality assurance function can perform perfunctory testing as part of the product shipment process, or can actually be relied upon by the project leaders to help them in debugging.

- There are no magical tools for any phase of the quality assurance process (including code inspection and testing). As one executive stated: "Lets face it, if any of us (systems houses) could get a true 10% software productivity advantage no one else could compete. Therefore, I don't believe there are any magic bullets out there. And, if I ever find one you can be sure I'm not going to tell anyone else about it!"

- Most of those engaged in operational quality assurance programs feel that the technical literature on productivity and quality assurance is the work of theorists who do not understand either the realities of a highly charged political environment (where emphasis is on getting the product out the door) or the economics of software quality assurance (the consensus was that it adds approximately 10% to the development costs). Therefore, the theoretical solutions to some of the problems are deemed unacceptable in the real world.

- There was one point on which there was consensus, and it was negative - regardless of the methodology employed, substantial amounts of documentation are generated. This adversely affects acceptance of the quality assurance function, and also impedes the quality assurance organization in performing its work.

- There was a general feeling that the cost of quality assurance is difficult to justify because the results cannot be measured. This brings us right back to the problem of measurement which INPUT identified as the need for "tools to evaluate tools". This is also an appropriate lead into a brief analysis of artificial intelligence and expert systems.

o In the United States, there is currently considerable enthusiasm about the potential of expert systems to solve many categories of productivity problems - including those inherent in the systems development process. INPUT listed ten major categories of expert systems and some potential personnel impacts in Artificial Intelligence and Expert Systems (1985). Exhibit II.3. Some of the conclusions reached in that study as they pertain to the development, testing, operation and maintenance of computer systems are as follows:

- The tendency in the computer industry has always been to under estimate the difficulties of software development. In the case of expert systems, it is INPUTs conclusion that the building of knowledge bases to support expert systems will be especially difficult for the following reasons:

EXHIBIT II-3

CATEGORIES OF EXPERT SYSTEMS AND REPRESENTATIVE IMPACTS

CATEGORY	SOME POSSIBLE IMPACTED PERSONNEL
Interpretive Systems	Data Entry Personnel Business Analysts
Prediction Systems	Forecasters (Including Consultants)
Diagnosis Systems	Doctors Field Engineers
Design Systems	Systems and VLSI Designers Accounting and Financial Systems Personnel
Planning Systems	Programmers Schedulers
Monitoring Systems	Project and Financial Control Personnel
Debugging Systems	Programmers Circuit Designers
Repair Systems	Field Engineers
Instruction Systems	Teachers Training Personnel
Control Systems	Industrial Engineers Investment Analysts

- The process of building knowledge bases presents a level of complexity which exceeds that of normal systems development and requires highly skilled personnel who are in short supply.

- There is every indication that, except in very narrow domains, the experts are either unwilling or unable to define how their knowledge is applied in complex problem solving (which frequently is highly intuitive) - this is especially true in such systems problems such as debugging.

- Because of the very nature of expert systems, the potential for hidden bugs is substantially greater than in conventional systems, and the expert may find it more difficult to analyse the "solution" presented by the expert system than it would have been to solve the problem from scratch.

- Having performance better than 90% of human "experts" may not be acceptable, when problem solutions are normally obtained from the 10% who specialize in solving those particular problems. (For example in medical diagnosis.)

- The question of human differences is especially important in the systems development process where is has long been recognized that ranges of programmer/analyst capability vary by several orders of magnitude. Barry W. Boehm quotes a range of 26 to 1 for programmer productivity in the debugging process, and INPUTs research confirms a range of at least that magnitude (in fact, there are those who state that the range is infinite because some programmer/analysts would <u>never</u> be able to solve certain problems).

- It is INPUTs opinion that the classic 90-10% rule (which states 90% of a system is normally developed in 10% of the time and the remaining 10% requires 90% of the effort), applies to individuals as well as projects - in other words, 10% of the programmer/analysts do 90% of the productive work, and the remaining 90% contribute only 10% and create a major share of the problems for the highly productive 10% of their fellow workers. It is, therefore, extremely dangerous to make the 90% more productive in producing systems of marginal quality which, in all probability, will require either replacement or substantial rework. This concern about expert systems applies as well to other applications development tools which facilitate the rapid development of low quality systems (that is the primary reason commitment to quality is at the base of the productivity pyramid). The fundamental conclusion remains the same - it is counter-productive to develop low quality systems.

