SYSTEM REQUIREMENTS AND SPECIFICATIONS

System requirements and specifications are essential, for without them there is no sure knowledge of the system to be built. They help us understand the relationships that exist within and across systems development, design, and implementation and are critical to the development process. They provide a means with which to validate and verify user needs, execute testing procedures, understand performance measures, and determine nonfunctional and functional character-

istics. Keys to understanding the relationships that exist among system requirements, design, construction, test, and implementation are to be found in these. Correct and accurate system requirements and specifications are critical to the delivery of a system that is on time and within budget.

These statements are investigated to ascertain feasibility and practicality and to examine tradeoffs. After the feasibility and practicality of the desired system have been determined, the resulting statements are analyzed for errors, difficult or incomplete concepts are prototyped, and the final product is transformed to formal or semiformal specification languages. The emphasis on error detection recognizes that between 50 and 80% of the errors found in delivered systems can be traced to errors in the interpretation of requirements (1–3).

System requirements and specifications are essential to the verification and validation that user needs are properly interpreted and to ensure that the outcomes are those intended. They provide visualization and understanding into the techniques necessary for system development and are used to validate the impact of changes, provide process control, and enable early risk management. Insights are provided to quality, consistency, completeness, impact analysis, system evolution, and process improvement. Traceability of requirements and specifications to their origin is equally needed. The value of correct requirements for a system are realized through the identification and resolution of risk, development of appropriate integration tests, and successful delivery of the final product (2). They provide the basis for the development of an audit trail for the project by establishing links within and across system entities, functions, behavior, performance, and the like.

PROBLEMS AND ISSUES CONCERNING SYSTEM REQUIREMENTS AND SPECIFICATIONS DEVELOPMENT

Before we examine how to develop system requirements and specifications, let's look at some of the problems. Although the necessity to provide adequate system requirements and specifications for development is widely accepted, there is considerable controversy as to the ultimate need, purpose, and cost of performing these activities. This arises primarily as a result of the need to acquire knowledge and the associated technical difficulties, the lack of automated approaches to implement the processes, and the concomitant time and effort that must be used to apply any of the presently available support tools. Difficulties generally revolve around elicitation and development of information from users and/or existing systems and lie at the interface between the system developer and the user. Transformation to the exact language of specification is another source of problems. There are also technical difficulties that relate to factors such as hardware, performance, capacity, and interfaces.

Issues and concerns often emanate from the complexity of a project. Each discipline (e.g., environmental monitoring systems, automated fingerprint systems, sediment extraction systems, C^3I systems, and health monitoring systems) has language, methods, and tools peculiar to the discipline. The same language constructs are not used across disciplines. This leads to potential errors in the development of requirements used to provide linkages within and across disciplines. Establishing threads across disciplines is difficult because of language, method, and tool peculiarities. Generally, system requirements and specifications are stated in natural language, which means a high potential for ambiguities, conflicts, inconsistencies, and incompleteness. The types of issues and errors that are typically found in system requirements and specifications include:

1. Conflict within and across requirements
2. Lack of consistency of requirements individually and in clusters of similar statements
3. Incompleteness across requirements and their clusters
4. Inability to determine the ripple impact of adding new requirements to existing systems
5. Issues related to storage and retrieval
6. Degree of volatility in the requirements generation
7. Failure to provide traceability throughout the life of the project (including maintenance)
8. Technically and economically unfeasible requirements

The key to success is the resolution of the problems and issues at the time of system initiation, not after deficiencies are noted during design or testing.

SYSTEM REQUIREMENTS AND SPECIFICATIONS PROCESSES

The development of the system requirements and specifications should move according to a stated plan. This plan minimally must include the development of system objectives, refinement of these objectives, and development and refinement of system constraints and variables; express as concisely as possible a high-level functional model; formulate a design and implementation strategy; and document the outcomes of these activities as specifications. The plan must include the essential functions necessary for developing system requirements and specifications. These are:

1. Elicitation
2. Organization of materials to form a logical order
3. Analyses of statements for problems such as consistency, errors, omissions, risk, and completeness
4. Modeling difficult to understand concepts
5. Transforming informal natural language to formal or semiformal specification languages

System requirements and specifications development requires approaches, such as those shown in the process model of Fig. 1, that address the following activities:

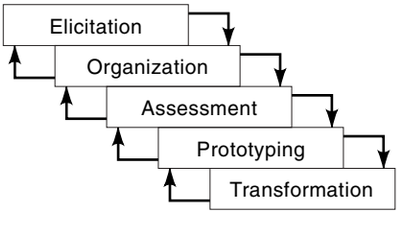

Figure 1. Process model for system requirements and specification development.

