Traceability

James D. Palmer

ABSTRACT

Traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation, and is critical to the development process because it provides a means of ascertaining how and why system development products satisfy stakeholder requirements, especially for large, complex systems. Traceability provides a path to the validation and verification of stakeholder requirements to assure these needs are met by the delivered system, as well as information on testing procedures, performance measures, nonfunctional characteristics, and behavioral aspects for the delivered system. Both syntactic and semantic information are needed to successfully implement tracing. It is not enough to know the form; it is also necessary to know the substance of the entities to be traced.

However, traceability is often misunderstood, frequently misapplied, and seldom performed correctly. There are many challenges to achieving traceability, particularly the absence of automated techniques to assist in the identification of linkages from requirements to design, or test, or operations needed to trace entities within and across the system development process. One of the particular challenges to providing traceability to and from system level requirements is that it becomes necessary to utilize both the constructs of language semantics as well as syntax.

In this paper, traceability is introduced, and its place in a development process, coupled with the values and pitfalls, are covered. The essentials of traceability are examined together with how to implement tracing within a development life cycle for large, complex systems. Working definitions and related terms are provided to assure common understanding of the terminology and application of tracing in system and software development. A review of contemporary approaches to implement tracing, along with an overview of several of the computer supported software (or system) engineering (CASE) tools that purport to support tracing, are given and future trends are examined.

INTRODUCTION

Successful system development depends on the ability to satisfy stakeholder needs and requirements and to reflect these in the delivered system. Requirements, design, and implementation that are complete, correct, consistent, and error free, play a major role in ensuring that the delivered system meets stakeholder needs. Critical keys to this are understanding and tracing the relationships that exist among system requirements, design, code, test, and implementation. Large-scale complex systems are initiated by stakeholder determination that a need exists that is not met by existing systems. From this beginning, system-level requirements are developed to broadly outline the desired capabilities, which, in turn, are investigated to ascertain feasibility and practicality and examine trade-offs. Once the feasibility and practicality of the desired system have been determined to be necessary and sufficient to launch a new system (or significant modification of an existing or legacy system), design is completed, and systems are constructed, tested, and fielded. It is essential to maintain traceability from the system requirements to operation and maintenance to assure that the delivered system meets the stated organizational needs of the stakeholder.

SYSTEM LIFE CYCLE FOR TRACEABILITY MANAGEMENT

Generally, a system or process development life cycle is followed to produce the desired system. There are many life cycle models [1], and one of the simplest is the system development or waterfall life cycle model depicted in Figure 1. It also serves as the basis for most life cycle models in use today, such as the spiral model, the evolutionary model, and the prototyping model. Within any system development life cycle, requirements must be traced both forward and backward to assure that the correct system is being designed and produced, and that the correct design and production approaches are used.

In the life cycle model of Figure 1, system requirements, usually prepared in natural language, are provided by the stakeholder to the developer. These system requirements, if they exist at all, may be poorly written and only vaguely define stakeholder desires for the new system. This may impact the ability to construct a system that will satisfy the stakeholder. From

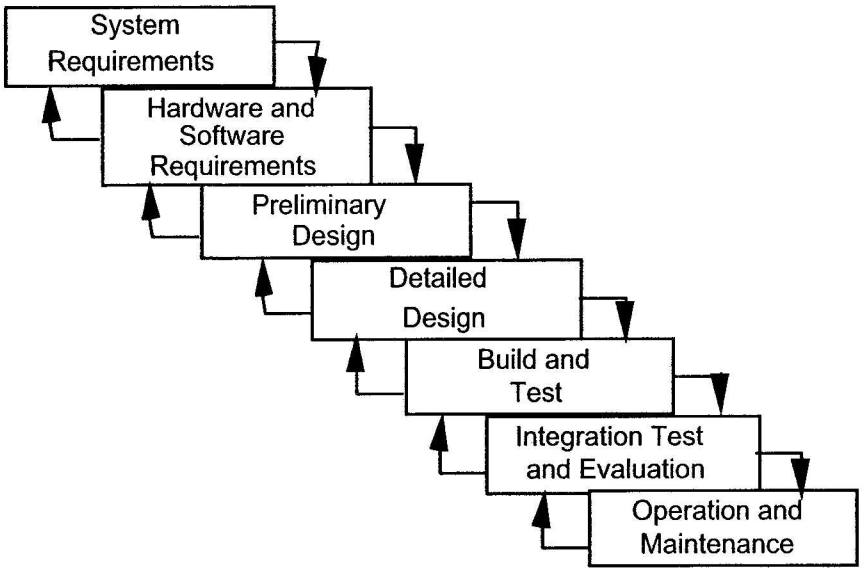

Figure 1. Typical system and software development life cycle.

these system requirements, hardware and software requirements and specifications are prepared. Requirement and specification development are followed by preliminary design; detailed design; construction of the system including hardware and software; system integration, testing and evaluation; and, finally, installation, including operation and maintenance.

