Technical Report
CMU/SEI-91-TR-008
ESD-TR-91-008

Issues in Tool Acquisition

Paul F. Zarrella
Dennis B. Smith
Edwin J. Morris

September 1991

Technical Report
CMU/SEI-91-TR-008
ESD-TR-91-008
September 1991

Issues in Tool Acquisition

Paul F. Zarrella
Dennis B. Smith
Edwin J. Morris

CASE Technology Project

Software Engineering Institute
Carnegie Mellon University
Pittsburgh, Pennsylvania 15213

This report was prepared for the

SEI Joint Program Office
HQ ESC/AXS
5 Eglin Street
Hanscom AFB, MA 01731-2116

The ideas and findings in this report should not be construed as an official DoD position. It is published in the interest of scientific and technical information exchange.

FOR THE COMMANDER

(signature on file)

Thomas R. Miller, Lt Col, USAF
SEI Joint Program Office

This work is sponsored by the U.S. Department of Defense.

This work was created in the performance of Federal Government Contract Number F19628-95-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 52.227-7013.

This document is available through Research Access, Inc., 800 Vinial Street, Suite C201, Pittsburgh, PA 15212. Phone: 1-800-685-6510. FAX: (412) 321-2994. RAI also maintains a World Wide Web home page. The URL is http://www.rai.com

Copies of this document are available through the National Technical Information Service (NTIS). For information on ordering, please contact NTIS directly: National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. Phone: (703) 487-4600.

This document is also available through the Defense Technical Information Center (DTIC). DTIC provides access to and transfer of scientific and technical information for DoD personnel, DoD contractors and potential contractors, and other U.S. Government agency personnel and their contractors. To obtain a copy, please contact DTIC directly: Defense Technical Information Center / Attn: BRR / 8725 John J. Kingman Road / Suite 0944 / Ft. Belvoir, VA 22060-6218. Phone: (703) 767-8274 or toll-free in the U.S. — 1-800 225-3842).

Table of Contents

List of Figures

Issues in Tool Acquisition

Abstract: This technical report identifies issues involved in the acquisition of Computer Aided Software Engineering (CASE) tools. Among the issues identified and discussed are cost, performance, process support, maintenance, data management, tool integration, and standardization. The report concludes with recommendations intended for individuals or groups responsible for acquiring CASE tools.

1 Introduction

In virtually every software engineering organization today, consideration of how to select and use software development and support tools is crucial in improving staff productivity and product quality. As each organization is different, so then does each organization have specific expectations and requirements to be addressed by tools. These tools must be carefully matched to the adopting body to facilitate the development process. The choice of the wrong tools can not only fail to improve the process, but can actually work to hinder it. Unfortunately, no specific, universal set of rules has ever been devised to aid users in their search for the “perfect” set of tools, nor does a set of tools currently exist that would satisfy the needs of all organizations.

Users often tend to react negatively to changes in technology rather than anticipating and preparing for them. Many times, users will find themselves anchored to a tool or methodology that has become outdated. This can be detrimental to the organization when it becomes necessary to upgrade the development environment. The organization may be locked out from technology advances made in the industry due to incompatibility of the new technology with existing processes and methods. To protect themselves from any of these risks, some users will purposely decide not to commit to any particular technology until some specific version has “matured” or until it has been generally accepted.

Currently, there are few accepted tool standards, and there are no guidelines pointing out how to make the best use of available tools. There are no absolute metrics with which the effects or influences of any tool or method can be determined. There is currently no one tool or method that is appropriate or suitable to every organization (no “silver bullet”) nor is there a means by which to predict the future methodologies, tools, or standards that will eventually emerge (no “crystal ball”). With this in mind, it becomes a matter of preparing for multiple technology paths and attention to trends in tool technology for an organization to be best positioned for the future directions of the tool market.

Even though a formidable task, it is not impossible to work within the framework of the currently unpredictable state of tool evolution and to begin implementation of an organizational tool policy. One productive method for developing a tool selection and adoption strategy is to examine the major issues confronting the tool implementation process and to plan for change,

even though the lack of substantive, available data makes this a difficult proposition. This paper was written to aid the reader in navigating through the complex considerations facing an organization when making informed tool decisions. It outlines the principal issues currently under discussion in the area of tool technology and offers suggestions on how to cope with the current uncertainty of tool selection and usage.

1.1 Overview of Current Issues

Numerous studies have shown that software is responsible for a large percentage of system complexity and cost. The demand for quality software is rising faster than the ability to produce it. Software systems are often on the critical path for system acquisition, and can be a leading cause of problems in the system.

Unfortunately, it is likely that the problems of software complexity and cost will worsen substantially. A survey of senior executives in government and the private sector indicates that the nation will have a serious problem in being able to produce mission-critical software over the next five years [18]. Results of the survey are summarized in Figure 1-1.

There is an obvious need to develop quality systems more quickly, and to develop techniques to support better and faster software development. Nevertheless, the software producing sector of the economy suffers from a relatively low level of capitalization relative to other industries. The cost of a good mechanic's toolset and garage equipment often far exceeds that of tools provided to a software engineer in a typical software industry setting. In fairness to the managers of software development and maintenance, the determination of the proper place to spend corporate money has not been easy. Managers must weigh the costs and potential benefits of improvements in hardware, staffing, and training against the less certain benefits of providing new software development and maintenance tools.

One potential method for meeting the rising demands for software is to provide better tools to software engineers to facilitate the production of more and better software. This approach is best represented by the growing market for sophisticated Computer Aided Software Engineering (CASE) tools and other productivity tools (see Section 1.2.1 for a list of productivity tools in a sophisticated environment).

Sophisticated software productivity tools are expected to:

- Improve software quality
- Increase developer productivity
- Improve control of the development process
- Lower development costs
- Lower maintenance costs
- Lead to improved customer satisfaction
- Reduce development backlogs

Percentage of executives who think that there will be a problem with the nation's software capacity to produce military software over the next five years [18]

	Expect a problem	Expect no problem	(N)
Industry Executives	88%	12%	90
Government Executives	87%	13%	16
All Executives	88%	12%	106

The assessed degree of severity of the problem for those who expect a problem *

	Mean Score	(N)
Industry Executives	3.9	80
Government Executives	4.4	14
All Executives	4.0	94

*Scale	Very Serious		Serious		Not Serious
	5	4	3	2	1

Figure 1-1 Executives' Assessment of National Software Capacity

Unfortunately, the literature concerning the effectiveness of such productivity tools is inconclusive. It is unclear whether individual CASE tools are effective. In addition, industry observers disagree concerning the types of projects that benefit most from the use of the tools [15 and 17], the point at which improvements become obvious [1and 2], and the degree of improvement [7 and 11]. Inconclusive results and contradictions offer the software manager little guidance in determining which tool or set of tools will be most useful.

Complex decisions concerning tool strategy and adoption require projections based not only on present conditions, but also on future trends, due to the long life of many software products. Among the major tool issues discussed here are:

- Cost
- Performance
- Support for the software process
- Maintainability
- Data management
- Integration

- Standardization

1.2 Cost

A large number of factors can influence initial and continuing cost of developing and maintaining a state-of-the-art software development environment. Among these factors are the large number of tools necessary in such an environment, the costs for the initial installation, long-term maintenance costs, the costs of the supporting hardware and software environments, and the costs for additional and better trained staff. These factors will be discussed in subsequent subsections.

1.2.1 Types Of Tools

An appreciation of the cost of developing a state-of-the-art environment should start with an understanding of the tools that such an environment will offer. While each state-of-the-art environment will probably consist of different sets of tools, it is clear that all will consist of more than code generation and documentation tools.

For the development and maintenance of software throughout the life cycle, representative tools from the following categories are likely to be included in such an environment:

- Project management tools, including estimating tools, tracking and scheduling tools, status reporting tools, and configuration management tools.
- Planning tools which provide capabilities for strategic data planning.
- Analysis and design tools, which provide support for one or more methods of requirements elicitation and design.
- Code level tools, including compilers, code generators, debuggers and performance analyzers.
- Documentation tools, which support sophisticated graphics and text manipulation capabilities.
- Testing support tools, which analyze resulting software complexity and aid in the generation of test suites.

1.2.2 Start-up Costs

We are unaware of any empirical data available on the cost of providing a full set of software development tools. However, instructive lessons can be learned from the projected costs associated with the implementation of a sophisticated suite of tools and the associated hardware to support a staff of 150 engineers. Figure 1-2 contains a list of the projected start-up costs from American Management Systems [11].

