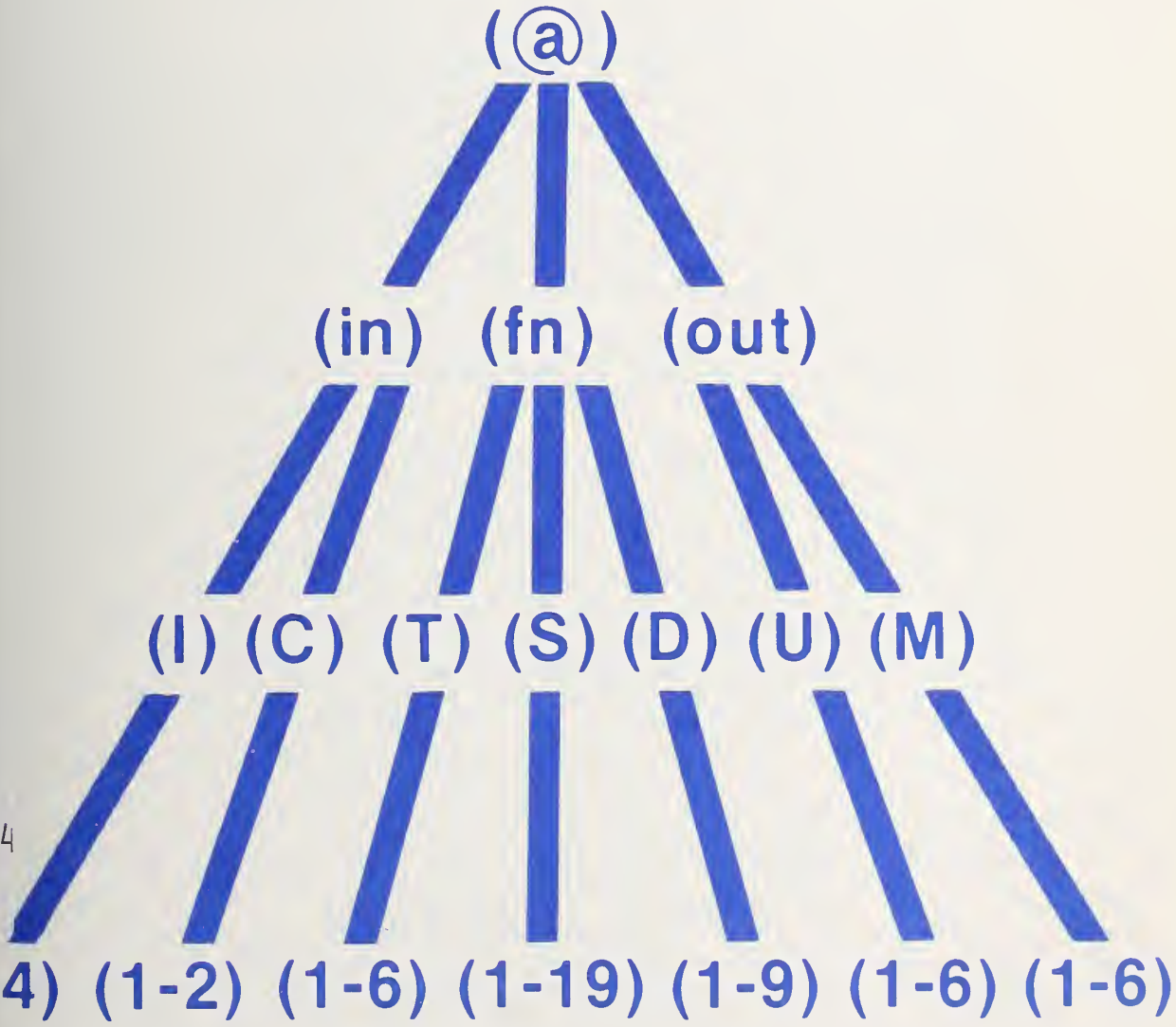
S. Department
Commerce
ational Bureau
Standards

Computer Science and Technology

NBS Special Publication 500-74

Features of Software Development Tools



NATIONAL BUREAU OF STANDARDS

The National Bureau of Standards[1] was established by an act of Congress on March 3, 1901. The Bureau's overall goal is to strengthen and advance the Nation's science and technology and facilitate their effective application for public benefit. To this end, the Bureau conducts research and provides: (1) a basis for the Nation's physical measurement system, (2) scientific and technological services for industry and government, (3) a technical basis for equity in trade, and (4) technical services to promote public safety. The Bureau's technical work is performed by the National Measurement Laboratory, the National Engineering Laboratory, and the Institute for Computer Sciences and Technology.

