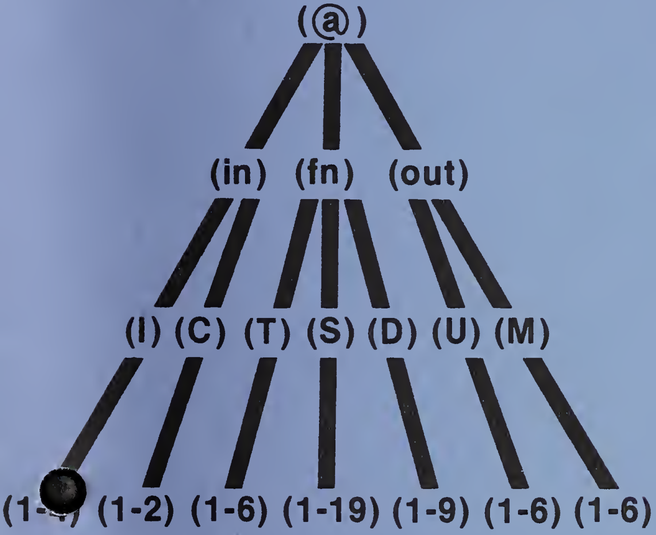
2C1127

SOFTWARE DEVELOPMENT TOOLS: A Reference Guide to a Taxonomy of Tool Features



U.S. DEPARTMENT OF COMMERCE
National Bureau of Standards
Center for Programming Science and Technology
Institute for Computer Sciences and Technology
Washington, DC 20234

Additional copies may be requested from:

Software Tools Project
Room A-265 Technology
National Bureau of Standards
Washington, DC 20234

February 1981

LC-1127

FOREWORD

This booklet has been prepared as a reference guide to a taxonomy. In particular, the taxonomy that will be presented is a hierarchical order of the features of software development tools. 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.

The features of software development tools allow one to distinguish one tool from another and to determine which tools are 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 tool and can compare these capabilities with those of other tools. The taxonomy of tool features presented in the next section provides a framework for communicating and a basis for understanding the capabilities of a tool. Following a period of comment and review, NBS plans to issue the taxonomy as a Federal Information Processing Standard (FIPS) to be used for the classification, acquisition, and evaluation of software development tools in the Federal Government. More details on the taxonomy are contained in NBS Special Publication 500-74.

A TAXONOMY OF SOFTWARE TOOL FEATURES

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 (I), control input (C), transformation (T), static analysis (S), dynamic analysis (D), user output (U), and machine output (M). Each of the 52 features has been placed into one of these classes. The second level of the taxonomy covers the basic processes of a tool. These are input (in), function (fn), and output (out). The highest level (@) covers all the levels below. A key has been assigned to each of the elements in the third and fourth levels. The keys provide a way to abbreviate the classification by features of a software tool. Classification will be discussed further in a later section.

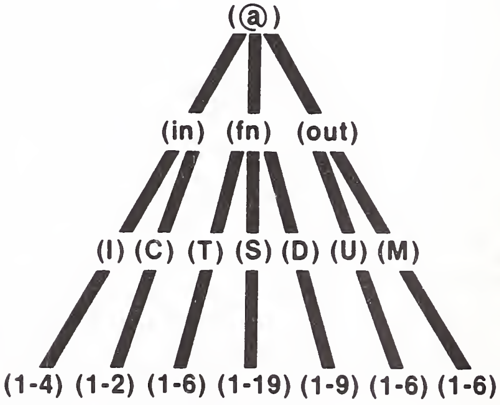


FIGURE 1. Taxonomy.

The taxonomy has been designed to be expandable. It is expected that future updates of the taxonomy will include additional features and additional levels.

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.).

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

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. Description languages are used to describe attributes of the input in high-level, non-procedural form.

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.

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.

Some tools, such as editors, operate on any of 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.

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 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.

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. 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.

atic Analysis (Key: S). Static analysis features specify operations on the subject without regard to the executability of the subject. They describe the manner in which the subject is analyzed. There are 19 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. 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 its parts present and if those parts are fully developed. 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, or is consistent with its specification. 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 th 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. 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.

Dynamic Analysis (Key: D). Dynamic analysis features specify operations that are determined during or after execution takes place. 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. 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. 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.

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 provide the link from the tool to the user. 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.

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.

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.

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 the hierarchical representation of the features provided by a tool which may be abbreviated using keys. For example, if we were to classify the compiler that was used previously as an example, it would have the following classification:

Input	Subject	Code
	Control Input	Commands
Function	Transformation	Optimization Translation
	Static Analysis	Cross Reference Error Checking
	Dynamic Analysis	Tracing
Output	User Output	Diagnostics Listings Tables
	Machine Output	Object Code

The compiler would have the following classification when individual keys are chosen:

I3.C1/T4.T6.S7.S9.D8/U2.U4.U6.M3

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 classification clearly and succinctly communicates the features provided by a tool and that the classification summarizes the tool attributes.

SCOPE

The taxonomy is primarily a classificati scheme for software development tools. Ther 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 systems, 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.