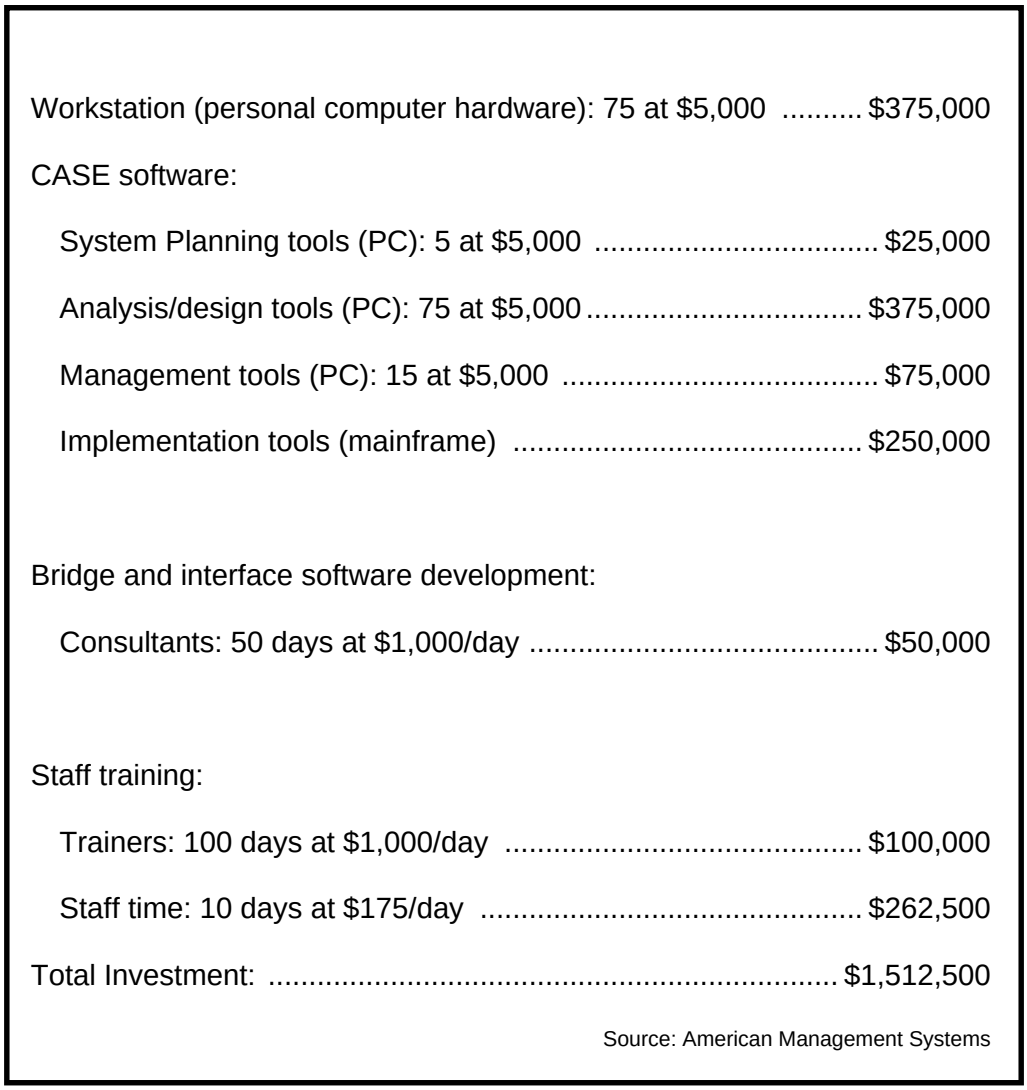


Figure 1-2 Start-Up CASE Investment

According to the data presented in Figure 1-2, the total start-up costs associated with the investment in a sophisticated tool set would exceed $1.5 million. It should be noted that this estimate includes $375,000 allocated toward the purchase of a set of workstations at $5,000 per unit. Such a purchase may or may not be necessary, depending on whether appropriate hardware is already available. However, should hardware have to be purchased, the cost per platform may exceed the $5,000 estimate.

Provision of a UNIX workstation environment, increasingly common in commercial systems and DoD settings, can be even more expensive. Sophisticated workstations running the UNIX operating system and providing adequate performance for modern CASE tools will typically cost $10,000–$20,000 per platform, although some discounting is common. In addition, the typical purchase price of a CASE tool running on the UNIX platform exceeds the figure of $5,000 per copy quoted. Many popular CASE tools available for the UNIX environment cost $15,000–$20,000 per copy, although again, discounting is the norm. Utilizing the new figures

for a UNIX implementation of $10,000 per workstation, and $15,000 per copy of the tool, the total cost of a tool implementation, with appropriate hardware, could exceed $3 million. Should an organization decide to implement a VMS workstation environment, costs would be expected to be similar.

1.2.3 Ongoing Costs

In addition to the start-up costs associated with the tool implementation, ongoing costs for a sophisticated tool implementation are also substantial. An estimate of these costs is provided in Figure 1-3 [11].

Once again, if hardware is upgraded to UNIX workstations and provided to all engineers, overall ongoing maintenance costs will increase.

1.2.4 Additional Hardware And Software Costs

In addition to the high costs of the tools and operating platforms, the less obvious cost of developing and maintaining the networked computing environment required for maximum advantage also needs to be considered. Whereas earlier generation software engineering tools, such as compilers, editors, and debuggers operated using terminals in a time-shared environment, many modern tools require a layered network approach, often using UNIX workstations, with additional support for various other computers, including personal computers and minicomputers. The costs of such a heterogenous network of computers can be significant, particularly when the migration from a time-share system is occurring concurrently with migration to CASE tools. Multi-user CASE tools place a significant burden on this environment. To achieve adequate performance, powerful and expensive tool servers must be purchased, and often dedicated, to the execution of the CASE tool. It is likely that even relatively sophisticated organizations will need to consider hardware upgrades to effectively support multi-user tools.

It should be pointed out that the cost for hardware to support CASE tools is declining. As new, more powerful Reduced Instruction Set Computer (RISC) processors become available, the price for computing resources adequate for CASE tools declines. In addition, the de facto standardization of the X11 windowing system offers the potential for significant reduction in hardware costs per engineer. Relatively inexpensive X11 terminals have the potential to replace workstations on some software developers' desks. Not only does this offer an immediate cost benefit per seat to systems administrators, but it also offers the potential for reduced costs for system administration staff. Along with the promise of reduced costs, however, X11 has brought confusion to licensing schemes used by tool vendors for multi-user versions of tools.

Software engineering group:

Three people at $50,000 (salary & benefits) $150,000

Hardware maintenance:

75 at $500 ..... $37,500

Software upgrades/maintenance:

Estimated at 10% a year for PCs $47,500

Estimated at 15% per year mainframes $37,500

On-going staff training (done by engineering):

Two days per person for staff of 150 at $175 ..... $52,500

Attendance at user meetings, symposia:

Two people to attend four meetings at $1500 ..... $12,000

Miscellaneous:

e.g., books and publications $10,000

Total annual ongoing cost: $347,000

Figure 1-3 Ongoing CASE Costs

Vendors are only beginning to develop schemes that deal with the potential for multiple users using X windows to access a single copy of the tool software.

1.2.5 Personnel Costs

Perhaps the greatest cost of the new, networked computer environments is that of providing the necessary system maintenance staff. Experience within the Software Engineering Institute suggests that networked environments require two to three times the number of maintenance staff members than that required for older, time-shared systems. This increase in staff (and

perhaps in technical skills as well) appears to be related to two main factors: the support of multiple platforms, and the maintenance of the sophisticated computing network capable of supporting the heterogeneous platforms in the environment. Heterogenous networks attempt to provide transparent file and process services to vastly different machine architectures and operating systems. For each supported operating system and architecture, a core computing staff is necessary to support both the different hardware and the different tools that populate the environment. The maintenance of the sophisticated computing network to provide the interactions between the various platforms itself requires an additional increase in staff.

The characteristics of the types of tools used in modern software development environments also contribute to the need for increasing numbers of system support staff. As tools become more sophisticated, both the level of sophistication necessary to maintain the tool and the amount of maintenance required of the tool increase. Customers of tool vendors demand that a sophisticated tool be tailorable to the user's environment. In response to the customers, the majority of sophisticated tools available on the market today provide some degree of tailorability. In fact, many sophisticated software engineering tools today now come with a maintenance manual as large and complex as the tool's user manual. This situation differs from earlier environments that had relatively simple editors and compilers in that it requires more system administration.

1.2.6 Trends in CASE Tool Costs

It is easy to predict the individual trends which will affect hardware and personnel costs related to adopting and maintaining current generation CASE tools. The price to performance ratio of hardware will continue to decline. Hardware will become more affordable. Likewise, one need not be prescient to know that the cost of employing or contracting for personnel to manage the systems, provide training, and design systems using CASE software will continue to escalate.

What is much harder to predict are the trends in costs for the actual purchase of a CASE tool set or environment. The actual amount an organization will need to allocate for future CASE purchases will be determined by the conflicting trends of a decrease in the costs per seat for a specific tool function, mitigated by a corresponding increase in the functionality provided by individual tools and by the overall CASE environment, and by the increasing number of users who need access to the tools and data. These users will grow beyond the traditional software developers to include system engineers, managers, quality control specialists, and documentalists.

It may not be apparent that costs per seat for a specific tool function will decline. Evidence of this trend already exists, however. As the CASE tool market has become more competitive, discounting of tools has become deeper. Even where tool costs have remained static or even increased, the price reflects tools which provide far greater functionality for each dollar that is spent. In addition, tools are incorporating new licensing schemes that allow for widespread access to tools at little increase in cost.

Another set of trends, now becoming apparent in the database market, may also affect the overall per seat cost of CASE tools. One such trend is the increased bundling of software with hardware purchases. It is expected that such bundling may act to reduce tool costs. In addition, database vendors have noticed a decrease in the amount of cost and effort purchasers are willing to spend on customization of tools. It is unclear whether this change reflects a decrease in customization, or a decrease in the effort necessary to customize new tools with built-in customization capabilities. Finally, a growing trend toward leasing of software rather than purchasing software is developing. By leasing software, a company can reduce its up-front expenditure, and amortize the cost of the software over some period.

1.3 Issues of Tool Performance

Before any organization can commit to the purchase of a full set of tools, including CASE tools, a complete evaluation of the cost effectiveness of such a purchase should be determined. Determining the cost effectiveness of a tool set is difficult, as costs are divided up among costs to initially acquire a sophisticated tool set, costs to maintain the tool set, costs to maintain the appropriate computing environment, as well as costs to train and support a large staff to maintain and use the tools. The limited data concerning tool costs (Figure 1-2 and Figure 1-3) make accurate cost projection difficult. Based on the little information available, total costs appear to be high, and it is unclear whether the long-term expected benefit represents a reasonable value for the investment. Not enough empirical evidence exists to either support or undermine the decision to purchase a sophisticated tool set or to support the claims of some tool vendors. Each organization considering the purchase of sophisticated tools must evaluate its own risk and potential benefits.