- However, it is not our intention to be negative about either expert systems or other applications development tools, and INPUT supports the profound observation of Ada Augusta, Countess of Lovelace, about Charles Babbage's analytical engine:

  . "It is good to be wary of exaggerating the ideas that arise from the process of the analytical engine. There is often a tendency, in considering any new topic, to initially <u>overrate</u> the technology, by emphasizeing the interesting or amazing aspects of it. Then, when we realize it doesn't meet our expectations, we tend to <u>undervalue</u> the true condition of the technology."

- It is INPUTs opinion that Lady Lovelace would probably be appalled at the current propensity to first overrate and then undervalue applications development tools - including the language which bears her name.

o All past INPUT research on software development productivity points to the fact that there are no easy or magical solutions to the software development productivity problem and the research for this study seems to confirm that conclusion.

III. Project Research

A. Ford Aerospace Western Development Laboratories

- Ford Aerospace was interviewed on-site during the course of INPUTs 1980 multi-client study and was selected as the primary research source for this study for the following reasons:

  - The productivity improvement efforts of Ford Aerospace were specifically directed towards software development of embedded, on-line, real-time computer systems. INPUT has concluded that these are of particular interest to Toshiba, and we agree that the complexity of todays computer/communications networks (even those of a commercial nature) present all computere systems vendors and users with development problems which are comparable to those which have been traditionally faced by the aerospace industry.

  - The original research indicated clearly what INPUT considers to be the complexity of evaluating the effectiveness of using software development tools.

  - Representatives of Ford Aerospace have been quite open in reporting the results of their productivity improvement efforts, and this has not been true of most comparable efforts.

  - Ford Aerospace western Development Laboratories is conveniently located to INPUTs corporate offices.

- At the time of INPUTs earlier research, INPUT found that Ford Aerospace was mounting a substantial effort to improve software development productivity through both environmental considerations (improved office

workspace and availability of terminals), and through the adoption and extension of the Programmer Workbench (PWB). (It was reported to INPUT that more than $1 million had been spent enhancing the PWB.) In 1980, a coordinated set of methods, standards and procedures were being proposed for all future projects, and there was quite a bit of enthusiasm at the executive level about the progress which was being made.

o As mentioned earlier INPUTs approach was to interview at various levels within the company, and as we proceeded down from the executive level, we encountered the following reactions to the proposed corporate standards for the development environment.

  - The manager of one major project was familiar with the standardization effort, but stated that his major strength was the ability to keep together a compatible team from one project to the next, and this was accomplished because his employees liked his management style and his project control system. His attitude was that any attempt to force a change would increase both costs and turnover on his project, and he did not think anyone would want to take responsibility for that. His support of the corporate effort was limited to new projects where new teams are being brought together - not new projects being given to an established work unit (such as his).

  - A project leader interviewed stated she had heard about the PWB and associated standards but wasn't "worried" because "my project has 2 or 3 years to run, and by that time everyone will have forgotten about the standards". Her general attitude was that she would

protect her projects from unnecessary intrusion from a bunch of "corporate types".

- A programmer interviewed merely stated that it sounded like a lot of "paperwork" to him, but he would depend upon his immediate manager to take care of the problem.

- While this highly individualized and resistant attitude may seem quite foreign to Japanese management, it is especially common in software development in the United States. There is a general tendency to permit managers and project leaders who are perceived as being highly productive to continue doing what they are doing as long as they meet rather poorly defined objectives. And, in all truth, there is some concrete evidence that introduction of changes in methodology or tools will adversely impact productivity of those accustomed to a different working environment.

o In 1984, the former software engineering director of Ford Aerospace Western Development Laboratories published a carefully thoughtout article on the results of the five year effort to improve software productivity. The first and most discouraging fact reported was that the cost per source instruction (adjusted for inflation) had gone up between 1979 and 1983 (the period covered by the all out effort). However, there were some reported changes to explain and offset this seeming adverse impact on productivity.

- First of all, it was pointed out that the systems being developed in 1983 much bigger and more complex with much greater "documentation and record-keeping demands".

- Design level work in Program Design Language (PDL) "far exceeded the kind of detail thought necessary in 1978-'79)".

- Unique devices were attached to the computers and they required special software and timing analysis.

- The burden placed on the system at the man-machine interface had greatly increased.