These life cycle activities require documentation of needs and outcomes. Each must trace forward to the subsequent activity and backward to the preceding one. Clearly, traceability, both forward and backward, is essential to verify that the requirements of one phase translate to outcomes of that phase, which become the requirements for the next phase, and so on through the development activity. Traceability is equally essential to validate that system requirements are satisfied during operation.

NEED FOR TRACEABILITY

Traceability is essential to verification and validation and is needed to better understand the processes used to develop the system and the products that result. It is needed for quick access to information, information abstraction, and to provide visualization into the techniques used for system development. Traceability is needed for change control, development process control, and risk control. Tracing provides insights to nonbehavioral components such as quality, consistency, completeness, impact analysis, system evolution, and process improvement. It is equally important to have the capability to trace a requirement or design or code module to its origin, as well as test. Stakeholders recognize the value of properly tracing within and across the entities of a system through risk management insights, appropriate integration tests, and the delivery of a project that meets the needs statements of the requirements [2].

Traceability supports assessment of under- or overdesigns; investigation of high-level behavior impact on detailed specifications, as well as nonfunctional requirements such as performance and quality factors. Moreover, traceability supports conflict detection by making it feasible to examine linkages within and across selected entities and by providing visibility into the entire system. Tracing provides the assurance that decisions made later in the system development life cycle are consistent with earlier decisions. Test cases check that coverage for code and integration testing and for requirements validation is provided. Traceability provides the basis for the development of an audit trail for the entire project by establishing the links within and across system entities, functions, behavior, and performance, for example.

Although there is widespread acceptance of the necessity to trace, there is considerable controversy as to the ultimate need, purpose, and cost of tracing from requirements to delivered product. The controversy arises primarily because of the lack of automated approaches to implement the process and the concomitant time and effort that must be applied with any of the presently available support tools. Developers simply do not see the benefits that may accrue to the final product when traceability is fully implemented compared to the time and effort required.

PROBLEMS AND ISSUES CONCERNING TRACEABILITY

Difficulties related to tracing generally revolve around the necessity to manually add trace elements to requirements documents and subsequent work products from software development. Since these products have little or no direct consequence to

the development team, assignment of trace elements generally has a low priority. The benefits of traceability are not seen until much later in the development life cycle, usually during validation testing and system installation and operation, and then primarily by integration testers and stakeholders rather than developers. Additionally, traceability is often misunderstood, frequently misapplied, and seldom performed correctly.

Issues and concerns emanate from the complexity of a project itself that must be confronted when implementing traceability. Each discipline, such as avionics, communications, navigation, security, or safety, may have languages, methods, and tools peculiar to the discipline. This results in a lack of ability to trace across disciplines, which, in turn, may lead to errors in traceability matrices used to provide linkages within and across disciplines. Some of the issues that need to be addressed by the stakeholder and developer at the time of system development include how to apportion projects by discipline, the type and nature of information that should be traced across different disciplines, and the types of tools that can be used to provide consistent and correct traceability across disciplines. Establishing threads across disciplines is also difficult due to language, method, and tool peculiarities.

Currently, there is no single modeling method or language sufficiently rich to represent all aspects of a large complex system and still be understandable to those involved. In tracing information across different disciplines and toolsets, and to provide threads across these, essential system properties and the classification schemes used are needed. Such properties and schemas do not usually exist. Thus, for verification and validation, traceability must always focus on a common denominator: the approved system requirements. Finally, internal consistency of the baseline documentation may not be adequate to support tracing. This latter is usually a significant problem in the modification of legacy systems.

Definition of Terms

There are many terms that describe, delineate, or relate to traceability. Some of these correlate to the "how and why" of traceability, whereas others connect to the outcomes or "what" of traceability. In general, the basic meaning of the terms is first that provided by *Webster's New Collegiate Dictionary* [3], whereas the final meaning is given in the context of systems and software engineering, as an example of usage.

Allocation. The act of distributing; allotment or apportionment, as to assign or apportion functions to specific modules.

Audit. A formal checking of records, to determine that what was stated was accomplished; to examine and verify, as to confirm a stated capability is met in the software product.

Behavior. The way in which a system acts, especially in response to a stimulus; stimulus–response mechanisms; activity or change in reliability across subsystems.

Bottom-up. A design philosophy or policy that dictates the form and partitioning of the system from the basic functions that the system is to perform and moving up to the top-level requirements, as a design policy that provides basic modules followed by top-level constructs.

Classification. A group of entities ranked together as possessing common characteristics or quality; the act of grouping or segregating into classes which have systematic relationships; a systematic grouping of entities based upon some definite scheme, as to classify requirements according to organizational or performance characteristics.