A second set of issues concern lack of adequate tool performance. As a system gets larger, the time required to perform simple actions can increase rapidly. For example, while some tools can regenerate data dictionaries transparently, others require complex rebuilds that take hours to perform. Such performance problems can be attributed to both hardware and software causes.

Some sophisticated CASE tools place heavy demands on hardware resources. This is particularly true of multiple user versions of tools, which place large demands on both the tool servers on a network and on the network itself. Several simultaneous users of a tool can tax even the most sophisticated computer facilities. One observer [14] suggests that before large-scale integrated software development environments will be feasible, new storage technologies that store more data and retrieve it more rapidly, such as optical disks, must be perfected.

In addition to the demands placed by software engineering tools on computing hardware, an equally stringent set of demands is placed by the tools on software technology, particularly in the area of data storage and retrieval. As discussed in Section 1.6.2, the needs of new, sophisticated software engineering tools cannot always be adequately met by simple file storage systems, nor is it clear that they are met by relational database technology [5]. New, repository-based technologies utilizing object-oriented databases are only now being developed.

1.4 Support For Software Process

According to Humphrey [13], the software process is the set of tools, methods, and practices used to create a software product. The tools used by a software organization, however, must reflect the software methods and practices in place within the organization. Smith [19] advises that an overall tool strategy for an organization or project should consider the process and methods in place, determine the functions of the lifecycle that need to be supported, and account for the environmental infrastructure that will let tools work together.

For a tool to be successful in a large-scale development project, it must support a process that is well entrenched rather than existing outside the bounds of the process. Those tools that offer a useful service, but do not fit smoothly in the existing process, are not likely to be successfully used on large projects. Likewise, tools that require manual reentering of data to go from the previous stage to the supported stage or from the supported stage to the next offer additional problems due to the loss of data during the transfer.

A number of modern environments have attempted to institutionalize the process environment into the toolset. Notable examples of this approach are ISTAR [10], based around a business contract model of software engineering, and SLCSE (Software Life Cycle Support Environment), based around a DOD-STD-2167A approach to software development.

It is important to note that the purchase of any tool, particularly modern tools that claim to offer cradle-to-grave support for the software process, entails a set of risks. We cannot know what new methods, tools, or systems will be devised, or which will fall into disfavor during the life of an application. While it is obvious that the purchase of tools is necessary, all purchases should be framed in terms of the costs, benefits, and risks associated with the tool.

1.4.1 Changing Models And Methods

The development of a strategy for tool selection is not simple.The software engineering community has not reached consensus on what processes, methods, and practices are most successful. In fact, both practices and methods within software engineering are currently undergoing a considerable transformation.

A major influence on the practices maintained within a software organization is the model of software development accepted by the organization. Through the 1970s and for much of the 1980s, the waterfall model of software development was the accepted model for the software development process. The software maintenance process was often viewed as a smaller scale analog of the development process. It is not surprising to find the major features of the waterfall model embedded in government standards such as DOD-STD-2167A, in the methodologies supported by commonly available tools, and in software environments based on these standards and tools, such as SLCSE.

Recently, alternative software development models have been suggested, including the cyclic model [3] and various models based on rapid prototyping and reuse of existing software components. It is unclear whether any of these new models will supercede the waterfall model as

the accepted standard. In fact, it is most likely that the software community is entering a period where multiple models of the software development cycle are common. What is less clear are the changes in the types and functionality of tools that will be necessary for effective support of new models of software development.

Just as the practices within software development are changing, so are the methods that are used to develop software as new object-oriented, rapid prototyping, and reuse-oriented methodologies are introduced. There is a significant field of debate in the software engineering community concerning the effectiveness of the established structured analysis and design techniques, particularly for large systems. Proponents of object-oriented methodologies believe that these newer methods offer better overall system design, facilitating reuse of software components. Proponents of rapid prototyping methodologies argue that they are best able to establish user requirements and reduce the risk of development. Secondary ongoing controversies concern the appropriateness of the new methodologies for various types and sizes of software, and for various stages in the development process.

There is little or no empirical data to suggest that one specific method will lead to greater success. Wood (1989) found little to support the choice of one structured specification method over another. It is unclear whether the use of newer object-oriented, prototyping-oriented, or reuse-oriented methods will enhance the chances for success. Successful, as well as unsuccessful, projects have been completed with all of the contending methodologies.

While it is possible that the object-oriented, prototyping-oriented, and reuse-oriented methods and processes being developed will be completely compatible with existing tools, the possibility also exists that they will be incompatible. Recently, tool vendors have attempted to develop methods of integrating standard structured analysis and design techniques with object-oriented methods. The level of success of these integration attempts is unclear. In the meantime other, low-technology support for new methods is being defined. For example, one of the most popular development practices for the object-oriented method, Class Responsibility Collaborators (CRC), uses only low-tech index cards as a design aid.

1.4.2 Tool Support For Software Development Methods

There have occasionally been some mistaken references to a “CASE methodology”. There is, in fact, nothing unique or original about the methods that CASE tools support. Some commonly available CASE tools support many of the software development methodologies that have been available for over a decade prior to the introduction of the tool.

All of the software development methods represented in CASE tools attempt to provide support for accepted standards of good software engineering. These methods provide, at a minimum, support for modeling of a system at various levels of abstraction using various views of the system. The methods often provide guidelines for decomposition of a system into appropriate software units. Actual choice of one method over another will continue to be a factor of the type of software to be developed, experience with the methods, and personal preference.

The representation of a method in a tool can be difficult, particularly when the tool is to be used to create large-scale software products. Two potential problems are incomplete support for the chosen method, and lack of tool support for team-oriented development. While CASE tools have improved considerably in accurate and complete representation of the supported method, support for team-oriented development requires a level of integration with other tools (such as document preparation tools, configuration management tools, and project planning tools) that remains generally unavailable today.

It is important to recognize that all available tools are the result of compromises between performance, features, and development costs. Just as it is unlikely that a single computer language will be appropriate for all projects, no single set of tool features are appropriate for all projects.

Tool builders must make compromises concerning tool performance and the look and feel, and between the conflicting needs of users.

One example of the compromises made by tool vendors can be seen in the provision of automated code generation capabilities. For the developers of small to medium-sized systems, automated code generation capabilities available in modern tools provide an excellent way of verifying the consistency of module interfaces. On the other hand, builders of large software systems may find that the level of modeling necessary to generate realistic code interfaces may be too low for large-scale development. These users may choose to model at a higher level, which is appropriate according to the methodology. In this case, sophisticated code generation capabilities are not required. Regardless of whether a project can make appropriate use of a feature, most tools are configured such that the user pays for the feature, both in dollars and cents and in increased complexity of the tool interface.

1.5 Maintainability of Tools

Large-scale development projects are likely to also be long-lived development projects. Often, the project is economically feasible only if it can be argued that the software will last for years. Therefore, the costs of the software should be weighed against years of benefit. It has been argued in the CASE literature that the primary benefits of CASE tools do not come from the savings during initial development, but rather from savings that become apparent during maintenance [1]. Thus, it can be expected that a tool that offers a large return on investment will be one that lives a long life, assisting in the maintenance of the software possibly for the life of the software. In fact, a number of agencies request delivery of the development tool set along with the finished software product.

Unfortunately, long-term maintenance of a set of tools presents problems due to the immaturity of existing tools and the potential that a tool may become obsolete during the life cycle of software products reliant on the tools.

1.5.1 Tool Obsolescence

First among these problems is the common problem of tool obsolescence. Tool vendors market their products today in a number of configurations for different operating systems, different windowing systems, and different hardware platforms. In addition, they sign large numbers of new agreements with other tool vendors, hardware suppliers, and framework suppliers to provide some degree of integration with other products. They regularly produce new, improved versions of their software.

The problem for the developers of software tools is obvious: the more people are devoted to maintenance of tools on the various platforms, the less people that are available to develop new products to compete in the market. The tool developers must either drop support for less popular platforms and frameworks, or face the risk of overextension.

The problem for the organizations that wish to buy a tool is also obvious: there is no guarantee that a tool will be supported 10 years, 5 years, or even 1 year from today. Without the tool that was originally built to maintain the software, maintenance may become more difficult, and the benefit from use of the tool during implementation may be lost.

It is not always the case, however, that the same tools that are used for software development are the best tools to use for software maintenance. For example, software maintainers may find re-engineering and restructuring capabilities more useful than other tool features. A tool may fall into disuse because it is no longer appropriate for the job, even when it continues to be supported by the vendor.

The effect of this problem in the DoD world is significant. Due to product liability for software systems produced for the DoD, contractors are justifiably cautious about purchasing tools other than those that are well established.

The difficulties of maintaining a tool over the life of a large piece of software are compounded by the complex interrelationships between current tools. Many modern tools are integrated with other tools such as documentation systems, windowing systems, and configuration management systems and are dependent on those systems. Unfortunately, system upgrades to tools in the environment do not happen simultaneously. It is common to find various tools at different parts of their life cycle: some just released (with the associated bugs), others having reached the stability of middle age, and still others aging, falling behind in interfaces to other tools and in technology. Often the version of a tool (or even an operating system) that an organization can run is more dependent on the interactions between tools than on the features the user wishes to have available. During the maintenance cycle, when the tools used to develop software (e.g., operating systems, methods, compilers) become 2, 5, 10, or even 20 years out of date, the maintenance of a tool environment can become difficult.