- Each computer performed many interleaved functions and the internal housekeeping function lowered effective throughput which, in turn, required further redesign and reprogramming.

- And, it was then pointed out that quality (in terms of detected discrepancies during final integration) had been improved, and both estimating and cost control had been improved. The quality of documentation was "so high that it wa probably excessive."

- The general conclusion reached was that quality and consistency had been achieved at the expense of productivity. The report went on to analyze the continuing "productivity problem" in more detail and this will be discussed under conclusions and recommendations.

o Discreet inquiries recently revealed that at least some projects at Ford Aerospace remain outside the standard development procedures which were being established in 1980, and some project leaders definitely feel the paperwork involved in using many of the new tools adversely affects their groups productivity. The general attitude seems to be that the requirements for tools varies from project to project and selection should be left to the project leaders.

- Based on the reported results at Ford Aerospace, it is also possible to raise other questions concerning the true impact of the productivity improvement effort which could not be pursued with in the limits of this study.

  - Since it is INPUTs experience that the envioronmental changes being pursued by Ford Aerospace (office space, terminal access, etc.) normally result in productivity improvement, was the impact of the standards, methods, and tools (PWB, PDL, etc.) actually more negative than the reported modest increase in the cost per source code instruction?

  - Were all of the costs of the productivity improvement program included in the costs being measured?

  - Did the tools employed contribute to the increase in complexity which was observed over the 5 year period?

  - Were the projects which did not employ the standard tools and methodologies included in the measurements? And, what was their realtive impact on the cost per source code instruction?

  - Were Ford Aerospace projects such as the York anti-aircraft gun (a $1.8 billion project which was scrapped in 1985) included in the cost per source code instruction calculations? (It became obvious from the published accounts that some of the embedded systems malfunctioned rather dramatically, and this calls into question any statements about improved quality.)

- INPUT believes the Ford Aerospace experience is especially important in demonstrating that, despite sincere efforts in establishing a comprehensive software development productivity improvement program, the problems of productivity improvement seem to remain. And, Ford Aerospace is not optimistic about the prospects for improvement because they have reached other conclusions based on their experience:

  - "Integration and testing, although a large part of the total cost, is not a very promising area for productivity improvement. It is near the end of the development cycle and consequently has little flexibility and very little multiplier effect. ... Clever simulators, automatic test sequencing and recording, and even partial automatic test plan generation would not cause large savings in this part of the development cycle."

  - And, they confirm what we all know about coding with the following statement: "Suppose we could go to machine instructions directly from a design language like PDL, all on the computer. That entire process now represents only 15% or less of the cost of producing the kind of software we are concerned with, and a good portion of that is occupied by module and unit testing. It appears we would be very lucky indeed to realize an overall improvement in productivity of 10% or more in this area. The technology to go to machine instructions directly from a design language is still a long way off. Such a large R&D investment for the potential productivity yield does not seem cost effective."

B. Hewlett-Packard

- It was pointed out earlier, that most computer and systems companies are not open about the software development productivity problem. This is true for two reasons:

  - The problem seems to remain regardless of how much effort is expended in pursuit of its solution.

  - The people attempting to solve the productivity problem become personally involved and are reluctant to publicize results which expose the persistant nature of problems which they are supposed to be able to solve. (In addition, if true progress were made it is probable that the particular solutions would be considered to give a competitive advantage and be kept proprietary.)

  - The net result is that whether there is success or failure the tendency is keep internal tools and aids proprietary.

- In order to obtain some insight into what is actually happening at Hewlett-Packard, a confidential interview was conducted with Gopal Kapur of Kapur and Associates who specializes in software productivity improvement consulting. Mr. Kapur has conducted seminars at HP, and Kapur and Associates has developed a life cycle and project management system (Kapur Method/2000) which is being used by some of their clients, and which is currently beginning to be marketed commericially.

- Mr. Kapur gave us the following information concerning HP productivity improvement efforts.

  - HP has a proprietary development environment called HP Life Cycle which is a life cycle and project management methodology, and

includes an integrated work bench. There are two versions of HP Life Cycle, one for engineering and one for internal HP business applications.

- Mr. Kapur stated that it is not HPs management style to enforce use of these tools, or to impose what he views as necessary discipline in the systems development process.

- He has observed a general reluctance to talk about the actual experience with HP Life Cycle, but his experience with specific projects (business applications) within HP would seem to demonstrate that significant progress in such areas as specification preparation, structured walkthroughs, and test beds has <u>not</u> been made.

