Doing Requirements Right the First Time

Theodore F. Hammer, *Goddard Space Flight Center*
Leonore L. Huffman and Linda H. Rosenberg, *Unisys Federal Systems*

The criticality of correct, complete, testable requirements is a fundamental tenet of software engineering. The success of a project, both functionally and financially, is directly affected by the quality of the requirements. Also critical is the complete requirements-based testing of the final product. This article addresses three critical aspects of requirements: definition, verification, and management. Project data collected from NASA Goddard Space Flight Center (GSFC) by the Software Assurance Technology Center (SATC) will be used to demonstrate these concepts and explain how any project, large or small, can apply this information.

It is generally accepted that requirements are the foundation upon which the entire system is built. Also accepted is that requirements verification and validation is needed to assure that the functionality specified in the requirements has been delivered. However, all too often, requirements are not satisfied, which means you fix what you can and accept that certain functionality will not be there. A better approach is to get the requirements right the first time. Complete, concise, and clear requirements will give the implementer a precise blueprint with which to build the system. Getting the requirements right is not done by magic but through the application of tools and metric analysis techniques in requirements specification, requirements verification, and requirements management.

Because both parties must understand the requirements that the acquirer expects the provider to contractually satisfy, specifications are usually written in natural language. The use of natural language to prescribe complex, dynamic systems has at least two severe problems: ambiguity and inaccuracy. Many words and phrases have dual meanings that can be altered by the context in which they are used. To define a large, multidimensional capability within the limitations imposed by the linear, two-dimensional structure of a document can obscure the relationships between individual groups of requirements. The first part of this article looks at types of requirements-specification terminology, some of which can contribute to ambiguity and misinterpretation.

Requirements-based testing is critical to the implementation of software systems. Automated tools, if properly used, open the door to assess the scope and potential effectiveness of the test program. A wealth of information can be obtained through proper implementation of a database that tracks requirements at each level of decomposition and the tests associated with the verification of these requirements. From this database, the project can gain important insight into the relationship between the test and requirements. The second part of this article outlines some of the important insights into NASA project test programs developed from analyses of this type.

Requirements management is a volatile, dynamic process. The skill with which project workers maintain, keep current, track, and trace the project's set of requirements affects every phase of the project's software development lifecycle—including maintenance. Months or years before project completion, effectively managed requirements determine how, when, and how expensively completion will take place.

Before processing requirements, the schema for the requirements management database must be developed. The final portion of this article describes some critical issues identified by the SATC that are needed to effectively manage requirements databases. It also discusses lessons learned on how to effectively design and maintain requirements databases.

Development Environment

To demonstrate how metrics can provide the insight needed to get the requirements right, data from a large NASA project, Project X, will be used. This anonymous project implements a large system in three main incremental builds.[1] The development of these builds is overlapping, design and coding of the second and third builds starting before the completion of the first build. Each build adds new functionality to the previous build and satisfies a further set of requirements.

NASA defines requirements in four levels of detail. "Mission-Level Requirements" for the spacecraft and ground system are System Level 1; they are the highest level and rarely, if ever, change. Level 1 requirements then undergo decomposition to produce "Allocated Requirements," called Level 2; these also are high level, and change should be minimal. Level 2 requirements are then divided into subsystems, and a further level is derived in greater detail, hence, "Level 3: Derived Requirements." Generally, contracts are bid using this level of requirements detail. Each requirement in Level 2 traces bidirectionally to one or more requirements in Level 3. "Detailed Requirements" are found in Level 4; these are used to design and code the system. There also is bidirectional tracing between Level 3 requirements and Level 4 requirements. To verify the requirements, two stages of testing are used. System tests are designed to verify the Level 4 requirements, then acceptance tests are used to verify the Level 3 requirements.