Flowdown. To move or circulate from upper to lower levels, as to trace a requirement from a top level to designs to code to test.

Function. The characteristic action or the normal or special action of a system; one aspect of a system is so related to another so that there is a correspondence from one to the other when an action is taken; an algorithm to provide the equations of motion.

Hierarchy. A series of objects or items divided or classified in ranks or orders; a type of structure in which each element or block has a level number (1 = highest), and each element is associated with one or more elements at the next higher level and lower levels, as when a single high-level requirement decomposes to lower level requirements and to design and code.

Impact analysis. Separation into constituent parts to examine or distinguish contact of one to another; a communicating force; to focus on software changes and the traceable consequences; relating software requirements to design components.

Policy. Management or procedure based primarily on material interest; a settled course or level to be followed for system security.

Requirement. A requisite condition; a required quality; to demand; to claim as by right or authority; to exact, as to demand system performance by the stakeholder.

Thread. To connect; to pass a thread through; string together; to link behaviors of a system together.

Top-down. A design philosophy or policy that dictates the form and partitioning of the system from the top-level requirements perspective to the lower-level design components, as in a design policy for all activities from high-level requirements to design and code.

Top-level requirement. A requisite condition leveled by the stakeholder; a system level requirement for security.

Traceability. The course or path followed; to follow or track down; to follow or study in detail or step by step, especially by going backward over evidence (as to trace requirements from design); to discover or uncover by investigation, as to trace to the source, follow requirements from the top level to design and code and back, or identify and document the allocation/flowdown path (downward) and derivation path (upward) of requirements into the hierarchy. The Department of Defense (DoD) defines traceability in the Standard for Defense System Software Development, DoD-Std-2167A to be a demonstration of completeness, necessity, and consistency. Specifically, DoD-Std-21267A defines traceability as: "(1) the document in question contains or implements all applicable stipulations of the predecessor document, (2) a given term, acronym, or abbreviation means the same thing in all documents, (3) a given item or concept is referred to by the same name or description in the documents, (4) all material in the successor document has its basis in the predecessor document, that is, no untraceable material has been introduced, and (5) the two documents do not contradict one another."

Traceability management. To control and direct; guide; administer; give direction to accomplish an end, as to control and direct tracing from top level through to design and code.

Tree. A diagrammatic representation that indicates branching from an original stem, as software components derived from a higher-level entity to more discrete lower-level entities.

State of the Practice of Traceability

Traceability management applies to the entire development life cycle from project initiation through operation and maintenance, as shown in Figure 2. It is presently feasible to manage tracing using a combination of manual and automated assistance, thus providing some assurance that the development of a system meets the needs as provided by the stakeholder. An essential element of successful traceability management, provided by currently available CASE tools, is the ability to provide links from requirements forward to designs, code, test, and implementation, and backward from any of these activities to requirements once these links have been manually entered into the CASE tool.

Techniques currently in use to establish and maintain traceability from requirements through designs, code, test, and operation begin with manual identification of linkages. These linkages may be subsequently supported by document managers, a database, or CASE tools specifically designed for requirements traceability management.

CONTEMPORARY TRACEABILITY PRACTICES

Traceability has traditionally been accomplished by manually assigning and linking unique identifiers; that is, a sentence or paragraph (or other partition) requirement is assigned a particular alphanumeric reference. This information is subsequently

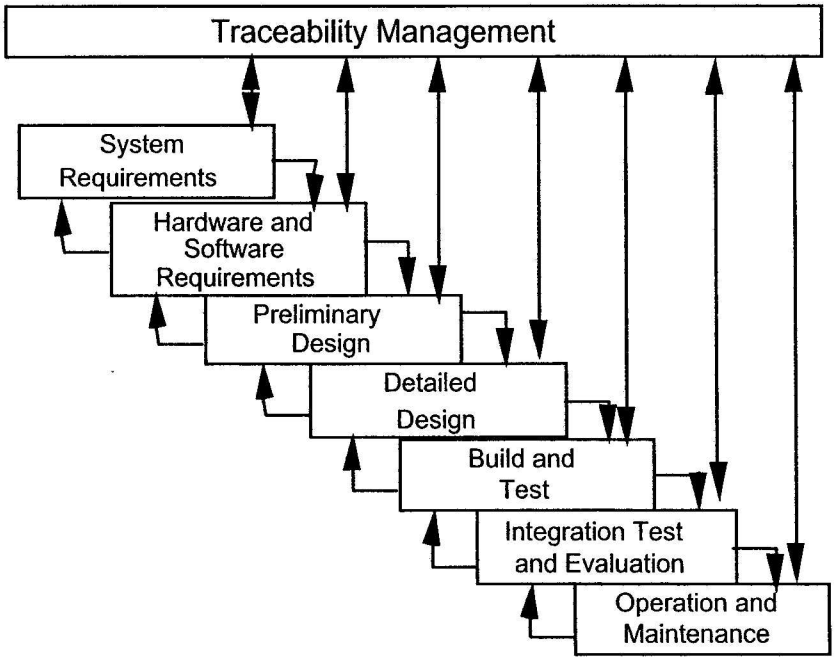

Figure 2. Traceability management across the system development life cycle.

