Software Requirements Engineering: What, Why, Who, When, and How

By Linda Westfall

Key words: requirements engineering, requirements elicitation, requirements analysis, requirements specification, requirements management, stakeholder

ABSTRACT

If software requirements are not right, companies will not end up with the software they need. This article will discuss:

- What: The various levels and types of requirements that need to be defined
- Why: The benefits of having the right software requirements
- Who: The stakeholders of the software requirements and getting them involved in the process
- When: Requirements activities throughout the software development life cycle
- How: Techniques for eliciting, analyzing, specifying, and validating software requirements

WHAT

Requirements must be determined and agreed to by the customers, users, and suppliers of a software product before the software can be built. The requirements define the "what" of a software product:

- *What the software must do* to add value for its stakeholders. These functional requirements define the capabilities of the software product.
- *What the software must be* to add value for its stakeholders. These nonfunctional requirements define the characteristics, properties, or qualities that the software product must possess. They define how well the product performs its functions.
- *What limitations there are on the choices* that developers have when implementing the software. The external interface definitions and other constraints define these limitations.

Most software practitioners just talk about "the requirements." However, by recognizing that there are different levels and types of requirements, as illustrated in Figure 1 adapted from Karl Wiegers (2004), practitioners gain a better understanding of what information they need to elicit, analyze, specify, and validate when they define their software requirements.

Business requirements define the business problems to be solved or the business opportunities to be addressed by the software product. In general, the business requirements define why the software product is being developed. Business requirements

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

are typically stated in terms of the objectives of the customer or organization requesting the development of the software.

User requirements look at the functionality of the software product from the user's perspective. They define what the software has to do in order for the users to accomplish their objectives. Multiple user-level requirements may be needed in order to fulfill a single business requirement.

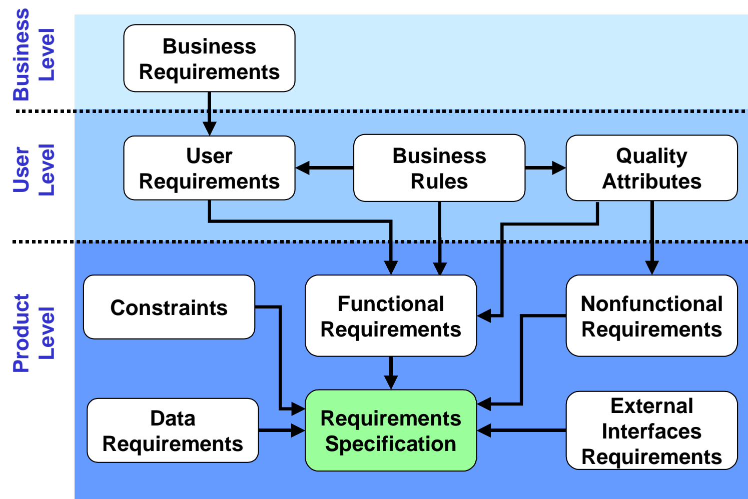

Figure 1 Levels and types of requirements

For example, the business requirement to allow the customer to pay for gas at the pump might translate into multiple user requirements, including requirements for the user to:

- Swipe credit or debit card
- Enter a security PIN number
- Request a receipt at the pump

The product's functional requirements that define the software functionality must be built into the product to enable users to accomplish their tasks, thereby satisfying the business requirements. Multiple functional level requirements may be needed to fulfill a user requirement. For example, the requirements that the users can swipe their credit card might translate into multiple functional requirements including requirements for the software to:

- Prompt the customer to put his or her card into the reader
- Detect that the card has been swiped
- Determine if the card was incorrectly read and prompt the customer to swipe the card again
- Parse the information from the magnetic strip on the card

As opposed to the business requirements, business rules are the specific policies, standards, practices, regulations, and guidelines that define how the users do business (and are therefore considered user-level requirements). The software product must adhere to these rules in order to function appropriately within the user's domain.

User-level quality attributes are nonfunctional characteristics that define the software product's quality. Sometimes called the "ilities," quality attributes include reliability, availability, security, safety, maintainability, portability, usability, and other properties. A quality attribute may translate into product-level functional requirements for the software that specify what functionality must exist to meet the nonfunctional attribute. For

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

example, an ease of learning requirement might translate into the functional requirement of having the system display pop-up help when the user hovers the cursor over an icon.

