FIPS PUB 99

FEDERAL INFORMATION
PROCESSING STANDARDS PUBLICATION

1983 MARCH 31

U.S. DEPARTMENT OF COMMERCE/National Bureau of Standards

GUIDELINE: A FRAMEWORK FOR THE EVALUATION AND COMPARISON OF SOFTWARE DEVELOPMENT TOOLS

ORY: SOFTWARE
TEGORY: SOFTWARE ENGINEERING

U.S. DEPARTMENT OF COMMERCE, **Malcolm Baldrige**, *Secretary*
NATIONAL BUREAU OF STANDARDS, **Ernest Ambler**, *Director*

Foreword

The Federal Information Processing Standards Publication Series of the National Bureau of Standards is the official publication relating to standards adopted and promulgated under the provisions of Public Law 89-306 (Brooks Act) and under Part 6 of Title 15, Code of Federal Regulations. These legislative and executive mandates have given the Secretary of Commerce important responsibilities for improving the utilization and management of computers and automatic data processing in the Federal Government. To carry out the Secretary's responsibilities, the NBS, through its Institute for Computer Sciences and Technology, provides leadership, technical guidance, and coordination of government efforts in the development of guidelines and standards in these areas.

Comments concerning Federal Information Processing Standards Publications are welcomed and should be addressed to the Director, Institute for Computer Sciences and Technology, National Bureau of Standards, Washington, DC 20234.

James H. Burrows, *Director*
Institute for Computer Sciences and Technology

Abstract

A framework for the evaluation and comparison of software development tools is introduced and presented. The framework is a hierarchical structure of tool features that provides the level of detail necessary to analyze and classify the capabilities of tools. Through a careful analysis of tool features, one can obtain a better understanding of the characteristics of a tool and can compare these characteristics with those of other tools.

Key words: dynamic analysis; Federal Information Processing Standards Publication; programming aids; software development; software engineering; software tools; static analysis; taxonomy.

Natl. Bur. Stand. (U.S.) Fed. Info. Process. Stand. Publ. (FIPS PUB) 99, 26 pages
(1983)
CODEN:FIPPAT

For sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161.

Federal Information
Processing Standards Publication 99

1983 March 31

ANNOUNCING THE

GUIDELINE: A FRAMEWORK FOR THE EVALUATION AND COMPARISON OF SOFTWARE DEVELOPMENT TOOLS

Federal Information Processing Standards Publications are issued by the National Bureau of Standards pursuant to the Federal Property and Administrative Services Act of 1949, as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973), and Part 6 of Title 15 Code of Federal Regulations (CFR).

Name of Guideline: A Framework for the Evaluation and Comparison of Software Development Tools.

Category of Guideline: Software; Software Engineering.

Explanation: This Guideline presents a structure that can be used as a basis for the analysis and classification of software development tools.

Approving Authority: U.S. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

Maintenance Authority: U.S. Department of Commerce, National Bureau of Standards (Institute for Computer Sciences and Technology).

Cross Index: None.

Applicability: This Guideline is a basic reference document for general use by Federal departments and agencies in the planning and acquisition of software development tools.

Implementation: This Guideline should be consulted when Federal agencies are: considering, acquiring, or implementing software development tools; developing policies and procedures for implementing and using software development tools; or reviewing the current use of software development tools.

Specifications: Federal Information Processing Standards Publication 99 (FIPS PUB 99), Guideline: A Framework for the Evaluation and Comparison of Software Development Tools (affixed).

Definitions: The following definitions apply in this document:

Software. (ISO) Computer programs, procedures, rules, and possibly associated documentation concerned with the operation of a data processing system.

Software Development Tools. Computer programs that aid the specification, construction, testing, analysis, management, documentation, or maintenance of other computer programs.

Qualifications: This Guideline represents a recommended framework for evaluating and comparing software tools; it is not offered as an unqualified recommendation. In applying this Guideline, it is important to bear in mind that software development tools are part of an emerging technology. Each Federal agency, office, or project should take into consideration their own specific circumstances when applying or referencing this publication.

Where to Obtain Copies of this Guideline: Copies of this publication are for sale by the National Technical Information Service, U.S. Department of Commerce, Springfield, VA 22161. When ordering, refer to Federal Information Processing Standards Publication 99 (FIPS PUB 99) and title. When microfiche is desired, this should be specified. Payment may be made by check, money order, or NTIS deposit account.

Federal Information
Processing Standards Publication 99

1983 March 31

Specifications for

GUIDELINE: A FRAMEWORK FOR THE EVALUATION AND COMPARISON OF SOFTWARE DEVELOPMENT TOOLS

Contents

FIGURES

TABLES

Note: Certain commercial products are identified in this Guideline for clarification of specific concepts. In no case does such identification imply recommendation or endorsement by the National Bureau of Standards, nor does it imply that the material identified is necessarily the best for the purpose.

1. PURPOSE

This Guideline is a basic reference document in evaluating and comparing software development tools. It presents a recommended framework, based on tool features, for identifying, discussing, evaluating, and comparing software development tools.

Federal departments and agencies should consult it when they are:

- acquiring or implementing software development tools;
- developing policies and procedures for acquiring, implementing, and using development tools; and
- reviewing the current use of software development tools.

Software development tools have become increasingly complex. Most early software tools performed a single function and were easy to understand. Current tools are multifunctional systems which, for commercial reasons, emphasize their uniqueness. For example, early compilers were simple tools that performed translation from high-level language to machine language. Current compilers, in addition to translating, may interpret, optimize, format, perform run-time checks, generate cross-reference tables, and provide other forms of documentation. The term "compiler" is no longer sufficient to clarify the functionality of the tool.

To compare and select existing tools, an accepted framework of individual tool features comprising more complex tools is needed. The classification scheme presented here provides such a framework. It establishes a basis for communication, understanding, and analysis of software development tools. Complex tools can be reduced to a collection of tool features. The classification scheme thus provides a basis for fundamental understanding and comparison of existing tools. The framework also presents a concise picture of software development tool technology, since it details a complete list of constituent tool features. It can be used as a checklist when considering desirable tool functionality.

The classification scheme, or framework, of software development tools presented here will be referred to in the remainder of the document as a taxonomy. To make the taxonomy easily understood, the tool features are organized into three major categories:

- Input
- Function
- Output

Most tools possess tool features in each of the major categories. Each major category is further subdivided into classes. For example, the function category has four classes: transformation, static analysis, dynamic analysis, and management. Any functional tool feature can be placed in one of the four classes. These functional classes are further subdivided into specific tool features. The details of this organization and definition of features are presented in the taxonomy itself in section 2.

The benefits of using this framework include:

- Facilitating the identification, evaluation, and selection of tools to satisfy a given set of user requirements.
- Providing a means for comparing the features of one tool with those of another.
- Fostering basic communication and understanding. The definitions of tool features in the taxonomy establish a common terminology for discussing software development tools.

The following sections expand on these benefits.

1.1 Identification

Careful identification of features establishes the essential characterization of candidate development tools and permits the determination of the specific tool most appropriate for a given application. However, feature identification requires detailed information about a tool including:

- what a tool accepts as input and how it accepts it.
- the way it manipulates and analyzes the input.
- what a tool produces as output for both the tool user and for further processing by other tools.

The taxonomy acts as a guide or checklist and provides the framework for feature identification.

Since tool developers use numerous terms to describe the capabilities of a tool, it is necessary for the tool analyst to translate these terms into features of the taxonomy. For example, the feature, "coverage analysis," has been discussed in developers' literature under the following names:

- Automated verification
- Path flow analysis
- Language analysis
- Decision-to-decision path analysis
- Segment testing
- Statement coverage
- Branch testing
- Testing metrics
- Execution monitoring
- Frequency analysis

Unfortunately, many of these terms have been used by other developers to refer to features other than "coverage analysis." Therefore, tool functionality must be clearly understood to identify tool features. The taxonomy, in addition to providing a comprehensive listing and structure of tool features, establishes a standard description for each feature and in most cases provides further references.

