INSIDE:
Software Tools, Software Techniques, Benefits, Classification of Software Tools and Techniques, Glossaries of Software Tools and Techniques

Software Tools and Techniques

Frederick Gallegos

PAYOFF IDEA. Software tools and techniques are most widely used by systems analysts and programmers for developing and maintaining systems. They can also be valuable aids for the auditor in performing compliance or substantive testing in the development, operation, or maintenance phases. This article describes these resources and discusses how they might be categorized for application.

INTRODUCTION

The complexities of writing and maintaining programs have caused software costs to outstrip computer hardware costs. Recent studies predict that by 1990, more than 90 percent of the cost of data processing will be attributable to software. During the 1970s, private industry and government spent more than $8 billion a year on software. Experts believe that such expenditures currently exceed $20 billion yearly. Because of such skyrocketing costs, software tools and techniques are emerging that facilitate the development effort through streamlined procedures or automation of some development tasks.

Many tools and techniques have been developed that offer significantly improved management control and reduced costs if properly applied. The number of new tools and techniques continues to grow.

This article discusses the use of software tools and techniques to alleviate the problems of development, maintenance, modification, operation, and conversion of applications software. Many of the software tools and techniques discussed are available for all types of computers. Software tools and techniques can be valuable aids to information resource managers; data processing design, development, and operations

staff; and EDP auditors. In addition, this article provides a glossary of the most common types of software tools and techniques and a method for productively classifying and managing them.

SOFTWARE TOOLS

A software tool is a program that automates some of the labor involved in the management, design, coding, testing, inspection, or maintenance of other programs. Commercially available tools range in size and complexity from simple aids for individual programmers and end users to complex tools that can support many software projects simultaneously. The following are some common tools:

- Preprocessors—Preprocessors perform preliminary work on a draft computer program before it is completely tested on the computer. Types of preprocessors include filters (also known as code auditors), which allow management to determine quickly whether programmers are adhering to specifications and standards, and shorthand preprocessors, which allow programmers and end users to write the programs in an abbreviated form that is then expanded by the preprocessor before it is tested on the computer. Shorthand preprocessors reduce writing, keypunching, and proofreading effort.
- Programmer or user support libraries—These automated filing systems can support the programming development projects of entire installations. Such a support library maintains files of draft programs, data, and documentation and can be used to provide management with progress reports.
- Program analyzers—These tools modify or monitor the operation of an applications program to allow information about its operating characteristics to be collected automatically. This information can then be used to help modify the program to reduce its run cost or to verify that the program operates correctly.
- Online programming support programs—These tools enable programmers and users to quickly correct and modify applications programs and test program results.
- Test data generators—These tools analyze a program and produce files of data needed to test the logic of the program.

Specific examples of software tools are provided in Figure 1.

SOFTWARE TECHNIQUES

Software techniques are methods or procedures for designing, developing, documenting, and maintaining programs, or for managing these activities. There are generally two types of software techniques: those used by personnel who work on programs and those used by managers to control the work.

Examples of software techniques useful to workers include:

- Structured programming—Developing programs in a certain style with standard constructs so that they will be more easily understood by others who must later maintain and modify them, which facilitates documentation, testing, and correction.

VENDOR	PRODUCT	ENVIRONMENT	DESIGN ANALYSIS	CODING		
AGS Management Systems Inc 880 First Ave King of Prussia PA 19406	Estiplan SDM/Structured	D A	X X	 X		
Aims+ Plus Inc 1701 Directors Blvd, Suite 400 Austin TX 78744	AIMS PLUS					X
Allen Ashley 395 Sierra Madre Villa Pasadena CA 91107	Hybrid Development System Source Module Development Utility				X	
Applied Data Research Rte 206 & Orchard Rd Princeton NJ 08540	ADR/Data Designer ADR/Ideal ADR/MetaCOBOL	B B B	X X	 X X	X	X
Azrex Inc 3 Mountain Rd Burlington MA 01803	AZ7	B		X		
Bristol Information Systems Inc 84 N Main St Fall River MA 02721	BIS Dataprint Report Generator	C		X		
Bytel Corp 1029D Solano Ave Berkeley CA 94706	Cogen Menupro	D A		X X		
Caine Faber & Gordon Inc 750 East Green St Pasadena CA 91101	PDL	A	X			
Capro Inc 12781 Pala Drive Garden Grove CA 92641	PRO-IV Software	D	X	X	X	
Cincom Systems Inc 2300 Montana Ave Cincinnati OH 45211	Mantis	B		X		
Computer Associates 125 Jericho Turnpike Jericho NY 11753	CA-EZTEST CA-symbug Jobdoc	B B B			X X	 X
Computing Productivity Inc Larrow House Waitsfield VT 05673	IP3	B	X	X	X	X
Consumers Software Inc Suite 106C 314E Holly St Bellingham WA 98225	Spreadsheet Auditor	D			X	X
Cortex Corp 55 William St Wellesley MA 02181	Application Builder Application Factory	C C		X		
Cullinet Software Inc 400 Blue Hill Dr Westwood MA 02090	Ada/Batch Ada/Online	B B	X X	X X		