1. Developing needs and desires of users
2. Determining potential problems or difficult factors (e.g., incompleteness, inconsistencies, redundancies, and ambiguities)
3. Encouraging alternative approaches to solve problems (e.g., discovery, prototyping, and simulation)
4. Providing for alternative courses of action at each step in development and appropriate methods to support the Course of Action (COA) (e.g., elicitation, assessment, and transformation)
5. Identifying personnel needed for each activity
6. Stimulating use of support tools (e.g., multimedia elicitation tools, decision support tools, classification techniques, transformation techniques, and prototyping)
7. Supporting development of high-quality products to meet time and cost constraints
8. Enabling management control

Domain knowledge plays an essential role in this process model. Analysts eliciting requirements information from domain experts, users, existing documentation, legacy systems, and current systems need to be domain experts or have a working knowledge of the domain for which the system is to be constructed. Domain knowledge might be represented as a taxonomy of concepts against which clusters of data may be compared to identify missing information, conflicts, or inconsistencies. For organization, assessment, and prototyping, classification processes used require domain knowledge to identify potential problems and issues, solutions, and architectures and structures. Transformation to specifications is a knowledge-intensive activity. Finally, documentation of domain knowledge ensures that baselines are consistent with domain semantics.

Elicitation

The initial task is most formidable: elicitation. The needs that are elicited are based on the objectives of the user, the constraints and variables that have an impact on potential solutions that can be implemented, as well as the impact of nonfunctional requirements such as management activities, quality factors, operating environment, and other nonbehavioral aspects. Development constraints must be identified and recognized. These might include the need to accommodate a particular target machine for system operation, the timing necessary for response between interactions, the need for real-time interrupts, or the length of time available for system development. Experience in elicitation will assist in acquiring this knowledge. Activities performed during elicitation include the following: (1) collection of statements in a database, (2) formulation of system-level issues, and (3) interpretation of the requirements that result. There are semiautomated approaches to assist elicitation that includes decision support systems, group decision support systems, questionnaires, interviews, prototypes, and capture of all information in a database. A typical elicitation activity can include convening groups of experts on all aspects of the new system and use of questionnaires and interviews with system personnel to ascertain needs and constraints and use the Delphi Techniques should the participants be geographically distributed. The elicitation activity may also employ presentations, large group meetings, and small groups supported by decision support systems.

Organization

The organization of system requirements and specifications is essential to overall system development and supports a variety of activities. Statements that have been collected in a database may be grouped to reflect common characteristics, similarity, or other meaningful attributes. These clusters use classification and clustering techniques to group statements together based on characteristics defined by user and developer. Typical attributes may include factors such as risk, design constraints, real-time functions, testing, or performance.

Classification has its basis in the utilization of categories, criteria, or taxonomies that are in turn based on specific characteristics (e.g., behavioral and nonbehavioral attributes), around which organization is to take place. Clusters that center on one or more degrees of functionality or one or more nonbehavioral aspects may be generated. Classification and clustering techniques also aid in identifying orthogonality or interdependence (i.e., those modules with the greatest degree of independence or the greatest degree of interdependence). Examples of classification include use of keywords to describe the contents of a document, categorize information, or catalog artifacts (4). This has two major advantages: *data simplification* and *information discovery.* Clustering objects into groups effectively summarizes information about each object increasing efficient organization and retrieval, reducing complexity and increasing understandability (5). Discovery leads to knowledge about possible structure and relationships of the objects (6). Clustering may result in a classification scheme (or taxonomy) that describes groups and the membership of each object in the cluster. The resulting classification scheme may be used to classify individual objects into appropriate clusters.

Clustering structures can be categorized as dissection, partitioning, hierarchical, and clumping. Dissection refers to the process of dividing the objects into clusters that are based on proximity. Partitioning divides the objects into disjoint groups, where each object belongs to one and only one group. Hierarchical structures are generally depicted as tree diagrams, where each level of the tree partitions the objects into disjoint groups. Clumping permits an object to belong to more than one group, producing "clumps" or overlapping groups.