1.2 Comparison and Evaluation

Successful acquisition and use of tools require the completion of a number of events in which the taxonomy can play an important role. The complete event sequence is contained in Appendix A. The following procedure formalizes the use of the taxonomy to compare and evaluate candidate tools:

1. Review the taxonomy and determine the features that have the best potential for satisfying the goals and objectives of the user organization (Events 1-2, Appendix A).
2. Obtain a catalog of software tools, such as NBS Special Publication 500-88 [Houg82], the DACS Software Tools Database [DACS82], the OSD/FCTC Software Tool Catalog [FSTC82], RCI's Software Tools Directory [RCI82], or SRA's Software Engineering Automated Tool Index [SRA82].
3. Review the catalog and make a list of the tools that provide the features of interest. Using the taxonomy as a checklist, identify the additional features that are provided by these tools.
4. Obtain from either the catalog or the tool supplier descriptive characteristics about the tool that are important environmental constraints. This can include cost data, implementation languages, application languages, hardware and software requirements, availability, performance, portability, support, and available documentation.
5. List the tools having the required features and meeting the environmental constraints.

The resulting list is a set of candidate tools that can be further considered for introduction into the user's organization (Events A4-A6, Appendix A).

Much of the above procedure can be automated by using a database of information on software tools. Since features permit the comparison of software tools, one of their uses is as a key for retrieval in a database. Just as library searches are based on key words, tool searches can be based on features. For example, a user who was interested in a tool that has the feature of "coverage analysis" might specify a retrieval request as follows:

> List all tools which perform the function of coverage analysis.

A retrieval of this request might yield the following list of tools:

NBS ANALYZER	JOVIAL TCA	EAVS
PET	RXVP	DYNA
NODAL	PACE	PACE-C
ITDEM	TEST PREDICTOR	JIGSAW
COTUNE II	TATTLE	TPT
LOGIC	PDS	JAVS
FAVS	CAVS	

After examining and comparing the application languages and the additional features offered by these tools, the retrieval request could be further specified as follows:

> List all tools which accept FORTRAN programs as input and perform the functions of dynamic analysis and either auditing or data flow analysis.

This retrieval request would narrow the above list to the following tools:

PET FAVS TPT

Note that the number of candidate tools to be considered by a potential user has been greatly reduced.

1.3 Classification

Classification formalizes the identification of tool features through use of the hierarchical structure of the taxonomy. For example, a tool that has the following features:

code input;
command control;
code instrumentation;
complexity measurement;
cross reference generation;
dynamic coverage analysis;
listing output;
cross reference and coverage reports output; and
instrumented code output

would have the following taxonomy classification:

INPUT
- SUBJECT (I)
  - CODE (I3)
- CONTROL INPUT (C)
  - COMMANDS (C1)

FUNCTION
- TRANSFORMATION (T)
  - INSTRUMENTATION (T3)
- STATIC ANALYSIS (S)
  - COMPLEXITY MEASUREMENT (S3)
  - CROSS REFERENCE (S6)
- DYNAMIC ANALYSIS (D)
  - COVERAGE ANALYSIS (D3)

OUTPUT
- USER OUTPUT (U)
  - LISTINGS (U4)
  - TABLES (U5)
- MACHINE OUTPUT (M)
  - SOURCE CODE (M6)

Note that indentation is used to show the input, functional, and output features. Following each feature name is the key that is associated with each feature (see sec. 2.1 for key definitions). The classification can be abbreviated by using the keys. The abbreviation is formed by collecting the individual feature keys according to their position in the taxonomy. For example, the abbreviated classification for the above tool would be:

I3.C1/T3.S3.S6.D3/U4.U5.M6.

The slash is used to separate the input, functional, and output features and the period is used to separate individual features. It can be seen from the example that the key provides a short-hand representation for the classification.

2. THE FRAMEWORK

2.1 Taxonomy of Tool Features

Input. Tool input features are based on the forms of input provided to a tool. As shown in table 2, these features fall into two classes, one based on what the tool should operate on, i.e., the subject, and the other based on how the tool should operate, i.e., the control.

Table 1. Input

Subject		Control input	
I1.	Text	C1.	Commands
I2.	VHLL	C2.	Parameters
I2A.	Description language		
I2B.	Requirements language		
I2C.	Design language		
I3.	Code		
I4.	Date		

a. *Subject (key: I).* The subject is usually the main input to a tool. It is the input which is subjected to the main functions performed by a tool. The four types of subjects are text, VHLL (very high level language), code, and data. Although the difference between these types is somewhat arbitrary, the taxonomy has very specific definitions for each.

I1. *Text*—accepts statements in a natural language form. Certain types of tools are designed to operate on text only (e.g., text editors, document preparation systems) and require no other input except directives or commands.

I2. *VHLL*—accepts a specification written in a very high level language that is typically not in an executable form. Tools with this feature may define programs, track program requirements throughout their development, or synthesize programs through use of some nonprocedural VHLL. There are three recognized types of VHLLs. Each is briefly described as follows:

I2A. *Description language*—accepts a formal language with special constructs used to describe the subject in a high-level nonprocedural form. An example of a description language is Backus-Naur Form (BNF).

I2B. *Requirements language*—accepts a formal language with special constructs and verification protocols used to specify, verify, and document requirements. Examples of requirements languages include the Problem Statement Language [Teic77] and the Requirements Statement Language [Bell77].

I2C. *Design language*—accepts a formal language with special constructs and verification protocols used to represent, verify, and document a design. Design languages are normally procedural (i.e., they specify how a program is going to work in an algorithmic manner). An example of a design language is Program Design Language [Cain75].

I3. *Code*—accepts a program written in a high-level language, assembler, or object level language. Code is the language form in which most programming solutions are expressed.

I4. *Data*—accepts a string of characters or numeric quantities to which meaning is or might be assigned. The input (e.g., raw data) is not in an easily interpreted, natural language form. A simulator that accepts numeric data to initialize its program variables is an example of a tool that has data as input.

Some tools, such as editors, operate on any of these four input forms. In these cases, the input form is chosen from the viewpoint of the tool. Since most editors view the input form as text, the correct subject for this tool is text.

b. *Control input (key: C).* Control inputs specify the type of operation and the detail associated with an operation. They describe any separable commands that are entered as part of the input stream.

C1. *Commands*—accept character strings consisting primarily of procedural operators, each capable of invoking a system function to be executed. A directive invoking a series of diagnostic commands (i.e., TRACE, DUMP, etc.) at selected breakpoints is an example. A tool performing a single function will not have this feature but will likely have the next.

C2. *Parameters*—accept character strings consisting of identifiers that further qualify the operation to be performed by a tool. Parameters are usually entered as a result of a prompt from a tool or may be embedded in the tool input. An interactive trace routine that prompts for breakpoints is an example of a tool with parametric input.

Function. The features for this class are shown in table 2. They describe the processing functions performed by a tool and fall into four classes: transformation, static analysis, dynamic analysis, and management.

Table 2. Function

Transformation		Static analysis		Dynamic analysis		Management	
T1.	Editing	S1.	Auditing	D1.	Assertion checking	G1.	Configuration control
T2.	Formatting	S2.	Comparison	D2.	Constraint evaluation	G2.	Information management
T3.	Instrumentation	S3.	Complexity measurement	D3.	Coverage analysis	G2A.	Data dictionary management
T4.	Optimization	S4.	Completeness checking	D4.	Resource utilization	G2B.	Documentation management
T5.	Restructuring	S5.	Consistency checking	D5.	Simulation	G2C.	File management
T6.	Translation	S6.	Cross reference	D6.	Symbolic execution	G2D.	Test data management
T6A.	Assembling	S7.	Data flow analysis	D7.	Timing	G3.	Project management
T6B.	Compilation	S8.	Error checking	D8.	Tracing	G3A.	Cost estimation
T6C.	Conversion	S9.	Interface analysis	D8A.	Breakpoint control	G3B.	Resource estimation
T6D.	Macro expansion	S10.	Scanning	D8B.	Data flow tracing	G3C.	Scheduling
T6E.	Structure preprocessing	S11.	Statistical analysis	D8C.	Path flow tracing	G3D.	Tracking
T7.	Synthesis	S12.	Structure checking	D9.	Tuning		
		S13.	Type analysis	D10.	Regression testing		
		S14.	Units analysis				
		S15.	I/O specification analysis				

a. *Transformation (key: T).* Transformation features describe how the subject is manipulated to accommodate the users' needs. They describe what transformations take place as the input to the tool is processed. There are seven transformation features. Each of these features is briefly defined as follows:

T1. *Editing*—modifying the content of the input by inserting, deleting, or moving characters, numbers, or data.

T2. *Formatting*—arranging a program according to predefined or user defined conventions. A tool that "cleans up" a program by making all statement numbers sequential, alphabetizing variable declarations, indenting statements, and making other standardizing changes has this feature.