Figure 1. Software Tools

VENDOR	PRODUCT	ENVIRONMENT	DESIGN ANALYSIS	CODING	TESTING	DOCUMENTATION
Datamate Co 4135 100th E Ave, Suite 128 Tulsa OK 74146	Datamate Reference Language GENIUS (Generator of Interactive User Systems)	A		X X		
Digital Research Inc 60 Garden Ct Monterey CA 93942	BT-80 Display Manager SID & ZSID	D D D		X X	 X	
Dylakor PO Box 3010 Granada Hills CA 91344	DYL-260 DYL-280	B B	 X	X X		
Dynatech Microsoftware Div 3 NE Executive Park Burlington MA 01803	C O R P Program Generator Codewriter Program Generator Techwriter Program Generator	D D D	X X X	X X X		
Forth Inc 2309 Pacific Coast Hwy Hermosa Beach CA 90254	Polyforth	C		X		
Generation Sciences Inc 10 E 40th St New York NY 10016	Gamma	B	X	X		X
Henco Software Inc 100 Fifth Ave Waltham MA 02154	INFO	A		X		
Higher Order Software Inc 2067 Massachusetts Ave Cambridge MA 02140	USE IT	A	X	X	X	X
Informatics General Corp 21031 Ventura Blvd Woodland Hills CA 91364	MARK IV MARK V	B B	X X	X X		
Information Builders Inc 1250 Broadway New York NY 10001	FOCUS PC-FOCUS	B D		X X		
Information Processing Inc 1850 Lee Rd S320 Winter Park FL 32789	BLISS/COBOL	C		X		
Ken Orr & Associates 1725 Gage Blvd Topeka KS 66604	DSSD Design Library STRUCTURE(S)	A A	X X	 X		 X
M Bryce & Associates 1248 Springfield Pike Cincinnati OH 45215	PRIDE	B	X			
Management and Computer Services—Computer Associates 498 N Kings Hwy Cherry Hill NJ	Datamacs Systemacs Tracmacs	B A B	 X		X X	
Manager Software Products Inc 131 Hartwell Ave Lexington MA 02173	TESTMANAGER	B			X	

Figure 1. (Cont)

VENDOR	PRODUCT	ENVIRONMENT	DESIGN ANALYSIS	CODING	TESTING	DOCUMENTATION
Martin Marietta Data Systems PO Box 2392 Princeton NJ 08540	RAMIS II UFO	B B	 X	X X		
Master Software 42 Pleasant St Watertown MA 02172	PrograMaster	B	X	X	X	X
Micro Focus Inc 2465 E Bayshore Rd S400 Palo Alto CA 94303	ANIMATOR Level II COBOL Sideshow	D D D	 X	 X X	X	
MicroPro International Corp 33 San Pablo Ave San Rafael CA 94903	DataStar	D		X		
Multiplications Software Inc 1050 Massachusetts Ave Cambridge MA 02138	Accolade	B	X	X		
National Information Systems Inc 20370 Town Center La S130 Cupertino CA 95014	DPL	B		X		
Netro Inc 99 St. Regis Crescent N Downview, Ontario, Canada M3J1Y9	Computer-Aided Programming	C		X		
Pansophic Systems Inc 709 Enterprise Dr Oak Brook IL 60521	EASYTRIEVE PLUS Pro/Grammar	B B		X X	 X	
Phoenix Systems Inc One Station Sq Pittsburgh PA 15219	Hercules System-80/2	D	X	X		
Progeni Systems Inc 715 N Central Ave Glendale CA 91203	Progeni Tools	B		X		
Pyramid Data Ltd 1050 W Katella SA Orange CA 92667	Number Cruncher	D	X	X		
Quantitative Software Management 1057 Waverley Way McLean VA 22101	SLIM	C	X			
RCI Suite 208 25550 Hawthorne Blvd Torrance CA 90505	ASSET Quick-Draw SoftCost	D D D	X X X			X X X
Relational Database Systems Inc 2471 E Bayshore Rd S600 Palo Alto CA 94303	Ace Perform	D D		X X		
Softool Corp 340 S Kellogg Ave Goleta CA 93117	SOFTOOL	C	X	X	X	