- He stated that both software engineering and data processing management regularly attend his seminars, and it is obvious that they recognize the need for improvement in the application of HP Life Cycle if not its technical and functional content.

o It is INPUTs opinion that HP is probably representative of the larger computer manufacturers - they have provided tools, but except for documentation, individual projects are permitted a great deal of flexibility in whether, or how, such tools are employed. Kapur concurs in INPUTs assessment that particular tools are not really so important as how they are applied, and the general physical and management environment in which they are applied. (For example, Kapur and Associates has adopted a policy of only answering telephone calls between 1:00 and 4:00 PM and feels this improves their own internal productivity as much as employing any specific methodology.)

C. Tektronix, Memorex Communications, CXI

- In order to get a general feel for how smaller companies approach productivity improvement, INPUT conducted a confidential interview with Warren Davidson who has served as Project Manager with Tecktronix, Manager of Technical Services with Memorex Communications Group, and is currently Manager of Software Services with CXI,Inc. (Micro-mainframe products). In all of his positions, Mr. Davidson has been responsible for control and release of software and performance measurement, and has been instrumental in attempting to establish Project Management and Control Systems.

- Mr. Davidson's general analysis is that software development teams normally prefer to develop their own development tools (or tailor others to suit their particular purposes), resist formal project management and control systems (or consider it an overhead exercise and do the minimum they can to comply), and work in collusion with marketing (and customers) to force products through the release process.

- It is his impression that systems software development (embedded communications systems, operating systems, etc.) is much more difficult to control than commercial applications development because systems programmers and engineers do not respect anyone without their detailed technical knowledge of their systems (and this extends to management as it gets removed from the day-to-day technical problems). It is difficult for anyone (quality control or line management) to impose project control on implementors of complex systems because the project leaders always fall back on the question of who is going to be responsible for the schedule. (Mr. Davidson stated: "the time when you really have to watch

out is when they agree to run their projects by your rules - then you Know they are going to blame any problems on you.)

- The result has been that effective project management and control systems are extremely difficult to install - for example, one does not currently exist at CXI and there is little hope of getting one installed despite the fact that the President of the company specifically hired Mr. Davidson to do just that. As he so aptly puts it: "When you don't know what is going on, you obviously can't measure it; if you can't measure it you can't control it; and if you can't control it, it is ridiculous to talk about improving productivity. We have problems."

- In his previous assignments, Mr. Davidson had succeeded in installing an "automated" software testing system (basically a custom VTAM simulator) which feed data into a project control and product performance evaluation system. However, the simulator was developed by an outside firm and the reporting systems were developed in house and were considered proprietary.

D. Dialogic Systems, Inc.

- Dialogic has developed a Development Center WorkBench which is based on IBMs Interactive System Productivity Facility (ISPF) and is designed to off-load COBOL development activities from IBM mainframes. The cost of less than $3,000 per user is a substantial reduction from the costs associated with interactive development using IBM mainframes, and improved development tools are reported to provide over a 20% productivity improvement compared to those on IBM mainframes. The Development Center WorkBench can be used either locally (directly

connected to the Block Multiplexor channel or connected remotely over multiple communication lines.

- o However, INPUTs interest was in determining how the Development Center WorkBench itself was developed for the DIALOGIC/10 MidFrame computer which consists of multiple functional processors (up to 31 Motorola 68000 microprocessors) operating in parallel. These high performance embedded microprocessors are as follows:

  - Host communications processor.
  - Remote terminal processor.
  - Local terminal processor.
  - Diagnostic processor.
  - Multiple application processors.
  - Multiple file processors.

- o It is our understanding, based upon a telephone interview, that the original development work for the Development Center WorkBench (approximately 250 man years of effort) was done in C Basic and Pascal on a VAX supermini under UNIX, but the enhancement and maintenance activities (which have been estimated to require another 250 man years) are being done on the WorkBench itself. If, in fact, this bootstrapping has occurred it would indicate language and target processor capability which could potentially be very attractive for developing high performance applications in an SNA network environment. Unfortunately, the scope of this study did not permit us to substantiate: 1) whether the

Development Center WorkBench is really being used for its own enhancement and maintenance, 2) exactly what the $3,000 per user cost figure contains, or 3) whether there is any plan for marketing another version of the WorkBench specifically aimed at distributed architectures.