T3. *Instrumentation*—adding sensors and counters to a program for the purpose of collecting information useful for dynamic analysis [Paig74]. Most code analyzers instrument the source code at strategic points in the program to collect execution statistics required for assertion checking, coverage analysis, or tuning. See D1, D3, and D9.

T4. *Optimization*—modifying a program to improve performance, e.g., to make it run faster or to make it use fewer resources. Many vendors' compilers provide this feature. Many tools claim this feature, but do not modify the subject program. Instead, these tools provide data on the results of execution which may be used for tuning purposes. See D9.

T5. *Restructuring*—reconstructing and arranging the subject in a new form according to well-defined rules. A tool that generates structured code from unstructured code is an example of a tool with this feature.

T6. *Translation*—converting from one language form to another. There are five types of translation features. Each is defined as follows:

T6A. *Assembling*—translating a program expressed in an assembler language into object code.

T6B. *Compilation*—translating a computer program expressed in a problem-oriented language into object code.

T6C. *Conversion*—modifying an existing program to enable it to operate with similar functional capabilities in a different environment. Examples include CDC FORTRAN to IBM FORTRAN, ANSI COBOL (1968) to ANSI COBOL (1974), and Pascal to PL/1.

T6D. *Macro expansion*—augmenting instructions in a source language with user defined sequences of instructions in the same source language.

T6E. *Structure preprocessing*—translating a computer program with structured constructs into its equivalent without structured constructs.

T7. *Synthesis*—generating programs according to predefined rules from a program specification or intermediate language. Tools having this feature include program generators, compiler compilers, and preprocessor generators.

b. *Static analysis (key: S).* Static analysis features specify operations on the subject without regard to the executability of the subject [Howd78]. They describe the manner in which the subject is analyzed. There are 15 static analysis features. Each is briefly described as follows:

S1. *Auditing*—conducting an examination to determine whether or not predefined rules have been followed. A tool that examines the source code to determine whether or not coding standards are complied with is an example of a tool with this feature.

S2. *Comparison*—determining and assessing similarities between two or more items. A tool that determines changes made in one file that are not contained in another has this feature.

S3. *Complexity measurement*—determining how complicated an entity (e.g., routine, program, system, etc.) is by evaluating some associated characteristics [Mcca76] [Hals77]. For example, the following characteristics can impact complexity: instruction mix, data references, structure/control flow, number of interactions/interconnections, size, and number of computations.

S4. *Completeness checking*—assessing whether or not an entity has all its parts present and if those parts are fully developed [Boeh78]. A tool that examines the source code for missing parameter values has this feature.

S5. *Consistency checking*—determining whether or not an entity is internally consistent in the sense that it contains uniform notation and terminology [Walt78], or is consistent with its specification [Robi77]. Tools that check for consistent usage of variable names or tools that check for consistency between design specifications and code are examples of tools with this feature.

S6. *Cross reference*—referencing entities to other entities by logical means. Tools that identify all variable references in a subprogram have this feature.

S7. *Data flow analysis*—graphical analysis of the sequential patterns of definitions and references of data [Oste76]. Tools that identify undefined variables on certain paths in a program have this feature.

S8. *Error checking*—determining discrepancies, their importance, and/or their cause. Tools used to identify possible program errors, such as misspelled variable names, arrays out of bounds, and modifications of a loop index are examples of tools with this feature.

S9. *Interface analysis*—checking the interfaces between program elements for consistency and adherence to predefined rules and/or axioms. A tool that examines interfaces between modules to determine if axiomatic rules for data exchange were obeyed has this feature.

S10. *Scanning*—examining an entity sequentially to identify key areas or structure. A tool that examines source code and extracts key information for generating documentation is an example of a tool with this feature.

S11. *Statistical analysis*—performing statistical data collection and analysis. A tool that uses statistical test models to identify where programmers should concentrate their testing is one example. A tool that tallies occurrences of statement types is another example of a tool with this feature.

S12. *Structure checking*—detecting structural flaws within a program (e.g., improper loop nestings, unreferenced labels, unreachable statements, and statements with no successors).

S13. *Type analysis*—evaluating whether or not the domain of values attributed to an entity are properly and consistently defined. A tool that type checks variables has this feature.

S14. *Units analysis*—determining whether or not the units or physical dimensions attributed to an entity are properly defined and consistently used. A tool that can check a program to ensure variables used in computations have proper units (e.g., hertz=cycles/seconds) is an example of a tool with this feature.

S15. *I/O Specification analysis*—analyzing the input and output specifications in a program usually for the generation of test data. A tool that analyzes the types and ranges of data defined in an input file specification to generate an input test file is an example of a tool with this feature.

c. *Dynamic analysis (key: D).* Dynamic analysis features specify operations that are determined during or after execution [Howd78a]. Dynamic analysis features differ from those classified as static by virtue of the fact that they require some form of symbolic or machine execution. They describe the techniques used by the tool to derive meaningful information about a program's execution behavior. There are 10 dynamic analysis features. Each is briefly described as follows:

D1. *Assertion checking*—checking of user-embedded statements that assert relationships between elements of a program. An assertion is a logical expression that specifies a condition or relation among the program variables. Checking may be performed with symbolic or run-time data. Tools that test the validity of assertions as the program is executing or tools that perform formal verification of assertions have this feature.

D2. *Constraint evaluation*—generating and/or solving path input or output constraints for determining test input or for proving programs correct [Clar76]. Tools that assist in the generation of or automatically generate test data have this feature.

D3. *Coverage analysis*—determining and assessing measures associated with the invocation of program structural elements to determine the adequacy of a test run [Fair78]. Coverage analysis is useful when attempting to execute each statement, branch, path, or iterative structure (e.g., DO loops in FORTRAN) in a program. Tools that capture this data and provide reports summarizing relevant information have this feature.

D4. *Resource utilization*—analysis of resource utilization associated with system hardware or software. A tool that provides detailed run-time statistics on core usage, disk usage, queue lengths, etc., is an example of a tool with this feature.

D5. *Simulation*—representing certain features of the behavior of a physical or abstract system by means of operations performed by a computer. A tool that simulates the environment under which operational programs will run has this feature.

D6. *Symbolic execution*—reconstructing logic and computations along a program path by executing the path with symbolic rather than actual values of data [Darr78].

D7. *Timing*—reporting actual CPU, wall-clock, or other times associated with parts of a program.

D8. *Tracing*—monitoring the historical record of execution of a program. There are three types of tracing features. Each is described as follows:

D8A. *Breakpoint control*—controlling the execution of a program by specifying points (usually source instructions) where execution is to be interrupted.

D8B. *Data flow tracing*—monitoring the current state of variables in a program. Tools that dynamically detect uninitialized variables or tools that allow users to interactively retrieve and update the current values of variables have this feature.

D8C. *Path flow tracing*—recording the source statements and/or branches that are executed in a program in the order that they are executed.

D9. *Tuning*—determining what parts of a program are being executed the most. A tool that instruments a program to obtain execution frequencies of statements is a tool with this feature.

D10. *Regression testing*—rerunning test cases which a program has previously executed correctly in order to detect errors spawned by changes or corrections made during software development and maintenance. A tool that automatically "drives" the execution of programs through their input test data and reports discrepancies between the current and prior output is an example of a tool with this feature.

d. *Management (key: G).* Management features aid the management or control of software development. There are three types of management features. Each is described as follows:

G1. *Configuration control*—aiding the establishment of baselines for configuration items, the control of changes to these baselines, and the control of releases to the operational environment.

G2. *Information management*—aiding the organization, accessibility, modification, dissemination, and processing of information that is associated with the development of a software system.

G2A. *Data dictionary management*—aiding the development and control of a list of the names, lengths, representations, and definitions of all data elements used in a software system.

G2B. *Documentation management*—aiding the development and control of software documentation.

G2C. *File management*—providing and controlling access to files associated with the development of software.

G2D. *Test data management*—aiding the development and control of software test data.

G3. *Project management*—aiding the management of a software development project. Tools that have this feature commonly provide milestone charts, personnel schedules, and activity diagrams as output.

G3A. *Cost estimation*—assessing the behavior of the variables which impact life cycle cost. A tool to estimate project cost and investigate its sensitivity to parameter changes has this feature.