Requirements Specification

The importance of correctly documenting requirements has caused the software industry to produce a significant number of aids [1] to create and manage requirements specification documents and individual specifications statements; however, few of these aids help evaluate the quality of the requirements document or the individual specification statements. The SATC has developed a tool to parse requirements documents. The Automated Requirements Measurement (ARM) software was developed to scan a file that contains the text of the requirements specification. The software searches each line of text for specific words and phrases that are indicated by the SATC's studies to be an indicator of the document's requirements specification quality. ARM has been applied to 56 NASA requirements documents, and seven measures have been developed.

- Lines of Text – Physical lines of text as a measure of document size.
- Imperatives – Words and phrases that command that something must be done or provided, e.g., shall, must, will, should, is required to, are applicable, and responsible for. The number of imperatives is used as a base requirements count.
- Continuances – Phrases that follow an imperative and introduce the requirements specification at a lower level for a supplemental requirements count, e.g., as follows, following, listed, in particular, and support.
- Directives – References provided to figures, tables, or notes, e.g., figure, table, for example, and note.
- Weak Phrases – Clauses that are apt to cause uncertainty and leave room for multiple interpretations or a measure of ambiguity, e.g., adequate, as applicable, as appropriate, as a minimum, be able to, be capable, easy, effective, not limited to, and if practical.
- Incomplete – Statements within the document that have "TBD" (to be determined) or "TBS" (to be supplied).
- Options – Words that seem to give the developer latitude to satisfy the specifications but that can be ambiguous, e.g., can, may, and optionally.

		Lines of Text	Imperatives	Continuances	Directives	Weak Phrases	Incomplete	Options
56 NASA Documents	Minimum	143	25	15	0	0	0	0
	Median	2,265	382	183	21	37	7	27
	Average	4,772	682	423	49	70	25	63
	Maximum	28,459	3,896	118	224	4	32	130
	Standard Deviation	759	156	99	12	21	20	39
	Project X	34,664	1,176	714	873	13	480	187

Table 1. *Requirements specification analysis example.*

It must be emphasized that the tool does not attempt to assess the correctness of the requirements specified. It assesses individual specification statements and the vocabulary used to state the requirements and also has the capability to assess the structure of the requirements document.[2]

To see how this tool would be used to assess the quality of a requirements document, the Project X Level 3 requirements document was analyzed using the ARM tool. Table 1 shows the results in contrast to statistics from the 56 previous documents.

From this analysis, several things become clear. First, the document shows some strengths: There appear to be a good number of imperatives, and the number of weak phrases is low compared to the family of NASA documents processed through the ARM tool to date; however, the document shows some significant weaknesses. The document has a large amount of text given the number of imperatives. This indicates a wordy document, which can obscure the requirements and prevent them from being clear and concise. The document also has a large number of incomplete requirements that contain TBDs and TBSs—on this point alone, the document can be judged not ready for use. Also, this document has a large number of options, which increases the uncertainty about what is required of the system to be developed. Options leave decisions about the system to the implementers, many times without sufficient direction or instruction about option selection criteria. As a result, the implementation varies widely, from some of the options to none.

Engineers have always wanted to get the requirements right in the specification, but there has been little available in terms of analysis tools to allow them to visualize the quality of the documentation. Now, with the ARM tool, the quality aspects of the documentation can be visualized, and necessary action can be taken to improve the documentation.

Requirements Volatility

Requirements testing is vital to getting the requirements right. Many times it is overlooked in favor of testing code, but if the software does not conform to the requirements, it is just as defective as if it were full of bugs. Good requirements testing relies on a good verification program, which in turn must rest on an analysis of requirements volatility and linkage. An effective verification program comprises a test profile made after linkage of requirements is analyzed and after considering requirements volatility. Again, data from Project X will demonstrate the utility of metrics in requirements verification.

