Making Requirements Management Work for You

Alan M. Davis, *Omni-Vista, Inc.*
Dean A. Leffingwell, *Rational Software, Inc.*

Requirements are capabilities and objectives to which software must conform and are the common thread for all development activities. Requirements management is the process of eliciting, documenting, organizing, and tracking changing requirements and communicating this information across the project team. Implementing a requirements management effort ensures that iterative and unanticipated changes are maintained throughout the project lifecycle. Without these measures, high-quality software is difficult if not impossible to achieve.

Identifying Requirements – How Will I Know One When I See One?

Requirements begin their lives when first elicited from customers or users. Elicitation may occur using any of a variety of techniques such as interviews, brainstorming, prototyping, questionnaires, and quality function deployment or techniques. Typically, requirements start out abstractly, e.g., "I need a system that controls elevators." As exploration continues, they become more specific, more detailed, and less ambiguous—they split and recombine in new ways (especially when multiple cases exist). Eventually, a set of highly detailed requirements emerges, e.g., "When the 'up' button is pressed, the light behind that button illuminates within one second."

Once captured, it is extremely important to maintain traces from each requirement to its more abstract predecessor requirements and to its more detailed successor requirements. Traceability aids in change management and is a fundamental component of quality assurance and sound requirements management.

The final, most detailed requirements are contained in a document called a *requirements specification*. This specification must be communicated and agreed upon by all relevant parties. It serves as the basis for design (it tells designers what the system is supposed to do) and for test (it tells testers what the system is supposed to do). Good require-

ments specifications exhibit the following characteristics [1].

- **Lack of ambiguity** – It is unlikely your product will satisfy users' needs if a requirement has multiple interpretations.
- **Completeness** – Although it may be impossible to know all future requirements for a system, you should at least specify all known requirements.
- **Consistency** – It is impossible to build a system that satisfies all requirements if two requirements are in conflict.
- **Traces to origins** – The source of each requirement should be identified. It may have evolved from the refinement of a more abstract requirement, or it may have come from a specific meeting with a target user.
- **Absence of design** – As long as requirements address external behaviors as viewed by users or by other interfacing systems, they are still requirements regardless of their level of detail. When a requirement attempts to specify the existence of particular subcomponents or algorithms, it is no longer a requirement but rather design information.
- **Enumerated requirements** – Most requirements specifications enhance their readability by including auxiliary types of information that are not requirements. This information includes introductory paragraphs or sentences, summary statements, tables, and glossaries. Actual requirements contained in the document should be somehow easily discernible, whether by unique font, identifying label, or other highlighting.

A complete list of principles to adhere to when performing requirements specification appears in Chapter 3 of [2].

Writing Your SRS – Getting Off to a Good Start

Many important documents exist within your development project: descriptions of user needs, design documents, and test plans. But one particular document, the software requirements specification (SRS), is a primary concern of the software developer. The purpose of this document is to define the complete external characteristics of the system to be built. It defines all the behavioral requirements, e.g., this system shall do A when the environment does B, and non-behavioral requirements, e.g., the system shall have an availability of 99.9 percent. Although standards are by no means a panacea, an organization that adopts a standard for the SRS achieves several benefits:

- The standard serves as a checklist of things to be addressed, so nothing is left out.
- It helps readers quickly locate and review requirements.
- It shortens the learning curve for new requirements writers and other members of the project team.

Numerous software specification standards can be used as a starting point in drafting an SRS. One that provides a good deal of guidance and flexibility is IEEE/ANSI 830-1993, IEEE Recommended Practice for Software Requirements Specifications. [3] Many other standards can be adopted to suit your needs. A good resource is the compilation by M. Dorfman and R. Thayer [4]. They have reprinted 26 different re-

quirements specifications under one cover including national, international, Institute of Electrical and Electronics Engineers (IEEE), American National Standards Institute (ANSI), NASA, and U.S. military standards.

It is important that your document outlines encourage accuracy, consistency, and a short learning curve. IEEE/ANSI 830-1993 serves as a good starting point for an SRS. Then, based on usage, you may find it beneficial to modify the standard and turn it into a corporate standard that better matches your company's specific processes and culture.