A quality attribute may also translate into product-level nonfunctional requirements that specify the characteristics the software must possess in order to meet that attribute. For example, an ease-of-use requirement might translate into nonfunctional requirements for response time to user commands or report requests.

The external interface requirements define the requirements for the information flow across shared interfaces to hardware, users, and other software applications outside the boundaries of the software product being developed.

The constraints define any restrictions imposed on the choices that the supplier can make when designing and developing the software. For example, there may be a requirement that the completed software use no more than 50 percent of available system memory or disk space in order to ensure the ability for future enhancement.

The data requirements define the specific data items or data structures that must be included as part of the software product. For example, a payroll system would have requirements for current and year-to-date payroll data.

The software may be part of a much larger system that includes other components. In this case, the business and user-level requirements feed into the product requirements at the system level. The system architecture then allocates requirements from the set of system requirements downward into the software, hardware, and manual operations components.

WHY

The following quote from Fredrick Brooks illustrates why requirements are so important: "The hardest part of building a software system is deciding precisely what to build. No other part of the conceptual work is as difficult as establishing the detailed technical requirements, including all of the interfaces to people, to machines, and to other software systems. No other part of the work so cripples the resulting system if done wrong. No other part is more difficult to rectify later" (Brooks 1995). Eliciting, analyzing, and writing good requirements are the most difficult parts of software engineering. However, to quote Karl Wiegers (2004), "If you don't get the requirements right, it doesn't matter how well you do anything else." One can end up doing a perfect job of building the wrong product.

There are many issues that can have a negative impact on software development projects and products if practitioners don't do a good job of defining their software requirements. These issues include:

- Incomplete requirements
- Lack of user involvement
- Requirements churn
- Wasted resources

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

- Gold plating
- Inaccurate estimates

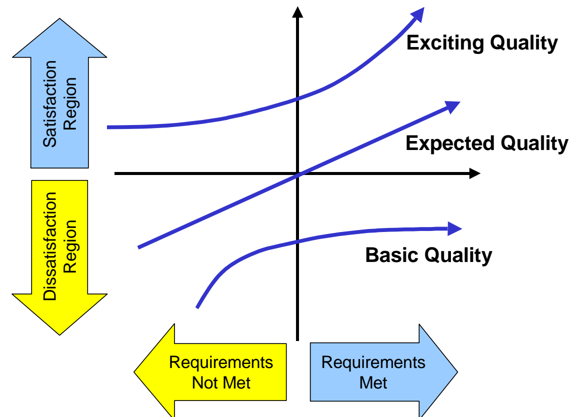

Figure 2 Kano model for quality requirements

If the requirements are incomplete, software practitioners end up building a software product that does not meet all of the customer and user's needs. As illustrated in Figure 2, Noritaki Kano developed a model of the relationship between customer satisfaction and quality requirements (Pyzdek 2000). The expected quality line represents those quality requirements that the customer explicitly states. For example, they will state their preferences for the make, model, options, and gas mileage when shopping for a car. The customer will be dissatisfied (and go buy a car somewhere else) if their explicit requirements are not met. The customer's satisfaction increases as more of their explicit requirements are met. When enough of their explicit requirements are met, the customer shifts from being dissatisfied with the product to being a satisfied customer. There is a basic level of quality requirements that a customer expects the product to have. These are requirements that are assumed by the customer and are typically not explicitly stated. For example, they expect a car to have four tires, a working engine, and a steering wheel. This level of requirements does not satisfy the customer. Note that the entire basic line is in the dissatisfaction region. Absence of this level of quality requirements will increase a customer's dissatisfaction. Exciting quality is the innovative requirements level and represents unexpected items. These are items that customers do not even know they want, but they love them when they see them. For example, remember when cup holders in cars were first introduced? Note that the entire excited quality line is in the satisfied region. It should be remembered, however, that today's innovations are tomorrow's expectations. The expected quality requirements are the ones practitioners can elicit fairly easily if they talk to the product's stakeholders. However, it is easy to miss both the basic and exciting quality requirements if they do not do a thorough job of detailed requirements elicitation and analysis. In addition, if practitioners miss a stakeholder group or if they do not get the users involved in the requirements process, they can end up with gaps even in their expected requirements.