managed in a word processor or database, often through use of a CASE tool. Even with the use of a CASE tool, the initial identification of trace entities and linkages must be accomplished manually. By establishing a unique identification system and following this scheme throughout the life of the project, it is possible to trace these specific entities both forward and backward from requirements to product. This unique identity may be linked within and across documents using manually derived traceability tables to assure full traceability over all aspects of the project.

A typical output of tracing is a traceability matrix that links high-level requirements to each and every other requirement or specification of the system. A typical traceability table for a large complex system is shown in Table 1. In this table, individual requirements in the systems requirements document (SRD) have been manually linked to more detailed system requirements in the systems specification, which in turn have been manually linked to particular specifications in the system segments.

Other matrices or tables may provide more details such as cryptic messages, partial text, critical values, or the entire text. The system represented in the traceability table is configured as in Figure 3. The SRD represents stakeholder input, the SS represents the initial interpretation of these high level requirements by developers, and the segment specifications provide more detailed information to design. The interface control document (ICD) provides linkages for all messages that occur within and across segments.

In most system development programs, there is the added expectation of continuous change in the system as requirements are added, modified, and deleted. Thus, the management of an ever-changing requirements base becomes a very important traceability function, as tracing provides a review of how the system requirements flow down to lower levels and how lower-level requirements are derived from higher levels. These traces may or may not contain information as to why the system is to be partitioned in a particular manner. As new requirements are added or existing ones are updated, deleted, or modified, the management process continues to provide traceability and impact analysis to assure that each of the changes is properly included in the system development process. This provides the major verification and validation procedure to assure stakeholder needs are met.

Traceability is especially critical for the operation and maintenance phase. This is when significant stakeholder changes may be made and change impacts and impact analyses must be performed. Such changes are difficult to trace; however, without tracing it is nearly impossible to ascertain the extent of the full impact of additions, deletions, or modifications to the system.

AN IDEAL PROCESS FOR TRACEABILITY

To understand what must be traced, we need a defined process for developing system architectural views and classification schemes, as well as processes for specifying and verifying the products to be constructed. This is generally provided by the stakeholder in concert with the developer. The development of these views is necessary to partition the project for design and construction.

An ideal traceability process consists of the steps of identification, architecture selection, classification, allocation, and flow-down as depicted in Figure 4. The process begins with the identification of requirements at the system level, specification of system architecture, and selection of classification schema. Following this, allocations are made in accordance with the selected schema. Following allocation, the requirements flow down to design, code, and test. This top-down approach has proven most effective in the management of traceability for large-scale, complex projects.

However, this approach is basically a manual activity that requires significant investment of time and effort on the part of skilled personnel. The outcomes represent a system hierarchy along the lines of the classification structure used for the architectural allocations. It is also necessary to provide threads through the various behavioral and nonbehavioral aspects of the project to complete the traceability process. These thread paths are manually assigned using approaches such as entity–relation–attribute diagrams. For example, tests are threaded back to requirements through code and design.

Once the system hierarchy, the architecture, and classification schema have been defined, identified system requirements are assigned to the top-level block of the hierarchy. At this time, they are added to the traceability database for storage, retrieval, and reuse. After appropriate analyses, these requirements are broken down and flow down into more detailed require-

Table 1. Traceability matrix for multisegment system

SRD	SS	Segment 1	Segment 2	Segment 3	ICD
3.1.2.1	3.3.4.5	3.2.2.5.6	3.5.3.2		3.1.4.6.7
	3.3.4.6	3.2.2.5.7			3.1.4.6.8
		3.4.5.6.2			3.1.4.6.9
3.4.3.1	3.6.7.2	3.5.2.5.1	3.7.4.3.1	3.6.4.5.2	3.3.2.4.5
	3.8.4.3		3.7.4.3.2		3.3.2.4.7