Selecting Requirements from Your Documents

Requirements documents contain some information that is not system requirements, e.g., introductions, general system descriptions, glossary of terms, and other explanatory information. Although important to an understanding of the requirements, they do not constitute requirements to be fulfilled by the system.

To ease communication of requirements and allow requirements management, writers should label those portions of text, graphics, or embedded objects that must be implemented and subsequently tested. Ideally, the requirements will be left in their original place rather than stored in multiple places; that is, they can be edited and maintained in the project documents even after they have been selected as individual requirements. This makes it easier to keep project documentation up to date as requirements change.

Organizing Your Requirements

Whether following a recognized standard or yours, you will need a section devoted to specific requirement descriptions. If you have isolated 500 requirements, for example, you should find a way to group them to aid in understanding rather than document them as a long list of bullets. We recommend organization by
- Mode of operation.
- Class of user.
- Object.
- Feature.

- Stimulus.
- Combining any of the above [1].

Applications that have clearly defined states (powered up, error recovery, etc.) could have their requirements grouped under their corresponding mode of operation. Systems that have a significant number of diverse users might be best organized by class of user. For example, a specification for an elevator control system could be organized into three major subsections: passenger, fireman, and maintenance employee. This provides a logical way to group specific requirements so that they can be reviewed and understood by each class of user.

Other applications may best be suited to organization by feature; that is, highlight the features and their intended behaviors as viewed by the user. Others, e.g., an air traffic control system, which is rich in real-world objects, may best be organized by grouping the behaviors of objects in the system. This approach may also be well suited to software organizations that have adopted object technology as their development paradigm.

Managing Requirements with Attributes

All requirements have attributes regardless of whether they are recognized. These attributes are a rich source of management information that can help you plan, communicate, and track your project's activities throughout the lifecycle. Each project has unique needs and should therefore select the attributes that are critical to its success. Following is a sample.

Customer Benefit – All requirements are not created equal. Ranking requirements by their relative importance to the end-user opens a dialogue with customers, analysts, and members of the development team.

Effort – Clearly, some requirements or changes demand more time and resources than others. Estimating the number of person-weeks or lines of code required, for example, is the best way to set expectations of what can or cannot be accomplished in a given time frame.

Development Priority – Only after considering a requirement's relative customer benefit and the effort required

to implement it can the team make feature trade-offs under the twin constraints of a project's schedule and budget. Priority communicates to the entire organization which features will be done first, which will be implemented if time permits, and which will be postponed. Most projects find that categorizing the relative importance of requirements into high, medium, and low or essential, desirable, and optional is sufficient, although finer gradations are possible.

Status – Key decisions and progress should be tracked in one or more status fields. During definition of the project baseline, choices such as *proposed*, *approved*, and *incorporated* are appropriate. As you move into development, *in progress*, *implemented*, and *validated* could be used to track critical project milestones.

Authors – The names of people (or teams) responsible for the requirement should be recorded in the requirements database, whether it is the person responsible for entering the text or the person responsible for identifying the need.

Responsible Party – The person who ensures the requirement is satisfied.

Rationale – Requirements exist for specific reasons. This field records an explanation or a reference to an explanation. For example, the reference might be to a page and a line number of a product requirement specification or to a minute marker on a video tape of an important customer interview.

Date – The date a requirement was created or changed should be recorded to document its evolution.

Version of Requirement – As a requirement evolves, it is helpful to identify the version numbers (and history) of requirements changes.

Relationships to Other Requirements – There are many relationships that can be maintained between requirements. For example, attribute fields can record
- The more abstract requirement from which this requirement emanated.
- The more detailed requirement that emanated from this requirement.
- A requirement of which this requirement is a subset.

- Requirements that are subsets of this requirement.
- A requirement that must be satisfied before this requirement is satisfied.

It is especially important to maintain linkages from requirements to all development products that emanate downstream from them. By providing these links, one can easily ascertain the impact of any changes and quickly determine development status (which should be an attribute of those downstream entities).

The above list is not exhaustive. Other common attributes include stability, risk, security, safety release implemented, and functional area. Whichever method is used to track them, attributes should be easily customized to adapt to the unique needs of each team and each application.