Requirements churn refers to changes in the requirements after they are initially agreed to and baselined. Some of this change is a part of refining developers' understanding as they develop the software. Changes also occur because of changes in the environment or the user's needs over time that occur as a natural part of a project of any significant duration. If requirements are poorly defined, however, requirements churn occurs because of missing requirements that should have been included in the original specification or because of poorly written or ambiguous requirements. These are the types of requirements churn that good requirement engineering practices will help avoid.

Requirements errors account for 70 percent to 85 percent of the rework costs on a software project (Wiegers 2003). If one finds a requirements defect during the requirements phase and it costs one unit to fix (for example, three engineering hours,

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

$500), the cost of fixing that same defect will typically increase as it is found later and later in the life cycle. In fact, studies show that it can cost more than 100 times more to fix a requirements defect if it is not found until after the software is released to the field.

Another waste of resources occurs when gold plating is added to the software. Gold plating can take place when a developer adds functionality to the software that was not in the requirements specification but that they believe “the user will just love” without putting that functionality through the requirements engineering process. Users may or may not want the new functionality. If they don’t, the cost of developing it is a waste. Cycling these “good ideas” through the requirements engineering process helps ensure that they truly are something that’s needed in the product so that gold plating does not occur. A second kind of gold plating comes from the users. For example, if practitioners ask the users “what they want” rather than “what they need to be able to do with the system,” they may end up with a wish list of nice to haves or things that they might want sometime in the future but do not really need right now. This is a good reason to prioritize the requirements and focus resources on the most important requirements first. Gold plating can result in wasting resources on implementing functionality that is not of real value or that’s never actually used. It also creates the risk that defects in that part of the functionality will cause reliability problems for the rest of the software.

The requirements define the scope of the products that are being developed. Without a clear picture of that scope, estimates of the project schedule, cost, and quality will be less accurate.

WHO

Stakeholders are individuals who affect or are affected by the software product and therefore have some level of influence over the requirements for that software product. The requirements engineering process provides the best opportunity to consider all of the various stakeholder’s interest in context with one another. There are three main categories of stakeholders: the acquirers of the software product, the suppliers of the software product, and other stakeholders.

The *acquirer* type stakeholders can be divided into two major groups. First there are the customers who request, purchase, and/or pay for the software product in order to meet their business objectives. The second group is the users, also called end-users, who actually use the product directly or use the product indirectly by receiving reports, outputs, or other information generated by the product.

The *suppliers of the software product* include individuals and teams that are part of the organization that develops the software product or are part of the organizations that distribute the software product or are involved in other product delivery methods (for example, outsourcing). The requirements analyst, also called the business analyst or system analyst, is responsible for eliciting the requirements from the customers, users, and other stakeholders, analyzing the requirements, writing the requirements specification, and communicating the requirements to development and other stakeholders. The designers are responsible for translating the requirements into the software’s architectural and detailed designs that specify how the software will be

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

implemented. The developers are responsible for implementing the designs by creating the software products. If the software is part of a larger system, hardware designers and developers may also be interested in the software requirements. The testers use the requirements as a basis for creating test cases that they use to execute the software under specific, known conditions to detect defects and provide confidence that the software performs as specified. The documentation writers are responsible for using the requirements as inputs into the creation of the user documentation including user/operations manuals, help files, installation instructions, and training materials as necessary. The project managers are responsible for planning, monitoring, and controlling the project and guiding the software development team to the successful delivery of the software. Technical support, also called operations or the help desk, is responsible for interfacing with the user community to support the software once it has been deployed to the field. Product change management, which may take the form of a change control board (CCB), is responsible for reviewing proposed changes to the requirements, analyzing their impacts, approving/disapproving changes, and ensuring that approved changes are implemented and validated.

There may also be *other stakeholders* interested in the requirements. Examples of other requirements stakeholders include:

- Legal or contract management
- Manufacturing or product release management
- Sales and marketing
- Upper management
- Government or regulator agencies
- Society at large

Identifying and considering the needs of all of the different stakeholders can help prevent software product requirements from being overlooked. For example, if a company is creating a payroll system and it does not consider charities as one of its stakeholders, it might not include the requirements for the software to:

- Allow the payees to specify from one to three charitable deductions
- Withhold charitable deductions from payee's checks each pay period
- Report current and year-to-date charitable deductions on payee's pay slip
- Print a check to each charity for the accumulated amount deducted from payees

The requirements analyst will never know as much about a stakeholder's work as that stakeholder. By identifying and involving key stakeholders, the analyst gains access to their experience base and domain knowledge. The analyst's job is then to analyze, expand on, synthesize, resolve conflicts in, and combine the inputs from all the stakeholders into an organized set of requirements at the appropriate level of abstraction for the target audience.

