Recommended Requirements Gathering Practices

Dr. Ralph R. Young
Northrop Grumman Information Technology

This article provides suggested conditions for performing requirements gathering and recommended requirements gathering practices. The author has conducted an extensive review of industry literature and combined this with the practical experiences of a set of requirements analysts who have supported dozens of projects. The sidebar on page 10 summarizes a set of recommended requirements gathering practices. Involving customers and users throughout the development effort results in a better understanding of the real needs. Requirements activities should be performed throughout the development effort, not just at the beginning of a project.

A "requirement" is a necessary attribute in a system, a statement that identifies a capability, characteristic, or quality factor of a system in order for it to have value and utility to a user [1]. According to Steve McConnell in *Software Project Survival Guide*, "The most difficult part of requirements gathering is not documenting what the users 'want'; it is the effort of helping users figure out what they 'need' that can be successfully provided within the cost and schedule parameters available to the development team."[1]

Each requirement should be necessary, verifiable, attainable, unambiguous, complete, consistent, traceable, concise, implementation-free, and have a unique identifier [1]. All of these characteristics of a good requirement should be self-evident, with the possible exception of "implementation-free." The reason that a requirement should be implementation-free is that requirements specify "what" shall be provided and not "how" – the how is a design aspect rather than a requirement. Documenting the rationale for each requirement (why it is required) is a good technique to reduce the number of requirements. Taking this one step, according to industry requirements consultant Ivy Hooks' experience, can eliminate "up to half" of the stated requirements [2].

Begin by understanding the organization's "business requirements." This leads to a "vision and scope" document that describes the background leading to the decision to develop a new or modified system or capability and describes the system to be developed. An agreed upon understanding of the capability is critical to a successful project. Consider having iterative scoping meetings with customers and users. The process of requirements elicitation itself generates more detailed and creative thinking about the problem that in turn can affect the scope. As the possibilities for a solution emerge, there are numerous decision points concerning what should and should not be included within the scope of the system.

The next step is to gather the stated requirements of the customers and users of the new capability. An effective requirements practice distinguishes "stated" requirements from "real" requirements [1]. Industry experience has shown that customers and system developers should jointly evaluate stated requirements to ensure that each is a verified need.

Part of the requirements process is to prioritize requirements.[2] This is important, because rarely is there enough time and money to provide everything that is wanted. It is also beneficial to focus on product benefits, not features [3]. Benefits refer to the necessary requirements. Adding unnecessary features adds design constraints and increases costs.

"Using peer reviews, scenarios, and walk-throughs to validate and verify requirements results in a more accurate ... specification and higher customer satisfaction."

It is estimated that 85 percent of the defects in developed software originate in the requirements [1]. Once defects are embedded in the requirements, they tend to resist removal. They are especially difficult to find via testing. Therefore it is crucial that training be required for requirements analysts and engineers that explains how to reduce the common types of requirements errors, including incorrect assumptions (49 percent), omitted requirements (29 percent), inconsistent requirements (13 percent), and ambiguities (5 percent) [2].

Use peer reviews and inspections to reduce defects in all your requirements representations. Peer reviews and inspections are a best practice way of eliminating defects. I recommend a peer review of all work products. The extent of the review should be based on the criticality of the work product. Peer reviews are a very effective method for reducing the costs of a project because they identify defects earlier. Rework is estimated at 45 percent of project costs, industry-wide [1]. Using peer reviews, scenarios, and walk-throughs to validate and verify requirements results in a more accurate requirements specification and higher customer satisfaction.

Inspections are a very rigorous form of peer reviews and should be considered for requirements representations. Gilb and Graham provide an excellent guide for inspections of any type of document [4]. According to Gilb, the capability to perform Gilb inspections requires five days of formal training and a lot of rigor.[3] One advantage of the Gilb approach is that he advocates "sampling" of work products rather than review of the entire product – the idea is that by identifying defects in the first few pages, the author can utilize this feedback to address similar problems throughout the document or work product.