One major problem that contributes to this situation is a lack of universal agreement on a standard operating system, windowing system, convention for tool interfaces, data dictionary, and data storage mechanisms. Without such accepted standards, long-term maintenance of a tool is difficult for tool vendors and users alike.

A software organization that wishes to use sophisticated tools must develop a strategy to manage the change in tools over the life cycle of the affected software products. Any decision to import new tools for beginning or existing projects should be tempered by the costs of reverse engineering the software product to the formats of the new tool. Without common interchange formats, or built-in reverse engineering capabilities, the updating of tools may offer little or no advantage.

Some tool vendors are beginning to recognize the problem of maintaining tools over the life of a software product and are starting to provide mechanisms to manage the long-term configurations of tools in relationship to the software developed with the tools. For example, Sun's Network Software Environment (NSE) [20] provides a method of creating and maintaining tool and software environments that can be managed as configuration objects (see Section 1.6.3). However, the costs of maintaining personnel trained in the use of old tools cannot be mitigated.

1.5.2 Immature Tool Technology

A second maintenance problem is due to the use of immature technologies in the available tools. The first available CASE tools were characterized by sophisticated graphical drawing capabilities with little or no page layout capabilities, limited integration with a select set of tools, little or no project management support, little or no data gathering or software metrics support, limited code generation support, methodological naivete (no methodological guidance), and no support for software reuse. Newer tools planned by tool vendors, however, offer features including improved ease of use, multi-user capabilities, project management support, metrics support, artificial intelligence (AI) assistance, autolayout capabilities, reverse engineering, methodological flexibility, and support for new methodologies.

The transition toward these newer CASE products, as well as the entry into the CASE market by IBM, Digital Equipment, Hewlett Packard, and Texas Instruments indicates that a market shake-out may be in order. Small vendors who do not have a strong niche in the market may be forced out of the industry. The small vendors' position is made more precarious by the tenuous grip they hold in the market. Gane [9], in his book *Computer-Aided Software Engineering*, listed projected 1988 revenues for 22 CASE vendors. Half of that group subsist on a market share of 1 to 3 percent. In a market subject to the competitive pressure applied by the large computing giants, the survival of small vendors is uncertain. In fact, the only chance for survival for many of the smaller vendors may be to rapidly transition to offering interfaces compatible with those offered by the larger vendors.

In addition to market shake-out, the immaturity of the marketplace is expressed in the development of competing frameworks for CASE tools. In the United States and Europe, various groups have banded together to form a variety of standardization committees (CAIS-A, CIS, PCTE, PCTE+, PCIS). Large computer manufacturers have introduced their own CASE frameworks. The situation can be confusing for a developer or potential purchaser of CASE tools. For example, experts familiar with the PCTE and PCTE+ efforts express little hope of convergence of the standards. In fact, they expect the two to remain distinct frameworks for

the foreseeable future. To determine which standards the vendor should sign up to is not a simple matter. In many cases, adhering to any of the proposed standards would require the CASE vendor to remove functionality from a tool. As we move toward environments containing integrated configuration management and database capabilities, these functions must be removed from existing tools. The vendor will need to choose to support one particular standard and tailor his tool to that standard, or must choose to remain independent with a full-featured product. The stakes are high for any vendor making these decisions.

1.6 Data Management

One of the major areas of concern to tool vendors and end users is the issue of data management. There are numerous facets to the data management issue. Many vendors have competing interests in definition of data interface protocols, structure of common storage, and data exchange formats. Data management concerns can be decomposed into the following issues:

- Tools and data
- Database technology
- Configuration management
- Data representation and semantics

1.6.1 Tools and Data

Tools operate on data objects that are generally stored in some form of proprietary database. The purpose of the database is to provide a vendor with a way of maintaining data consistency within the tool set. Users who wish to share complex data between tools are generally constrained to tools offered by a single vendor due to the current limitations of inter-tool data integration. Most vendors do not offer a full life-cycle complement of integrated tools nor do they adhere to any accepted standard for data representation, effectively isolating their tool databases from access by the tools of other vendors.

Most tools use localized, non-distributed databases to store tool objects. This creates problems with data consistency between tools and users and puts the burdens of data synchronization and reconciliation onto the user or the tool framework. Also, end user tools are often precluded from interfacing with the data directly due to the proprietary nature of most tool databases. These problems increase data maintenance overhead throughout the development life cycle as provisions must be made for integration of data from a variety of diverse sources (e.g., local and remote databases, tools, users).

1.6.2 Database Technology

As tools and their interfaces have become more sophisticated, so have the requirements of data types and data storage facilities. Over time, these data storage requirements have evolved from simple, sequential file access to complex, database management technologies. Data management must not only take into account the increased volume of information that is

being stored by tools, but also must be able to distinguish between a growing number of complex and derived data types.

Unfortunately, the properties of an object management system (e.g., object distribution and locking, flexible data types, versioning) are not fully served by available data storage techniques. There is a contention that current database technology cannot effectively support full storage of all object data in a central repository.

According to Brown [5], relational databases cannot readily accommodate the flexible, complex object/data types (e.g., code segments, design diagrams, user processes) required by large-scale development technologies (such as CASE). They are generally unable to handle the amount of data that it takes to implement a fine-grained object management system (i.e., objects are composed of data items and interrelationships that are more complex than the level afforded by a singular file or character string representation). Software engineering databases must handle continuously evolving data, continuous schema modifications performed by a large group of software engineers, and data of many different types including ASCII text, various document formats, source and object images of a program, and graphical design formats. In addition, these databases are loaded with little initial data, but must expect rapid growth both in terms of structure of the data and the contents of the data. They must also handle multiple versions or configurations of the data and support long-lived transactions with many individual units of information that are smaller than those locked by relational databases. This combination of demands can lead to significant performance problems as the volume of the data in the database grows.

To overcome the limitations of relational databases, many tool vendors are currently exploring alternatives to the local dictionary for data storage/access. This new technology is generally referred to as a data “repository.” The repository is a database designed to hold the types of data necessary for large-scale software development and maintenance. Repositories are generally built on top of relational or object-oriented databases.

Object-oriented databases differ from relational databases in that they store data maintenance and access rules along with the object data. This technology provides extended capabilities (e.g., multiple object views, modifiable rules and object types) and enhances the distribution/performance aspects of a shared database. However, object-oriented database technology is relatively new (no single data model) and many vendors, users, and standards organizations have existing commitments to relational databases.

1.6.3 Configuration Management

An important requirement of any tool used for large-scale development is that the tool provide adequate support for maintaining multiple versions of the data in the tool, or that the tool provide output that can be archived in existing configuration management (CM) tools. In a large software development project, data and source code may exist in hundreds of different configurations and intermediate forms. If a tool is to prove valuable to users during this process, it must be capable of maintaining many forms, or must provide mechanisms to allow some oth-

er tool to maintain them. Unfortunately, there are no existing standards for versioning and CM tools.

Two common approaches are chosen by tool builders today. The simplest, most frequently chosen approach is to require the user to provide his own configuration management capabilities. This approach is often unworkable for the user because the nonstandard data formats utilized by tools cannot by handled by many configuration management systems. In addition, the unit of configuration management must, by default, become the tool builder's unit of data storage in the file system (most often a file). Users, however, often wish to manage items with smaller granularity than the data file as the need evolves for control over a set of object types more varied than simple source code. However, the users are prohibited from doing so.

The second alternative chosen by some tool builders is to build in a proprietary CM capability specifically to support the needs of the tool. A primary example of this approach is the Ada program library, which maintains information about the source, object, and intermediate forms of an Ada program. Unfortunately, this approach causes considerable difficulty in integrating tools with other tools and frameworks in the environment. For example, the automated build facilities of sophisticated CM tools are often unable to work in the context of Ada program libraries, which provide their own (conflicting) build facilities. Also, extension of tools to include configuration support must be given careful consideration so as not to provoke the scalability problems (particularly data storage and performance limitations) previously outlined.

A third approach, likely to become more common as large vendors begin to offer framework products, is to tie into existing configuration management tools. Configuration management system vendors have begun to recognize the potential to integrate tools through CM capabilities, and are offering features that support inter-tool communication [20 and 12].

1.6.4 Data Representation And Semantics

Resolution of the data integration issue is also impeded by the problem of reconciliation of the different internal representations of the subject data. Most vendors not only utilize incompatible data interfaces, but also define incompatible data formats.

Some vendors have developed import/export facilities and tool extensions to deal with the problems of data dictionaries that are not capable of being shared among different tools or among multiple users. However, there is an inherent problem of data loss/mismatch when attempting to merge two (possibly) incompatible data sets. There is the possibility that insufficient data is being either exported or imported. Data objects/relationships used by the exporting tool may simply be extraneous to the importing tool or may not fit into its data model (e.g., type, format, naming conventions).

Many vendors realize the importance of publishing their tool interfaces to help other vendors (and users) interface their own tools with the vendor tools. However, this does not address the larger problem of data incompatibility as the semantics of the data are still not fully understood. Not only might the format of the data be different, but the contents of the dictionaries may be

inconsistent. The weaknesses of relational databases are also exacerbated when objects are transformed. Since data rules are imposed by the tools (as opposed to the database), the consistency of objects (the “view”) may well be distorted as meaning is lost or misinterpreted when moving the data between tools.

1.7 Tool Integration

Despite the existence of numerous tools directed at facilitation of the software development process, a major impediment to realization of a comprehensive solution is the current state of tool integration. Users are hesitant to commit to a single vendor for tool support. They are also wary of being locked out of technology advances that may be reflected in tools that are incompatible with those that are already in use. At the same time, vendors realize that no one company can afford to attempt to provide a complete set of tools that spans all of the phases of the development life cycle.