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

Identifying the stakeholders and getting them involved in the requirements engineering process brings different perspectives to the table that can aid in a more complete set of requirements early in the software development life cycle. As Wiegers puts it, getting stakeholders involved eliminates the need for two of the most ineffective requirements elicitation techniques: clairvoyance and telepathy (Wiegers 2003).

Remember, people hate change and a new software product means changing the way the stakeholders will perform part or all of their jobs. Obtaining stakeholder input and participation gets them involved in the solution to their needs. Involved stakeholders are more likely to buy in to the completed software product, which can create champions for the software product in the stakeholder community. This can be beneficial in transitioning the software product into the operational environment.

The first step in identifying the stakeholders is to make sure that one considers all of the potential stakeholders. The following checklist can help in identifying potential stakeholders:

- What types of people will use the software product?
- What business activities are supported by the software product and who performs, is involved in, or manages those activities?
- Whose job will be impacted by the introduction of the new software product?
- Who will receive the reports, outputs, or other information from the software product?
- Who will pay for the software product?
- Who will select the software product or its supplier?
- If the software product fails, who could be impacted?
- Who will be involved in developing, supporting, and maintaining the software product?
- Who knows about the hardware, other software, or databases that interface with this software product?
- Who established the laws, regulations, or standards governing the business activities supported by the software product?
- Who should be kept from using the software product or from using certain functions/data in the software?
- Who does this software product solve problems for?
- Who does this software product create problems for?
- Who does not want the software product to be successful?

It is almost impossible for the development of a software product to take into consideration the needs of all of the potential stakeholders. The needs of stakeholders may also contradict each other. For example, the need to keep unfriendly hackers from

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

breaking into the payroll software conflicts with the accountant's need for quick and easy access to the software.

The second step in identifying the stakeholders is for the practitioners to decide how they are going to deal with these conflicts. They accomplish this by determining which stakeholders have higher priorities based on their contribution to the success of the software product. This allows practitioners to make appropriate trade-offs when conflicts occur. Gause and Weinberg (1989) discuss what they call the user-inclusion strategy, which the author has expanded to the stakeholder-inclusion strategy. In this strategy, software practitioners can look at the list of potential stakeholders and determine if:

- They want to be friendly to this stakeholder and consider how to accommodate their needs in the software product (for example, make access quick and easy to the software for the accountant)
- They can afford to simply ignore their needs because they are not key to the success of the software product (for example, decide that the specific needs of a little known charity are not important enough to consider when defining the requirements)
- They want to be unfriendly and consider how to counteract their needs in the software product (for example, prevent hackers from breaking into the software)

The third step is to decide who will represent each stakeholder group that one has designated as friendly or unfriendly. There are three main choices:

- *Representative*. Select a stakeholder champion to represent the group of stakeholders. For example, if there are multiple testers who will be testing the product, the lead tester might be selected to represent this stakeholder group. The lead tester would then participate in the requirements engineering activities and be responsible for gathering inputs from other testers and managing communication with them.
- *Sample*. For large stakeholder groups or for groups where direct access is limited for some reason, sampling may be appropriate. In this case, it would be necessary to devise a sampling plan for obtaining inputs from a representative set of stakeholders in that particular group. For example, if the company had several thousand employees, it may decide to take a sample set of employees to interview about their needs for the new accounting system.
- *Exhaustive*. If the stakeholder group is small or if it is critical enough to the success of the system, it may be necessary to obtain the input from every member of that stakeholder group. For example, if the software product has only one customer or a small set of customers it might be important to obtain input from each of the customers.

Software practitioners must determine when the stakeholder group needs to participate in the requirements engineering activities. Are they going to participate throughout the entire process or only at specific times? For example, practitioners may want the stakeholder champion for the accountants to be a member of the requirements team and

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

to participate throughout the requirements engineering process. They may not need the human resource stakeholder representative to participate on the requirements team, but may need them on a continuous, on-call basis in case there are questions. However, practitioners may only need to sample the employees during the requirements elicitation activities.