Assessment

When the organization is complete, review and analysis of omissions, commissions, constraints, or issues are initiated. This process may employ some of the same methods used in the organization function (e.g., classification and clustering) as well as techniques to determine risk, uncover errors of commission (e.g., ambiguities), and errors of omission (e.g., conflicts). These methods are used to assist in the detection and identification of correct statements, errors, issues, risk, and incomplete sets of functions.

Potential problems, omissions, and commissions are discovered by searching the database using a variety of techniques or queries. For example, testing is crucial to the operation of a real-time life critical system, thus search criteria would center around all aspects of testing. These might include reliability, mean time to failure, timing sequences, and

similar features. Another major activity is the detection and correction of errors within and across requirements using classification and clustering techniques. Comparison lists of ambiguous terminology, risk factors, performance characteristics, and the like, make it possible to detect errors related to factors such as conflict, inconsistency, incompleteness, and ambiguity. Detection is enabled by using rules and/or predefined taxonomies (classification structures). Following detection, errors of these types are presented to the user for resolution.

Finally, new statements are developed from the information prepared and documented during assessment. This includes definition of criteria for evaluation and assessment of any changes that may occur.

Prototyping

Candidates for prototyping are identified by the organization and assessment processes. Prototyping is an approach to increase the utility of system requirements and specifications information for user and developer. It is used to clarify and/or generate user needs and to transform these to system specifications. It assists in understanding the operating environment and modeling potential operation environment changes and assisting understanding of intended and desired functionality in terms of what the system should accomplish. Prototyping is also used to assist developers examine various structural approaches to design (see MODELING AND SIMULATION).

Prototyping is important in determining elements of risk, as well as statements that are incomplete or difficult to understand. Many techniques for prototyping are in common use, including animation and simulation, which are supported by several automated tools. Prototyping is a common activity and is included in most development approaches.

Transformation to Specifications

System models are prepared from the preceding outputs and transformed to formal or semiformal languages of specification or design. Various approaches may be used to construct these models. Characteristics of natural languages hold the keys to these approaches (i.e., syntactic and semantic structure of the language as the basis of linguistic approaches) (7).

Several automated approaches to transformation use either syntactic and/or semantic information. Generally they require statements presented in a form amenable to the particular technique. Most involve using databases, term frequency identification, lexical affinity identification, and some application of semantics. Term-frequency-based techniques have been used for many years to develop key words to describe book and article content and to develop abstracts (8). Statistical methods have been used to analyze text based on the frequency distribution of terms in the text. A primary concept is that the importance of a term is proportional to the frequency of occurrence. A method used in conjunction with term frequency is lexical affinity to identify concepts within a text. Generally, the concept is used to identify pairs of terms such as verb-noun, adjective-noun, and adverb-verb pairs to provide semantic information. Together these provide specific detailed specifications based on entity-action pairs (i.e., the object acted upon by a particular function).

Finite state machines (FSM), such as state transition diagrams (STD), are used to model system dynamic behavior (9). A key in developing FSMs is the identification of dynamic behavior concepts, such as events, conditions, actions, transitions, and states. A pattern-matching and knowledge-based approach that uses both syntactic and semantic knowledge to process statements may be used. A primary feature of this approach is to establish correspondence, using a semantic scheme, between linguistic patterns and system dynamic behavior concepts, such as events, conditions, actions, transitions, and states. A behavior concept classification structure may be used to identify concepts and a parsing grammar and associated pattern rules and mapping rules to provide automated support to identification of dynamic behavior concepts.

Management as Part of the Process

System requirements and specifications are generally grouped in two categories: functional and nonfunctional. Functional represents the way the internal system components interact with their environment. They intend to present a precise set of properties, or operational needs that the system must satisfy. Nonfunctional results from restrictions or constraints that are placed on the types of solution paths that may be considered for system design. Nonfunctional includes quality factors, management processes, cost, and development tools.

System requirements and specifications stipulate functional requirements and performance objectives for the system. Management specifications include:

1. Operating environment concerns
2. Overall design concept or objectives
3. Trustworthiness of requirements, including quality factors
4. Maintenance concerns, including the need for system evolution over time
5. Economic resources and time allocation

A systems management plan, or set of systems management needs, is developed and incorporated as part of the process. Thus, there are two components: systems management and technical or functional system requirements and specifications. Technical or functional aspects are analyzed to produce exact specifications, whereas systems management needs are analyzed to produce explicit management strategies for system development.