Figure 1. (Cont)

VENDOR	PRODUCT	ENVIRONMENT	DESIGN ANALYSIS	CODING	TESTING	DOCUMENTATION
SofTech Microsystems Inc 16875 W Bernardo Dr San Diego CA 92127	Advanced Development Kit for P-System Native Code Generator for P-System	D D		X X	X	X
The Software Store 706 Chippewa Sq Marquette MI 49855	INFO-88 Application Development System	D	X	X		
STSC Inc 2115 E Jefferson St Rockville MD 20852	APL* Plus/80 Application Development System	B		X	X	
System Support Software Inc 5230 Springboro Pike Dayton OH 45439	Quikjob III Quikwrite	B B		X X	X	
Systems & Software Inc 1319 Butterfield Rd Downers Grove IL 60515	Rex-Tools	A		X	X	
TOM Software 127 SW 156th St Seattle WA 98166	EZ-SPEED SPEED UTILITY	D D	X X	X		
Tominy Inc 4221 Malsbary Rd Cincinnati OH 45242	DATABASE-PLUS	A	X	X		
TSI International 187 Danbury Rd Wilton CT 06897	DATA ANALYZER Pro/Test	B B		X	X	

Notes:

Operating Environment
- A Multiple systems
- B Mainframes
- C Minicomputers
- D Microcomputers

Figure 1. (Cont)

- Top-down development—Designing, coding, and testing systems by building program modules starting with those at the general level and proceeding down to the most specialized, detailed level.
- Performance improvement—Analysis and modification of programs to make them run more efficiently without affecting user requirements. Performance may be improved by various software tools, including program analyzers.
- Concurrent documentation—The development of documentation concurrently with program development to provide better project control, increase completeness of the documentation, and save money.

Examples of techniques useful to managers include:

- Third-party inspection of software to improve quality—It is now feasible to require such inspection because current tools can automate much of the work involved.
- Chief programmer team method—The team nucleus is a skilled chief programmer, a backup programmer, and a programming librarian.
- Alternatives to software development—This applies to both software tools and applications software.

BENEFITS OF SOFTWARE TOOLS AND TECHNIQUES

Software tools and techniques can be powerful aids in the design, development, testing, and maintenance of software. Several studies have reported that the application of tools and techniques result in significant benefits, including improved management control, equipment procurements that could be deferred, and reduced software costs.[1] Specifically, the use of software tools and techniques can:

- Reduce adverse impact on user tasks—Structured programming produces programs that are easier to test and, once tested, easier to modify. Therefore, structured programming can reduce the chances of errors in the user results (e.g., overpayments) and make it easier to respond quickly to future user requests for modifications. In addition, appropriate tools can reduce the work of verifying that test data has actually exercised a program. This improves the chances of removing errors from the program before it is placed into production.
- Reduce overruns and delays—Current design and development techniques, including structured programming, can make software development more visible to management and more controllable.
- Reduce redundant software projects—Software tools and techniques make it easier for organizations to reuse existing software and avoid the expense and delay of developing their own software. Tools reduce the labor of analyzing software for suitability; modern techniques give a better idea of what to analyze for.
- Reduce software conversion costs—As noted in various studies and expounded by conversion contractors, appropriate tools can significantly reduce the labor of making programs written for one type of computer run on another.
- Allow equipment purchases to be deferred—Newly written software requires fewer machine resources to run, and existing software can be modified to reduce required machine resource utilization.
- Reduce operating costs—This includes the labor costs of maintenance, modification, and conversion, as well as the cost of the machine resources required to run the software.
- Improve software quality—Improved quality reduces testing and revision and simplifies future maintenance, modification, and conversion.