G3B. *Resource estimation*—estimating the resources attributed to an entity. Tools that estimate whether or not memory limits, input/output capacity, or throughput constraints are being exceeded have this feature.

G3C. *Scheduling*—assessing the schedule attributed to an entity. A tool that examines the project schedule to determine its critical path (shortest time to complete) has this feature.

G3D. *Tracking*—tracking the development of an entity through the software life cycle. Tools used to trace requirements from their specification to their implementation in code have this feature.

Output. Output features, which provide the link from the tool to the user, are illustrated in table 3. They describe what type of output the tool produces for both the human user and the target machine (where applicable). Again using a compiler as an example, the user output would be diagnostics and possibly listings and tables (cross reference), and the machine output would be object code or possibly intermediate code.

Table 3. Output

User output	Machine output
U1. Computational results	M1. Assembly language
U2. Diagnostics	M2. Data
U3. Graphics	M3. Intermediate code
U3A. Activity diagrams	M4. Object code
U3B. Data flow diagrams	M5. Prompts
U3C. Design charts	M6. Source code
U3D. Histograms	M7. Text
U3E. Milestone charts	M8. VHLL
U3F. Program flow charts	
U3G. Tree diagrams	
U4. Listings	
U5. Tables	
U6. Text	

a. *User output (key: U).* User output features describe the types of information that are returned from the tool to the human user and the forms in which these outputs are presented. There are six user output features. Each is briefly described as follows:

U1. *Computational results*—output that simply presents the result of a computation. The output is not in an easily interpreted natural language form (e.g., text or tables).

U2. *Diagnostics*—output that simply indicates what software discrepancies have occurred. An error flag from a compiler is an example.

U3. *Graphics*—a graphical representation with symbols indicating operations, flow, etc. There are seven types of graphics output, each described as follows:

U3A. *Activity diagrams*—diagrams presenting actions or states of a software development activity and their interrelationships. Also, diagrams representing or summarizing the major aspects of system performance. Examples of activity diagrams are control diagrams, pert charts, and system profiles.

U3B. *Data flow diagrams*—diagrams that represent the path of data in the solving of a problem and that define the major phases of the processing as well as the various data media used. Examples of data flow diagrams are Jackson diagrams [Jack75], DFD's [DeMa78], and Bubble Charts [Your75].

U3C. *Design charts*—charts which represent the software architecture components, modules, and interfaces for a software system. Examples of design charts are HIPO diagrams, structure charts, and block diagrams.

U3D. *Histograms*—graphic representations of frequency distributions where the graph height is proportional to a class frequency. Tools that perform statistical analysis or coverage analysis often provide histograms.

U3E. *Milestone charts*—a chart which represents the schedule of events used to measure the progress of software development efforts. Milestone charts are often provided by project management tools.

U3F. *Program flow charts*—graphical representation of the sequence of operations in a computer program. Examples of program flow charts include FIPS Flow Charts [FIPS72], Chapin Charts [Chap74], and Nassi-Shneiderman Charts [Nass73].

U3G. *Tree diagrams*—diagrams that represent the hierarchical structure of software modules or data and are generated from a root node. A tree diagram does not show iteration or cycles. An example of a tree diagram is a "call tree" or module hierarchical diagram.

U4. *Listings*—an output from the tool is a computer listing of a source program or data and may be annotated. Many different forms of listings can be generated. Some may be user controlled through directives.

U5. *Tables*—an output from the tool is arranged in parallel columns to exhibit a set of facts or relations in a definite, compact, and comprehensive form. A tool that produces a decision table identifying a program's logic (conditions, actions, and rules that are the basis of decisions) is an example.

U6. *Text*—an output from the tool is in a natural language form. The output may be a choice of many different types of reports and the formats may be user defined.

b. *Machine output (key: M).* Machine output features handle the interface from the tool to either another tool or a machine environment. They describe what a machine or tool expects to see as output. There are eight machine output features. Each is briefly described as follows:

M1. *Assembly code*—a low level code whose instructions are usually in one-to-one correspondence with computer instructions.

M2. *Data*—a set of representations of characters or numeric quantities to which meaning has been assigned. A tool generating input to a plotter is an example.

M3. *Intermediate code*—a code between source code and assembly code. A tool producing P-code for direct machine interpretation is an example.

M4. *Object code*—a code expressed in machine language which is normally an output of a given translation process. A tool producing relocatable load modules for subsequent execution is an example.

M5. *Prompts*—a series of procedural operators used to interactively inform the system in which the tool operates that it is ready for the next input.

M6. *Source code*—a code written in a high level procedural language that must be input to a translation process before execution can take place.

M7. *Text*—statements in a natural language form. A tool producing English text which is passed to a word processor is an example.

M8. *VHLL*—statements written in a very high level language. A tool which produces a requirements language or design language for use by another tool is an example.

2.2 Expansion Issues

Although it would be structurally satisfying to have all features expanded to the same level (it would keep the taxonomy looking like a trimmed pine), there are many features which represent relatively new technology that make this task impossible. These areas are maturing and are not yet well understood. For example, features such as synthesis, restructuring, complexity measurement, cost estimation, regression testing, and constraint evaluation are the subject of current research. Expansion of these features to the same level as features such as translation, tracing, or project management is premature.

Another reason for expanding to a lower level is to keep the granularity of the taxonomy consistent at the bottom level. For example, translation and instrumentation are both transformation features at the fourth level. Translation, however is broader in scope than instrumentation. Consequently, translation is further expanded to keep the granularity of its features consistent with other transformation features.

2.3 Missing Features

On occasion, a tool may have a characteristic that can not be easily classified with the taxonomy. It may be that the taxonomy is missing a feature. When this situation occurs, the classification for this tool remains at the lowest level possible. For example, it has been proposed that test generation and query languages be added to VHLL Input. Since these languages are not currently part of the taxonomy, the classification would not proceed below VHLL Input. Test generation and query languages will be reviewed for need, desirability, and appropriateness in future revisions of the taxonomy.

All occurrences of missing features are important to ICST. The taxonomy reflects tool technology at the time of revision. New features and new tools may require future changes to the taxonomy to keep it up to date. Anyone with comments or problems relating to the taxonomy is encouraged to forward them to: The Institute for Computer Sciences and Technology, National Bureau of Standards, Technology Bldg., Room B266, Washington, DC 20234.

3. REFERENCES

|Bel177| Bell, T. E.; Bixler, D. C.; Dyer, M. E. An extendable approach to computer-aided software requirements engineering. IEEE Transactions on Software Engineering, Vol. SE-3, No. 1; 1977.

|Boeh78| Boehm, B. W.; Brown, J. R.; Kaspar, H.; Lipow, M.; MacLeod, G. J.; Merritt, M. J. *Characteristics of Software Quality.* New York: North-Holland Publishing Company; 1978.

|Cain75| Caine, S. H.; Gordon, E. K. PDL: A tool for software design. Proceedings of the National Computer Conference; 1975.

|Chap74| Chapin, N. New format for flowcharts. Software—Practice and Experience; 1974 Oct.-Dec.

|Clar76| Clarke, L. A. A system to generate test data and symbolically execute programs. IEEE Transactions on Software Engineering, Vol. SE-2; September 1976.

|DACS82| ----------. Software tools—custom searches. DACS Newsletter, Data & Analysis Center for Software, Griffiss AFB, NY; 1982 September.

|Darr78| Darringer, J. A.; King, J. C. Applications of symbolic execution to program testing. Computer; 1978 April.

|DeMa78| DeMarco, T. *Structured Analysis and System Specification.* Prentice-Hall, Inc.; 1978.

|Fair78| Fairley, R. E. Tutorial: Static analysis and dynamic testing of computer software. Computer; 1978 April.

|FIPS72| U.S. Dept. of Commerce. Flowchart symbols and their usage in information processing. Natl. Bur. Stand. (U.S.) Fed. Info. Process. Stand. (FIPS PUB) 24; 1972 June.

|FSTC82| ----------. Software tool catalog. Federal Software Testing Center, Report No. OSD 82-013, Falls Church, VA; 1982 April.

|Hals77| Halstead, M. H. *Elements of Software Science.* New York: Elsevier-North Holland Publ. Co.; 1977.

|Hech82| Hecht, H. The introduction of software tools. Natl. Bur. Stand. (U.S.) Spec. Publ. 500-91; 1982 September.

[Houg81] Houghton, R., ed. NBS/IEEE/ACM software tool fair. Natl. Bur. Stand. (U.S.) Spec. Publ. 500-80; 1981 October.