Requirements stability impacts the verification effort because testing cannot be planned or designed when the requirements are continually in a state of flux. Figure 1 shows how metrics provide insight into requirements stability while also demonstrating the

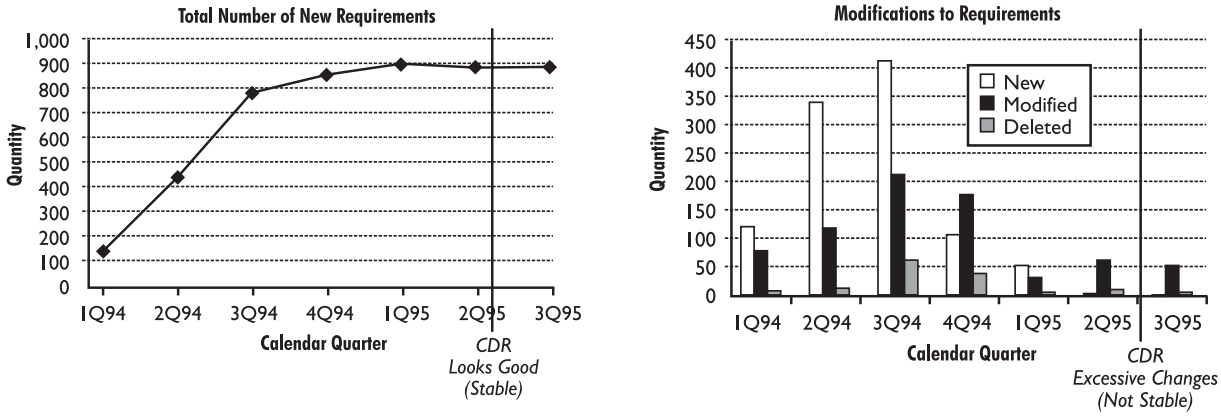


Figure 1. *Requirements stabilization—volatility. Combination of* both *views indicates risk area: Requirements are* not yet *stable.*

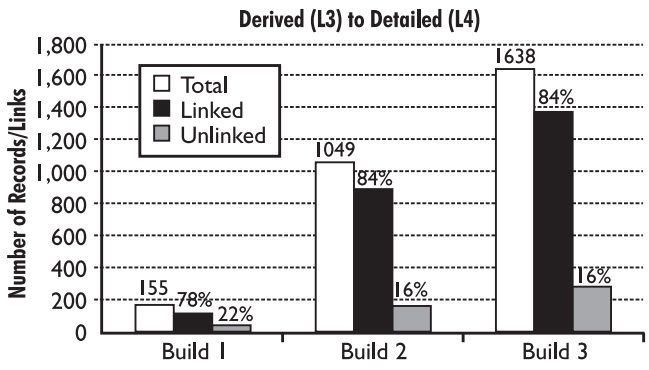
importance of examining an issue from more than one angle. According to the graph on the left side of the figure, the total number of requirements has stabilized in time for the Critical Design Review (CDR); however, the graph on the right shows that the requirements are *not* stable—modifications and deletions are still taking place. This almost constant change in the requirements will endanger the verification program.

Requirements stability can also be viewed in terms of the completeness of requirements traceability. Requirements traceability is the linkage of the requirements at one level to the requirements at the next lower level. Missing linkage may indicate missing requirements. Figure 2 shows the linkage of Level 3 requirements to Level 4 requirements. In all cases, there is missing linkage (white bar of graph) between Level 3 and Level 4 requirements, indicating that the Level 4 requirements may be incomplete for a CDR held for any one of these builds.

Requirements Verification

The objective of an effective verification program is to ensure that every requirement is tested, the implication being that if the system passes the test, the requirement's functionality is included in the delivered system [1, 2]. The traceability of the requirements to test cases therefore needs to be assessed. It is expected that a requirement will be linked to a test case and may well be linked to more than one test case, as shown in Figure 3 [3, 4].

Figure 2. *Requirements traceability.*



The important aspect of this analysis is to determine which requirements have not been linked to any test cases.