The activities to be carried on include:

1. Evaluation of system functional requirements and specifications for completeness and feasibility
2. Evaluation of the system-level requirements and specifications for compatibility with other systems operational in the user environment
3. Exploration of the user environment, including existing systems, to determine how the new system might be best deployed
4. Identification of other organizations and units that have interfaces with the new system
5. Evaluation of user-stated available resources, including funding and time available

6. Development of a systems management strategy to incorporate these considerations

Major objectives to be met by the systems management plan include definition and scope of effort, specification of required resources, development of specific schedules, and cost estimates. This information, in conjunction with technical information, is used by the user for a go/no go decision relative to continuing the development effort. This decision is made on the basis of (1) technical requirements feasibility, (2) system quality or trustworthiness factors, (3) system maintenance, (4) evolution of products over time, and (5) systems management needs.

CHARACTERISTICS OF SYSTEM REQUIREMENTS AND SPECIFICATIONS THAT INFLUENCE THE PROCESS

There is a set of characteristics most important to the user. The qualities in this list distinguish those system requirements and specifications that will produce the desired system. These characteristics, which are user-centered and straightforward, are summarized in Table 1.

The qualities most important to the developer relate to correctness and realizability. These are of critical importance in determining whether system requirements and specifications represent an accurate representation of need (1,10,11). Such a set reduces the possibility of errors, and therefore the risk of misinterpretation during later activities of the life cycle. These qualities are summarized in Table 2.

Errors introduce the potential for multiple interpretations, which may cause disagreements between users and developers and may result in costly rework, lawsuits, or unproductive systems. Risk management must be initiated as a necessary component to produce a quality system (see DECISION THEORY).

Brooks (12) feels that it is extremely difficult for users to articulate "completely, precisely and correctly" an accurate set of system requirements and specifications without first iterating through versions of the system. Different versions allow users to "visualize" how the system satisfies their needs and helps to "stimulate" unstated needs. Developers and users often view system issues from very different perspectives. Errors occur because the user may not clearly understand system needs and/or may use imprecise or ambiguous terms to describe these needs. Developers may lack the necessary communication skills needed to elicit system needs. Developers may not be acquainted with the domain and are unable to determine whether system requirements and specifications reflect system needs. Users and developers may speak different languages, lacking a common ground to communicate. The language of the user is usually specific to the domain, whereas the language of the developer is based on the technology used to attempt to solve user needs. Users may not be able to visualize how a system will satisfy their needs (12). Developers may not be able to represent the system via paper-based requirements in a form that users can understand and that is the result of a transformation process that may not accurately record the intent of the user (1). This disparity in understanding must be bridged to have a successful transfer of user needs to requirements specifications.

Table 1. System Characteristics from a User's Perspective

Characteristic	Definition
Realizable	User needs achieved in delivered system
Accurate	Reflects user needs and desires
Affordable	System realization is possible within available resources
On time	System completion and operation within timeframe determined

Table 2. System Characteristics from a Developer's Perspective

Characteristic	Definition
Complete	Everything system is required to do
Consistent	No conflicts
Correct	Accurate representation of user need
Feasible	Achievable and within scope of project resources
Maintainable	Changes achieved easily, completely, and consistently
Precise	Stated clearly and specifically
Testable	Test cases developed for each function
Traceable	Origin and responsibility is clear
Unambiguous	Only one interpretation
Understandable	Comprehensible to user and developer
Validatable	Authenticated at project completion
Verifiable	Confirmed at the end of each development activity

TOOL SUPPORT FOR MANAGEMENT OF SYSTEM REQUIREMENTS AND SPECIFICATIONS

Management activities for system requirements and specifications apply to the entire process depicted in Fig. 1, using a combination of manual, semiautomated, and automated techniques. Essential elements of successful management provide methods for the usual management functions as well as for error detection, risk management, and change control. These are provided, in part, by currently available computer assisted software (or system) engineering (CASE) tools, which include the ability to link requirements forward to designs, code, test, and implementation and backward from any of these activities to system requirements and specifications. Techniques currently in use establish, maintain, and provide assistance to development beginning with elicitation and continuing through to transformation. This assistance is essential for large complex systems because the sheer number of statements that must be elicited, organized, analyzed, and transformed may run into the several thousands.