E. Other Interviews

o As mentioned earlier in this report, it was decided to review briefly the status of Software Engineering Economics (as presented by Barry W. Boehm in his book on the subject), Information Engineering (as originally presented by James Martin and Clive Finklestein), and Queuing Networks (as pursued by Dr. Ralph L. Disney). The reasons for INPUTs interest in these three areas will become apparent when our conclusions and recommendations are presented. Briefly stated, the current status of the three areas are as follows:

- While INPUT still feels that Mr. Boehm's book has made a substantial contribution to the metrics of software development, and could serve as the basis for workable models for project estimating, productivity (performance) measurement, and project control; there are some problems with the practical application of much of his knowledge.

- Mr. Boehm as Director of Software Research and Technology at TRW, Inc. is concerned primarily with government contracts in aerospace and defense. Detailed analysis of the specific success or failure is difficult because the specific tools are considered proprietary, and the projects are classified.

. A confidential interview with the TRW credit operation disclosed that Mr. Boehm's work was economically justified only on government projects with stringent quality control requirements. (TRW estimates that quality assurance using SDM 70 adds 15% to development costs on commercial projects even after the documentation required is reduced substantially.)

- Mr. Martin and Mr. Finklestein came to a parting of the ways shortly after writing their book on Information Engineering. The problem seemed to revolve around the practical application of Information Engineering and what the term really meant. Since then Mr. Martin has become associated with DDI, and it appears that the product is being used to define the concept. One thing becomes clear - Information Engineering is not a science, and the term has even less meaning than Fourth Generation Languages (another ill-defined term from the fertile mind of Mr. Martin).

- Dr. Disney has specialized in Queuing Networks for over 25 years, and gradually this Operations Research tool has become accepted by computer scientists as being of considerable assistance in complex computer scheduling and resource allocation problems. Essentially, much to the consternation of some mathematicians, Queuing Networks have been remarkably accurate in predicting device utilizations and throughputs (errors seldom exceed 5%); and, while less reliable as predictors of queue length and waiting time, error rates usually less than 25%. The author of this study has consulted with Dr. Disney over the years, and there are particular reasons to be concerned about Queuing Networks at this time:

- Hierarchical computer/communications networks with systems (and data) distributed over mainframes, minicomputers, and microprocessors present resource allocation problems which are substantially more complex than any encountered by operating systems on central processors.

- It has long been the authors intuitive belief that the organization of large software development projects and the attendant communications (data and information flow) problems are amenable to Queuing Network modeling; and, this intuitive feeling even stronger as project teams go on-line with advanced software development tools.

- Dr. Disney agrees that Queuing Networks are promising for the more complex distributed systems which are currently being developed; he stated: "everyone seems to use the term 'robust' these days, and I guess Queuing Networks are robust."

- He was not prepared to comment about software development work units, but he was sufficiently interested to schedule an April meeting with us in Palo Alto.

o INPUT has also stated that APL might provide the necessary meta-language for concise communication across systems interfaces. We conducted a confidential interview with an IBM employee who has been associated with APL since its beginning in the IBM Research Center in Yorktown Heights. Not surprisingly, he agrees with our assessment of the potential of APL, and he stated that recent releases have provided enhanced capability which would facilitate reliable "foreign language" execution across

systems interfaces. However, the battle within IBM concerning APL continues unabated after more than 20 years.

IV. Conclusions and Recommendations

A. Conclusions

o The specific conclusions reached from this study are as follows:

- There are no general all encompassing solutions to the productivity improvement problem - no one tool or set of tools is available which are either acceptable of workable in all situations.

- Major computer manufacturers and systems developers are inclined to develop their own set of tools (or heavily modify those which are commercially available), and attempt to standardize on the use of those tools for purposes of project management and quality control. There is little reason to believe that these efforts have been effective in either producing clearly demonstratable productivity improvement or in standardizing the way software is developed or managed.

- Smaller companies (with smaller work units) are especially resistant to the imposition of methodologies which address the life cycle because of increased paperwork. There is still the classic tendency to rush into programming with inadequate requirements, specifications, analysis and design; and programmers continue to resist documentation. There is also a general feeling that highly productive individuals and work units should not be disturbed for fear that productivity will be adversely impacted (which may be true).

- Generally speaking, there seems to be concensus that tools should address the life cycle and that some type of integrated software

development work bench with ready access to centralized, on-line documentation is required. (Massive volumes of paper documentation is a sure sign of a project which is in trouble.)