Figure 4 shows the traceability of requirements to test cases for Project X around the CDR time frame for Build 2. The profiles show several problems. First, the poor traceability between the requirements and test cases for Build 1 indicates that the requirements management tool was not used effectively early in the project lifecycle. Second, there seems to be a mix-up in the test priorities by the implementer. The test program for Build 3 is farther along than that for Build 2, even though Build 2 will be developed and tested before Build 3. Resources may have been inappropriately allocated to the development of the test program for Build 2. Last, the test program for the Level 4 requirements is behind that for the test program for the Level 3 requirements. Again, this is backward. The first tests to be executed should be those for the Level 4 requirements—the system tests—and after that, tests for the Level 3 requirements—the acceptance tests—should be executed.

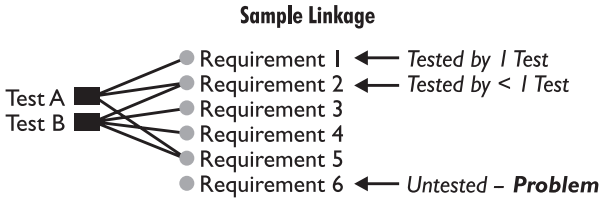


Figure 3. *Requirements verification – trace to test linkage.*

Requirements Test Cases

Not only is it important to understand whether all the requirements are linked to test cases, the character of the test program also needs to be understood. This can be done by looking at the profile and relationship of requirements to test cases. Figure 5 shows an expected profile of unique requirements per test case based on data from NASA projects [5].

This profile shows the expectation that there will be a large number of requirements tested by only one test case and that there will be some requirements that will be tested by

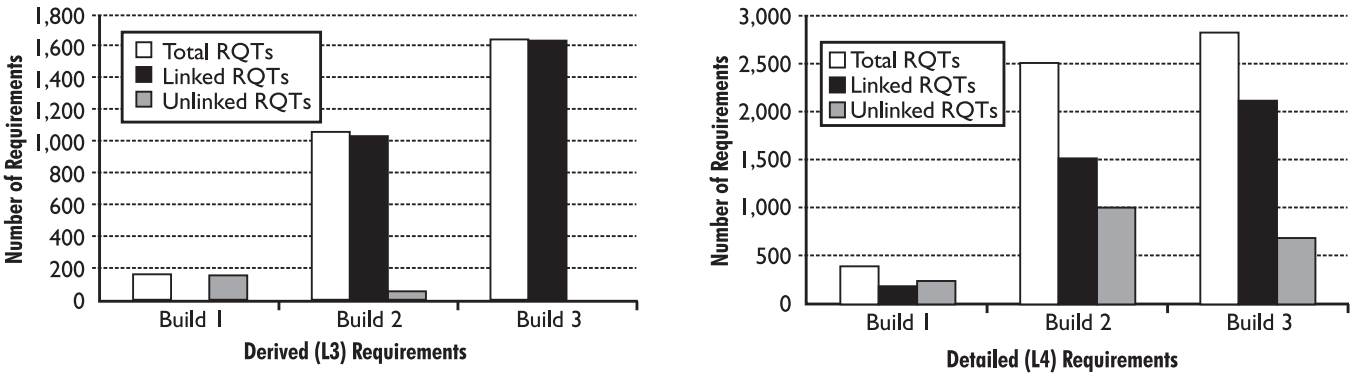


Figure 4. *Requirements verification trace to test.*

multiple test cases. It is expected that the upper bound of multiple test cases will range in the double-digits because more complicated requirements may require different test cases to thoroughly verify all aspects of the requirements. However, there is a logistical limit on the number of test