Some believe that all requirements should be listed in a requirements document such as a Software Requirements Specification. Experience has shown that it is helpful to think of "several" artifacts comprising your requirements specification: the database in your automated requirements tool, the vision and scope statement for the project, the requirements document, and other requirements lists or descriptions provided by customers and users. These can include lists of requirements met by related legacy (historical) systems and the list of system-level (real) requirements evolved by the requirements manager/requirements engineer. This enables us to have a more comprehensive understanding of the real requirements that

Recommended Requirements Gathering Practices

The following is a list of recommended requirements gathering practices. They are based on the author's extensive review of industry literature combined with the practical experiences of requirements analysts who have supported dozens of projects.

1. Write and iterate a project vision and scope document.
2. Initiate a project glossary that provides definitions of words that are acceptable to and used by customers/users and the developers, and a list of acronyms to facilitate effective communication.
3. Evolve the real requirements via a "joint" customer/user and developer effort. Focus on product benefits (necessary requirements), not features. Address the minimum and highest priority requirements needed to meet real customer and user needs.
4. Document the rationale for each requirement (why it is needed).
5. Provide training for requirements analysts and selected customer/user representatives that explains the following:
   - The role of the requirements analyst, e.g., to evolve real requirements working with customers and users, not to invent requirements independently or to "gold plate."
   - How to write good requirements.
   - The types of requirements errors and how these can be reduced.
   - The value of investing more in the requirements process.
   - The project and/or organization's "requirements process."
   - Overview of the methods and techniques that will be used.
   - How to use the project's automated requirements tool.
   - The role of validation and verification during requirements definition.
6. Establish a mechanism to control changes to requirements and new requirements.
7. Prioritize the real requirements to determine those that should be met in the first release or product and those that can be addressed subsequently.
8. When the requirements are volatile (and perhaps even when they are not), consider an incremental development approach. This acknowledges that some of the requirements are "unknowable" until customers and users start using the system.
9. Use peer reviews and inspections of all requirements work products.
10. Use an industry-strength automated requirements tool.
    - Assign attributes to each requirement.
    - Provide traceability.
    - Maintain the history of each requirement.
11. Use requirements gathering techniques that are known, familiar, and proven in the organization such as requirements workshops, prototyping, and storyboards.
12. Provide members of the project team (including requirements analysts) who are domain/subject matter experts.
13. Evolve a project and organizational approach based on successful use of policy, process, methods, techniques, and tools. Provide a mechanism such as working groups to share information and "best practices" among projects.
14. Establish a continuous improvement ethic, teamwork approach, and a quality culture.
15. Involve customers and users throughout the development effort.
16. Perform requirements validation and verification activities in the requirements gathering process to ensure that each requirement is testable.

is communicated effectively to all stakeholders.

One of our most common problems is taking on too much work – attempting to exceed requirements rather than addressing the minimum requirements to meet real needs. Thus, meeting minimum requirements is in the customers' best interests. It helps avoid the problems of late deliveries, budget overruns, low morale, and poor quality [5].

Preferred Requirements Gathering Techniques

Following are a set of recommended requirements elicitation techniques. Among almost 40 such techniques available [1], only a few have proven most effective. These techniques can be used in combination. Their advantages are that they are effective in emerging the real requirements for planned development efforts. Kotonya and Sommerville [6] provide a good discussion of the context for requirements elicitation and analysis. More detailed discussions of these techniques are provided in Leffingwell and Widrig [7] and in Sommerville and Sawyer [8].

Interviews. Interviews are used to gather information. However, the predisposition, experience, understanding, and bias of the person being interviewed influence the information obtained. The use of context-free questions by the interviewer helps avoid prejudicing the response [9]. A context-free question is a question that does not suggest a particular response. For example, who is the client for this system? What is the real reason for wanting to solve this problem? What environment is this product likely to encounter? What kind of product precision is required?