THE NATIONAL MEASUREMENT LABORATORY provides the national system of physical and chemical and materials measurement; coordinates the system with measurement systems of other nations and furnishes essential services leading to accurate and uniform physical and chemical measurement throughout the Nation's scientific community, industry, and commerce; conducts materials research leading to improved methods of measurement, standards, and data on the properties of materials needed by industry, commerce, educational institutions, and Government; provides advisory and research services to other Government agencies; develops, produces, and distributes Standard Reference Materials; and provides calibration services. The Laboratory consists of the following centers:

Absolute Physical Quantities[2] — Radiation Research — Thermodynamics and Molecular Science — Analytical Chemistry — Materials Science.

THE NATIONAL ENGINEERING LABORATORY provides technology and technical services to the public and private sectors to address national needs and to solve national problems; conducts research in engineering and applied science in support of these efforts; builds and maintains competence in the necessary disciplines required to carry out this research and technical service; develops engineering data and measurement capabilities; provides engineering measurement traceability services; develops test methods and proposes engineering standards and code changes; develops and proposes new engineering practices; and develops and improves mechanisms to transfer results of its research to the ultimate user. The Laboratory consists of the following centers:

Applied Mathematics — Electronics and Electrical Engineering[2] — Mechanical Engineering and Process Technology[2] — Building Technology — Fire Research — Consumer Product Technology — Field Methods.

THE INSTITUTE FOR COMPUTER SCIENCES AND TECHNOLOGY conducts research and provides scientific and technical services to aid Federal agencies in the selection, acquisition, application, and use of computer technology to improve effectiveness and economy in Government operations in accordance with Public Law 89-306 (40 U.S.C. 759), relevant Executive Orders, and other directives; carries out this mission by managing the Federal Information Processing Standards Program, developing Federal ADP standards guidelines, and managing Federal participation in ADP voluntary standardization activities; provides scientific and technological advisory services and assistance to Federal agencies; and provides the technical foundation for computer-related policies of the Federal Government. The Institute consists of the following centers:

Programming Science and Technology — Computer Systems Engineering.

[1]Headquarters and Laboratories at Gaithersburg, MD, unless otherwise noted; mailing address Washington, DC 20234.
[2]Some divisions within the center are located at Boulder, CO 80303.

Computer Science and Technology

NBS Special Publication 500-74

Features of Software Development Tools

Raymond C. Houghton, Jr.

Center for Programming Science and Technology
Institute for Computer Sciences and Technology
National Bureau of Standards
Washington, DC 20234

U.S. DEPARTMENT OF COMMERCE
Malcolm Baldrige, Secretary

U.S. **National Bureau of Standards**
Ernest Ambler, Director

Issued February 1981

Reports on Computer Science and Technology

The National Bureau of Standards has a special responsibility within the Federal Government for computer science and technology activities. The programs of the NBS Institute for Computer Sciences and Technology are designed to provide ADP standards, guidelines, and technical advisory services to improve the effectiveness of computer utilization in the Federal sector, and to perform appropriate research and development efforts as foundation for such activities and programs. This publication series will report these NBS efforts to the Federal computer community as well as to interested specialists in the academic and private sectors. Those wishing to receive notices of publications in this series should complete and return the form at the end of this publication.

National Bureau of Standards Special Publication 500-74
Nat. Bur. Stand. (U.S.), Spec. Publ. 500-74, 24 pages (Feb. 1981)
CODEN: XNBSAV

Library of Congress Catalog Card Number: 80-600193

U.S. GOVERNMENT PRINTING OFFICE
WASHINGTON: 1981

For sale by the Superintendent of Documents, U.S. Government Printing Office, Washington, DC 20402
Price $1.75
(Add 25 percent for other than U.S. mailing).

TABLE OF CONTENTS

Features of Software Development Tools

Raymond C. Houghton, Jr.

Abstract

Software tools are powerful productivity and quality aids that in many cases are not being used effectively. This report discusses an effort to lessen this problem by providing a formal way in which tools can be classified according to the features that they provide.