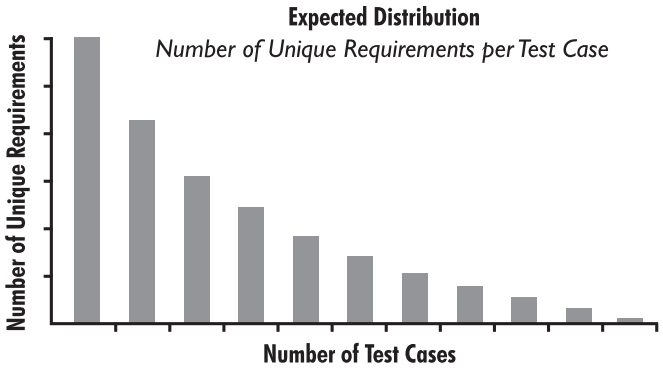


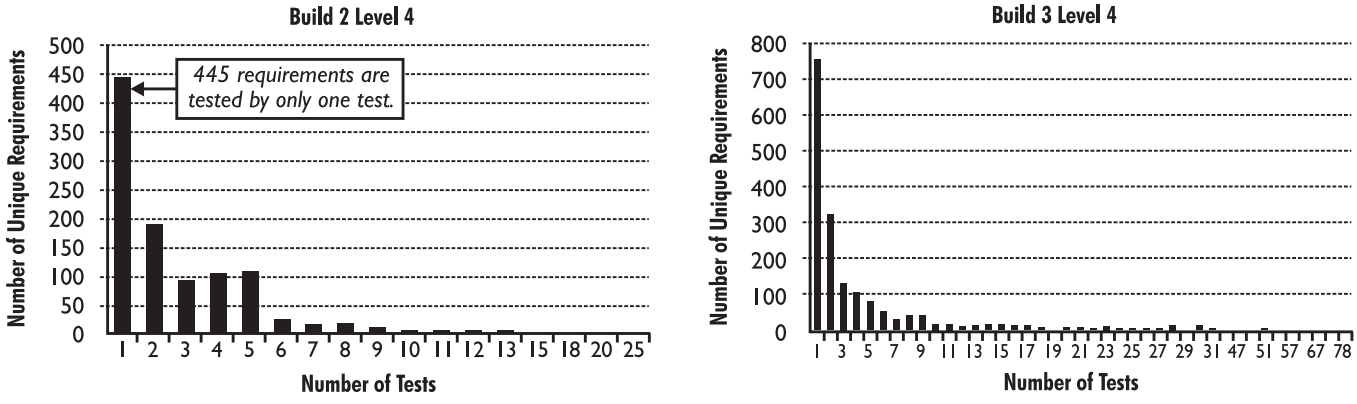
Figure 5. *Test program characterization tests per requirement. Some requirements will be tested only once or can be group tested. Complex requirements need multiple tests.*

cases that can be performed; as the number of test cases increases, the difficulty in verifying the requirements increases due to the complication in data analysis, understanding the results of the multiple tests cases, and understanding the impact of multiple test case results on the verification of the requirements. Figure 6 shows the requirements-to-test-case profile for Project X. There is a good indication that a large number of requirements are covered by just one test, which makes for a simple, easy-to-evaluate test program for a significant part of the system requirements. However, in several instances for both Build 2 and Build 3, there are several tests for unique requirements. Notice that for Build 2, one requirement has been linked to 25 test cases, and in Build 3, that same requirement is linked to 51 test cases. This large number of test cases may well make it impossible to verify that these requirements have been implemented.

Requirements Management Tools

The use of tools to aid in requirements management has become an important aspect of system engineering and design because of the size and complexity of development efforts. The tools that requirements managers use for automating the requirements engineering process have reduced the drudgery in maintaining a project's requirements set and added the benefit of significant error reduction. Tools also provide capabilities far beyond those obtained from text-based maintenance and processing of requirements. Requirements management tools are sophisticated and complex—the nature of the material for which they are responsible is finely detailed, time-sensitive, highly internally dependent, and can be continuously changing. Tools that simplify complex tasks require skill and a thorough understanding of their capabilities if they are to perform effectively over the lifetime of a project [6].