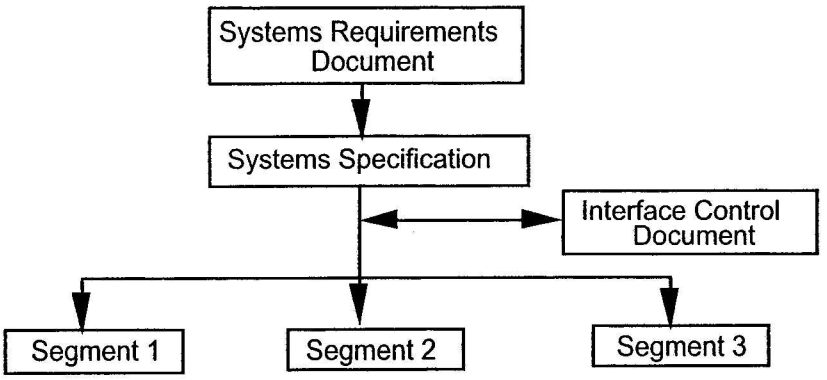

Figure 3. Typical requirements classification schema for a large, complex system.

ments for each of the lower-level blocks to which the requirement was allocated, as was shown in the example of Figure 3. The higher-level requirements are sometimes referred to as parents and the lower-level ones as children. Change notification should be rigorously traced to determine the impact of such activities on changes in cost, schedule, and feasibility of system design and implementation, on tests that must be conducted, and on support software and hardware.

ACTUAL PRACTICE FOR IMPLEMENTING TRACEABILITY

In actual practice, tracing is a labor intensive and aggravating task. Domain experts follow a break-down process that is similar to that depicted Figure 3. Once appropriate systems architectures are identified, a classification schema or schemas for purposes of allocation of requirements to system specific architectures is prepared and requirements are assigned to specific units. As examples of the types of classification schemes used, one may be centered on functional aspects of the project, such as navigation, communications, or threat assessment; another may concentrate on performance and security; while yet another may be focused on stakeholder organization. It is not feasible to enumerate, a priori, all the ways in which the project may need to be partitioned and viewed; thus, traceability becomes a continuous process as perspectives change and as requirements change. To validate these various views, there is only one common basis from which to form trace linkages: the system requirements.

The next step, after receipt of the requirements documents and delineation of the system architecture, is to determine the nature of the tracing to be accomplished. Several options are feasible; these include working with statements that contain "shall," "will," "should," "would," or similar verbs; or with entire paragraphs; or the total set of statements provided by the stakeholder. The strongest selection is "shall" statements, which may be the only contractually acceptable designation for a requirement. This is followed by the development of classification schemes according to function, data object, behavior, organization, or other approaches. Once the option(s) has been selected, the requirements are parsed according to the option and assigned a unique identity. For example, if "shall" has been selected as the option, sentences with "shall" as a verb are collected and are identified sequentially, while also retaining the original identification system provided by the stakeholder. This new identification system is maintained throughout the life of the project.

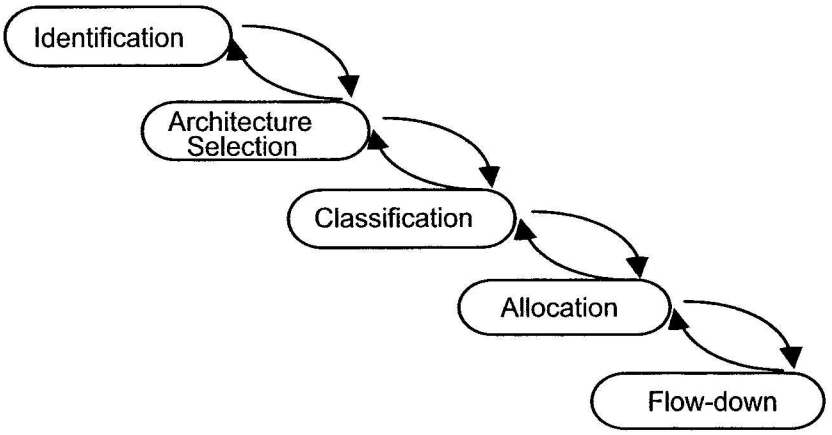

Figure 4. The ideal traceability process.