Key words: Dynamic analysis; programming aids; software development; software engineering; software tools; static analysis; taxonomy.

1. Introduction

Software development tools are computer programs that aid the specification, construction, testing, analysis, management, documentation, and maintenance of other computer programs. Thus, software development tools include the traditional tools of a programmer (e.g. compilers, editors), more recently developed tools (e.g. design aids, program analyzers, testing tools), and tools currently in the research stage (e.g. formal verifiers, programming environments). Tools are important because they can be used to increase software productivity and quality. As a result, their use has evolved as an important part of software development. Most importantly, they represent a class of software that can be used and reused within many different software development environments. The use of software tools provides the opportunity to reduce cost and improve competitive advantage.

Although software tools are an important part of quality software development, there does not currently exist a clear body of techniques for making effective use of such tools. This situation exists in part due to the confusion

---

NOTE: Certain commercial products are identified in this paper 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.

generated by the lack of standard terminology for the functions and characteristics of software tools. It is difficult to compare and to evaluate alternative tool packages for use in specific operating environments. As a result, there is a lack of technology transfer by tool developers to the user community.

This report discusses an approach aimed at correcting some of these problems. It presents a framework for tool comparison and a scheme for classifying tools by the features they provide. Features of software tools are presented in a hierarchical classification system called a taxonomy. Following a period of comment and review, NBS plans to issue the taxonomy as a Federal Information Processing Standards Publication (FIPS-PUB) Guideline to provide a common reference within the Federal Government for terms and definitions associated with software development tools.

2. The Importance of Tool Features

Software development tools have grown in complexity over recent years as have other software applications. Most early tools performed a single function and were very simple to operate. These early, simpler tools have since given way to more complex tools which in some cases have their own built-in command languages. This growth is leading to tool systems where the programmer works totally within the system to develop software.

This evolution of tool development has caused major problems in communicating exactly what a tool does. For example, a compiler once was a simple tool that performed the single function of translating high level language to machine language. Today, we have compilers that interpret, optimize, perform run-time checks, and generate cross-references, profiles, formatted output, and other forms of documentation. Does the term compiler adequately describe these multi-functional systems, or do people still think of a compiler as a simple tool that performs a single function? There are other examples of tools whose complexity has outgrown their names. These include verification systems, program analyzers, program preprocessors, flow analyzers, software development systems, simulators, and programming environments. In each of these examples, one should press for further details about the tool to determine exactly what it does.

The features incorporated in a software tool are the details one should seek. Careful identification of features allows one to distinguish one tool from another and to determine which is more appropriate for a given application. In order to do this, one must acquire detailed information on a tool. This should include what a tool accepts as input and how it accepts it, the way it manipulates and analyzes that input, and what a tool produces as output for both the tool user and for further processing by other tools. With a careful analysis of this information, one can begin to understand the real capabilities of a tool and can compare these capabilities with those of other tools. The taxonomy of tool features presented in Section 4 provides a framework for communicating and a basis for understanding the capabilities of a tool.

3. Formalizing the Use of Tool Features

For now let us assume that we have a way to carefully and succinctly discover the individual features of a software tool. The sections that follow will address how one could use the resulting information. Examples of each of these uses will be shown in Section 5.

Comparison & Evaluation of Software Tools. Once all the features of a tool are known, its features can be compared to the features of other tools. The presence (or absence) of a specific feature in a given tool is apparent. Thus, a potential tool user has a rational means for determining whether to include a tool for consideration. For example, if a tool user is interested in a tool that has the features of high level language input, program instrumentation, dynamic coverage analysis, and output report generation, then the user may find several tools that fall within this classification. The user can then focus on the additional features offered by each of the tools and perform a cost benefit analysis on these additional features. Once a set of candidate tools has been determined based on desired features, the user can focus on other considerations, such as implementation languages, application language, hardware and software requirements, availability, performance, portability, support and available documentation.

Basis for Information Organization. Since features provide a vehicle for the comparison of software tools, one of their apparent uses is as a basis for retrieval in a database. Just as library searches are based on keywords, tool searches can be based on features. Relationships can be formed between the features of a tool and its name. Thus, when one wants to know all the tools available that have

certain features, then an additional relationship may be defined that relates these features only with those tools that possess them. From here it is a simple matter to provide any other information that may be desired about the tools in question.