[Houg82] Houghton, R. Software development tools. Natl. Bur. Stand. (U.S.) Spec. Publ. 500-88; 1982 March.

[Howd78] Howden, W. E. A survey of static analysis methods. Tutorial: Software Testing and Validation Techniques, IEEE Cat. No. EHO138-8; 1978.

[Howd78a] Howden, W. E. A survey of dynamic analysis methods. Tutorial: Software Testing and Validation Techniques, IEEE Cat. No. EHO138-8; 1978.

[Jack75] Jackson, M. A. *Principle of Program Design.* Academic Press; 1975.

[Mcca76] McCabe, T. J. A complexity measure, IEEE Transactions on Software Engineering, Vol. SE-2; 1976 December.

[Nass73] Nassi, I.; Shneiderman, B. Flowchart techniques for structured programming. SIGPLAN Notices of the ACM; 1973 August.

[OSD82] ----------. A software tools project: A means of capturing technology and improving engineering. Office of Software Development, Report OSD-82-101, GSA; 1982 February.

[Oste76] Osterweil, L. J.; Fosdick, L. D. DAVE—a validation error detection and documentation system for FORTRAN programs. Software—Practice and Experience; 1976 October.

[Paig74] Paige, M. R.; Benson, J. P. The use of software probes in testing FORTRAN programs. Computer; 1974 July.

[RCI82] ----------. *Software Tools Directory.* Torrance, CA: Reifer Consultants, Inc.; 1982.

[Robi77] Robinson, L.; Levitt, K. N. Proof techniques for hierarchically structured programs. Communications of the ACM; 1977 April.

[SRA82] ----------. *Software Engineering Automated Tool Index.* San Francisco, CA: Software Research Associates; 1982.

[Teic77] Teichroew, D.; Hershey III, E. PSL/PSA: A computer-aided technique for structured documentation of information processing systems. IEEE Transactions on Software Engineering, Vol. SE-3, No. 1; 1977.

[Walt78] Walters, G.; McCall, J. The development of metrics for software reliability and maintainability. Proceedings of the Annual Reliability and Maintainability Symposium; 1978 January.

[Your75] Yourdon, E.; Constantine, L. *Structured Design,* New York: Yourdon Press; 1975.

NBS Special Publications can be ordered from the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402.

APPENDIX A
EVENT SEQUENCES FOR THE ACQUISITION OF TOOLS

A.1 Purpose of Event Sequences

The management of any significant project requires that the work be divided into tasks for which completion criteria can be defined. The transition from one task to another is called an event and to permit orderly progress of the activities, here the introduction of a software tool, the scheduling of these events must be determined in advance. A general outline for such a schedule is provided by the event sequence described in the next section. The actual calendar time schedule will depend on many factors to be determined for each specific tool use (particularly on the time required for procurement of the tool and training). One of the formats used for the event sequence is consistent with the critical path method (CPM) of project scheduling and can be used with that technique for the development of an optimum calendar time schedule.

Most of the activities included in the event sequence are obviously necessary but a few were included specifically to avoid difficulties encountered in tool procurements. Quite frequently tools are obtained 'through the side door' without adequate consideration of the resources required for the effective employment of the tool and without determination by a responsible manager that the tool serves a primary need of the organization. Tools acquired in this manner are seldom used in an optimal way and are sometimes discarded. Experiences of this type are not conducive to gaining widespread acceptance of tools in programming environments where the activities required for the introduction of tools will impose a drain on resources. A key feature of the proposed approach is, therefore, that tool usage will be initiated only in response to an expressed management goal for software development or for the entire computing function.

Difficulties in the introduction of tools can arise in three areas:

- organizational obstacles
- problems arising from the tools
- obstacles in the computer environment

The individual activities described below as well as the ordering of the event sequence are designed to eliminate as many of these difficulties as possible. They are most effective with regard to the first category and probably least effective with regard to the last category. The need for involving a reponsible management level in the tool introduction has already been mentioned, and this is indeed the key provision for avoiding organizational obstacles. "Responsible management" is that level that has the authority to obligate the resources required for the introduction process. The scope of the resource requirement will become clearer after all introduction activities have been described. Because the criterion for the selection of the management focus is its ability to commit funds, this management level is hereafter referred to as funding management. In some organizations this may be the project management, in some it may be functional management, and in yet others it may be an agency or department management not specifically identified with a computing function. It should be involved in at least the following activities associated with the introduction of tools:

1. Identifying the goals to be met by the tool (or by the technique supported by the tool), and assigning responsibility for the activities required to meet these goals.
2. Approving a detailed tool acquisition plan that defines the resource requirements for procurement and in-house activities.
3. Approving the procurement of tools and training if this is not explicit in the approval of the acquisition plan.
4. Determining after some period of tool use whether the goals have been met.

Additional organizational obstacles must be overcome by actions of the software management (local management of the organization that will introduce the tool). A pitfall to be avoided is assigning the details of the tool acquisition as a sideline to an individual who carries many other responsibilities. Even in a small software organization (up to 14 programmers), it should be possible to make the tool introduction the principal assignment of an experienced individual with adequate professional background. This person is referred to as the software engineer. In medium size organizations (15 to 39 programmers), several

individuals may be involved in software engineering tasks (not restricted to tool usage), and this may constitute a software engineering function.

Further, the event sequence includes activities of a toolsmith who, in most cases, will not be the same person as the software engineer. The former assignment requires expertise in systems programming and specialized knowledge of the tool to be introduced. The duties of the software engineer involve planning, project management and obtaining cooperation from a variety of individuals and organizations. Where there is a software engineering function, the toolsmith is typically a member of it.

Obstacles arising from the tools themselves are expected to be avoided in the event sequence by a careful, methodical selection of tools. In particular, distinct contributions to the tool selection are specified for software management and the software engineer. Software management is assigned responsibility for:

- identifying tool objectives;
- approving the acquisition plan (it may also require approval by funding management);
- defining selection criteria; and
- making the final selection of the tool or the source.

The software engineer is responsible for:

- identifying candidate tools;
- applying the selection criteria (in informal procurement) or preparing RFP inputs (in formal procurement); and
- preparing a ranked list of tools or sources.

Further, the ultimate user of the tool is involved in the recommended event sequence in reviewing either the list of candidate tools or, for formal procurement, the tool requirements.

This distribution of responsibilities reduces the chances of selecting a tool that (1) does not meet the recognized needs of the organization, (2) is difficult to use, (3) requires excessive computer resources, or (4) lacks adequate documentation. The repeated exchange of information required by the process outlined above will also avoid undue emphasis on very short-term objectives which may lead to selection of a tool on the basis of availability rather than suitability.

The obstacles to tool usage that reside in the computer environment are primarily due to the great diversity of computer architectures and operating system procedures, and to the lack of portability in most software tools. Activities associated with the introduction of tools can only modestly alleviate these difficulties. The event sequence provides the following help in this area:

1. A methodical process of identifying candidate tools and selecting among these on the basis of established criteria. This will avoid some of the worst pitfalls associated with "borrowing" a tool from an acquaintance or procuring one from the most accessible or persuasive tool vendor.
2. The assignment and training of a toolsmith who can make minor modifications to both the computer environment and the tool. This is expected to provide relief where there are version-related or release-related incompatibilities with the operating system, or where the memory requirements of the tool exceed the capabilities of the installation. In the latter case, remedies may be provided by removing tool options or by structuring the tool program into overlays.

The event sequence described below is conceived as a procedure generally applicable to the introduction of tools to Federal agencies falling into pertinent programming environment categories. For this reason, a systematic reporting of the experience with the introduction process as well as with the tool is desirable. The evaluation plan and the evaluation report specified in the event sequence support these goals.

A.2 Recommended Event Sequence

The event sequence described in this subsection is applicable to both business and scientific programming environments. The general scope of the introduction activities and their sequence are identical for the two environments. Because of differences in tool requirements, personnel qualifications, and organizational structure, some differences in the content of the individual events will be expected. The event sequence addresses only the introduction of existing tools. Where a newly developed tool is introduced, a considerable modification of the activities and their sequence will be necessary.

The recommended event sequence allows for two procurement methods: informal procurement (e.g., by purchase order) or formal procurement (by request for bids). Obviously, the latter is much more time consuming but it may lead to the procurement of better or cheaper tools. Acquisition of tools from the General Services Administration or from other Government agencies should follow the informal

procurement steps even when there is no procedural requirement for this. As mentioned above, tool acquisitions which do not obtain the concurrence of all affected operational elements frequently do not achieve their objectives.