An organization that adopts a carefully selected group of software tools and techniques can better predict software costs and provide better documentation. For example, some organizations have adopted and required the use of a group of modern programming tools and techniques, including structured programming, a program support library, structured design, concurrent documentation, and preprocessors.[2] The reported benefits include improved project control, better end products, better organization for the maintenance phase, and estimated annual savings of more than $1 million in the development and maintenance of systems.

Software quality can be improved by applying appropriate tools and techniques in the development phase. Software tools can reduce the labor of preparing test data and verifying that the test data has exercised all program logic. More thorough testing becomes feasible and more reliable programs result.

One management technique to improve the quality of software systems requires the use of quality control groups independent of the software developers. This quality control can be performed by performance evaluation groups or internal auditors. These groups can review either software development or maintenance projects. For example, two recent government reports highlight cases in which internal auditors' use of software tools and techniques resulted in the detection and correction of errors before system implementation and eliminated unnecessary program instructions.[3]

CLASSIFICATION OF SOFTWARE TOOLS AND TECHNIQUES

Software tools and techniques assist the analyst, manager, programmer, and user by providing meaningful information and can be used to automate parts of the software effort, thereby increasing software reliability and productivity. Most important, they can be reused for multiple projects with diverse needs, distributing their development costs and thus lowering the cost to individual projects.

Figure 2 illustrates a life cycle concept for developing a method of classifying software tools and techniques. The basic model of the software life cycle process has been developed in accordance with the standard definitions of a software life cycle. This model illustrates the managerial, methodological, and evaluative techniques required throughout the cycle. The managerial techniques necessary over a software life cycle involve:

- Managing people—The users, systems analyst, programmers, project manager, test personnel
- Managing a project—Planning, coordination, direction, control, review
- Managing the configuration—Change control, documentation control, modification control, upgrade, optimization
- Using quality assurance as a verifying agent

The methodological techniques involve:

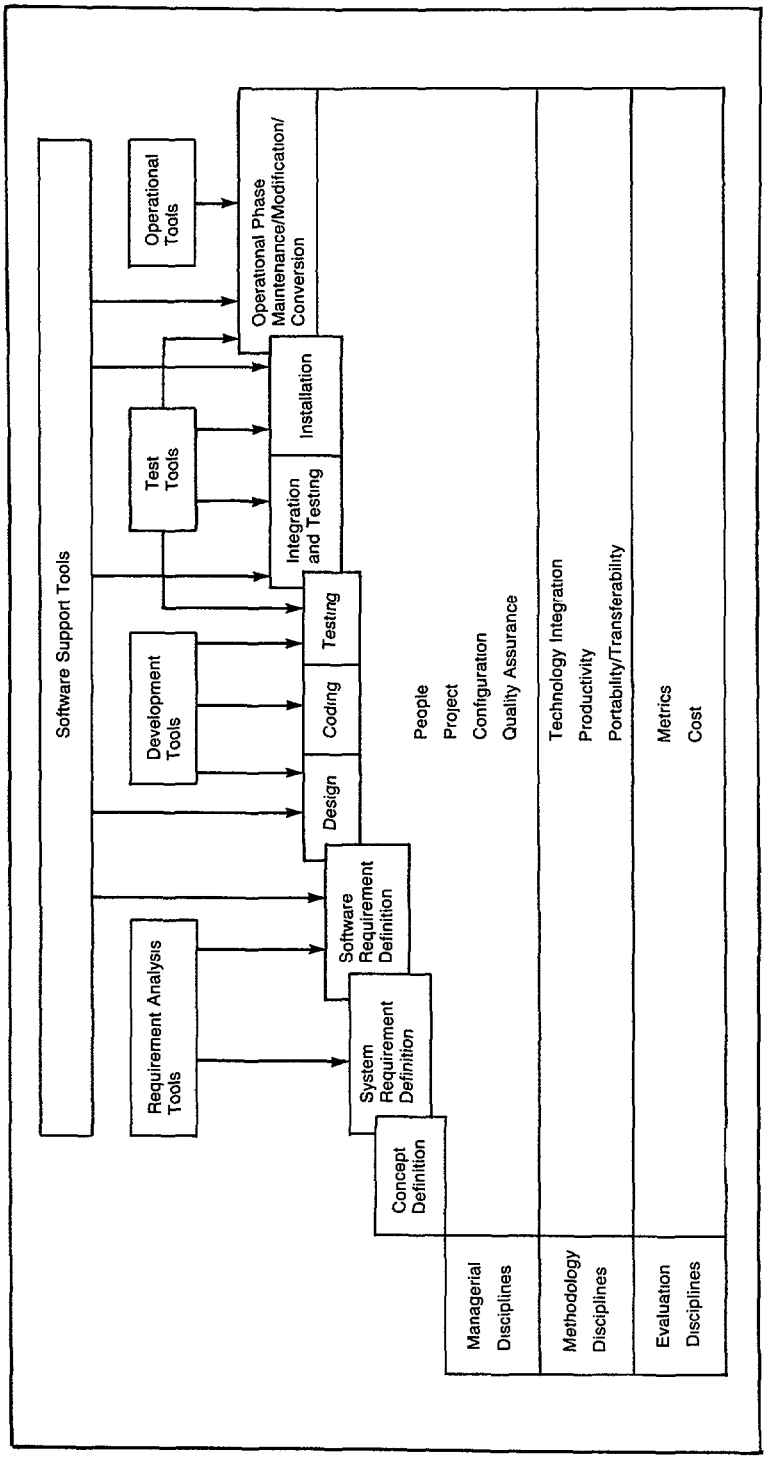

Figure 2. Software Life Cycle Taxonomy

- Integrating current and new technology (e.g., structured design concepts, programming, programmer's workbench, utilities, new software tools) into the design, development, and operational stages of the software life
- Using methods to improve the productivity of the designers, developers, testers, quality assurance personnel, and auditors
- Using methods for transferring the knowledge gained to other projects and people (e.g., training, seminars, professional papers, conferences, software decomposition and migration, tool migration)

The evaluative techniques required involve the better use of metrics and cost data to assess the implications of and risk caused by environmental changes. Examples of such metrics are resource estimators, factor analyzers, reliability models, and product measures. The basis for software life cycle evaluation is cost; therefore, accurate cost accounting for DP resources, especially in software, is a critical element.

The National Bureau of Standards (NBS) has developed a taxonomy for classifying general-purpose software tools in the DP environment. The taxonomy was published as Federal Information Processing Standard (FIPS) 99, "Guideline: A Framework for the Evaluation and Comparison of Software Development Tools." Because such a wide range of tool packages could be encompassed by the term *general purpose,* that term was applied only to those software tool packages usually not provided by the vendor as part of the purchased system. The goals established for the taxonomy were to:

- Permit existing tools to be uniquely classified
- Allow tool needs to be easily specified
- Provide a simple and meaningful set of descriptors for each tool classification
- Permit tool capabilities, costs, and benefits to be compared within a given classification

The characterization of software tools represents a major challenge because most tool descriptions fail to provide sufficient information. Considerable effort may be required to glean the information necessary to identify what the tool does and how it interfaces with the external environment. Once identified, these facts are extremely useful. Specifying what a tool does allows meaningful comparison of the capabilities of competing tools. It also permits the establishment of criteria for selecting tools (i.e., costs and benefits associated with tool usage can be related to tool capabilities and evaluated accordingly). Specification of a tool's interface enables a user to determine if the tool can produce the output needed and if it can work within the given operational environment. For example, the tool may become part of a programming environment (e.g., an integrated collection of tools used to support software development). In this case, the tool's interfaces with other tools are an important factor in tool evaluation.

This Federal Information Processing Standard is explicit in the specifications of features in all three dimensions (i.e., input, functions,

output), achieving a primary objective—a unique classification of an individual tool or a tool need. Products are classified according to a features designator called the taxonomy key. The key is formed by combining the individual feature keys for each of the three dimensions of the taxonomy. As many individual designators are chosen in each dimension as are necessary to completely describe the tool. The key, as illustrated in Figure 3, clearly and succinctly communicates the results of classification in all three dimensions.