Document Analysis. All effective requirements elicitation involves some level of document analysis such as business plans, market studies, contracts, requests for proposals, statements of work, existing guidelines, analyses of existing systems, and procedures. Improved requirements coverage results from identifying and consulting all likely sources of requirements [10].

Brainstorming. Brainstorming involves both idea generation and idea reduction. The goal of the former is to identify as many ideas as possible, while the latter ranks the ideas into those considered most useful by the group. Brainstorming is a powerful technique because the most creative or effective ideas often result from combining seemingly unrelated ideas. Also, this technique encourages original thinking and unusual ideas.

Requirements Workshops. Requirements workshops are a powerful technique for eliciting requirements because they can be designed to encourage consensus concerning the requirements of a particular capability. They are best facilitated by an outside expert and are typically short (one or a few days). Other advantages are often achieved – participant commitment to the work products and project success, teamwork, resolution of political issues, and reaching consensus on a host of topics. Benefits of requirements workshops include the following:

- Workshop costs are often lower than are those for multiple interviews.
- They help to give structure to the

requirements capture and analysis process.
- They are dynamic, interactive, and cooperative.
- They involve users and cut across organizational boundaries.
- They help to identify and prioritize needs and resolve contentious issues.
- When properly run, they help to manage user's expectations and attitude toward change [11].

A special category of requirements workshop is a Joint Application Development (JAD) workshop. JAD is a method for developing requirements through which customers, user representatives, and developers work together with a facilitator to produce a requirements specification that both sides support. This is an effective way to define user needs early. Wood and Silver in *Joint Application Development* [12] assert that quality systems can be built in 40 percent less time utilizing JAD. They explain how to perform JAD and provide diagrams, forms, and a sample JAD design document.

Prototyping. Prototyping is a technique for building a quick and rough version of a desired system or parts of that system. The prototype illustrates the capabilities of the system to users and designers. It serves as a communications mechanism to allow reviewers to understand interactions with the system. See Sommerville's *Software Engineering* [13] for a good discussion of prototypes and how they can be used. Prototyping sometimes gives an impression that developers are further along than is actually the case, giving users an overly optimistic impression of completion possibilities. Prototypes can be combined effectively with other approaches such as JAD and models.

Use Cases. A use case is a picture of actions a system performs, depicting the actors [14]. It should be accompanied by a textual description and not be used in isolation of other requirements gathering techniques. Use cases should always be supplemented with quality attributes and other information such as interface characteristics. Many developers believe that use cases and scenarios (descriptions of sequences of events) facilitate team communication. They provide a context for the requirements by expressing sequences of events and a common language for end users and the technical team.

Be cautioned that use cases alone do not provide enough information to enable development activities. Other requirements elicitation techniques should also be used in conjunction with use cases. Requirements consultant Ivy Hooks recommends using operational concepts as a simple, cost-effective way to build a consensus among stakeholders and to address two large classes of requirements errors: omitted requirements and conflicting requirements [2]. Operational concepts identify user interface issues early, provide opportunities for early validation, and form a foundation for testing scenarios in product verification.

Storyboards. A storyboard is a set of drawings depicting a set of user activities that occur in an existing or envisioned system or capability. Storyboards are a kind of paper prototyping. Customers, users, or developers start by drawing pictures of the screens, dialogs, toolbars, and other elements they believe the software should provide. The group continues to evolve these until real requirements and details are worked out and agreed upon. Storyboards are inexpensive and eliminate risks and higher costs of prototyping. Another related technique is storytelling: the writing of vignettes to envision new products and services based on perceived user needs and the possibilities offered by emerging technologies.

Interfaces Analysis. Missing or incorrect interfaces are often a major cause of cost overruns and product failures. Identifying external interfaces early clarifies product scope, aids risk assessment, reduces product development costs, and improves customer satisfaction. The steps of identifying, simplifying, controlling, documenting, communicating, and monitoring interfaces help to reduce the risk of problems related to interfaces. Hooks and Farry provide a thorough discussion and recommendations [2].