Contemporary Requirements Practices

The development of system requirements and specifications has suffered from lack of formal or standard processes supported by appropriate automated tools. The most commonly used approach has been that of entering requirements information in a database and using the database capabilities to organize and manage requirements.

An organized approach to the activities noted in Fig. 1 includes the following activities:

1. Formulate system-level concepts and determine requirements issues.

1.1. Users identify needs, constraints, and variables, including budget and any operational and legal requirements. With developers, users determine what the system requirements and specifications should be, how they should be stated, and how they are to be derived.
1.2. Identify objectives of user groups and ways to determine how system-level objectives can be met.
1.3. Identify specifications that are affected by existing systems and determine the degree to which the existing system may be retained.
2. Organize and analyze issues and select the approach.
2.1. Organize elicited information and review and analyze any constraint or issues related to system-level requirements and specifications to include technical, operational, and economic aspects.
2.2. Define criteria for evaluation and assessment of requirements.
3. Interpret information gathered previously in 2.
3.1. Assess the requirements statements for potential issues and errors and for risk.
3.2. Develop a validation plan for evaluation of the delivered product.
3.3. Review performance demands.
3.4. Analyze cost and benefits.
4. Develop prototypes as appropriate.
4.1. Identify candidates for prototyping.
4.2. Develop and operate prototypes and interact with user to ascertain whether a proper solution has been developed.
4.3. Archive the results in the requirements database.
5. Transform statements to formal or semiformal specification languages.
5.1. Maintain a database of the initially transformed requirements. Use *basis, version 0* as the database for the system development throughout the life of the project.
5.2. Maintain subsequent versions in a database.

Continuous change in the system is common as needs are added, modified, and deleted, and management and maintenance of a *basis* set becomes essential. As new needs are added or existing ones are updated, deleted, or modified, the process continues to provide analysis to ensure that each change is properly included in the system development process and that new problems, if introduced, are resolved. This provides the major verification and validation procedure to ensure that user needs are met. Change notification is traced to determine the impact of such activities on cost, schedule, and feasibility of system design and implementation and tests that must be conducted.

Tool Characteristics

A wide variety of semiautomated and automated CASE tools assist various activities involved in the development and analysis of requirements. The tools range from those that use a variety of methods to those that are single purpose only. Several tools that represent a number of approaches to requirements development are examined in the next section. No single tool has either captured the market nor is deemed to be up to the general task of requirements development. The various approaches that are used in these tools have many common characteristics that include:

1. A means for information analysis
2. An approach for functional representation
3. An approach for nonfunctional representation
4. A definition of interfaces
5. A mechanism for partitioning
6. Support for abstraction
7. A database for requirements statements
8. Representation of physical and logical views

There are common tool characteristics that are deemed to be minimally necessary to provide support for management. The tool(s) must be well understood by and be responsive to users and match the characteristics of the development environment used by the developers. Tools must accept and use the data that are supplied in the form provided. In addition, the tool(s) must be flexible and capable of operating 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. Management tools for this process will never be *fully* automated because human decision making is essential to the establishment of classification schema and system architecture designation. Human interaction and decision making is both desirable and necessary to maximize the interaction of user/developer in development of the project.

Typical of the currently available automated (or semiautomated) assistance approaches to requirements management are tools that provide support through a variety of syntactic language analyses that include hypertext linking, syntactical similarity coefficients, preselected or predefined verbs and nouns, 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 linkages to the words or phrases selected. Use of syntactic similarity coefficients ascertains whether or not a predefined number of words of a given statement are found in another statement. When the value of similarity is above a predefined threshold, the two statements in question are said to be similar. Using preselected or predefined objects, nouns, and verbs permits the user to describe information in a context-free manner without affecting the ultimate application.

There are problems with each of these approaches. Hypertext linking finds the search text without regard to the placement in the text and without regard to the way in which the words are used. Syntactic similarity coefficient is like hypertext linking in that it does not pay attention to the meaning and context of the requirement. Using preselected or predefined objects provides access only to those statements so identified without searching for a meaning other than the predefined designation.