With the realization that cooperation is necessary to survival, vendors are now examining the various integration characteristics of tools and exploring the specific mechanisms that can be used to standardize tool interfaces. Analysis of tool integration (and associated standards activity) is generally separated into three functional areas (see Figure 1-4).

- Presentation integration
- Control integration
- Data integration

1.7.1 Presentation Integration

In general, one of the basic concerns of integrating tools is the issue of a consistent user interface/presentation. The investment that an organization makes in selecting tools is more than just the cost of the tools. It involves the time and cost of comprehensive training and support (both from the vendor and from the users). Standardized user interfaces and tool functions will help keep these costs under control as well as offer greater tool/choice flexibility to the users.

Presentation integration represents the development of consistent user interfaces for widely different tools. Some vendors are looking at a standardized user presentation format to decrease the tool learning curve and provide smoother transition between different vendor offerings. Vendors also realize that most users operate in a diverse network of heterogeneous systems and that commonality among these systems is an important consideration when selecting a standard.

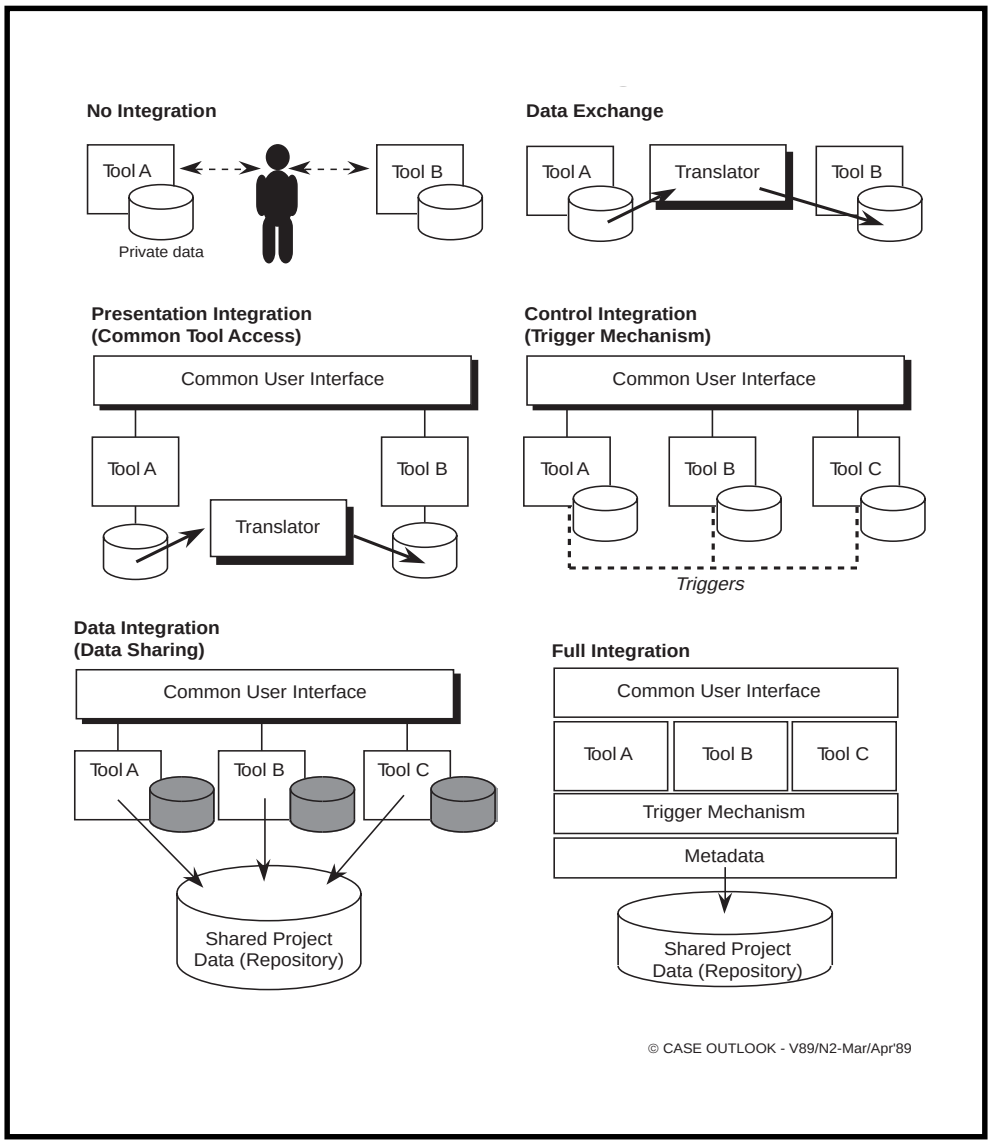


Figure 1-4 Levels of CASE Integration

1.7.2 Control Integration

Control integration refers to the ability of tools to inform other tools of their actions and to request actions by other tools. One mechanism used to facilitate control integration requires interface formats to be defined by specific tools. A tool wishing to communicate with another tool must invoke the tool via the defined interface, thus providing the potential for control integration. Unfortunately, the development of these tool-specific interfaces is labor-intensive and often occurs only through joint agreement of specific tool vendors. The user who wishes to utilize tools with these control integration capabilities is thus limited in their selection of tools.

A second, more complex mechanism requires an intervening monitor that receives tool notifications or requests and subsequently sends appropriate notifications and requests to other

tools in the environment. Each tool would maintain its internal database, but would provide an interface to import, export, and perform operations on objects in the database. This second technique requires that the monitor maintain an overview of both the other tools in the environment and the process that is to be implemented.

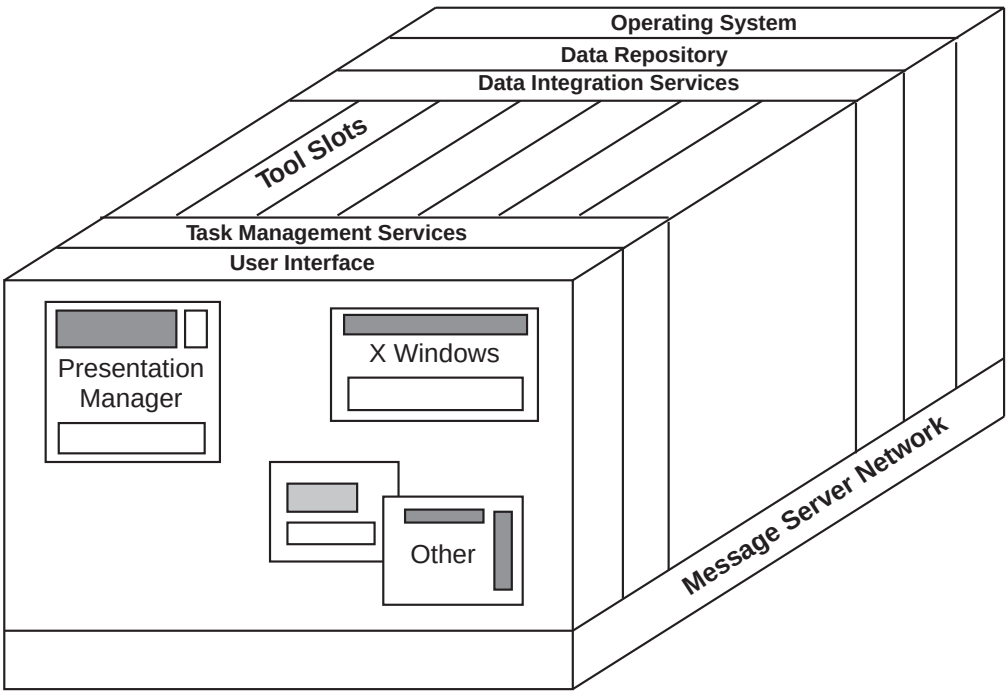
1.7.3 Data Integration

Data integration refers to the transfer of information between tools and the establishment of relationships between data maintained by different tools. One method requires that individual tools agree with specific data interfaces. This approach is relatively simple to implement and widely applicable to many types of tools. This method, however, provides only for the exchange of data and is neither effective at establishing links between data maintained by different tools nor at maintaining the semantic context of data.

A second method for achieving data integration involves the development of a shared repository in which a variety of tools store information. A fully functioning repository would provide the capability of maintaining a core semantic content of objects together with tool-specific views and would permit several tools to work together because of the common dictionary. Repositories have been discussed widely but are still several years from maturity, due to the complex database requirements.

1.7.4 Framework Technology

In addition to consideration of the forms of tool integration are the concerns over the processes involved in combining these aspects under a single point of control. Currently, efforts are being expended in definition and development of tool “frameworks.” A framework is sometimes referred to as an Integrated Project Support Environment (IPSE) (see Figure 1-5). The intent of a framework is to provide for greater interoperability of tools across the software life cycle and to enable the software development and maintenance cycles to become more integrated as objects created by different tools flow between phases.



Source: Hewlett-Packard, Software Engineering Systems;
as printed in SOFTWARE MAGAZINE, October 1989,
Sentry Publishing Co., Inc., Westborough, MA 01581

Figure 1-5 Full IPSE Model

Frameworks provide for the life-cycle integration goals of tools via “plug-in” tool integration with a tool management executive or “backplane.” The executive handles the overhead involved with coordination (control integration) of the tool suite (e.g., tool registration and instantiation, error reporting). At the front end of the framework is the mechanism that provides a consistent user interface (presentation integration) across the tool set. At the back end of the framework is a common data interface/repository (data integration), messaging system, and operating system services manager. These interfaces unburden the tool modules of the particulars of the host environment and allow for a broader, more interchangeable product set.