Modeling. A model is a representation of reality that is intended to facilitate understanding. The CORE requirements tool has behavioral modeling capabilities. Behavior is allocated to physical components of the planned system. See Vitech's Web page at <www.vtcorp.com> for information concerning this tool and a trial version that can be downloaded. Uses of the tool for modeling and example problems are described in Buede [15]. In a recent study of 15 requirements engineering teams supporting relatively small projects (average of 10 person-years of effort with project duration of 16.5 months), use of prototypes and models helped eliminate ambiguities and inconsistencies and correlated with the most successful projects [10].

Performance and Capacity Analysis. Hofmann and Lehner [10] provide an insight based on their study of 15 requirements engineering efforts: Stakeholders emphasized that concentrating on system functions and data resulted in a lack of attention to the total system requirements and in incomplete performance, capacity, and external interface requirements. Thus, it is vital to ensure that the requirements gathering process provides for all requirements (requirements coverage).

An Innovative Concept

For an innovative approach to gathering requirements, see "A Quick, Accurate Way to Determine Customer Needs [16]." The authors of this article believe customers tend to say one thing during requirements elicitation and then do something entirely different. They feel that this problem is largely due to reliance on traditional requirements gathering approaches such as focus groups, surveys, and interviews that do not deal effectively with contradictions in peoples' responses.

The authors advocate "a new technology" called imprint analysis. Imprint refers to the collection of associations and emotions unconsciously linked with a word, concept, or experience. They believe this method produces findings that remain consistent over time because it takes human emotions into account. Emotion is the trigger to action; emotions in the present dictate peoples' emerging needs. The authors believe that imprint analysis can actually forecast customer behavior.

A Cautionary Note

Everyone involved in a particular project should use a common set of methods and techniques. To that end, it is advisable to have project discussions and training sessions to evolve the desired "project approach." Projects should use methods and techniques that have been used successfully on previous projects in that organization. If there is no local precedent, hire staff from outside the organization who have previous successful experience. And above all, I strongly recommend that the project involve people who have previously successfully used all methods and techniques that are to be employed. Providing formal training for developers who are expected to use new methods and tools is a valuable investment.

Automated Requirements Tools

I recommend the use of an automated requirements tool to support a development effort of any size. Tiny projects might get away with using Microsoft Word or Microsoft Excel; however, most proj-

ects require an industry-strength requirements tool such as DOORS, Requisite Pro, or Caliber RM with capabilities that extend beyond "requirements management."

Using a requirements tool facilitates requirements elicitation because it enables better understanding of the requirements by both the customer and the developer. Also, an effective requirements tool helps prioritize requirements, provides requirements traceability throughout the development effort, allows assignment of multiple attributes (characteristics of requirements) to all requirements, and facilitates managing requirements changes [1].

Conclusions and Recommendations

There is a wealth of information and guidance available in back issues of CROSSTALK, books, articles, and industry conference publications, and also from the "lessons learned" on projects in our own organizations. Much has been written, but perhaps too little has been conscientiously applied on actual projects. Do not try to do everything at once. Rather, encourage the project team to select and commit to a few improved practices that make sense in your environment.

Establish a few useful metrics that enable evaluation of implementation effectiveness and institutionalization of selected practices. Remember, the things that are measured and tracked are the ones that improve. Make a concerted effort to improve project communications as well as teamwork. A committed, highly motivated team can accomplish most anything.◆