- It is generally felt that systems complexity is increasing more rapidly than any efforts to improve productivity, but this especially difficult to quantify because there are no accepted metrics or agreement on what contitutes software productivity.

- Despite changes in terminology (such as software engineering), software development remains a relatively undisciplined art rather than a science. Twenty years ago, computer architects (such as Dr. Fred Brooks and Dr. Gene Amdahl) felt that engineering discipline could be applied to the software development process - to date, there has been very little progress in doing this.

- Past experience has shown that success or failure of software projects seems to depend more on "good management" rather than the application of particular tools, methodologies, or formal approaches to productivity improvement. Unfortunately, there does not seem to be any clear indication of what consitutes "good management" of software development. It is possible to isolate vastly different management "styles" which seem to have contributed to both spectacular successes and failures in software projects.

o INPUT is convinced that the productivity pyramid (Exhibit II.1) forms a general basis for the management of software projects and for the improvement of productivity. We are also convinced that productivity must be viewed at four performance levels:

- Hardware-software
- Human-machine dyad
- Work unit
- Institutional

o When viewed in this manner, several additional conclusions can be reached:

  - There is tendency to believe that hardware costs are decreasing so rapidly that sloppy software can be tolerated and overcome by processing power. INPUT believes that commitment to quality extends to hardware-software performance and that "quick and dirty" solutions (including tools which create hardware performance problems) are counter productive.

  - Most attention at the human-machine dyad has been addressed to making the human more productive. In the United States this has taken the form assuming that English is an appropriate human-machine interface. INPUT does not agree with this assumption and feels that unrestrained endorsement of English language tools will be counter productive. E. W. Dijstra has expressed the feeling that English tends to obscure the very power of the computer, and our current research on APL elicited the same reaction - "Everyone complains about all of the symbols in APL, but there are many problem solutions which can't be descibed in English." In addition, human capabilities vary so much that no single man-machine interface is best in all cases.

- At the work unit, there has long been a general awareness that adding more people does not normally improve schedules, and large work units are not as productive as small work units. Despite this awareness, there has been a general tendency to add personnel to both the development work units, and "in support" of the work unit. There is every indication adding more people and making those of lower ability "more productive" tends to lower work unit performance substantially. During the course of research for this study, Gopal Kapur stated he has a guaranteed way of improving software productivity at least 50% - he challenges his clients to remove the least productive 50% of their employees, and he agrees that the last thing which should be done is to improve the productivity of poor performers (because their work is really counter-productive to work unit performance).

- Institutional performance can only be improved by having end user involvement and broadbased management. There is a general tendency for various work units to shift responsibility to others. For example, computer architects have been inclined to leave certain problems to software in order to keep costs down. A classic case of this was the decision <u>not</u> to provide indirect addressing in the IBM System/360 architecture - there is no question that this was counter productive from the point of view of both the corporation and its customers. (The current trend towards RISC machines is especially susceptible to these types of counter-productive trade-offs, and must be carefully managed with institutional performance in mind.)

- The analysis of the Ford Aerospace experience with productivity inmprovement provided the following list of reasons that software development is so difficult:

  - Incomplete or inaccurate software requirements.(1)
  - Incomplete or inaccurate interface definition.(2)
  - Trying to fit too much in a given memory space.(3)
  - Trying to do too much computing in too little processing time.(4)
  - Trying to do too many things simultaneously inside a single computer.(5)
  - Internal conflict for resources.(6)
  - Queuing.(7)
  - Too many types and/or numbers of interfaces with outside hardware.(8)
  - Error detection and correction.(9)
  - Documentation.(10)

- This list can then be classified as follows:

  - Items 1 and 2 are familiar project management and systems engineering problems which have been discussed in terms of the productivity pyramid.
  - Items 3 through 7 are concerned with software system architecture and all represent parts of the same problem - todays software

architectures were fundamentally designed for another technological era. Perhaps no where is this more evident than in IBMs Systems Network Architecture with its heavy emphasis upon large host mainframes with multiple levels of operating systems overhead. It is felt that advances in microprocessor technology and economics make this problem amenable to solution. (Fundamentally, this demonstrates the importance of the hardware-software performance level - housekeeping and overhead consume so much of the systems resources that they are not applied to solving the problem at hand.)