For commercially available requirements tools, data must be input manually, and manually developed information must be used to generate relationships across requirements. This is followed by automated management of information after input is made to the tool database. At present, no standards are available to support tools for requirements management. This 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. One common aspect for all tools is the manual development of architectural perspectives and classification schemes.

CASE Tools for the Requirements Process

Some commercially available tools that use a single method within a single phase have been developed for requirements management information, whereas others have been developed specifically to link requirements to other activities within the development life cycle. SADT, a product of Softech, Inc., provides assistance through the use of activity diagrams to accomplish system definition, requirements analysis, and design (13). General-purpose system analysis tools also are used for requirements management. Some of the more robust of these tool sets include Requirements Driven Design (RDD-100) by Ascent Logic (14), which is used to document system conceptual models, and Foresight (15) which is used to maintain a data dictionary and document system simulation. Other tools and techniques that support requirements management include Software Requirements Methodology (SREM) (16); and ARTS, a database management system for requirements (17). Yet another method used by commercial tool vendors is the hypertext approach. In this approach, keywords or phrases are identified and linked by hypertext throughout the document or set of documents. An example of a tool that uses this approach is Document Director (18).

There are also tools that are intended for the explicit purpose of requirements tracing; however, they also provide for requirements management. These tools link information from multiple disciplines and phases and include Requirements Traceability Manager (RTM) (Marconi Corporation) (19), SLATE (TD Technologies) (20), and DOORS (Zycad Corporation) (21). These tools use an entity-relation-attribute-like schema to capture information in a system database, either relational or object-oriented, to enable formation of queries about traceable entities, and to generate reports. 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, user/server, object-oriented approach that provides dynamic representation of the system as it evolves.

Several other CASE tools are in various stages of development. They will provide support to the requirements process, even though they have not seen wide-spread application beyond the group that did the original research and investigation. These include the following efforts: (1) the work at Southern California University by Johnson et al. (21) on a tool named Acquisition of Requirements and Incremental Evolution of Specifications (ARIES). This tool supports requirements analysis through evaluation and codification of the results in formal specifications. (2) Mays et al. (22) of IBM have developed a requirements planning process described as the Planning and Design Methodology (PDM) that includes requirements acquisition, problem definition, development of external functional descriptions to address the problems, and provision for system and product designs from the descriptions. (3) Rich and Waters (23), in work at MIT, describe the development of the Programmer's Apprentice, a knowledge-based approach to development. The intent is to take a set of disorganized imprecise statements and provide a coherent requirements representation. (4) Faulk et al. (24) at the Software Productivity Consortium have developed an approach they call the Consortium Requirements Engineering method (CoRE). In this method a coherent approach is taken for specifying real-time requirements information. (5) Kaindl (25) has produced a hypertext tool for semiformal representation of natural languages. The tool is named Requirements Engineering through Hypertext (RETH). (6) Palmer and Evans (26) at George Mason University have developed and applied a syntactic and semantic analyzer they call Advanced Integrated Requirements Engineering System (AIRES). In this approach, natural language statements are entered into a database and automatically analyzed for similarities, redundancies, inconsistencies, ambiguities, and the like. The output is a database of organized requirements. These represent the diverse activities being undertaken in an attempt to provide an automated CASE tool for requirements engineering and management.

FINAL OBSERVATIONS

The future of system requirements and specifications support lies in the development of the capability to deal directly with requirements in natural language (the language of choice of most users), the ability to provide automated assistance to allocation of requirements to various architectural and classification systems, and the management of these activities to ferret out errors, issues, and risks. The following areas presently are being addressed through ongoing research and development programs in both industries and universities:

- Automated allocation of entities to architectures and classifications
- Requirements management that is independent of methods used to develop architectures and classifications
- Derivation of explicit attributes that are consistent from the highest-level system requirements to the lowest levels of decomposition

Because of the very nature of this area, which is a human-intensive activity, the process will never be a fully automated one. Just the elicitation activity itself is sufficiently human-intensive to justify both users and developers to spend many hours to ensure that the system to be constructed is properly represented by the requirements. The other activities require less human interaction; however, this interaction is necessary to ensure that we do the best we can do for the product.