Framework development work has been going on for several years. Many vendors acknowledge the value of tool frameworks, but few are actively looking to integrate their tools into existing backplanes. Here, the issue of standards comes into play. The framework must be tightly integrated with respect to the control interfaces involved in a proper implementation. All of the tools in the framework must use standard protocols upon which to support the various control mechanisms (e.g., invocation format, parameter passing, results notification). Since no

one vendor can currently define a standard for framework integration by offering a truly comprehensive set of single-vendor CASE tools, and no defined standard has yet to be agreed upon, no single version of framework technology has been able to claim market dominance.

1.7.5 Tool Standardization

There are approximately 250 tool interconnection standards efforts currently in progress [16]. These range from government backed efforts to industry efforts to ad hoc standards committees. Many of the standards are aimed directly at the process of data management (data storage and/or data exchange). These efforts also deal with issues of tool portability (tool environment), tool integration (tool frameworks), and/or tool architecture (tool/interface consistency).

Much of the interest in current standardization efforts stems from the fact that each vendor has invested significant amounts of time and money in the development of their own proprietary tool interfaces. Each would ultimately like theirs to be the basis on which the industry standardizes. Realistically, vendors want to expend the least amount of effort necessary to adhere to any finalized standard and certainly do not want to be forced to support multiple conflicting standards. To this end, many vendors are examining the alternative approaches to standardization, including forming vendor partnerships and lobbying for acceptance of specific standards.

1.8 Formal Standards

One problem facing standardization is the “wait and see” attitude expressed by some tool vendors. These vendors are unwilling to support any specific standards effort or participate in several conflicting efforts in an attempt to “hedge their bets.” In these regards, they reduce the risk that they will adopt the wrong standard and also maintain the possibility that their interfaces already meet or approximate the (yet to be agreed upon) standard. This posture actually detracts from the standards efforts because failure to work actively toward a single standard is, in essence, a position against standardization.

The overall standards effort is further confounded by the fact that different standards address different aspects of integration. Some standards focus primarily on repository specifications, others on framework definition, and others on data exchange. Many of the standards efforts do overlap, but not all members of each standards body belong to all of the standards bodies. Thus, different viewpoints and agendas are driving standards for similar functions, which will most probably result in incompatible standards definitions.

All of this means that the emergence of multiple standards for tool integration is a possibility. Also, since most of the standards efforts had been progressing virtually independently of each other (at least until the inception of the International Workshop on CASE [IWOC] Standards Coordination Committee, and with the exception of any pending standards mergers), it could very well come to pass that the first defined standards predominate the industry or that no real consensus emerges at all.

1.8.1 De Facto Standards

While the formal standards efforts attempt to find common ground, the tool industry is seeing another form of standards process taking shape in the form of de facto standards. Generally when speaking of de facto standards, one is discussing a market driven by larger vendors. Such is the case with software tools. There are both positive and negative implications of this type of standards process.

On the one hand, de facto standards have the most likely chance of causing any type of real tool integration in the foreseeable future. Unfortunately, they are generally dictated by the larger vendors to the rest of the tool community. This means that smaller vendors and end users are forced to accept the unilateral decisions of the larger vendors without much chance for input on the shape of the interface standards.

Most vendors cannot be convinced to expend effort to adhere to an interface that may be superceded in the near future. This is the concern with de facto standards, basically because vendor integration of the tool/data interfaces of another vendor is a very costly proposition. The cost ultimately includes the indirect support and maintenance associated with the tools and interfaces of the other vendor. The costs are multiplied exponentially by the number of nonstandard interfaces and the number of platforms and environments supported by these other vendors. It is unlikely that smaller tool vendors will be able to continue to compete in this type of environment for long.

While there are always inherent problems in this type of standards adoption, at least there would be a focal point on which to build the forward process of enhancing the tool market. More users would be able to get started on their own tool implementation plans and a wider range of integrated tools would ultimately result. Also on the positive side, de facto standards should cause other vendors to finally come to standards “realization.” That is, as the larger vendors get users to start accepting their products and then start touting their particular definition of a standard, other vendors will be forced to join up with the standard or quickly consolidate around acceptance of other (hopefully well defined) standard(s).

1.8.2 Accepted (Unofficial) Standards

In addition to de facto standards, some tool interfaces and environmental and operating standards, such as X-windows and UNIX (more specifically, POSIX compliance) have already been generally accepted by the vendor/user community. The standards in use today have proven to be effective in enhancing the viability of many vendor products. This ultimately allows users to “mix and match” the most effective tools available from a variety of different sources.

Many vendors are looking at a standardized user presentation format to decrease the learning curve and provide smoother transition between different vendor offerings. Some vendors have chosen X-windows as a standard upon which windowing systems for tools presentation are built. This provides a definitive “look and feel” to the diverse tool set offerings and, in some cases, allows the familiar user interface to be extended to user-integrated tools as well. Al-

though no set standard has been officially declared, X-window acceptance is an example of how support by a wide range of vendors for a particular interface can help drive the standards definition process.

Another consideration is the emergence of a standard for the UNIX operating system. The UNIX system has been showing promise as a “cross-over” operating system catering to the needs of both the technical and commercial markets. In this respect, tool vendors would be relieved of the burden of maintaining separate product lines for multiple operating systems by targeting the UNIX system. This would also alleviate many of the rehosting issues facing the vendors because product porting would become an issue of hardware (UNIX system platform) only. Tool and database distribution would be greatly enhanced through the accessibility of existing network support functions (e.g., NFS, TCP/IP).

In addition to UNIX networking functions, the Open System Interconnection (OSI) standard for local area network data communications has emerged in the international software community. The OSI reference model consists of a clearly defined, seven layer structure for reliable connections between heterogeneous computing networks. The working model defines the capabilities of the different layers but leaves the particulars of the interface implementation to the vendor. This helps to make the standard more attractive to vendors by resolving ambiguity without mandating a specific solution. In fact, most vendors adhering to the OSI standard do not incorporate distinct protocols for each of the layers.

2 Conclusions

To make reasonable decisions about tool selection, an organization should be concerned with both the short-term selection issues and the long-term implications of the current trends in tools and tool use. In many ways, these selection issues and implications are defined by the changing tool market. Recommendations to assist in the making of informed decisions about tool purchases in this volatile market are presented in the following sections.

2.1 Make Judicious Use of Today's Technologies

Limited fixes to the problems remaining in tool integration are beginning to emerge. Not only are tool vendors beginning to develop common operating formats (e.g., inputs, error messages, editors) to be shared with strategic partners, but other vendors of important "substrate" products, such as configuration management systems, are developing new technologies to allow the sharing of information between engineers and tools.

One area of interest is the provision of tool notification functions in products such as Sun's NSE [20] and Hewlett Packard's Softbench Encapsulator [12]. These tools provide a method to notify tools in the environment of changes made by other tools. Notification functions respond to modifications made to the system and perform automatic update and/or notification of the appropriate resultant actions. These workspace management attributes then provide the capacity for improved source/data modification traceability. While requiring significant effort on the part of the tool user to establish and maintain relationships, this functionality represents an initial step at providing traceability between tools.

A second interesting development allows software engineers to manage the configuration of tools in concert with the data and software produced by use of the tools. This feature, available in sophisticated configuration management systems such as Sun's NSE [20], simplifies the management of tool configurations over the life cycle of a software product.

There is a general feeling that the user interface contributes significantly to the acceptance and learning time for a product. A third type of developing technology allows the provision of a consistent user interface across tools, thus facilitating tool acceptance and reducing learning time. Tool vendors such as Hewlett Packard [12] are developing common interface generators that allow for this simplified integration of proprietary (or third-party) tools into the vendor tool set.

2.2 Find Tool Support for Chosen Methods

Another trend that must be considered in the area of tools is that of changing software development methods. With the onset of next generation design methods (e.g., object-oriented design, rapid prototyping), users will find it necessary to carefully consider the impact on the organization of these methods and the associated tools.

Users should not jump to adopt a new, strictly imposed process for software development if they are best served by some internally developed or hybrid process. However, the most effective tools may be those that can be adapted to evolutionary changes in the software engineering process/model. Specifically, tools that are not adaptable to these changes will quickly become useless.

2.3 Make Better Use Of Existing Tools

While complete, integrated solutions remain out of reach, and the tool market remains in a state of flux, tool purchase will be a risky business. Although the situation with new tools remains unclear, much is to be gained by making better use of existing tools and technologies. According to Humphrey [13], organizations that score poorly on the process maturity scale are unlikely to make successful use of existing software tools. Most organizations are at the lowest level of process maturity and therefore are making poor use of tools. A first step in making better use of existing tools is to improve the software process that the tools support. Humphrey recommends that to make effective use of CASE tools, an organization must have established at least basic project controls, including project planning, management oversight, software quality assurance, and change control. Key problem areas to be addressed include training, technical practices involving reviews and testing, and the process focus involving standards and process groups.

Perhaps the most common method used to make better use of existing tools is to use practical, though rudimentary, methods of integration to achieve a unified product. Several users have successfully taken commercial off-the-shelf software and integrated these into tool sets. These users have selected those tools that provide well-documented public interfaces. Such efforts, which have been primarily undertaken by large corporations, are notable both in the relatively modest degree of success and in the amount of effort required. In addition to user-driven integration efforts, strategic marketing agreements between tool vendors have provided primitive integration between CASE analysis and design tools, and documentation tools. In both user-integrated toolsets and vendor-integrated toolsets, the level of integration in many cases is limited to primitive callable interfaces. Little progress has been made in the complex area of data integration.