Requirements Traceability – Ensuring Quality and Managing Change

Requirements traceability is explicitly required in most Department of Defense software contracts and is typically practiced by manufacturers of all high-reliability products and systems. In the health-care industry, requirements traceability is governed by the proposed changes in the Good Manufacturing Practices Regulation. However, most companies outside these industries do not routinely practice requirements traceability.

Traceability is a link or definable relationship between two entities. Those who use requirements traceability find that it provides a level of project control and assured quality that is difficult to achieve by any other means. At Abbott Laboratories, where traceability was instituted in 1987, they like to say, "You can't manage what you can't trace." [5] This makes intuitive sense if for no other reason than to emphasize that full requirements test coverage is virtually impossible without some form of requirements traceability.

Benefits of Requirements Tracing

In its simplest terms, requirements tracing demonstrates that software does what it is supposed to do. The key benefits of this process include

- Verification that all user needs are implemented and adequately tested.
- Verification that there are no "extra" system behaviors that cannot be traced to a user requirement.
- Understanding the impact of changing requirements.

Implementing Requirements Traceability

Figure 1 shows a sample hierarchy of project documents. In this example, the product requirements document is the "source" of all requirements. In other examples, the source document could be a user-needs document or system specification.

A hierarchical relationship between two documents in Figure 1 is an indication that interrelationships may exist among specific elements in those documents. For example, the relationship shown between the product requirements document and the software requirements specification implies that any specific product requirement could be satisfied by one or more software requirements. Similarly, any software requirement may help to satisfy one or more product requirements. Clearly, a product requirement with no related software requirements or hardware requirements will not be satisfied. The reverse also is true: A software requirement with no related product requirements is extraneous and should be eliminated.

In addition to establishing document relationships to support traceability, you will need to employ some form of system to maintain links between individual items within the hierarchy. This can be done by embedding links and identifiers directly within the document or by using a separate spreadsheet or database that manages the linkages outside the document. There are advantages and disadvantages to each of these approaches. A new class of requirement management tools automatically maintains traceability links. Ideally, this capability is integrated in the same tool that manages and manipulates the documents and their individual requirements.

Change Management

Traceability provides a methodical and controlled process to manage the changes that inevitably occur as an application is developed. Without tracing, every change requires that documents be reviewed on a moment-to-moment basis to see which, if any, elements of the project require updating. Because it is difficult to establish whether all affected components have been identified, changes tend to decrease system reliability over time.

With traceability, management of a change can proceed in an orderly fashion. The impact of a change can now be understood by following the traceability relationships through the document hierarchy. For example, when a user need changes, a developer can quickly identify which software elements must be altered, a tester can pinpoint which test protocols must be revised, and managers can better determine the

Figure 1. *Example document hierarchy.*
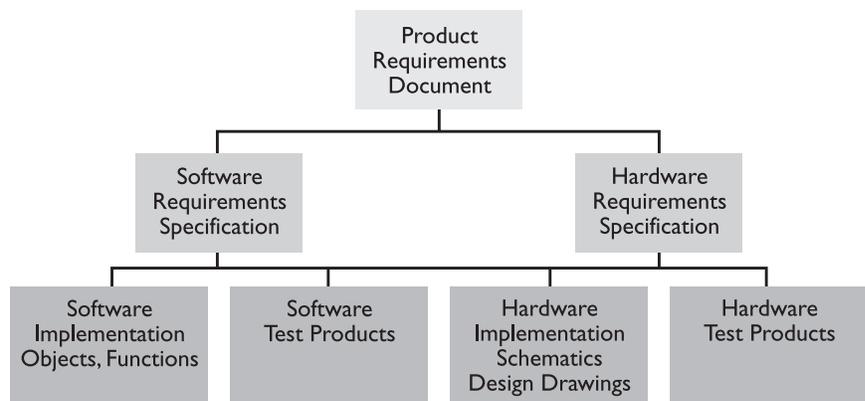
What You Can Do

- Continue to educate yourself on the benefits of requirements management. Secure training. Read. We have included a suggested list at the end of this article.
- Explore and use the new tools that make requirements management easier.
- Adopt a personal strategy to better communicate the requirements you own.

potential costs and difficulty to implement the change.