The NBS Software Tools Database [Houg80] is a relational database that possesses this capability. One of the purposes of the database is to initiate a means by which persons in the Federal Government can determine what tools are available and what their capabilities are. An example of tool information retrieval from the database is shown in Section 5.

Classification of Software Tools. Since features provide a means by which we can describe software tools, then they can also provide a means by which we can classify software tools. In order to do this, we must discover and describe all the features for every known software development tool, gather them all together, eliminate duplication, and order them. The resulting order is called a taxonomy. The taxonomy is used to classify tools by associating the ordered features with the tool being classified. An example of this will be given in section 5 after first presenting the taxonomy.

4. A Taxonomy of Tool Features

The taxonomy that will be discussed was developed to satisfy the following goals:

a. The taxonomy shall allow NBS to classify all currently available software development tools.

b. Each class shall include terms that have the most specific meaning where possible.

c. The classes shall allow easy comparison and evaluation of tools.

d. The classes shall be related to the life cycle of an automated data system.

All of the goals were met except for the last. Life cycle relationships, although possible, were found to be outside the scope of the taxonomy that was developed.

The initial development of the taxonomy was performed under contract [1] to the National Bureau of Standards and

---

[1] Contract NB79SBCA0273 to SoHar, Inc. and SoHaR Subcontract No. 102 to Software Management Consultants.

is reported in detail in [Reif80]. The present taxonomy is an update of the one presented in that report. Further updates are anticipated as the taxonomy approaches adoption as a FIPS. Therefore, comments on the usefulness, implementation, structure, and content are solicited by the author.

The taxonomy is a hierarchical order of software tool features and is illustrated in figure 1. At the bottom or feature level of the hierarchy are a total of 52 tool features. Each of these features will be defined and discussed in the sections that follow. At the third level are the classes of tool features. These are subject, control input, transformation, static analysis, dynamic analysis, machine output, and user output. Each of the 52 features have been placed into one of these classes. In order to provide a way to abbreviate a tool classification, keys have been assigned to each of the elements in the third and fourth levels. Each of the keys will be identified in the sections that follow and examples of their use will be shown in Section 5.

The second level of the taxonomy covers the basic processes of a tool. These are input, function, and output. The highest level, software development tool features, covers all the levels below. The taxonomy has been designed to be expandable. It is expected that future updates of the taxonomy will include additional features and possibly additional levels.