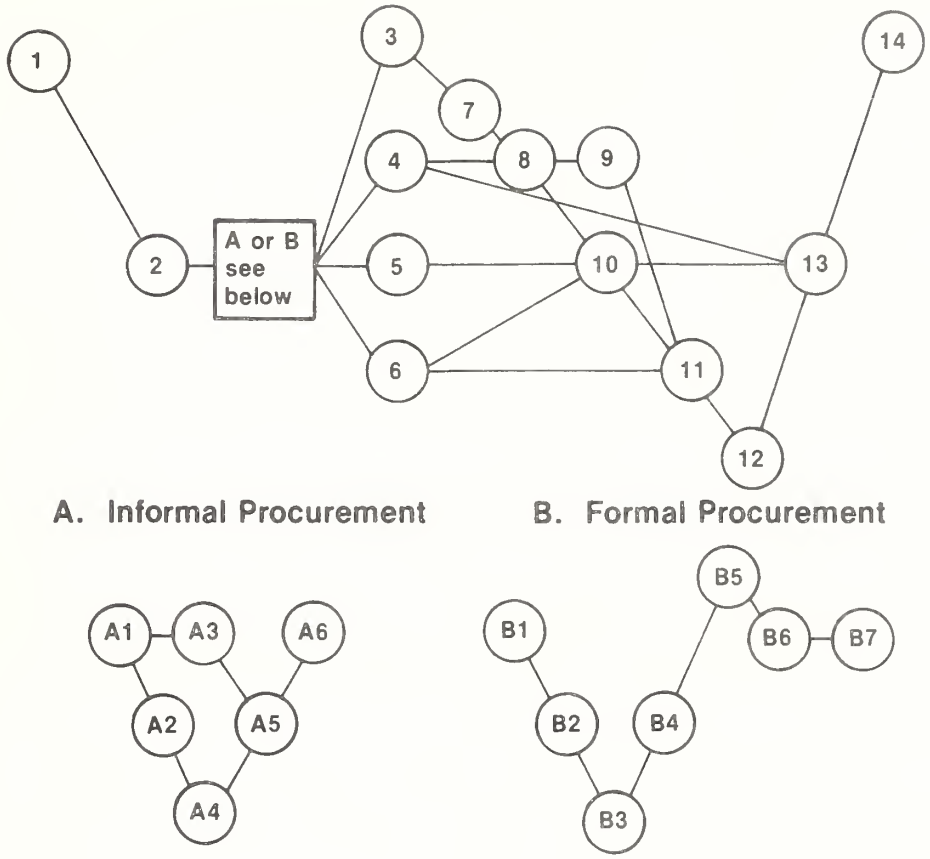
The presentation of the event sequence in table A.1 is tailored to tools which are being introduced for the first time into a user community which shares software support information (e.g., a Federal agency or a private sector company). As a result, some steps are shown which can be combined or eliminated where less formal control is exercised or where plans or modifications required for the introduction of a tool are available from a prior user. The event sequence is intended to cover a wide range of applications, and it was constructed with the thought that it is easier for the tool user to eliminate steps than to be confronted with the need for adding some that had not been covered in this volume.

The key functions which contribute to the introduction of tools are listed across the top of table A.1, and events for which each function is responsible are listed in the column under it. The preferred order of tasks for each function can thus be directly found from this table. The precedence relationships between events is shown in graph form in figure A.1. This figure will be found particularly helpful for scheduling activities by the critical path method and for the general development of a project schedule. The numbering of events is the same in table A.1 and figure A.1. A detailed description of each of the numbered events, and of the activities associated with it, is presented after the table and figure.

Table A.1. Event sequence for tool introduction

Funding management	Software management	Software engineer	Tool user
1. Goals			
	2. Tool objectives		
← Acquisition, see A or B below →			
3. Procure tool	A ←	4. Evaluation plan	
	A ←	5. Toolsmithing plan-S	
	A ←	6. Training plan	participates
7. Receive tool			
		8. Acceptance test	
	9. Orientation		
		10. Modifications-S	
		← 11. Training →	
			12. Use
A ←	A ←	13. Evaluation report	
14. Goals met?			
	A. Acquisition Activities for Informal Procurement		
A ←	A1. Acquisition plan		
	A2. Select'n criteria		
		A3. Ident. candidates	A4. Review
		A5. Score candidates	
	A6. Select tool		
	continue with step 3 above.		
	B. Acquisition Activities for Formal Procurement		
A ←	B1. Acquisition plan		
		B2. Technical req'mts	B3. Review
A ←	A ←	B4. Generate RFP	
B5. Issue RFP		B6. Proposal Evaluation	
	B7. Select source		
	continue with step 3 above.		

A = Approval required S = Toolsmith responsibility



Events

1. Goals
2. Tool objectives
3. Procure tool
4. Evaluation plan
5. Toolsmithing plan
6. Training plan
7. Receive tool
8. Acceptance test
9. Orientation
10. Modifications
11. Training
12. Use
13. Evaluation report
14. Goals met?

A1. Acquisition plan
A2. Selection criteria
A3. Identify candidates
A4. User review
A5. Score candidates
A6. Select tool

B1. Acquisition plan
B2. Technical requirements
B3. User review
B4. Generate RFP
B5. Issue RFP
B6. Proposal evaluation
B7. Select source

Figure A.1. Precedence relation for event sequence

1. Goals

The goals to be accomplished should be identified in a format that will later permit determination (event 14) that they have been met. Typical goal statements are: reduce average processing time of COBOL programs by one-fifth, achieve complete interchangeability of programs or data sets with organization Y, and adhere to an established standard for documentation format.

The statement of goals shall also identify responsibilities, in particular the role that headquarters staff organization may have and coordination requirements with other organizations. Where a decentralized management method is employed, the statement of goals may have associated with it a not-to-exceed budget and a desired completion date. Once these constraints are specified, funding management may delegate the approval of the acquisition plan to a lower level.

2. Tool Objectives

The goals generated in event 1 are translated into desired tool features and requirements arising from the development and operating environment are identified. Constraints on tool cost and availability may also be added at this event. A typical statement of tool objectives for a program formatter is: Provide header identification, uniform indentation, and the facility of printing listing and comments separately for all FORTRAN X3.9-1978 and ABC Extended FORTRAN programs. Program must run on our ABC computer under XOSnn. Only tools which have been in commercial use for at least 1 year and at no less than *N* different sites shall be considered.

At this point the sequence continues with either A1 or B1 below

A. Acquisition Activities for Informal Procurement

A1. Acquisition Plan

The acquisition plan communicates the actions of software management both upward and downward. The plan may also be combined with the statement of the tool objectives (event 2). The acquisition plan should include the budgets and schedules for subsequent steps in the tool introduction, a justification of resource requirements in the light of expected benefits, contributions to the introduction expected from other organizations (e.g., the tool itself, modification patches, or training materials), and the assignment of responsibility for subsequent events within the software organization, particularly the identification of the software engineer. Minimum tool documentation requirements shall also be specified in the plan.

A2. Selection Criteria

The criteria shall include a ranked or weighted listing of attributes that will support effective utilization of the tool by the user. Typical selection criteria are:

- accomplishment of specified tool objectives
- ease of use
- ease of installation
- minimum processing time
- compatibility with other tools
- low purchase or lease cost

Most of these criteria need to be factored further to permit objective evaluation, but this step may be left up to the individual who does the scoring. Together with the criteria (most of which will normally be capable of a scalar evaluation), constraints which have been imposed by the preceding events or are generated at this step should be summarized.

A3. Identify Candidate Tools

This is the first event for which the software engineer is responsible. The starting point for preparing a listing of candidate tools is a comprehensive tool catalog, such as [Houg82].* A desirable but not mandatory practice is to prepare two lists. The first does not consider the constraints and contains all tools meeting the functional requirements. The cross-reference by tool features in the appendices of [Houg82] will be found particularly valuable in generating this list of candidates. For the program formatting tool used in event 2, 16 entries are found. Some of these may be eliminated by further review of their description in the body of the catalog (e.g., because they do not process the specified FORTRAN dialects). For the remaining viable candidates, literature should be requested from the developer and examined for conformance with the given constraints. At this point a second list is generated, containing tools meeting both the functional requirements and the constraints. If this list does not have an adequate number of entries, relaxation of some constraints will have to be considered.