- Item 8 can clearly be demonstrated at all levels in the processing hierarchy, but the problem is particularly acute in office automation (local area networks) where diverse devices must be accommodated and new technology, such as optical memories, must interface with obsolete systems software (both operating systems and data base management systems). Once again the problems listed in 3 through 7 become apparent and there is great risk that performance at the hardware-software level will result in the development of unworkable systems where response times become unacceptable. Standards are a possible solution in this area, but IBMs dominance of the US computer industry permits the establishment of de facto standards and virtual control over most industry standards activity. (For example, it is impossible to establish standards for optical storage devices before IBM embraces the technology.)

- The complexity of shared software architectures (in the current environment, the VM/MVS/UNIX strategy of IBM, under the SNA architecture, establishes levels of complexity which threaten to

extend down to the microprocessor level) make error detection and correction (Item 9) extremely difficult and this in turn has obvious quality and productivity impacts. Testing and integration in an environment dictated by IBM will become increasingly difficult, and the general tendency to be IBM compatible will turn out to be a vicious trap.

- Ford Aerospace describes documentation (Item 10) as a "massive, costly, and time -consuming element of software development" which is presently overdone and "is used mostly to convince semiskilled project managers and customers that the software is going together properly and addressing all the system requirements."

o INPUT concludes that much of our past analysis of the productivity problem is being confirmed for all types of systems development projects. It is an extremely complex problem which will not be solved by any magical tool which places the emphasis on generating code and getting fast results. As the environment becomes more complex there is no less need for tools to facilitate program development and more need for tools to analyse data and information flow.

o The final conclusion is that the scope of this study did not permit sufficient analysis of any specific productivity tools being employed by computer manufacturers or those available from commercial vendors. However, it is our opinion that most will work if employed after assuring the base of the productivity pyramid is in place, but none will improve productivity without that foundation.

B. Recommendations

- It is not recommended that Toshiba attempt to adapt the productivity programs and/or tools of other vendors to its own use. There is no indication that any of these tools have achieved dramatic improvements in productivity. The problem is too complex for simple solution and undue emphasis upon tools can be a distraction for establishing a comprehensive productivity improvement program.

- However, we do recommend that Programmer Work Benchs and the latest developments in Life Cycle and Project Management Methodologies be explored if you have not already done so. While INPUT does not endorse any products, both the Kapur Method/2000 and the Dialogic Development Center WorkBench were both of sufficient interest to warrant additional analysis, and Maestro (developed by Softlab GmbH and implemented on Four-Phase Systems hardware) has provided a general programming environment which has been been commercially available for sometime. Addresses for marketing outlets here in the United States are provided in Appendix A.

- INPUT obviously endorses the work it has done (and is doing) in productivity improvement. Specifically, we believe the following:

  - It is necessary to understand IBMs software directions because, for better or for worse, IBM is creating the systems software environment within which other software systems must be integrated. Of particular importance are SNA, Operating Systems, and Data Base Management Systems. All of these areas have been explored in detail, in past INPUT reports, and this year special emphasis is being given to Operating Systems at all levels in the processing hierarchy.

- We believe that integration of multiple languages, operating systems, and data bases systems is inevitable and there is a tremendrous need for a powerful meta-language(s) for concise communication across hardware-software systems, program modules, user languages, and data base management systems. It is strongly recommended that APL be considered as a possible tool for this purpose.

- We also believe that performance at the hardware-software level is becoming and increasing critical issue, and it is not possible to depend on advances in computer technology to compensate for the increasingly complex systems which are being developed. It is well Known that there are limits on computer processing power, regardless of cost, and that of these limitations can be exceeded by many of the algorithms of both artificial intelligence and operations research. There is great need to be able to predict, monitor and control hardware-software performance if these limits are not to be exceeded (and unworkable systems developed).

- It is necessary to become familiar with the tools of AI and OR as both possible solutions to the productivity problem and as potentially being part of the problem. We strongly recommend those responsible for complex software development become familiar with such important subjects as Game Theory, Information Theory, Decision Theory, and Queuing Networks etc. before proceeding with the development of Decision Support and Expert Systems which may fail with disastrous and even catastrophic consequences.

- In summary, we recommend that the productivity pyramid be understood and used as the foundation for the development of a productivity improvement program, and that the effectiveness of any productivity improvement program be measured against the four performance levels established in this report (hardware-software, human-machine dyad, work unit, and institutional)