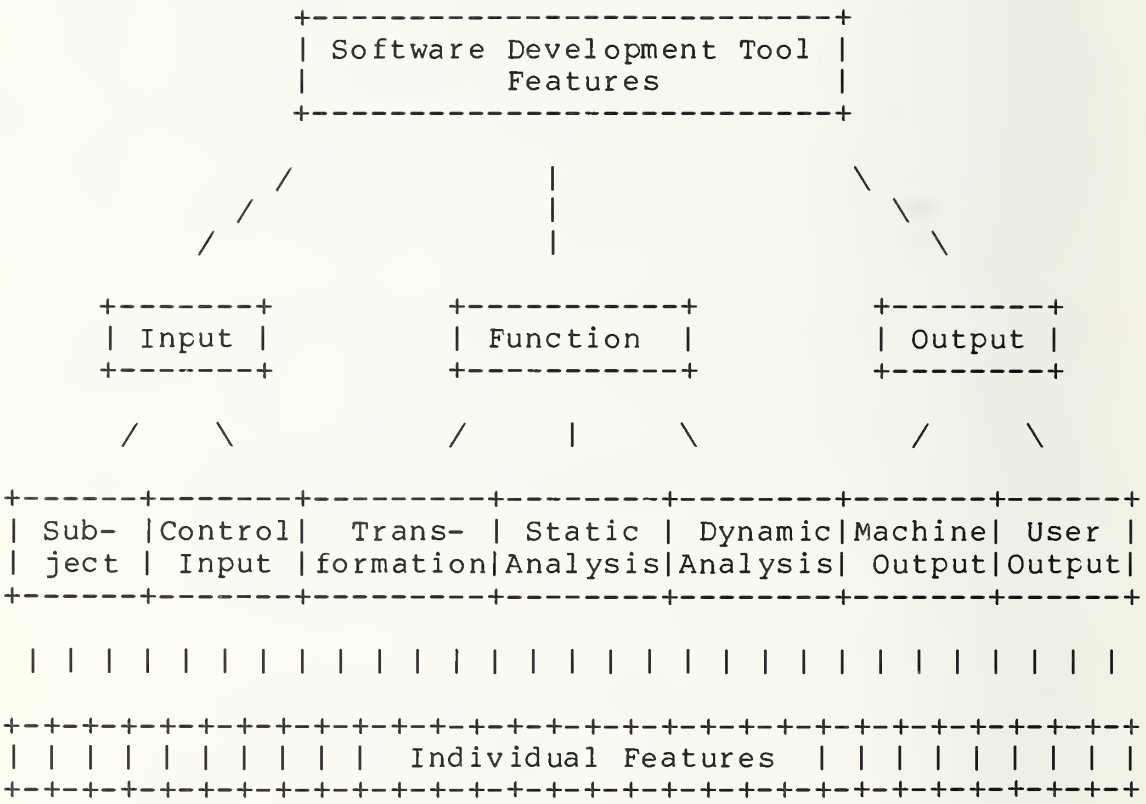


Figure 1. Taxonomy of Software Development Tool Features

Input. Input features are based on the forms of input which can be provided to a tool. These features fall into two classes, one which is based on how the tool should operate (control input) and the other based on what the tool should operate on (subject). Using a compiler as an example, the subject is the code that will be compiled and the control input is the set of commands which specify compiler options (optimize, debug, cross reference, etc.).

INPUT	
Subject	Control Input
I1. Text	C1. Commands
I2. VHLL	C2. Parameters
I3. Code	
I4. Data	

Table 1. Input

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 data, code, VHLL (very high level language), and text.

I1. Text - The input to the tool is presented 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 - The input to the tool is a program written in a very high level language that is typically not executed. Three recognized types of VHLL's are requirements languages, design languages, and description languages. Requirements and design languages are both used to define programs and to provide a means for generating automatic documentation. Examples of requirements languages include the Problem Statement Language [Teic77] and the Requirements Statement Language [Bell77]. An example of a design language is Program Design Language [Cain75]. Description languages are used to describe attributes of the input in high-level, non-procedural form. An example of a description language is Backus-Naur Form (BNF).

I3. Code - The input to the tool is a program written in a language that is subject to a given translation process before it is executed. Code is the language form in which most programming solutions are expressed and includes high level languages, assembly languages, object representations or parametric representations. High level languages are problem-oriented (e.g., COBOL and FORTRAN), while assembly languages are machine oriented but symbolic in nature. Object code represents an input that is typically executable without further translation, while parametric representation (e.g., NAMELIST in FORTRAN IV) inputs variables to be translated.

I4. Data - The input to the tool is a string of characters 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 if the four of these input forms. In cases such as this, the input form is chosen from the viewpoint of the tool. Since editors view the input form as text, then text would be the correct choice for this tool.

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 - The control input to the tool consists 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 that performs a single function will not have this feature but will most likely have the next.

C2. Parameters - The control input is a series of parameters that are associated with the functions performed by the 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 three classes: transformation, static analysis, and dynamic analysis. Again using a compiler as an example, the transformation features would be translation and possibly optimization, the static analysis features would be error checking and possibly cross reference, and a dynamic analysis feature may be tracing.

FUNCTION

Transformation	Static Analysis	Dynamic Analysis
T1. Editing	S1. Auditing	D1. Assertion Checking
T2. Formatting	S2. Comparison	D2. Constraint Evaluation
T3. Instrumentation	S3. Complexity Measurement	D3. Coverage Analysis
T4. Optimization	S4. Completeness Checking	D4. Resource Utilization
T5. Restructuring	S5. Consistency Checking	D5. Simulation
T6. Translation	S6. Cost Estimation	D6. Symbolic Execution
	S7. Cross Reference	D7. Timing
	S8. Data Flow Analysis	D8. Tracing
	S9. Error Checking	D9. Tuning
	S10. Interface Analysis	
	S11. Management	
	S12. Resource Estimation	
	S13. Scanning	
	S14. Scheduling	
	S15. Statistical Analysis	
	S16. Structure Checking	
	S17. Tracking	
	S18. Type Analysis	
	S19. Units Analysis	

Table 2. Function

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

T1. Editing - modifying the context 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 in order to collect execution statistics required for coverage analysis and tuning (see D2 and D9).