A4. User Review of Candidates

The user reviews the list of candidate tools prepared by the software engineer. Because few users can be expected to be very knowledgeable in the software tools area, specific questions may need to be asked by software management such as: "Will this tool handle the present file format? Are tool commands consistent with those of the editor? How much training will be required?" Adequate time should be budgeted for this review and a due date for responses should be indicated. Because the user views this as a long-term task and of lower priority than many immediate obligations, considerable follow-up by line management will be required. If tools can be obtained for trial use, or if a demonstration at another facility can be arranged, this step will be much more significant.

* See references in section 3 of this Guideline.

A5. Score Candidates

For each of the criteria previously identified, a numerical score is generated on the basis of information obtained from vendor's literature, from demonstration of the tool, from the user's review, from observation in a working environment, or from comments of prior users. If weighting factors for the criteria are specified, the score for each criterion is multiplied by the appropriate factor and the sum of the products represents the overall tool score. Where only a ranking of the criteria was provided, the outcome of the scoring may be simply a ranking of each candidate under each of the criteria headings. Frequently a single tool is recognized as clearly superior in this process.

A6. Select Tool

This decision is reserved for software management to provide review of the scoring and to permit additional factors not expressed in the criteria to be taken into consideration. For example, a report just received from another agency might indicate that the selected vendor did not provide adequate service. If the selected tool was not scored highest, the software engineer should have an opportunity to review the tool characteristics thoroughly to avoid unexpected installation difficulties. The selection concludes the separate sequence for informal procurement. Continue with event 3.

B. Acquisition Activities for Formal Procurement

B1. Acquisition Plan

The plan generated here must include all elements mentioned under A1 plus the constraints on the procurement process (e.g., set aside for high labor surplus areas) and the detailed responsibilities for all procurement documents (statement of work, technical and administrative provisions in the Request for Proposal, etc.).

B2. Technical Requirements Document

The technical requirements document is an informal description of the tool requirements and the constraints under which the tool has to operate. It will utilize much of the material from the acquisition plan but should add enough detail to support a meaningful review by the tool user.

B3. User Review of Requirements

The user reviews the technical requirements for the proposed procurement. As in event A4, the user may need to be prompted with pertinent questions, and there should be close management follow-up in order to get a timely response.

B4. RFP Generation

From the technical requirements document and the user comments on it, the technical portions of the RFP can be generated. Usually these include:

1. *A specification of the tool as delivered.* This should include applicable documents, a definition of the operating environment, and the quality assurance provisions.
2. *A statement of work governing the tool procurement.* This should state any applicable standards for the process by which the tool is generated (e.g., configuration management of the tool), and documentation or test reports to be furnished with the tool. Training and operational support requirements are also identified in the statement of work.
3. *Proposal evaluation criteria and format requirements.* Evaluation criteria are listed in the approximate order of importance. Subfactors for each may be identified. Restrictions on proposal format (major headings, page count, desired sample outputs) may also be included.

B5. Solicitation of Proposals

This activity is carried out by administrative personnel. Capability lists of potential sources are maintained by most purchasing organizations. Where the software organization knows of potential bidders, their names should be given to the procurement office. When responses are received, they are screened for compliance with major legal provisions of the RFP.

B6. Technical Evaluation

Each of the proposals received in response to the RFP is evaluated against the criteria previously established. Failure to meet major technical requirements can lead to outright disqualification of a proposal. Those deemed to be in "the competitive range" will be assigned point scores that will then be used together with cost and schedule factors separately evaluated by administrative personnel.

B7. Source Selection

On the basis of the combined cost, schedule, and technical factors, a source for the tool is selected. If this was not the highest rated technical proposal, prudent management will require additional reviews by software management and the software engineer to determine that it is indeed acceptable.

The source selection concludes the separate sequence for formal procurement. Continue with event 3.

3. Procure Tool

In addition to determining that the cost of the selected tool is within the approved budget, the procurement process will also consider the adequacy of licensing and other contractual provisions and compliance with the "fine print" associated with all Government procurements. The vendor's responsibility for furnishing the source program, for meeting specific test and performance requirements, and for tool maintenance need to be identified. In informal procurement, a period of trial use may be considered if this had not already taken place under one of the previous events.

If the acquisition plan indicates the need for outside training, the ability of the vendor to supply the training and the cost advantages from combined procurement of tool and training should be investigated. If substantial savings can be realized through simultaneous purchase of tool and training, procurement may be held up until outside training requirements are defined (event 6).

4. Evaluation Plan

The evaluation plan is based on the goals identified in event 1 and the tool objectives derived from these in event 2. It describes how the attainment of these objectives is to be evaluated for the specific tool selected. Typical items to be covered in the plan are milestones for installation, dates, and performance levels for the initial operational capability and for subsequent enhancements. Where improvements in throughput, response time, or turn-around time are expected, the reports from which these data are to be obtained should be identified. Responsibility for tests, reports, and other actions should be assigned in the plan. A topical outline of the evaluation report should be included.

The procedure for the acceptance test is a part of the evaluation plan, although in a major tool procurement it may be a separate document. It lists the detailed steps necessary to test the tool in accordance with the procurement provisions when it is received, to evaluate the interaction of the tool with the computer environment (e.g., adverse effects on throughput), and for generating an acceptance report.

5. Toolsmithing Plan

The plan will describe the selection of the toolsmith, the responsibilities for the adaptation of the tool, and the training which will be required. The toolsmith should preferably be an experienced system programmer, familiar with the current operating system. Training in the operation and installation of the selected tool in the form of review of documentation, visits to current users of the tool, or training by the vendor must be arranged. The toolsmithing plan is listed here as an event for which the software engineer is responsible, and in the discussion of further events it is assumed that the toolsmith will work under the direction of the software engineer. The toolsmithing plan should be approved by software management.

6. Training Plan

The training plan should first consider the training inherently provided with the tool, e.g., documentation, test cases, on-line diagnostics, HELP. These features may be supplemented by standard training aids supplied by the vendor for in-house training such as audio or video cassettes and lecturers. Because of the expense, training sessions at other locations should be considered only where none of the previous categories is available. The number of personnel to receive formal training should also be specified in the plan and adequacy of in-house facilities (number of terminals, computer time, etc.) should be addressed. If training by the tool vendor is desired, this should be identified as soon as possible to permit training to be procured with the tool (see step 3). User involvement in the preparation of the training plan is highly desirable and coordination with the user is considered essential. The training plan is normally prepared by the software engineer and approved by software management. Portions of the plan should be furnished to procurement staff if outside personnel or facilities are to be utilized.

7. Tool Received

The tool is turned over by the procuring organization to the software engineer.

8. Acceptance Test

The software engineer or staff test the tool. This is done as much as possible in an "as received" condition making only those modifications that are essential for bringing it up on the host computer. A report on the test is issued. Approval by software management constitutes official acceptance of the tool.

9. Orientation

When it has been determined that the tool has been received in a satisfactory condition, software management holds an orientation meeting for all personnel involved in the use of the tool and tool products (reports or listings generated by the tool). The main purpose is to communicate as directly as possible objectives of the tool use (such as increased throughput or improved legibility of listings). Highlights of the evaluation plan should also be presented and any changes in duties associated with the introduction of the tool should be described. Personnel should be reassured that allowance will be made for problems encountered during the introduction and that the full benefits of the tool may not be felt for some time.

10. Modifications

This step is carried out by the toolsmith in accordance with the approved toolsmithing plan. It includes modifications of the tool itself, of the documentation, and of the operating system. In rare cases some modification of the computer proper may also be necessary (channel assignments, etc.). Typical tool modifications involve deletion of unused options, changes in prompts or diagnostics, and other adaptations made for efficient use in the prevailing environment. Documentation of the modifications is an essential part of this event.

Vendor literature for the tool is reviewed in detail and is tailored for the prevailing computer environment and for the tool modifications which have been made. Deleting sections which are not applicable can be just as useful as adding material that is required for the specific programming environment. Unused options shall be clearly marked or removed from the manuals. If there is some resident software for which the tool should not be used (e.g., because of language incompatibility or conflicts in the operating system interface), warning notices should be inserted into the tool manual.