Software practitioners must also determine how they are going to have each stakeholder group participate. Are they going to send them questionnaires; do one-on-one interviews; have them participate in a focus group or in a facilitated requirements elicitation workshop; or show them prototypes, mock-ups, or simulations, and gather their inputs?

The final decision is to establish the priorities of the stakeholder team members based on their relative importance to the success of the software product. These established priorities help one determine whose voice to listen to when conflicts arise between the needs of various stakeholders or stakeholder groups.

WHEN

Requirements development encompasses all the activities involved in identifying, capturing, and agreeing upon the requirements. This includes the definition and integration of the business requirements, the user requirements, and software product-level requirements. The majority of the requirements development activities occur during the early concept and requirements phases of the life cycle. Continued elaboration of the requirements, however, can progress into the later phase of the software development life cycle.

Requirements management encompasses all of the activities involved in requesting changes to the baselined requirements, performing impact analysis for the requested changes, approving or disapproving those changes, and implementing the approved changes. Requirements management also includes the activities used for ensuring that all work products and project plans are kept consistent and tracking the status of the requirements as one progresses through the software development process. Requirements management is an ongoing activity throughout the software development life cycle.

The implemented software product is validated against its requirements during the testing phases of the life cycle to identify and correct defects and to provide confidence that the product meets those requirements.

Requirements engineering is an iterative process. Practitioners must first develop the business requirements and baseline them. The business requirements are input into the development of the user-level requirements. Based on that effort, practitioners may discover gaps in their business requirements that result in their further refinement. They can then use the information they gain from refining the business requirements for further update of the user-level requirements. The business and user-level requirements feed into the definition of the product-level requirements. This may lead to further refinement of the business and user-level requirements. The product requirements are then input into the software design and development process, which may uncover implicit requirements or the need for further refinement of the business, user, and product-level requirements.

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

HOW

Software requirements engineering is a disciplined, process-oriented approach to the definition, documentation, and maintenance of software requirements throughout the software development life cycle. As illustrated in Figure 3, software requirements engineering is made up of two major processes: requirements development and requirements management. Requirements development encompasses all of the activities involved in eliciting, analyzing, specifying, and validating the requirements.

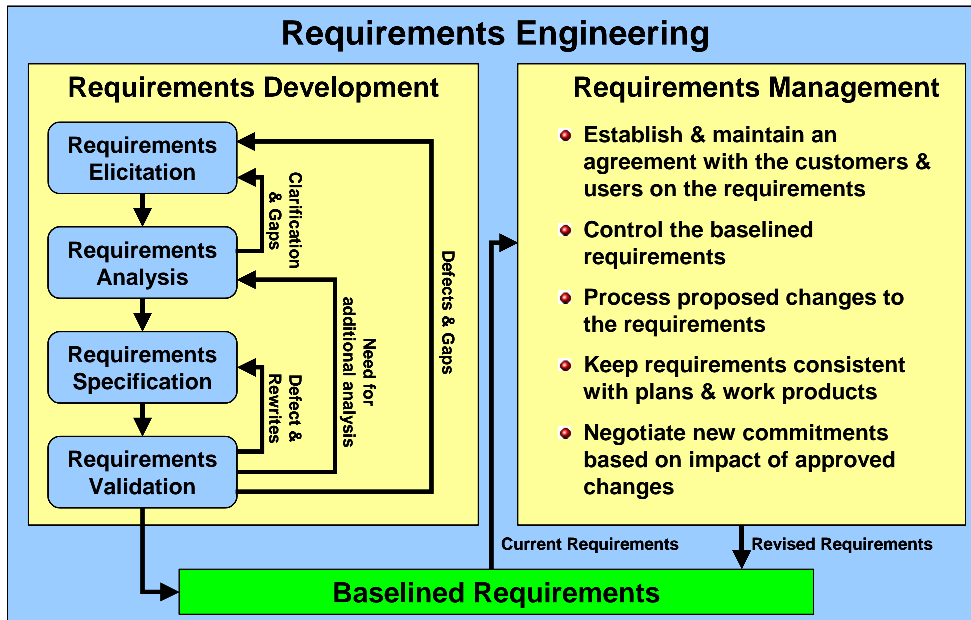

Figure 3 Requirements Engineering Process (based on Wiegers 2003)