References

1. Young, Ralph R. Effective Requirements Practices. Boston: Addison-Wesley, 2001. See also <ralph young. net>, a Web site devoted to requirements-related topics.
2. Hooks, Ivy F., and Kristin A. Farry. Customer-Centered Products: Creating Successful Products Through Smart Requirements Management. New York: AMACOM (publishing arm of The American Management Association), 2001.
3. Smith, Preston G., and Donald G. Reinertsen. Developing Products in Half the Time. 2nd ed. New York: John Wiley & Sons, Inc., 1998.
4. Gilb, Tom, and Dorothy Graham. Software Inspection. Reading, Mass.: Addison-Wesley, 1993. See also <www.result-planning.com>.
5. Whitten, Neal. "Meet Minimum Requirements: Anything More Is Too Much." PM Network Sept. 1998.
6. Kotonya, Gerald, and Ian Sommerville. Requirements Engineering: Processes and Techniques. Chichester, England: John Wiley & Sons, 1998.
7. Leffingwell, Dean, Don Widrig, and Edward Yourdon. Managing Software Requirements. Boston: Addison-Wesley, 2000.
8. Sommerville, Ian, and Pete Sawyer. Requirements Engineering: A Good Practice Guide. New York: John Wiley & Sons, 1997.
9. Gause, Donald C., and Gerald M. Weinberg. Exploring Requirements: Quality Before Design. New York: Dorset House Publishing, 1989.
10. Hofmann, Hubert F., and Franz Lehner. "Requirements Engineering as a Success Factor in Software Projects." IEEE Software July/Aug. 2001: 58-66.
11. Graham, Ian. Requirements Engineering and Rapid Development: An Object-Oriented Approach. Reading, MA: Addison-Wesley, 1998.
12. Wood, Jane, and Denise Silver. Joint Application Development. New York: John Wiley & Sons, 1995.
13. Sommerville, Ian. Software Engineering. 6th ed. Harlow, England: Addison-Wesley, 2001.
14. Schneider, Geri, Jason P. Winters, and Ivar Jacobson. Applying Use Cases: A Practical Guide. Reading, Mass.: Addison-Wesley, 1998.
15. Buede, Dennis M. The Engineering Design of Systems: Models and Methods. New York: John Wiley & Sons, 2000.
16. Afors, Cristina, and Marilyn Zuckerman Michaels. "A Quick, Accurate Way to Determine Customer Needs." Quality Progress, July 2001, 82-87.
17. McConnell, Steve. Software Project Survival Guide. Redmond, Wash.: Microsoft Press, 1998.
18. Wiegers, Karl E. "First Things First: Prioritizing Requirements." Software Development Magazine Sept. 1999: 24-30.

Notes

1. Adapted from Steve McConnell, *Software Project Survival Guide* [17]. See Chapter 8 for valuable suggestions concerning requirements development.
2. Visit Karl Wiegers' Web site <process impact.com/goodies.shtml> to download a Microsoft Excel spreadsheet useful for prioritizing requirements. See also Wiegers' article, *First Things First: Prioritizing Requirements* [18].
3. Industry consultant Robert Sabourin trains and facilitates Gilb inspections. He advises that the basic training can be accomplished in four hours, including one example inspection. A mentor or champion is required to train moderators, scribes, and process administrators. Sabourin's experience is that Gilb inspections provide good value, for example, to inspect requirements against sources and to inspect all downstream work from requirements. Performing inspections can foster communication and gain buy-in. Inspections can be used to test artifacts that otherwise would be nearly impossible to test objectively. Inspections can be implemented with minimal impact on the normal workflow. See <www. amibug.com>.
4. See Chapter 3 of Graham's *Requirements Engineering and Rapid Development* for a detailed discussion of organizing and running workshops.

About the Author

Ralph R. Young, DBA, is the director of Software Engineering, Systems and Process Engineering, Defense Enterprise Solutions at Northrop Grumman Information Technology, a leading provider of information technology and systems-based solutions. Dr. Young leads a requirements working group of requirements engineers. He teaches a 10-hour Requirements Course for Practitioners and consults frequently concerning both requirements engineering and process improvement. Dr. Young has received awards for teamwork, leadership, continuous improvement, and publishing, and is often recognized for his contributions in process management and improvement. He is the author of *Effective Requirements Practices.*

Northrop Grumman
Information Technology
Mail Stop 5S3
1500 PRC Drive
McLean, VA 22102
Phone: (703) 556-1030
E-mail: young_ralph@prc.com