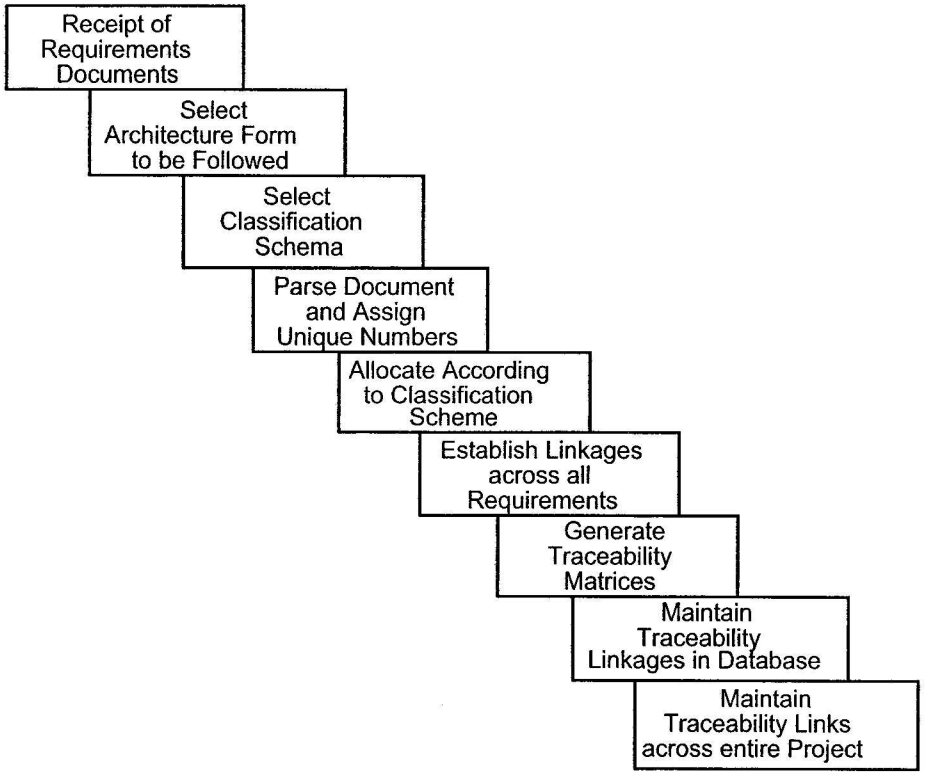

Figure 5. Steps to accomplish traceability.

Syntactic and semantic information are both necessary to perform tracing. Language semantics are needed to assure that the trace is related to the meaning or context of the requirement or set of requirements, whereas syntax is necessary to trace to a specific word or phrase, without regard to meaning or context. Integration of both constructs is required to provide for full traceability from natural language statements to the other steps as shown in Figure 2. Manual verification of outcomes is required to assure compliance with the intent and purpose of the tracing activity.

Next comes allocation according to the classification scheme. This, likewise, is a manual task, even with automated assistance from one of the available CASE tools, as most of these tools require the operator to physically establish the links from one entity to another for traceability. All linkages must be designated and maintained and traceability matrices are generated from these outcomes. If a CASE tool has been used that supports generation of traceability matrices, these are created automatically; otherwise, these matrices must be manually prepared. These steps are depicted graphically in Figure 5. These results are usually stored in a traceability database. The traceability linkages are subsequently designated and maintained across the entire development project from design to code to test to operation and maintenance.

RETURN ON INVESTMENT FOR TRACEABILITY

It is not feasible to measure the return on investment (ROI) for traceability. Although most of the costs associated with implementation can be documented, the benefits are quite difficult to ascertain unless comparative case studies are conducted. Costs of implementation include the investment of time and effort of domain experts to provide system architectural perspectives and classification schema, the initial cost of acquiring CASE tools to manage requirements traceability, and the expended costs of training and maintenance in the use of such tools. Due to the manual approaches required to establish architectural perspectives, classification schema, allocation, linkage, and system maintenance, fixing costs, while manageable, is a difficult task. These costs may be either estimated or accounted for with some degree of accuracy. This may be done for an ongoing project or by estimating the time, effort, capitalization costs, and expended costs involved.

The benefits are largely intangible and are related to the avoided costs associated with rework and possible failure of the product to satisfy stakeholders. To estimate the benefits, it would be necessary to prepare various scenarios, simulate the outcomes due to failure of various aspects of the development process, and estimate the value of avoiding these failures. Risk factors must also be taken into consideration in calculation of the potential benefits, including the potential that the project will not meet stakeholder needs. Assessing benefits without comparative analyses is generally not possible. Generating this infor-

mation is considered to be unfeasible due to the costs of running such experiments and the need to develop realistic scenarios that may or may not ever be replicated in actual practice.

CURRENT TRACEABILITY TOOLS

Typical of the currently available automated (or semiautomated) assistance approaches to traceability are those that provide for traceability through a variety of syntactic language components: hypertext linking, unique identifiers, syntactical similarity coefficients, or combinations of these. In hypertext linking, the "hotword" or word/phrase to be linked to other requirements is manually identified and entered into the hypertext tool. Links are automatically made and maintained by the tool to provide forward and reverse traceability for the word selection. In the unique identifier approach, an identifier is assigned that remains with the individual requirement throughout the life of the project. To assure traceability, this unique identifier provides a "fan-out" capability within a hierarchical structure such that one system-level ("A" level) requirement may be the parent to many "B" level requirements, which, in turn, may be the parents for great numbers of "C" level requirements, as depicted in Table 1. Use of syntactic similarity coefficients ascertains whether or not a predefined set of words of a given requirement are found in another requirement. When the degree of similarity is above a predefined threshold, the two requirements in question are said to trace.