T4. Optimization - modifying a program to improve performance, e.g. to make it run faster or to make it use fewer resources. Optimizing compilers have this feature.

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

T6. Translation - to convert from one language form to another. Tools that have this feature include compilers, structured language preprocessors, and conversion tools.

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 nineteen 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 - Assessing similarities between two or more items. A tool that compares programs or test runs for maintaining version control has this feature.

S3. Complexity Measurement - Determining how complicated an entity (e.g., routine, program, system, etc.) is by evaluating some number of 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. 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.

S7. Cross Reference - referencing entities to other entities by logical means. Tools that generate call graphs or tools that identify all variable references in a subprogram have this feature.

S8. 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.

S9. 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.

S10. 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.

S11. Management - aiding the management or control of software development. Tools that control access, updates, and retrievals of software; tools that maintain and control data definition and use; and tools that manage test data sets are examples of tools with this feature.

S12. 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.

S13. 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.

S14. 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.

S15. 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.

S16. Structure Checking - detecting structural flaws within a program (e.g. improper loop nestings, unreferenced labels, unreachable statements, and statements with no successors).

S17. 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.

S18. 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.

S19. 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.

c. Dynamic Analysis (Key: D). Dynamic analysis features specify operations that are determined during or after execution takes place [Howda78]. 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 nine 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 constraints for determining test input [Clar76]. Tools that assist 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 (i.e., 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 times associated with parts of a program.

D8. Tracing - tracking the historical record of execution of a program. Tools that produce trace histories or allow the setting of breakpoints for tracking down errors have this feature.

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.

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.

OUTPUT

User Output		Machine Output	
U1.	Computational Results	M1.	Data
U2.	Diagnostics	M2.	Intermediate Code
U3.	Graphics	M3.	Object Code
U4.	Listings	M4.	Prompts
U5.	Text	M5.	Source Code
U6.	Tables	M6.	Text

Table 3. Output

a. User Output (Key: U). User output features handle the interface from the tool to the human user. They describe the types of information that the tool provides for the user and the form in which this output is presented. There are six user output features. Each is briefly defined as follows:

U1. Computational Results - an output from the tool is simply the result of a computation. The output is not in an easily interpreted natural language form (e.g. text).

U2. Diagnostics - an output from the tool simply indicates that a software discrepancy has occurred. An error flag from a compiler is an example.

U3. Graphics - an output from the tool is graphically presented with symbols indicating operations, flow, etc. A tool providing a flowchart of a program is an example.

U4. Listings - an output from the tool is a 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. 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.

U6. 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.

b. Machine Output (Key: M). Machine output features handle the interface from the tool to a target machine. They describe what the machine expects to see as output from the tool. There are six machine output features. Each is briefly described as follows:

M1. Data - the input to the machine is representations of characters or numeric quantities to which meaning has been assigned. A tool generating input to a plotter is an example.

M2. Intermediate Code - the input to the machine is between source code and machine code. A tool producing P-code for direct machine interpretation is an example.

M3. Object Code - the input to the machine is a program 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.

M4. Prompts - the input to the machine is a series of procedural operators that are used to interactively inform the system in which the tool operates that it is ready for the next input.

M5. Source Code - the input to the machine is the program written in a procedural language that must be input to a translation process before execution can take place.

M6. Text - the input to the machine is presented in a written form that can be read without machine interpretation. A tool producing English text which is fed to the machine is an example.

5. Using the Taxonomy

Section 3 discussed in a very general level the use of tool features. Now that we have defined the features of a tool and given each of them individual keys, we can discuss some very specific uses of the taxonomy.

Tool Classification. To classify tools with the taxonomy, one must identify a tool's input, functional, and output features. As many features are identified as necessary to fully describe the capability of a tool. Identifying tool

features, in some cases, can be a major challenge because most tool descriptions fail to provide sufficient information. Considerable effort may be required to acquire the information necessary to identify what the tool does and how it interfaces with the external environment.

The result of classification is a features designator called the taxonomy key. The key is formed by collecting the individual feature keys according to the their position in 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
Instrumented Code Output

would have the following classification when individual keys are chosen:

I3.C1/T3.S3.S7.D3/U4.U5.M5

Note that the slash is used to separate the input, functional, and output features and the period is used to separate individual keys. It can be seen from the example that the key clearly and succinctly communicates the results of classification and summarizes the tool attributes.