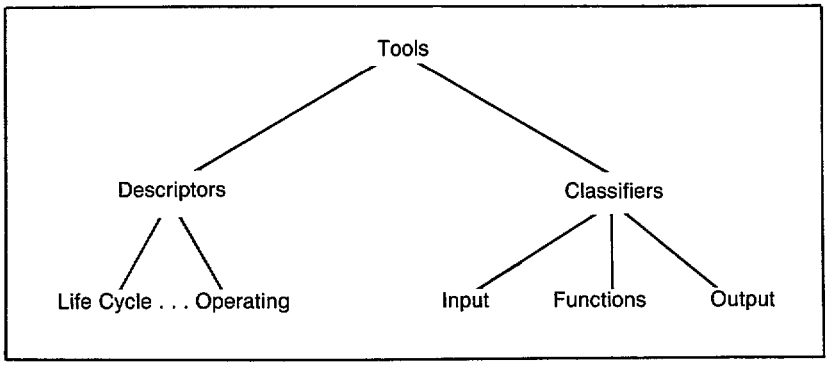

Figure 3. Tool Description and Classification Interrelationships

A GLOSSARY OF SOFTWARE TOOLS

The glossary of software tools is divided into the following classes:

- Transformation
- Static analysis
- Dynamic analysis
- Management

The types of software tools that can be classified under these areas are listed in Table 1.

Transformation

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. These features are briefly defined in the following sections:

Editing. Editors modify the content of the input by inserting, deleting, or moving characters, numbers, or data.

Formatting. Formatting arranges a program according to predefined or user-defined conventions. A tool having this feature can clean up a program by making all statement numbers sequential, alphabetizing variable declarations, indenting statements, and making other standardizing changes.

Table 1. Software Tools

Transformation
Editing
Formatting
Instrumentation
Optimization
Restructuring
Translation
  Assembling
  Compilation
  Conversion
  Macro expansion
  Structure preprocessing
Synthesis

Dynamic Analysis
Assertion checking
Constraint evaluation
Coverage analysis
Resource utilization
Simulation
Symbolic execution
Timing
Tracing
  Breakpoint control
  Data flow tracing
  Path flow tracing
Tuning
Regression testing

Static Analysis
Auditing
Comparison
Complexity measurements
Completeness checking
Consistency checking
Cross reference
Data flow analysis
Error checking
Interface analysis
Scanning
Statistical analysis
Structure checking
Type analysis
Units analysis
I/O specification analysis

Management
Configuration control
Information management
  Data dictionary management
  Documentation management
  File management
  Test data management
Project management
  Cost estimation
  Resource estimation
  Scheduling
  Tracking

Instrumentation. This adds sensors and counters to a program for the purpose of collecting dynamic analysis information. 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.

Optimization. Optimization is the process of modifying a program to improve performance (e.g., to make it run faster or use fewer resources). Many vendors' compilers provide this feature. Many tools claim to have 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.

Restructuring. Restructuring rearranges the subject in a new form according to well-defined rules. A tool that generates structured code from unstructured code is an example.

Translation. There are five types of translation features, which convert from one language form to another. They are defined as follows:

- Assembling—Translating a program expressed in an assembly language into object code.
- Compilation—Translating a program expressed in a problem-oriented language into object code.

- 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 1974 to ANSI COBOL 1985, and Pascal to PL/1.
- Macro expansion—Augmenting instructions in a source language with user-defined sequences of instructions in the same source language.
- Structure preprocessing—Translating a program with structured constructs into its equivalent without structured constructs.

Synthesis. Program generators, precompilers, and preprocessor generators generate programs according to predefined rules from a program specification or intermediate language.

Static Analysis

Static analysis features specify operations on the subject without regard to its ability to be executed. They describe how the subject is analyzed. The 15 static analysis features are briefly described in the following sections:

Auditing. Auditing conducts an examination to determine whether predefined rules have been followed. Examples include a tool that examines the source code to determine whether coding standards are complied with.

Comparison. Comparison determines and assesses similarities between two or more items. A tool with this feature can identify changes made in one file that are not contained in another.

Complexity Measurement. This determines how complicated an entity (e.g., routine, program, system) is by evaluating some associated characteristics. For example, complexity can be affected by instruction mix, data references, structure and control flow, number of interactions and interconnections, size, and number of computations.