There are problems with each of these approaches. They do not consider the semantics or context in which the tracing is to occur. Hypertext linking finds the searched text without regard to the placement in the text and without regard to the way in which the words are used. Use of a unique identifier provides access only to those requirements so identified, with no perspective as to meaning or context. Syntactic similarity coefficient traceability is like hypertext linking in that it is indiscriminate as to the meaning and context of the requirement to be traced.

Commercially available requirements tools utilize straightforward traceability links that must be manually developed to relate requirements to other requirements and to design, code, and implementation. Current methods for implementing traceability with these commercial tools generally involve the manual provision of links within and across documents and then automated management of these documents. Traceability links are used to establish the one-to-one, one-to many, many-to-one, or many-to-many relationships that may exist, as may be seen from Table 1. As noted previously, linkages are not automatically established by tools during the development process, but must be manually generated. From this point, automated assistance is provided by the tool to manage traceability.

At present, there are no standards available to support tools for traceability, which has led to the development and use of a large number of commercial tools, each with differing methods, as well as proprietary tools developed by certain industries because it is considered to be a competitive advantage for most large, complex projects. A number of commercially available tools have been developed to support traceability and a number of general CASE tools provide support to traceability management, especially from requirements forward to design, code, test, and operation. One of the common activities for all tools is manual development of architectural perspectives and classification schemas. Another common feature is the need to manually establish the initial linkages within and across all traceable entities. Once the initial linkages have been established, these tools effectively and efficiently manage a traceability database or word processor document.

COMMON TOOL CHARACTERISTICS

There are some common tool characteristics that are deemed to be minimal to provide support for traceability. The tool must be well understood by and be responsive to users and match the characteristics of the development environment used by the developers. Tools must also accept and utilize the data that is provided in the form provided. In addition, the tool must be flexible, capable of operation in an automated assistance mode to support various activities and services such as active and passive data checking; batch as well as on-line processing; addition, deletion, and modification of a requirement; customization to specific domain applications; dynamic database structure for change management; and a tailorable user interface. Traceability tools will never be fully automated, as human decision making is essential to the establishment of classification schema and system architecture designation. Human interaction and decision making are both desirable and necessary to maximize the interaction of the stakeholder/developer in the development of the project.

COMMERCIAL CASE TOOLS FOR TRACEABILITY

Some commercially available tools have been developed for traceability link information expressed by a single discipline within a single phase, whereas others have been developed specifically to link requirements to other activities within the development life cycle. Cadre TeamWork for Real-Time Structured Analysis (CADRE) is a tool that operates in a single discipline within a single phase. Tools that link information from multiple disciplines and phases include Requirements Traceabili-

ty Manager (RTM) (Marconi Corporation) [4], SLATE (TD Technologies) [5], and DOORS (Zycad Corporation) [6]. These tools use an entity–relation–attribute-like schema to capture information on a system database, either relational or object-oriented, and enable formation of queries about traceable entities and for report generation. RTM uses a relational database structure to capture information and provide management, whereas DOORS provides an object-oriented database for management of information. SLATE follows a multiuser, client–server, object-oriented approach that provides dynamic representation of the system as it evolves.

Another method used by commercial tool vendors is the hypertext approach. In this approach, keywords or phrases are identified as being indicative of traces. These are linked through hypertext throughout the document or set of documents that comprise the requirements. An example of a tool that uses this approach is Document Director [7].

Some general-purpose analysis tools are also used for tracing. Some of the more robust tool sets include: Requirements Driven Design (RDD-100 by Ascent Logic) [8], which is used to document system conceptual models, and Foresight [9]. which is utilized to maintain a data dictionary and perform document system simulation.

Other tools and techniques that support requirements traceability include Software Requirements Methodology (SREM), Problem Statement Language/Problem Statement Analyzer (PSL/PSA), N2 charts, Requirement Networks (R-Nets), and ARTS (a database management system for requirements). Not all of the CASE tools support requirements traceability; however, most do support some form of requirements management.

FUTURE TRENDS AND CONCLUSIONS

The future of traceability support lies in the development of the capability to deal directly with requirements in natural language, the ability to provide automated assistance to allocation of requirements to various architectural and classification systems, and the management of these. With this automated assistance, it becomes feasible to provide for and manage a traceable baseline for the entire system.

The following issues are being addressed in ongoing research programs:

- Automated allocation of entities to architectures and classifications
- Traceability that is independent of methods used to develop architectures and classifications
- Tracing product attributes from requirements to the lowest levels

Several research programs are working on the problems associated with natural language. The two addressing traceability are from George Mason University and Trident Systems. The Center for Software Systems Engineering at George Mason University has developed and applied an automated assistance approach to the problems of allocation of entities to architectures and classification, called the Automated Integrated Requirements Engineering System (AIRES) [10]. Trident Systems intends to develop a CASE tool called RECAP (Requirements Capture) which is intended to manage natural language requirements [11].