Since the identification of tool features can be difficult for some types of tools, it is anticipated that a user's guide to the taxonomy will be published. This guide will provide a step-by-step procedure for identifying the features of tools with many actual examples for a wide variety of software development tools.

Comparison of Software Tools and Information Retrieval. In section 3, we described a user who was interested in a tool that has the features of high level language input, program instrumentation, dynamic coverage analysis, and output report generation. Using individual keys from the taxonomy, a retrieval request could be specified as follows:

I3/T3&D3/U5

where I3, T3, D3, and U5 represent the features, the slash is used to separate input and output, and the & is a logical 'and' of the features. In English this retrieval request

would be: "Give me all the tools which accept high level language programs as input, perform the functions of program instrumentation and dynamic coverage analysis, and provide reports as output." A retrieval of this request from the NBS Software Tools Database would 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:

I3'FORTRAN/T3&D3&(S1^S8)/U5

where S1 represents code auditing, S8 represents data flow analysis, the ' delineates the actual language of the source code, and ^ is a logical 'or'. This request in English would be: "Give me all the tools which accept Fortran programs as input, perform the functions of program instrumentation and dynamic analysis and either code auditing or data flow analysis, and provide reports as output." This retrieval request would yield the following tools:

PET	FAVS	TPT

Note that the number of tools that a potential tool user must consider has been greatly reduced.

6. Scope of the Taxonomy

The taxonomy is primarily a classification scheme for software development tools. There are, however, many tools that are broader in function and application that can not be easily classified by the taxonomy. These tools include operating systems, system utilities (linkage editors, loaders, tape handlers, file system, sorters), data base management systems, and management information systems. These tools serve as an extended part of a system and are outside the scope of the taxonomy. Since most of the features in the taxonomy are specific to the software development process, only tools specific to software development should be classified.

7. Conclusion

Software tools are powerful productivity and quality aids that are not being effectively used in many Federal programming environments. This report has discussed an effort to lessen this problem by providing a formal way in which tools can be classified. The use of a taxonomy of tool features can be used to (1) categorize currently available tools, (2) standardize terminology associated with tools, and (3) ease the task of comparing and evaluating tools. Use of the taxonomy should help users determine precisely what a given tool can offer and consequently what its benefits are.

8. References

[Bell77] T. E. Bell, D. C. Bixler and M. E. Dyer, "An Extendable Approach to Computer-Aided Software Requirements Engineering", IEEE Transactions on Software Engineering, Vol SE-3, No 1, 1977.

[Boeh78] B. W. Boehm, J. R. Brown, H. Kaspar, M. Lipow, G. J. MacLeod and M. J. Merritt, "Characteristics of Software Quality", North-Holland Publishing Company, NY, 1978.

[Cain75] S. H. Caine and E. K. Gordon, "PDL: A Tool for Software Design", Proceedings of the National Computer Conference, 1975.

[Clar76] L. A. Clarke, "A System to Generate Test Data and Symbolically Execute Programs", IEEE Transactions on Software Engineering, Vol SE-2, September 1976.

[Darr78] J. A. Darringer and J. C. King, "Applications of Symbolic Execution to Program Testing", Computer, April 1978.

[Fair78] R. E. Fairley, "Tutorial: Static Analysis and Dynamic Testing of Computer Software", Computer, April 1978.

[Hals77] M. H. Halstead, "Elements of Software Science", Elsevier - North Holland Pub. Co., New York, 1977.

[Houg80] Raymond C. Houghton, Jr. and Karen A. Oakley, eds., "NBS Software Tools Database", NBSIR 80-2159, October 1980.

[Howd78] W. E. Howden, "A Survey of Static Analysis Methods", Tutorial: Software Testing and Validation Techniques, IEEE Cat. No. EHO138-8, 1978.

[Howda78] W. E. Howden,"A Survey of Dynamic Analysis Methods", Tutorial: Software Testing and Validation Techniques, IEEE Cat. No. EHO138-8, 1978.

[Mcca76] T. J. McCabe, "A Complexity Measure", IEEE Transactions on Software Engineering, Vol SE-2, December 1976.