Several authors, [4 and 6], suggest that users can make better use of existing tool technologies by improving the process through which tools are adopted. These authors contend that failures to achieve the desired results with CASE tools are often due to a failed process for dealing with the significant technological change represented by the tools. Andrews [1] has noted that an organization can maximize the benefit of CASE tools and minimize adoption problems by gaining management and engineering support, providing proper education and training, selecting appropriate tools, developing internal standards, and managing the expectations of management and engineers.

2.4 Develop Strategies For Tool Selection

When deciding on a strategy for tool selection, users must also take into account the amount and extent of adaptation that will be required by them (or on their behalf) to fully integrate a set of tools originating from diverse sources.

One important consideration in selecting an “open” tool set is that of tool flexibility. For tools to be fully integrated, users must be able to modify the characteristic behavior of the tool (e.g., design rules, object types, process), not just the display interface.

Users are faced with an interface support problem similar to that of the tool vendors. Users must decide if the effort required to integrate a set of proprietary tools would be greater than that required to develop a specific, tailored tool interface. Users also may not be willing or able to wait for a standard to emerge, or to integrate their tools to a (potentially) nonstandard interface. By adopting a private solution, the user also maintains control over tool definition as well as data, interface, and expansion characteristics (i.e., flexibility) and faces less risk of forced re-adaptation of tools due to interface and/or environment changes.

Given the current state of standardization efforts, ultimately the end users themselves may have to act as systems integrators and write the code necessary to effectively integrate a set of vendor tools. This assumes, of course, that the tool interfaces are documented by the vendors and that the user can afford to expend the time and cost necessary to implement the tool integration. In the interim, users are left to the task of manually coordinating changes between tool databases.

2.5 Develop Internal Standards

When considering the effectiveness of productivity tools, users should take into account that tools will be best utilized when tailored to the specific software development environment. Internal standards should be developed to facilitate the selection and usage of tools and to facilitate the transition of information to other internal and external organizations.

Users should be sure that they understand which capabilities and support technologies they require from a productivity tool. Users should then purchase tools that track developing technologies and are capable of being adapted to evolutionary changes in the software engineering process/model.

Tools should be considered as an extension of the operating environment. As such, users should be sure to consider any modifications to operating systems, support software, and/or system hardware that would be required to support the selected tool(s).

To gain the most from a tool set, users should work to a specific standard for tool data, process description, and documentation production. This will help prevent misinterpretation of tool data, facilitate the communication process between project members, and ultimately expedite tool effectiveness.

Finally, a defined set of metrics should be developed that would be used to quantify tool results. This will help users to measure their progress with respect to both productivity and quality improvements.

2.6 Consider Personal And Group Productivity Tools

Before making a final decision on tools, users should also consider the possibilities of personal or group productivity tools. It is not always necessary to select a fully distributed, multi-user tool set to improve group productivity.

In many cases, the costs associated with adopting a new set of state-of-the-art tools (from initial purchase, training, etc.) are prohibitive. In these instances, an organization might be better served by focusing instead on localized tool functions.

With the opening of the UNIX market, a number of productivity tools have become available for a growing base of UNIX platforms. Virtually every computer from PC to mainframe now has some flavor of the UNIX operating system available, and tool vendors are now offering products that run on many of these versions.

When using personal tools, decisions have to be made concerning how to handle/synchronize user data. In the case of local databases, the organization must have a plan for database update and distribution throughout the project(s) involved. In the case of a central, single-threaded database there are several complex issues (e.g., security, roll-back, integrity) that have to be addressed. These solutions require different levels of user tool interaction and coordination of data.

An organization should also determine how best to organize its staff for optimum tool use and support efficiency. Some organizations could utilize local tool “experts” to facilitate the process. Projects should also be organized to allow efficient communication of data/information throughout the development life cycle.

2.7 Identify the Most Cost Effective Tools

Due to the expense of purchasing a tool, few organizations can afford to do it without first attempting to analyze the actual tool costs. Before settling on purchase of a tool, it is useful to examine the overall process for areas that can be improved with other potentially cheaper tools. For example, if the CM process suffers from lack of tool support, it may well make sense to invest in a CM tool prior to investing in a more expensive tool to support other activities. Similarly, simple tools such as notes facilities can have a significant impact on productivity and quality by providing a better means to communicate project decisions and history. For example, one group of developers at Digital Equipment Corporation have found that the greatest benefit from the use of a tool set for software development came through use of technologically unsophisticated communication tools. At the recent SETA1 conference, when asked what tool improvements would be most beneficial to them, real-time developers cited improvements in such commonly available tools as compilers, debuggers, profilers, and linkers.

To select the most cost effective tool, the user must analyze their software development process to identify the most expensive areas and perform an analysis of individual tools similar to that presented in Figure 1-2 and Figure 1-3. Unfortunately, due to the lack of historical metric data about productivity and costs in much of the industry, it is likely that many organizations do not know where the majority of their software development and maintenance budgets are spent. General guidance is available from experts in the field of software metrics. However, general guidelines cannot replace data collected from within the organization.

In Jones [14], data on over 3500 software projects was presented and indicates that the work breakdown characteristics for software vary along the dimensions of system size (in SLOC and function points), system type (MIS, systems, real time, AI, etc.), system sponsor (internal use, commercial, DoD), and life-cycle phase (development or maintenance). The data implies that different tools will be more or less cost-effective depending on these characteristics. For example, Jones suggests that paperwork costs account for 45% of total costs in an average military system, but only 25% of costs in an MIS system. It is unlikely that a tool that is cost effective in one setting will be equally cost-effective in the other.

Differing work breakdown characteristics for software will influence not only the type of tool that is most cost effective in a specific setting, but also will influence which tool should be considered as the best for a specific setting. Many sets of guidelines are available for tool assessment, among them one published by the Software Engineering Institute [8]. Care must be taken, however, to apply appropriate weights to tool functions based on the work breakdown characteristics when determining which tool is most appropriate for the organization.

2.8 Track Emerging Standards Carefully

To anticipate the direction of the tool market as opposed to simply reacting to it, users should get more involved in the process that will determine the future of the practice. This can be accomplished via several constructive approaches.

Users should get involved in selected standards efforts. By attaining representation on the standards committees that most directly affect the users' end product or service, users can get their input considered directly by the standards body. This will help reduce product interface restructuring once a standard is adopted.

Users should look for complementary standards efforts. When adopting tools from different vendors, try to select tools that will integrate with little or no end-user adaptation. Users should look for vendors with interests common to their own. There are a number of vendor partnerships that were established to address the issues of development and integration of comprehensive tool suites.

Users should be sure to consider basic standards. Make sure that tools are purchased from a vendor that adheres to the accepted standards. Small vendors who choose to "go it alone" will likely not be in existence for long. Choosing tools with standard interfaces will save time and

money in the future, especially if end-user enhancements or adaptations have been performed.

2.9 Make Selective Use Of Influence

In addition to tracking standards, users should work to influence tool decisions as is reasonable. Users should first determine what level of influence is desired. Any user efforts in this regard are bound to place a burden on the user organization and would best be expended in protecting one's interests rather than being spread out too widely.

Users should also try to determine what level of influence is practical given their level of expertise and interest. For example, attempting to define a standard alone is likely to be less effective than participating in the standards definition. Users would best be served by meshing their own desires and expectations with similar interests in other organizations.

Finally, users should try to anticipate the expected payoff period of their participation. If involvement in tool definition would produce the "ideal" tool but take too long to produce it, then the effect of the influence would be negated by missing the window of application need. It is also likely that ongoing advances in technology would render this resulting tool obsolete. In this case, users would be better served by using off-the-shelf tools and working to help facilitate the tool integration and adaptation aspects of the implementation.

2.10 Maintain A Flexible Posture

Currently, the emphasis of tools is on the "bigger is better" approach. That is, tools are often developed that contain multiple software development functions (e.g., graphics editor, version control, data management) in one complete package. This may partly be due to the standardization and external integration problems previously mentioned. With the exception of specific vendor partnerships, multiple vendors generally seem unable to come to terms and produce a comprehensive set of integrated software tools.

It is still uncertain what will happen in the area of tool frameworks. There are many considerations that have to be addressed with respect to standardized interfaces for the three principal integration areas: control, data, and presentation. Full, open tool set integration is not likely to occur in this sense. It is most likely that framework standards will be adopted by limited private vendor partnerships and that the resulting proprietary flavor will act to exclude general vendor and user acceptance.

In the future, tools are most likely to become more specialized. As de facto tool standards emerge, vendors will start to produce tools that fit with the defined interfaces. It is possible that the larger vendors will continue to produce monolithic tool sets, but smaller vendors will likely develop specific specialized "point" tools that will be compatible with the established formats.

2.11 Understand The Changing Tool Market

It is important to keep in perspective the relatively small size of the software development market for tools in relation to the overall tool market. The majority of tools available on the mass tool market today are not specifically aimed at software developers, but rather at end users. Even tools such as documentation systems, which have gained wide acceptance in the software engineering market, were not originally intended for that market. In many cases, software developers have been a late group to take advantage of developing tool technology. One need only compare the vast array of personal productivity tools available for personal computers to the relatively small number of personal productivity tools available for the workstation market to realize that software developers do not control the general market.

It is also important to note the DoD market is smaller still. According to Jones [14], military software makes up 22% of U.S. software production. This 22% total is further divided into almost every conceivable type of software. If military budgets shrink relative to the rest of the economy, the impact of DoD software will be even smaller.