AIRES provides an assessment framework and techniques for integrated application of both semantic and syntactic rules for effective, efficient, and comprehensive identification of traceable and nontraceable requirements in large, complex multiple-segment systems. The framework provides for the categorization of requirements in classification structures through the application of a diverse combination of rules and procedures, each of which applies unique combinations of both semantic and syntactic classification rules and tables for the categorization of requirements. These serve as the basic building blocks of the assessment framework and may be applied either singly or in combination. AIRES supports automated development of linkages that may be transferred electronically to commercially available traceability tools such as RTM for management of a requirements database and report generation. AIRES is presently available in prototype form and has been utilized in support of several large, complex systems for traceability support [12].

RECAP, presently a conceptual design, is intended to provide a set of interfaces that permit the operator to manipulate natural language requirements. RECAP proposes to combine the information management and extraction capabilities of information retrieval system approaches with knowledge-based rules. It also intends to provide sequential and string search access to any portion of the document set. Quick access to information is proposed through keywords, sentence identifiers, or rule-based queries. The user will be required to provide information for resolution of ambiguity, mistakes in statements, and addition of missing items. RECAP is intended to aid the user in making these decisions [11].

Information linked by these tracing tools is not dependent upon a model or discipline. It is possible to link entities as needed; for example, it may be desirable to link the estimated footprint, weight, and power usage of a piece of computer equipment (stored in a hardware modeling tool) to the estimated throughput and memory requirements for a piece of software (stored in a software modeling tool). To efficiently use these tracing tools, it is necessary to automatically transfer the information cap-

tured to CASE tools used downstream in the development life cycle. This is accomplished by tracing system definitions, system development processes, and interrelationships across system units.

Although tracing from origination to final product is a difficult and arduous, manually intensive task at the present time, advances in technology should soon be commercially available to assist in automated allocation and classification procedures. These advances will make the traceability task much more reasonable, feasible, and supportable for large, complex system developments due to the automated assistance provided for allocation and classification, the most labor-intensive aspects of tracing. In each of the approaches, the CASE tool provides automated assistance to tracing, but requires human operator inputs only for decision-making activities. These tools represent a significant advance over the present state of the practice for traceability.

REFERENCES

[1] Sage, Andrew P. and Palmer, James D., *Software Systems Engineering,* Wiley, 1990.

[2] White, Stephanie, "Tracing Product and Process Information when developing Complex Systems," in *CSESAW '94,* July 19–20, 1994, pp. 45–50, NSWCDD/MP-94/122.

[3] *Webster's New Collegiate Dictionary,* Sixth Edition, G. & C. Merriam Co., Springfield, MA, 1951.

[4] *RTM-Requirements and Traceability Management, Practical Workbook,* GEC-Marconi Limited, October, 1993.

[5] Nallon, John, "Implementation of NSWC Requirements Traceability Models," in *CSESAW,* White Oak, MD, July 19–20, 1994, pp. 15–22, NSWCDD/MP-94/122.

[6] Rundley, Nancy and Miller, William D., "DOORS to the Digitized Battlefield: Managing Requirements Discovery and Traceability," CSESAW, White Oak, MD, July 19–20, 1994, pp. 23–28. White, Stephanie, "Tracing Product and Process Information when Developing Complex Systems," in *CSESAW,* White Oak, MD, July 19–20, 1994, pp. 45–50, NSWCDD/MP-94/122.

[7] *Document Director—The Requirements Tool,* B.G. Jackson Associates, 17629 E. Camino Real, Suite 720, Houston, TX 77058, 1989.

[8] "RDD-100-Release Notes Release 3.0.2.1, October, 1992," Requirements Driven Design, Ascent Logic Corporation, 180 Rose Orchard Way, #200, San Jose, CA 95134, 1992.

[9] Vertal, Michael D., "Extending IDEF: Improving Complex Systems with Executable Modeling," in *1994 Annual Conference for Business Re-engineering,* IDEF Users Group, Richmond, VA May, 1994.

[10] Palmer, James D. and Evans Richard P., "An Integrated Semantic and Syntactic Framework for Requirements Traceability: Experience with System Level Requirements for a Large Complex Multi-Segment Project," in *CSESAW '94,* July 19–20, 1994, pp. 9–14, NSWCDD/MP-94/122.

[11] Hugue, Michelle, Casey, Michael, Wood, Glenn, and Edwards, Edward, "RECAP: A REquirements CAPture Tool for Large Complex Systems," in *CSESAW,* White Oak, MD, July 19–20, 1994, pp. 39–44, NSWCDD/MP-94/122.

[12] Palmer, James D. and Evans Richard P., "Software Risk Management: Requirements-Based Risk Metrics," in *Proceedings of IEEE 1994 International Conference on SMC,* October 2–6, 1994, San Antonio, TX.