[Oste76] L. J. Osterweil and L. D. Fosdick, "DAVE - A Validation Error Detection and Documentation System for FORTRAN Programs", Software~Practice and Experience, October 1976.

[Paig74] M. R. Paige and J. P. Benson, "The Use of Software Probes in Testing FORTRAN Programs", Computer, July 1974.

[Reif80] Donald J. Reifer and Harold A. Montgomery, "Final Report, Software Tool Taxonomy", Software Management Consultants Report No. SMC-TR-004, June 1980.

[Robi77] L. Robinson and K. N. Levitt, "Proof Techniques for Hierarchically Structured Programs", Communications of the ACM, April 1977.

[Teic77] D. Teichroew and E. Hershey III, "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] G. Walters and J. McCall, "The Development of Metrics for Software Reliability and Maintainability", Proceedings of the Annual Reliability and Maintainability Symposium, January 1978.

NBS-114A (REV. 2-8C)

U.S. DEPT. OF COMM. BIBLIOGRAPHIC DATA SHEET (See instructions)	1. PUBLICATION OR REPORT NO. SP 500-74	2. Performing Organ. Report No.	3. Publication Date February 1981

4. TITLE AND SUBTITLE

Features of Software Development Tools

5. AUTHOR(S)

Raymond C. Houghton, Jr.

6. PERFORMING ORGANIZATION *(If joint or other than NBS, see instructions)*

NATIONAL BUREAU OF STANDARDS
DEPARTMENT OF COMMERCE
WASHINGTON, D.C. 20234

7. Contract/Grant No.

8. Type of Report & Period Covered

Final

9. SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS *(Street, City, State, ZIP)*

10. SUPPLEMENTARY NOTES

Library of Congress Catalog Card Number: 80-600193

☐ Document describes a computer program; SF-185, FIPS Software Summary, is attached.

11. ABSTRACT *(A 200-word or less factual summary of most significant information. If document includes a significant bibliography or literature survey, mention it here)*

Software tools are powerful productivity and quality aids that in many cases are not being used effectively. This report discusses an effort to lessen this problem by providing a formal way in which tools can be classified according to the features that they provide.

12. KEY WORDS *(Six to twelve entries; alphabetical order; capitalize only proper names; and separate key words by semicolons)*

Dynamic analysis; programming aids; software development; software engineering; software tools; static analysis; taxonomy

13. AVAILABILITY

☒ Unlimited

☐ For Official Distribution. Do Not Release to NTIS

☒ Order From Superintendent of Documents, U.S. Government Printing Office, Washington, D.C. 20402.

☐ Order From National Technical Information Service (NTIS), Springfield, VA. 22161

14. NO. OF PRINTED PAGES

24

15. Price

$1.75

☆ U.S. GOVERNMENT PRINTING OFFICE : 1981 O—340-997 (1526)

USCOMM-DC 6043-P80

ANNOUNCEMENT OF NEW PUBLICATIONS ON COMPUTER SCIENCE & TECHNOLOGY

Superintendent of Documents,
Government Printing Office,
Washington, D. C. 20402

Dear Sir:

Please add my name to the announcement list of new publications to be issued in the series: National Bureau of Standards Special Publication 500-.

Name _____

Company _____

Address _____

City _____ State _____ Zip Code _____

(Notification key N-503)

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 $13; foreign $16.25. Single copy, $3 domestic; $3.75 foreign.

NOTE: The Journal was formerly published in two sections: Section A "Physics and Chemistry" and Section B "Mathematical Sciences."

DIMENSIONS/NBS—This monthly magazine is published to inform scientists, engineers, business and industry leaders, teachers, students, and consumers of the latest advances in science and technology, with primary emphasis on work at NBS. The magazine highlights and reviews such issues as energy research, fire protection, building technology, metric conversion, pollution abatement, health and safety, and consumer product performance. In addition, it reports the results of Bureau programs in measurement standards and techniques, properties of matter and materials, engineering standards and services, instrumentation, and automatic data processing. Annual subscription: domestic $11; foreign $13.75.

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 Services, 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 Services, Springfield, VA 22161, in paper copy or microfiche form.

U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Washington, D.C. 20234

OFFICIAL BUSINESS

Penalty for Private Use, $300

POSTAGE AND FEES PAID
U.S. DEPARTMENT OF COMMERCE
COM-215

SPECIAL FOURTH-CLASS RATE
BOOK