Requirements Reporting – Easing Management Reviews

A requirements repository gives managers a powerful tool to track and report project status. Critical milestones are more easily identified. Schedule risks are better quantified. Priorities and ownership are kept visible. Querying the repository can quickly uncover facts that provide answers to important questions, such as

- How many requirements are there on this project? How many are high priority?
- What percentage of the requirements are incorporated in the baseline?
- When will they be implemented?
- Which requirements changed since the last customer review?
- Who is responsible for the changes?
- What is the estimated cost impact of the proposed changes?

High-level reports aid management reviews of product features. Requirements can be prioritized by user safety considerations or by customer need, difficulty, and cost to implement. These specialized reports help managers better allocate scarce resources by focusing attention on key project issues. The net result is that managers make better decisions and thereby improve the outcomes of their company's application development efforts.

Conclusion

Software development is one of the most exciting and rewarding careers of our time. Unfortunately, many of us carry the scars from applications that missed expectations. It is common for applications to overshoot their schedule by half, deliver less than originally promised, or be canceled before release. To keep pace

with rising complexity and increased user demands, we must begin to mature the ways in which we develop, test, and manage our software projects. The first step in this advancement is improved requirements management.

Requirements management provides a "live" repository of application requirements and their associated attributes and linkages. This repository establishes an agreement on exactly what the software is supposed to do. It provides a wealth of information that can be used to manage and control your projects. Your quality will improve, and the software you build will better fit your customer's needs. And with requirements data available to all members, team communication is greatly improved.

Start now. You can cut project costs significantly by catching requirement errors early. Try to write down, in plain English, all the requirements of your current project. (Hint: if this is difficult or was not already done, ask why.) Compile these requirements in a suitable, short report and share them with your customers and peers. Get their feedback. Are any requirements missing, incomplete, or wrong? There is a good chance the answer is yes to all three. If it is still early enough to correct these errors, you have saved a lot of money. If it is too late, ask what is it about your process that could change, then propose a first step. ◆

About the Authors

Alan M. Davis is founder and chief executive officer of Omni-Vista, Inc., which develops and markets software development decision-making support tools. He serves as professor of Computer Science and El Pomar professor of software engineering at the University of Colorado at Colorado Springs. He is author of *Software Requirements: Objects, Functions, and States* and *201 Principles of Software Engineering* and is author or co-author of more than 100 papers on software and requirements engineering. He was editor in chief of *IEEE Software Magazine* from 1994 to 1998.

Dean A. Leffingwell is a vice president of Rational Software and is general manager of Rational University, where he is responsible for methodology, the Rational Unified Process, and customer education and training. Before 1997, he was chief executive officer and co-founder of Requisite, Inc., developers of the RequisitePro requirements management product and Requirements College™. He is considered an authority in requirements management and software quality and is a frequent speaker on these topics. He has a master's degree in engineering from the University of Colorado.

References

1. *CHAOS*, The Standish Group International, Inc., Dennis, Mass., 1994.
2. Davis, A., *201 Principles of Software Development*, McGraw-Hill, New York, 1995.
3. http://standards.ieee.org/catalog/olis/arch_swe.html
4. Dorfman, M. and R. Thayer, *Standards, Guidelines and Examples of System and Software Requirements Engineering*, IEEE Computer Society, Los Alamitos, Calif., 1991.
5. Watkins, R. and M. Neal, "Why and How of Requirements Tracing," *IEEE Software*, July 1994, pp. 104-106.

Suggested Reading

1. Davis, Alan M., *Software Requirements – Objects, Functions, and States*, Prentice-Hall, Englewood Cliffs, N.J., 1993.
2. Gause, Donald C. and G. Weinberg, *Exploring Requirements – Quality Before Design*, Dorset House, New York, 1989.

For information on how to order these books or for addresses of the available Internet forums that discuss requirements management, please write, call, or send E-mail to

Rational Software Corporation
18880 Homestead Road
Cupertino, CA 95014
Voice: 800-728-1212
Fax: 408-863-4120
E-mail: info@rational.com
Internet: http://www.rational.com.