As the percentage of DoD software declines when compared to the total software market, the influence of the DoD on the tools market will likely follow. The DoD will find it difficult to take the sole lead in developing enterprises like ARPANET, or in enforcing standards like TCP/IP. DoD organizations that wish to maintain their current level of influence must develop strategic partnerships with like-minded, non-DoD organizations. The influence of DoD organizations may be made most effective through active participation in a variety of commercial standards efforts, some of which are advancing standards only peripherally related to software engineering, such as document interchange formats.

2.12 Consider Specific DoD Needs

While the DoD must become part of the larger software development and software user communities, it still possesses unique needs in the areas of security, hardware and systems software, legal issues, and documentation.

The security needs of the DoD will continue to be unique both in the level of security necessary and in the determination of access to information. In this area, defense organizations must continue to take a leading role for the foreseeable future, although orientation will be on influencing commercial standards to include functionality for DoD needs rather than developing unique security requirements. One case where this is already taking place is in the upgrade of PCTE security standards to those of PCTE+. This upgrade was initiated due to the efforts to migrate the European commercially-funded PCTE toward the U.S. DoD-backed CAIS-A standard.

In the area of hardware and systems software, DoD needs will continue to require exceptionally high performance systems for some hard real-time applications. On the other hand, as hardware improves, an increasingly greater portion of DoD needs can and will be met by commercially supported UNIX systems. Many groups within the military have recognized the grow-

ing acceptance of UNIX as a standard within the computing community, and have begun the massive effort to transition to the UNIX environment. A larger number of compilers running on UNIX workstations and providing for generation of code for military embedded computers are becoming commercially available. This will reduce the need for the military to support the building of specialized development environments, such as ALS/N, in the future.

Prior to the acceptance of systems built with sophisticated toolsets, the military must work out legal and practical issues concerning the purchasing of tools and support environments specific to applications. Recognition of the difficulty in maintaining DoD-specific support environments is evident in the recent emphasis of the STARS program to produce commercially viable support environments that interface with existing tools and technologies. It should be recognized that military organizations purchasing a large, long-lived system must purchase not only the system, but also the technology to maintain the system. In the near future, it is possible that the development environment may become as important in awarding contracts as the contracted system itself.

2.12.1 Find Ways To Deal With Documentation Costs

According to Jones [14], overall productivity for military software production lags behind that of commercial systems software production by greater than 30%, yet it appears that productivity during implementation is equivalent. The vast majority of the difference between productivity of military vs. commercial systems is attributed to documentation costs. Large-scale DoD tasks, unlike commercial systems software or MIS tasks, spend the largest portion of their project budget on tasks related to paperwork.

The large documentation costs in the military sector can be reduced in one of two ways: by finding better methods of producing required documentation or by reducing the amount of required documentation. Both methods are possible in the military environment. Modern CASE tools often provide documentation capabilities specific to DOD-STD-2167A, which can be used in conjunction with documentation tools. In fact, most users of CASE tools agree that the tools offer a significant advantage in the generation of documentation.

Perhaps an even more significant reduction in documentation costs and a corresponding gain in productivity can come from a concerted effort to determine what form and volume of documentation is of use to the software maintainer. Informal IBM studies revealed that much of the documentation generated during development at IBM was of minimum use to maintainers. A similar effort on the part of the military sector to determine an optimal form and amount of documentation could be used to tailor the level of documentation required of system builders, even within the framework of DOD-STD-2167A.

3 Summary

The computing environment in which military software is developed is changing rapidly, and it will continue in a state of flux until at least the year 2000. The basic issues presented in this paper are not likely to be solved quickly. Military organizations must realize that they will be most effective in influencing the character that the solutions take by working in concert with the larger, commercial software community of developers and users. It is imperative that military organizations move in the direction of developing standards for hardware and software by advancing the military computing environment in the direction of the best commercial standards. Failure to do so will mean that military organizations will not be capable of taking advantage of technological advances in computing environments as they occur.

References

1 Andrews, D. C. “CASE Users Find Implementation Trail Tough, But Following Correct Path Results in Gains.” *MIS Week*, 10, 6 (Feb 1989), 48-50.

2 Blechar, M. J. “Is it CASE or is it a Pretender? Tools Must Address All Stages of the Development Process.” *Computing Canada*, 14, 22 (Oct 1988), 44.

3 Boehm, B. “A Spiral Model of Software Development and Enhancement” *Computer*, 21, 5 (May 1998), 61.

4 Bouldin, B.M. “Out With Wimps, In With Benign Despots (Implementing CASE Tools Requires Understanding how People Learn)” *Software Magazine*, 9, 5 (April 1989), 38.

5 Brown, A. W. *Database Support for Software Engineering.* New York: Wiley, 1989.

6 Ellison, R. J., Huff, C. H., Morris, E. J., Smith, D B., & Zarrella, P. F. “Tools Within Software Engineering Practice.” *Software Engineering Institute Technical Review*, (1989), 11-20.

7 Feuche, M. “Implementing CASE? Learn From the Users.” *MIS Week*, 10, 37 (Sep 1989), 1-2.

8 Firth, R., Mosley, V., Pethia, R., Roberts, L., & Wood, W. *A Guide to the Classification and Assessment of Software Engineering Tools*. Software Engineering Institute Technical Report CMU/SEI-87-TR-10 (DTIC: ADA213968) (Aug 1987).

9 Gane, C. *Computer Aided Software Engineering.* Prentice Hall, 1988.

10 Graham, M., & Miller, D. *ISTAR Evaluation*. Software Engineering Institute Technical Report CMU/SEI-88-TR-3 (DTIC: ADA201345) (Jul 1988).

11 Grochow, J. M. “Cost Justifying CASE Requires Specific Identification.” *MIS Week*, 9, 26 (Jun 1988), 35.

12 *HP Encapsulator 1.0: Technical Data*. Hewlett Packard Co., 1989.

13 Humphrey, W. S. *CASE Planning and the Software Process*. Software Engineering Institute Technical Report CMU/SEI-89-TR-26 (DTIC: ADA219066) (May 1989).

14 Jones, C. *Software Measurement and Estimation*, DCI Seminar Notes, Boston, Mass., Aug 2-3, 1990.

15 Nelson, R. R., & Loh, M. “Reaping CASE Harvests.” *Datamation*, 35, 13 (Jul 1989), 31-33.

16 Poston, R. M. “Proposed Standard Eases Tool Interconnection.” *IEEE Software*, 6, 6 (Nov 1989), 69-70.

17 Powell, M. “The Madness in the Method.” *Computer Weekly*, 1164 (May 1989), 22-23.

18 Siegel, J., Stewman, S., Konda, K., Larkey, P.,& Wagner, W. G. *National Software Capacity: Near Term Study*. Software Engineering Institute Technical Report CMU/SEI-90-TR-12 (DTIC: ADA227564) (May 1990).

19 Smith, D. B., & Oman, P. W. “Software Tools in Context.” *IEEE Software*, 7, 3 (May 1990), 14-18.

20 *Network Software Environment: Reference Manual*. Sun Microsystems, Inc., 1988.

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION
Unclassified

1b. RESTRICTIVE MARKINGS
None

2a. SECURITY CLASSIFICATION AUTHORITY
N/A

3. DISTRIBUTION/AVAILABILITY OF REPORT
Approved for Public Release
Distribution Unlimited

2b. DECLASSIFICATION/DOWNGRADING SCHEDULE
N/A

4. PERFORMING ORGANIZATION REPORT NUMBER(S
CMU/SEI-91-TR-8

5. MONITORING ORGANIZATION REPORT NUMBER(S)
ESD-TR-91-8

6a. NAME OF PERFORMING ORGANIZATION
Software Engineering Institute

6b. OFFICE SYMBOL (if applicable)
SEI

7a. NAME OF MONITORING ORGANIZATION
SEI Joint Program Office

6c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

7b. ADDRESS (City, State and ZIP Code)
ESD/AVS
Hanscom Air Force Base, MA 01731

8a. NAME OFFUNDING/SPONSORING ORGANIZATION
SEI Joint Program Office

8b. OFFICE SYMBOL (if applicable)
ESD/AVS

9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
F1962890C0003

8c. ADDRESS (City, State and ZIP Code)
Carnegie Mellon University
Pittsburgh PA 15213

10. SOURCE OF FUNDING NOS.

PROGRAM ELEMENT NO	PROJECT NO.	TASK NO	WORK UNIT NO.
63756E	N/A	N/A	N/A

11. TITLE (Include Security Classification)
Issues in Tool Acquisition

12. PERSONAL AUTHOR(S)
Paul F. Zarrella, Dennis B. Smith, Edwin J. Morris

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Yr., Mo., Day)	15. PAGE COUNT
Final	FROM TO	September 1991	48

16. SUPPLEMENTARY NOTATION
Paul F. Zarrella, Dennis B. Smith, and Edwin J. Morris

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse of necessary and identify by block number)
FIELD	GROUP	SUB. GR.	Software development, tools, CASE

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

This technical report identifies issues involved in the acquisition of Computer Aided Software Engineering (CASE) tools. Among the issues identified and discussed are cost, performance, process support, maintenance, data management, tool integration, and standardization. The report concludes with recommendations intended for individuals or groups responsible for acquiring CASE tools.

(please turn over)

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT
UNCLASSIFIED/UNLIMITED ■ SAME AS RPT □ DTIC USERS ■

21. ABSTRACT SECURITY CLASSIFICATION
Unclassified, Unlimited Distribution

22a. NAME OF RESPONSIBLE INDIVIDUAL	22b. TELEPHONE NUMBER (Include Area Code)	22c. OFFICE SYMBOL
Charles J. Ryan, Major, USAF	(412) 268-7631	ESD/AVS (SEI)

ABSTRACT —continued from page one, block 19