The requirements elicitation step includes all of the activities involved in identifying the requirement's stakeholders, selecting representatives from each stakeholder class, and determining the needs of each class of stakeholders. The elicitation process is the information-gathering step in the requirements development process. Many different techniques may be used to elicit requirements, including stakeholder interviews, focus groups, facilitated requirements workshops, observations of current work processes, questionnaires and surveys, analysis of competitor's products, and benchmarking of industry practices. Requirements elicitation may also involve documentation studies of:

- Industry standards, laws, and/or regulations
- Product literature (one's own or the competition's)
- Process documentation and work instructions
- Change requests, problem, or help-desk reports
- Lessons learned from prior projects or products
- Reports and other deliverables from the existing systems

During the requirements analysis step the stakeholder's needs, assumptions, and other information identified during requirements elicitation are melded together and refined

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

into further levels of detail. This step includes representing the requirements in various forms including prototypes and models, performing trade-off analysis, establishing priorities, analyzing feasibility, and looking for gaps that identify missing requirements. The information gained in the analysis step may necessitate iteration with the elicitation step as clarification is needed, conflicts between requirements are explored, or missing requirements are identified.

The requirements are formally documented during the specification step so they can be communicated to the product stakeholders. The requirements specification can take one of many forms. For example, on small projects the requirements information may be documented in a single Software Requirements Specification (SRS) document. On larger projects, the requirements may be specified in multiple documents. For example:

- Business requirements may be documented in a business requirements document (BRD), marketing requirements document (MRD), or project vision and scope document.
- User requirements may be documented in a set of use cases in a tool or in a user requirements specification (URS) document.
- If the software is part of a larger system, the product-level requirements may be documented in a system requirements specification and a system architecture specification may document the allocations of those requirements to the software, hardware, and manual operations components of that system.
- The software functional and nonfunctional requirements and the constraints may be documented in an SRS.
- External interfaces may be included in the SRS or in separate external interface requirements documents.

One good practice for requirements specification is to have predefined requirements specification templates. These templates allow the requirements analyst to focus on content instead of format and can help ensure that key items are not overlooked as the requirements are being documented. For example, if assumptions are not documented, they may lead to future misunderstandings or other problems. Another good practice is to use a requirements tool (or database). A requirements tool can be a major asset in both requirements management and project management.

The last step in the requirements development process is to validate the requirements to ensure that they are well written, complete, and will satisfy the customer needs. Validation may lead one to iterate the other steps in the requirements development process because of identified defects, gaps, additional information or analysis needs, needed clarification, or other issues.

One of the primary tools for requirements validation is to conduct formal peer reviews of the requirements specification documents before they are baselined. Empirical evidence from many different companies shows that peer reviewing the requirements documentation has the highest return on investment of any defect detection activity. The peer review process should look at the requirements specification as a whole to ensure that it is:

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

- *Complete*. The requirements document includes all of the necessary requirements information. For example, the SRS includes all the functions and nonfunctional requirements, constraints, external interface requirements, and data requirements that must be satisfied.
- *Consistent.* Internal conflicts do not exist between requirements in the document that result in the requirements contradicting each other. The requirements also do not conflict with higher-level requirements including business, user, or system-level requirements. Terminology should also be used consistently within the document:
  - A word has the same meaning every time it is used.
  - Two different words are not used to mean the same thing.
- *Modifiable*. The requirements document is organized and written in a manner that will facilitate making future change:
  - Nonredundant: Each requirement is stated in only one place.
  - Changeable: Each requirement can be changed without excessive impact on other requirements.

The peer review process should also look at each individual requirement to ensure that it is:

- *Unambiguous*. Each requirement statement should have only one interpretation, and each requirement should be specified in a coherent, easy-to-understand manner. For example, the author searches for words that end in "ly" (for example, quickly, user-friendly, automatically) because adverbs by nature are almost always ambiguous.
- *Concise*. Each requirement should be stated in short, specific, action-oriented language.
- *Finite*. The requirement should not be stated in an open-ended manner. For example, words like "all," "every," and "throughout" should be avoided in requirements statements.
- *Measurable*. Specific, measurable limits or values should be stated for each requirement as appropriate.
- *Feasible*. The requirement can be implemented using available technologies, techniques, tools, resources, and personnel within the specified cost and schedule constraints.
- *Testable*. There exists a reasonably cost-effective way to determine that the software satisfies the requirement.
- *Traceable*. Each requirement should be traceable back to its source (for example, user-level requirements, system-level requirements, standard, enhancement request). It should also be specified in a manner that allows traceability forward into the design, implementation, and tests.