Figure 6. *Test program characterization tests per requirement.*



There are many requirements management tools from which to choose. These range from simple word processors to spreadsheets to relational databases to tools designed specifically for the management of requirements, such as DOORS (Quality Systems & Software, Mount Arlington, N.J.) or RTM: Requirements Traceability Management (Integrated Chipware, Inc., Reston, Va.). The key to selecting the appropriate tool is the functionality provided and the capability to develop metrics from the data.

The metric capability of the tool is important. It should be noted that most of the metrics presented in this article were developed from the data contained in a requirements management tool. Table 2 shows a comparison of the metric capability associated with the various tools. Clearly, the relational database and requirements management tool provide the capabilities needed to effectively support requirements management.

Tool selection is only part of the equation. A thorough understanding of the tool's capabilities and the management processes that will use the tool also is necessary. The tool should not be plugged into the management processes with no thought to the impact on the tool's capabilities. Adjustments may be needed in the management processes and employment of the tool to bring about an efficient requirements management process. Briefly, Project X had the following problems with the requirements management tool.

Project X's focus on establishing a requirements management process was influenced by project organization. The way the project chose to use the tool appeared reasonable on the surface but was fraught with flaws stemming from inexperience, and ultimately it worked against clear management. Specifically, many classes (tables or relations) mirrored organizational structure instead of a single class existing for each development phase. With a multiple test class and requirements class approach, there was a natural tendency for the organizations to "improve" the data schema definitions assigned to them. The result was losses in data integrity and restricted access to important information about the requirements. Some information that should have been available to all project organizations became specific to a particular organization [6].

Because multiple classes were implemented at the test-by-build level, fields were duplicated to each of the test classes; common information then became self-contained within each class. However, confusion developed between the test organizations as to which one was responsible for populating common data, all of which lead to inconsistent data entries and prevented effective data mining [7]. Also, due to the multiple-class approach, links that traced requirements to tests also became extensive and conflicting. Because the project decided to organize the database schema along the lines of the organization, it was necessary to provide the traceability of requirements to requirements and test case to requirements by connections between many classes. This resulted in a complex, undocumentable traceability relationship between the system test cases and the two levels of requirements. Most requirements tools are designed to use minimal classes and effect decomposition within a class, not between classes [6].

	Word Processor	Spreadsheet	Relational Database	Requirement Tool
Document Size	X			
Dynamic Changes Over Time				X
Release Size	X	X	X	X
Requirement Expansion Profile			X	X
Requirement Types	X	X	X	X
Requirement Verification			X	X
Requirement Volatility	X	X	X	X
Test Coverage			X	X
Test Span			X	X
Test Types	X	X	X	X

Table 2. *Requirement repository metric capabilities.*

Conclusion

To do requirements right the first time, the following components must be present: quality documentation, a complete and appropriately structured verification program, and effective requirements management. Quality documentation is complete, clear, and concise—concepts that used to be considered ethereal and difficult to measure or visualize. Now, with the advent of tools like ARM, metrics can be developed to show the strengths and weaknesses of the requirements documentation. The completeness of the verification program used to be the only aspect that was easily understood. Now, through the use of metrics, project workers not only can gain insight into the completeness of the test program but also can understand the overall characteristics of the verification program. Effective requirements management now demands the appropriate use of management tools or databases or both through the development lifecycle. Through their use, the development of metrics to gain insight into the nature of the requirements is enabled. Metrics provide a powerful tool to gain insight into each of these areas and give the project the ability to get the requirements right the first time. It is no longer a dream but a reality. ◆

About the Authors

Theodore F. Hammer is the NASA manager for the SATC at NASA's GSFC. He is responsible for managing software quality assurance activities for selected spacecraft implementation projects. Prior to this position, he was a member of the Assur-

ance Management Office, where he was responsible for managing the overall quality assurance activities for specific ground system implementation projects, with special emphasis on software quality assurance. He has more than 22 years experience in software development and assurance. He joined NASA GSFC in 1989, where he supported NASA Headquarters Software Management Assurance Program and participated in the review of the early versions of the military software development standard, MIL-STD-498, as well as NASA software development and assurance standards and guidebooks. He has a bachelor's degree in electrical engineering from the University of Maryland and is a member of the American Society for Quality.