11. Training

Training is a joint responsibility of the software engineer and the tool user. The former is responsible for the content (in accordance with the approved training plan); the latter should have control over the length and scheduling of sessions. Training is an excellent opportunity to motivate the user to utilize the tool. The tool user should have the privilege of terminating steps in the training that are not helpful and of extending portions that are helpful but in which greater depth is desired. Training is not a one-time activity. Retraining or training in the use of additional options after the introductory period is desirable and provides an opportunity for users to talk about problems with the tool.

12. Use in the Operating Environment

The first use of the tool in the operational environment should involve the most qualified user personnel and minimal use of options. The first use should not be on a project with tight schedule constraints. Any difficulties resulting from this use must be resolved before expanded service is initiated. If the first use is successful, then use by additional personnel and use of further options may commence.

User comments on training, first use of the tool, and use of extended capabilities are prepared and furnished to the software engineer. Desired improvements in the user interface, speed or format of response, and in utilization of computer resources are appropriate topics. Formal comments may be solicited shortly after the initial use, after 6 months, and again after 1 year.

13. Evaluation Report

The software engineer prepares the evaluation report, using the outline generated in event 4. The user comments and observations of the toolsmith form important inputs to this document. Most of all, it must discuss how the general goals and the tool objectives were met. The report may include, observations on the installation and use of the tool, cooperation received from the vendor in installation or training, and any other "lessons learned." It may contain a section of comments useful to future users of the tool. Tool and host computer modifications shall also be described in the report. The report is approved by software management and preferably also by funding management.

14. Determine If Goals Are Met

Funding management receives the evaluation report and determines whether the goals established in event 1 have been met. This determination shall be in writing and should include:

- attainment of technical objectives
- adherence to budget and other resource constraints
- timeliness of the effort
- cooperation from other agencies
- recommendations for future tool acquisitions

APPENDIX B
TAXONOMY BACKGROUND INFORMATION

Appendix B includes background information on the taxonomy including a brief history of its development, the most recent updates, and references.

B.1 History

The initial development of the taxonomy was performed under contract* to the National Bureau of Standards. Since its initial development, the taxonomy has evolved through in-house and public review. The first public review of the taxonomy took place at a workshop held at NBS on 11 April 1980. Continued development by ICST has clarified the definitions and removed several inconsistencies. The taxonomy was released for general public review through two NBS reports** [Houg81a] [ICST81] and two conference papers** [Houg81b] [Reif81a].

The workshop and the response to the publications led to the formation of a review group composed of people from industry, Government, and academe. Comments from this group plus adjustments made as a result of in-house classification experience resulted in the taxonomy reported in this Guideline. This revision was distributed to the review group on 18 July 1981. The comments received with discussion, resolution, and summary statistics for each proposed change are available from the Institute for Computer Sciences and Technology, National Bureau of Standards, Room B266, Technology Bldg., Washington, DC 20234.

B.2 Changes from SP 500-74

The taxonomy reported in section 2 of this publication has evolved, through use, from the taxonomy presented in NBS Special Publication 500-74 [Houg81a]. To enhance the usefulness of the taxonomy, several areas have been expanded and several new features have been added. The following list summarizes the changes:

- Expansions
  - VHLL to description language, requirements language, and design language.
  - Translation to compilation, conversion, macro expansion, and structure preprocessing.
  - Management to configuration control, information management, and project management.
  - Information management to data dictionary management, documentation management, files management, and test data management.
  - Project management to cost estimation, resource estimation, scheduling, and tracking.
  - Tracing to breakpoint control, data flow tracing, and path flow tracing.
  - Graphics to activity diagrams, data flow diagrams, design charts, histograms, milestone charts, program flow charts, and tree diagrams.
- Additions
  - Synthesis under transformation.
  - I/O specification analysis under static analysis.
  - Regression testing under dynamic analysis.
  - VHLL under machine output.

* Contract NB79SBCA0273 to SoHar, Inc., and SoHaR Subcontract No. 102 to Software Management Consultants (now Reifer Consultants, Inc.).

** See references in section B.3 of this Guideline.

B.3 References

[Houg81a] Houghton, R. Features of software development tools. Natl. Bur. Stand. (U.S.) NBS Spec. Publ. 500-74; 1981 February.

[Houg81b] Houghton, R. An inverted view of software development tools. Proceedings of the 20th Annual Technical Symposium of the Washington, D.C. Chapter of the ACM; 1981 June.

[ICST81] ----------. Software development tools: a reference guide to a taxonomy of tool features. U.S. Department of Commerce, LC-1127; 1981 February.

[Reif81a] Reifer, D. Tool standards—It's about time. Proceedings of the Software Engineering Standards Application Workshop, IEEE No. 81CH1633-7; 1981 August.

NBS TECHNICAL PUBLICATIONS

PERIODICALS

JOURNAL OF RESEARCH—The Journal of Research of the National Bureau of Standards reports NBS research and development in those disciplines of the physical and engineering sciences in which the Bureau is active. These include physics, chemistry, engineering, mathematics, and computer sciences. Papers cover a broad range of subjects, with major emphasis on measurement methodology and the basic technology underlying standardization. Also included from time to time are survey articles on topics closely related to the Bureau's technical and scientific programs. As a special service to subscribers each issue contains complete citations to all recent Bureau publications in both NBS and non-NBS media. Issued six times a year. Annual subscription: domestic $18; foreign $22.50. Single copy, $5.50 domestic; $6.90 foreign.

NONPERIODICALS

Monographs—Major contributions to the technical literature on various subjects related to the Bureau's scientific and technical activities.

Handbooks—Recommended codes of engineering and industrial practice (including safety codes) developed in cooperation with interested industries, professional organizations, and regulatory bodies.

Special Publications—Include proceedings of conferences sponsored by NBS, NBS annual reports, and other special publications appropriate to this grouping such as wall charts, pocket cards, and bibliographies.

Applied Mathematics Series—Mathematical tables, manuals, and studies of special interest to physicists, engineers, chemists, biologists, mathematicians, computer programmers, and others engaged in scientific and technical work.

National Standard Reference Data Series—Provides quantitative data on the physical and chemical properties of materials, compiled from the world's literature and critically evaluated. Developed under a worldwide program coordinated by NBS under the authority of the National Standard Data Act (Public Law 90-396).

NOTE: The principal publication outlet for the foregoing data is the Journal of Physical and Chemical Reference Data (JPCRD) published quarterly for NBS by the American Chemical Society (ACS) and the American Institute of Physics (AIP). Subscriptions, reprints, and supplements available from ACS, 1155 Sixteenth St., NW, Washington, DC 20056.

Building Science Series—Disseminates technical information developed at the Bureau on building materials, components, systems, and whole structures. The series presents research results, test methods, and performance criteria related to the structural and environmental functions and the durability and safety characteristics of building elements and systems.

Technical Notes—Studies or reports which are complete in themselves but restrictive in their treatment of a subject. Analogous to monographs but not so comprehensive in scope or definitive in treatment of the subject area. Often serve as a vehicle for final reports of work performed at NBS under the sponsorship of other government agencies.

Voluntary Product Standards—Developed under procedures published by the Department of Commerce in Part 10, Title 15, of the Code of Federal Regulations. The standards establish nationally recognized requirements for products, and provide all concerned interests with a basis for common understanding of the characteristics of the products. NBS administers this program as a supplement to the activities of the private sector standardizing organizations.

Consumer Information Series—Practical information, based on NBS research and experience, covering areas of interest to the consumer. Easily understandable language and illustrations provide useful background knowledge for shopping in today's technological marketplace.

Order the **above** *NBS publications from: Superintendent of Documents, Government Printing Office, Washington, DC 20402.*

Order the **following** *NBS publications—FIPS and NBSIR's—from the National Technical Information Service, Springfield, VA 22161.*

Federal Information Processing Standards Publications (FIPS PUB)—Publications in this series collectively constitute the Federal Information Processing Standards Register. The Register serves as the official source of information in the Federal Government regarding standards issued by NBS pursuant to the Federal Property and Administrative Services Act of 1949 as amended, Public Law 89-306 (79 Stat. 1127), and as implemented by Executive Order 11717 (38 FR 12315, dated May 11, 1973) and Part 6 of Title 15 CFR (Code of Federal Regulations).

NBS Interagency Reports (NBSIR)—A special series of interim or final reports on work performed by NBS for outside sponsors (both government and non-government). In general, initial distribution is handled by the sponsor; public distribution is by the National Technical Information Service , Springfield, VA 22161, in paper copy or microfiche form.