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

Another major tool for validating the requirements is to start writing the test cases for functional (black box) testing of the software. Since functional testing only requires knowledge of the requirements and not of the internals of the software product, practitioners can start writing test cases against their requirements as soon as they start writing them. The major advantage to writing the functional test cases early in the life cycle is that it will uncover defects in the requirements. Writing test cases early may result in some rework if the requirements change, but the cost of that rework will be more than offset by the savings resulting from finding and fixing more requirements defects earlier.

Requirements development is an iterative process. One should not expect to go through the steps in a one-shot, linear fashion. For example, the requirements analysts may talk to a user, then analyze what the user had to say. They may go back to that user for clarification and then document what they understand as that part of the requirements. They may then go on to talk to another user, or hold a joint requirements workshop with several user representatives. Their analysis may then include building a prototype that they show to a focus group. Based on that information the analyst documents additional requirements and holds a requirements walk-through to validate that set of requirements. The analyst then moves on to eliciting the requirements for the next feature and so on.

After one or more iterations through the software requirements development process, part or all of the requirements are deemed “good enough” to baseline and become the basis for software design and development. A good saying to remember at this point is, “When is better the enemy of good enough?” The requirements will never be perfect -- one can always do more refinement and gather more input. Requirements baselining is a business decision that should be based on risk assessment.

Requirements management starts with getting stakeholder buy-in to the baselined requirements. Requirements management encompasses the activities involved in requesting changes to the baselined requirements, performing impact analysis for the requested changes, approving or disapproving those changes, and implementing the approved changes. Requirements management also includes the activities used for ensuring that work products and project plans are kept consistent and tracking the status of the requirements as one progresses through the software development process.

CONCLUSION

Practitioners must understand the different types and levels of requirements to do a good job of requirements engineering. It requires an understanding of the benefits of having good requirements so that adequate resources and time are dedicated to the requirements engineering process throughout the software development life cycle. Doing requirements engineering correctly requires an interdisciplinary approach that considers the needs of multiple stakeholder groups. It also requires expertise in the various skills of requirements engineering including requirements elicitation, requirements analysis, requirements specification, requirements validation, and requirements management.

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.

REFERENCES

Brooks, F. 1995. *Mythical man-month: essays on software engineering, 20th anniversary edition. Addison-Wesley Professional.* Gause, D., and G. Weinberg. 1989. *Exploring requirements, quality before design*. New York: Dorset House Publishing.

Pyzdek, T. 2000. *Quality engineering handbook*. New York: Marcel Dekker.

Wiegers, K. E. 2003. *Software requirements*, 2nd Edition. Redmond, Wash.: Microsoft Press.

Wiegers, K. E. 2004. *In search of excellent requirements.* Process Impact Web site. See URL: http://www.processimpact.com.

BIOGRAPHY

Linda Westfall is the President of The Westfall Team, which provides training and consulting services in Software Engineering, Software Quality and Software Project Management. Prior to starting her own business, Linda was the Senior Manager of the Quality Metrics and Analysis at DSC Communications where her team designed and implemented a corporate wide metrics program. Linda has more than twenty-five years of experience in real time software engineering, quality and metrics. She has worked as a Software Engineer, Systems Analyst, Software Process Engineer and Manager of Production Software.

Very active professionally, Linda is the past chair of the American Society for Quality (ASQ) Software Division. She has also served as the Software Division's Program Chair and Certification Chair and on the ASQ National Certification Board. Linda is a past-chair of the Association for Software Engineering Excellence (ASEE). Linda is a Professional Engineer in Software Engineering from the state of Texas and is an ASQ Certified Software Quality Engineer (CSQE) and Certified Quality Auditor (CQA).

You can contact Linda by sending email to lwestfall@westfallteam.com or visit her web site at www.westfallteam.com.

Copyright © 2005-2006 The Westfall Team. All Rights Reserved.