Completeness Checking. Completeness checking determines whether all the parts of an entity are present and whether those parts are fully developed. Examples include a tool that examines the source code for missing parameter values.

Consistency Checking. This determines whether an entity is internally consistent in the sense that it contains uniform notation and terminology or adheres to its specification. Examples include tools that check for consistent use of variable names or tools that check for consistency between design specifications and code.

Cross Reference. Entities are referenced to other related entities by logical means. Examples include a tool that identifies all variable references in a subprogram.

Data Flow Analysis. Data flow analysis entails a graphic analysis of the sequential patterns of definition and references of data. Tools that identify undefined variables on certain paths in a program have this feature.

Error Checking. Discrepancies, their importance, and their cause are determined. Examples include a tool used to identify possible program errors (e.g., misspelled variable names, arrays out of bounds, and modifications of a loop index).

Interface Analysis. This feature checks the interfaces between program elements for consistency and adherence to predefined rules or axioms. A tool with this feature could examine interfaces between modules to confirm adherence to axiomatic rules for data exchange.

Scanning. Scanning entails examining an entity sequentially to identify key areas or structure. Examples include a tool that examines source code and extracts key information for generating documentation.

Statistical Analysis. This performs statistical data collection and analysis. Examples include a tool that uses statistical test modules to identify where programmers should concentrate their testing and a tool that tallies occurrences of statement types.

Structure Checking. Structure checking detects structural flaws within a program (e.g., improper loop nesting, unreferenced labels, unreachable statements, and statements with no successors).

Type Analysis. This evaluates whether the domains of values attributed to an entity are properly and consistently defined. A tool that type checks variables has this feature.

Units Analysis. Units analysis determines whether the units or physical dimensions attributed to an entity are properly defined and consistently used. Examples include a tool that checks a program to ensure that variables used in computations have proper units (e.g., hertz ◻ cycles/second).

Input/Output Specification Analysis. The input and output specifications in a program are analyzed, usually for the generation of test data. An example of an application of this feature is the analysis of the types and ranges of data defined in an input file specification for the purpose of generating an input test file.

Dynamic Analysis

Dynamic analysis features specify operations that are determined during or after execution. Dynamic analysis, unlike static analysis, requires some form of symbolic or machine execution. Dynamic analysis features describe the techniques used by the tool to derive meaningful

information about a program's execution behavior. The 10 dynamic analysis features are briefly described in the following sections.

Assertion Checking. This reviews 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 have this feature, as do tools that perform formal verification of assertions.

Constraint Evaluation. This generates or solves path input constraints for test data. Typically, tools that help generate test data specifications have this feature.

Coverage Analysis. This process helps determine and assess measures associated with the development of application program structural elements. This is extremely helpful in determining the adequacy of a modular or integrated test run. For example, coverage analysis is useful when attempting to execute a program statement, branch, or iterative structure.

Resource Utilization. Resource utilization associated with system hardware or software is analyzed. A tool that provides detailed run-time statistics on core usage, disk usage, queue durations is an example.

Simulation. Simulation entails 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.

Symbolic Execution. The logic and computations along a program path are reconstructed by executing the path with symbolic rather than actual data values.

Timing. This reports actual CPU, clock, or other times associated with parts of the program.

Tracing. Tracing monitors the historical record of execution of a program. The three types of tracing features are described as follows:

- Breakpoint control—Controlling the execution of a program by specifying points (usually source instructions) at which execution is to be interrupted.
- Data flow tracing—Monitoring the current state of variables in a program. Tools that dynamically detect uninitialized variables have this feature, as do tools that allow users to interactively retrieve and update the current values of variables.
- Path flow tracing—Recording the source statements or branches of a program in the order that they are executed.

Tuning. Tuning determines which parts of the program are being executed the most. Examples include a tool that instruments a program to obtain the execution frequencies of its statements.

Regression Testing. Test cases that a program has previously executed correctly are rerun to detect errors resulting from 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 previous output has this feature.

Management

Management features help in the administration or control of software development. The three types of management features are described in the following sections.

Configuration Control. This helps establish baselines for configuration items, the control of changes to these baselines, and the control of releases to the operational environment.