Goddard Space Flight Center
Code 302
Greenbelt, MD 20771
Voice: 301-286-7475
Fax: 301-286-1701
E-mail: thammer@pop300.gsfc.nasa.gov

Lenore L. Huffman is a principal engineer with SATC. She has more than 14 years software engineering and quality assurance experience. She is expert in the design, implementation, and execution of data collection, database structures, and metrics reporting and analysis. She also is expert in the design and use of state-of-the-art database reporting systems. She has extensive experience automating configuration management and problem reporting systems and adapting their capabilities to satisfy unique project requirements. She has successfully planned, designed, and implemented software quality assurance projects. Prior to joining the SATC, she developed metrics for software at the Space Telescope Institute, and while working at a chemical research center, was awarded several U.S. patents. She has a master's degree in business administration.

Goddard Space Flight Center
Code 300.1, Building 6
Greenbelt, MD 20771
Voice: 301-286-0099
E-mail: Lenore.L.Huffman.1@gsfc.nasa.gov

Linda H. Rosenberg is an engineering section head at Unisys Government Systems in Lanham, Md. She is contracted to manage the SATC through the System Reliability and Safety Office in the Flight Assurance Division at NASA GSFC. She is responsible for risk management training at all NASA centers, and the initiation of software risk management at NASA GSFC. As part of the SATC outreach program, she has presented metrics and quality assurance papers and tutorials at GSFC, the Institute of Electrical and Electronic Engineers (IEEE), and the Association for Computing Machinery (ACM) local and international conferences. She also reviews for ACM, IEEE, and military conferences and journals. She holds a doctorate in computer science from the University of Maryland, a Master's of Engineering Science in computer science from Loyola College, and a bachelor's degree in mathematics from Towson State University. She is a member of IEEE, the IEEE Computer Society, ACM, and Upsilon Pi Epsilon.

Goddard Space Flight Center
Code 300.1, Building 6
Greenbelt, MD 20771
Voice: 301-286-0087
E-mail: Linda.H.Rosenberg.1@gsfc.nasa.gov

References

1. Brooks Jr., Frederick P., "No Silver Bullet: Essence and Accidents of Software Engineering," *IEEE Computer*, Vol. 15, No. 1, April 1987, pp. 10-18.
2. Hammer, T., L. Huffman, L. Rosenberg, W. Wilson, L. Hyatt, "Requirements Metrics for Risk Identification," Software Engineering Laboratory Workshop, Goddard Space Flight Center, December 1996.
3. NASA, *Software Assurance Guidebook*, NASA Goddard Space Flight Center Office of Safety, Reliability, Maintainability, and Quality Assurance, September 1989.
4. Wilson, W., L. Rosenberg, and L. Hyatt, "Automated Analysis of Requirements Specifications," Fourteenth Annual Pacific Northwest Software Quality Conference, October 1996.
5. Hammer, T., "Measuring Requirements Testing," Eighteenth International Conference on Software Engineering, May 1997.
6. Hammer, T., "Automated Requirements Management – Beware How You Use Tools," Nineteenth International Conference on Software Engineering, April 1998.
7. Chen, M., J. Han, and P. Yu, "Data Mining: An Overview from a Database Perspective," *IEEE Transactions on Knowledge and Data Engineering*, Vol. 8, No. 6, December 1996.

Notes

1. Various names are used—deliveries, releases, builds—but the term *build* is used in this article.
2. This tool is available at no cost from the SATC Web site http://satc.gsfc.nasa.gov.