From origination to final product, system development is a difficult, arduous, and manually intensive task at the present time. Advances in technology should provide some relief to assist in automating allocation and classification procedures and generally to provide more assistance to the user and de-

veloper. However, the manual aspects of examining each of the needs for major large systems is not likely to be replaced by automated processes soon. This is desirable because it is truly necessary for users and developers to exercise human judgment on each and every need to ascertain whether it is the correct one for the system desired and it is correctly stated to avoid interpretation problems.

BIBLIOGRAPHY

1. J. D. Palmer and N. A. Fields, An integrated environment for software requirements engineering, *Software,* 80–85, May 1992.
2. A. P. Sage and J. D. Palmer, *Software Systems Engineering,* New York: Wiley, 1990.
3. B. W. Boehm, Improving software productivity, *Computer,* **20** (9): 43–57, 1987.
4. A. D. Gordon, *Classification,* New York: Chapman and Hall, 1981.
5. J. D. Palmer and Y. Liang, Indexing and clustering of software requirements specifications, *Inf. Decision Technol.,* **18**: 283–299, 1992.
6. M. R. Anderberg, *Clustering Analysis for Application,* New York: Academic Press, 1973.
7. J. D. Palmer and S. Park, Automated support to system modeling from informal software requirements, *Sixth Int. Conf. Software Eng. Knowledge Eng.,* Latvia, June 20–23, 1994.
8. G. Salton and M. J. McGill, *Automatic Information Organization and Retrieval,* New York: McGraw-Hill, 1983.
9. *IEEE Software Engineering Standards,* 1987.
10. H. Gomaa, A software design method for real-time systems, *Commun. ACM,* **27** (9): 657–688, 1984.
11. A. M. Davis, *Software Requirements: Objects, Functions, and States,* Englewood Cliffs, NJ: Prentice-Hall, 1993.
12. F. P. Brooks, Jr., No silver bullet: Essence and accidents in software engineering, *Computer,* **20** (4): 10–19, 1987.
13. D. T. Ross and K. E. Shoman, Jr., Structured analysis for requirements definition, *IEEE Trans. Softw. Eng.,* **SE-3** (1): 69–84, 1977.
14. *RDD-100—Release Notes Release 3.0.2.1, October, 1992, Requirements Driven Design,* Ascent Logic Corporation, San Jose, CA, 1992.
15. M. D. Vertal, Extending IDEF: Improving complex systems with executable modeling, *1994 Annu. Conf. Bus. Re-eng., IDEF Users Group,* Richmond, VA, May 1994.
16. M. W. Alford, SREM at the age of eight: The distributed computing design system, *Computer,* **18** (4): 36–46, 1985.
17. R. F. Flynn and M. Dorfman, The automated requirements traceability system (ARTS), *AIAA Third Computers in Aerospace Conf.,* AIAA, pp. 418–428, 1981.
18. *Document Director—The Requirements Tool,* B.G. Jackson Associates, Houston, TX, 1989.
19. J. Nallon, Implementation of NSWC requirements traceability models, *CSESAW 94,* White Oak, MD, NSWCDD/MP-94/122, pp. 15–22, July 19–20, 1994.
20. N. Rundley and W. D. Miller, DOORS to the digitize battlefield: Managing requirements discovery and traceability, *CSESAW 94,* White Oak, MD, NSWCDD/MP-94/122, pp. 23–28, July 19–20, 1994.
21. W. L. Johnson, M. Feather, and D. Harris, Representing and presenting requirements knowledge, *IEEE Trans. Softw. Eng.,* **SE-18** (10), 853–869, 1992.
22. R. G. Mays et al., PDM: A requirements methodology for software system enhancements, *IBM Systems J.,* **24** (2): 134–149, 1985.
23. C. Rich and R. Waters, The programmers apprentice: A research overview, *Computer,* **21** (11): 10–25, 1988.
24. S. Faulk et al., The CoRE method for real-time requirements, *Software,* **9** (5): 22–33, 1992.
25. H. Kaindl, The missing link in requirements engineering, *ACM SIGSOFT Softw. Eng. Notes,* **18** (2): 30–39, 1993.
26. J. D. Palmer and R. P. Evans, An integrated semantic and syntactic framework for requirements traceability: Experience with system level requirements for a large complex multisegment project, *CSESAW 94,* White Oak, MD, NSWCDD/MP-94/122, pp. 9–14, July 19–20, 1994.

JAMES D. PALMER
George Mason University

SYSTEMS. See BUSINESS DATA PROCESSING.