Information Management. Information management aids in the organization, accessibility, modification, dissemination, and processing of information associated with the development of a software system. Some specific areas of information management are:

- Data dictionary management—Aiding in the development and control of a list of the names, lengths, representations, and definitions of all data elements used in a software system
- Documentation management—Aiding in the development and control of software documentation
- File management—Providing and controlling access to files associated with the development of software
- Test data management—Aiding in the development and control of software test data.

Project Management. Tools aiding in the management of a software development project commonly provide milestone charts, personnel schedules, and activity diagrams as output. Some specific areas of project management are described in the following items:

- Cost estimation—Assessing the behavior of the variables that affect life cycle cost. A tool to estimate project cost and investigate its sensitivity to parameter changes has this feature.
- Resource estimation—Estimating the resources attributed to an entity. Examples include tools that estimate whether storage limits, input/output capacity, or throughput constraints are being exceeded.
- 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.

- Tracking—Tracking the development of an entity through the software life cycle. Examples are tools used to trace requirements from their specification to their implementation in code.

A GLOSSARY OF SOFTWARE TECHNIQUES

Software techniques are methods or procedures for designing, developing, documenting, and maintaining computer programs or for auditing or managing these activities. The following are nine major categories of software techniques, which can be managerial, methodology, or evaluation oriented:

- Code arrangements—These include structured programming, structured coding, and top-down coding.
- Descriptive documentation—This is documentation external to the code, including HIPO charts, structure charts, and pseudocode, as well as conventional flowcharts.
- Embedded documentation—This is documentation embedded in the code itself, including comments, variable-naming conventions, and indentation conventions.
- Performance documentation—This includes files of test data sets and their results and execution monitor results.
- Programming practices standards—These include adherence to structured programming conventions, avoidance of non-ANSI features of programming languages, modularity requirements, and standard data names.
- Reuse of already written code—This includes the use of COBOL COPY libraries and library subroutines.
- Design—Design techniques include top-down design, structured design, and user design reviews.
- Quality assurance organization and management—These techniques include a quality assurance testing organization independent of the developer, user reviews, and user acceptance tests.
- Programming organization and management—These techniques include chief programmer teams, structured walkthrough, and programming librarians.

CONCLUSION

Software tools and techniques can be as useful to auditors as they are to systems developers. Auditors should familiarize themselves with these tools and techniques before purchasing a new tool or adopting a new technique.

Frederick Gallegos, CISA, CDE, is manager, Management Science Group, U.S. General Accounting Office, Los Angeles and lecturer, Computer Information Systems Department, California State Polytechnic University, Pomona.

Notes

1. B.W. Boehm, et al., *Characteristics of Software Quality* (New York: North-Holland Publishing Company, 1978).
D. Fife, *Software Configuration Management: A Primer for Project Control*, (NBSSP: July 1977) 500–511.
W.E. Howden, "A Survey of Dynamic Analysis Methods" *Tutorial: Software Testing and Validation Techniques*, IEEE Catalog No. EH0138-8 (1978).
W.E. Howden, "A Survey of Static Analysis Methods" *Tutorial: Software Testing and Validation Techniques*, IEEE Catalog No. EH0138-8 (1978).
Office of Software Development, *A Software Tools Project: A Means of Capturing Technology and Improving Engineering*, Report OSD-82-101 (General Services Administration: February 1982).
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).
U.S. General Accounting Office, *Wider Use of Better Computer Software Technology Can Improve Management Control and Reduce Costs*, Report to the Congress, FGMSD-80-38 (April 29, 1980).
U.S. General Accounting Office, *Governmentwide Issues Regarding ADP Software, Areas of Growing Concern—Present and Future*, Internal Report to FGMSD-ADP by Los Angeles Regional Office (November 2, 1979).
2. H. Hecht, *The Introduction of Software Tools.* (NBSSP: September 1982) 500–91.
R. Houghton, "An Inverted View of Software Development Tools," *Proceedings of the 20th Annual Technical Symposium of the Washington, D.C. Chapter of the ACM* (June 1981).
U.S. General Accounting Office, *Improving COBOL Applications Can Recover Significant Computer Resources*, Report to the Congress, AFMD-82-4 (April 1, 1982).
3. U.S. General Accounting Office, *Wider Use of Better Computer Software.*
U.S. General Accounting Office. *Improving COBOL Applications.*

NOTES

33156

NOTES