The What, Why, Who, When and How of Software Requirements

Linda Westfall
President
The Westfall Team
3000 Custer Road, Suite 270, PMB 101
Plano, TX 75075
lwestfall@westfallteam.com
www.westfallteam.com

SUMMARY

If the software requirements aren't right, you won't end up with the software that you need. This paper will discuss:

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

WHAT

Requirements are what the customers, users and suppliers of a software product must determine and agree on 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 the 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 we can better understand what information we need to elicit, analyze, specify and validate when we are working to define our 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 set of business requirements define why the software product is being developed. Business requirements 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 perspectives of the various users of that product. They define what the software has to do in order for the users to accomplish their objectives. In order to fulfill a business requirement, there may be multiple user level requirements. For example, the business requirement to allow the customer to pay for the gas at the pump might translate into multiple user requirements including requirements for the user to:

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

The software's functional requirements specify the software functionality that the developers must build into the product to enable users to accomplish their tasks, thereby satisfying the business objectives" (i.e., business requirements). [Wiegers-03] In order to fulfill a user requirement, there may be multiple functional level requirements. 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 their card into the reader
- Detect that the card has been swiped
- Determine if the card was incorrectly read and prompt the customer to reenter the card if it wasn't
- 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 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 curser over an icon.

A quality attribute may also translate into product level nonfunctional requirements that specify the characteristics that 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% 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.

Sometimes software requirements are allocated downward from a set of system requirements or the software requirements may be dependant upon the limitations and resource availability of its system's environment.

WHY

The following quote from Fredrick Brooks illustrates 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 & 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-87] Eliciting, analyzing and writing good requirements are the most difficult part of software engineering. However, to quote Karl Wiegers, "If you don't get the requirements right, it doesn't matter how well you do anything else." [Wiegers-04]

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

- Incomplete requirements
- Lack of user involvement
- Requirements churn
- Wasted resources
- Gold plating
- Inaccurate estimates

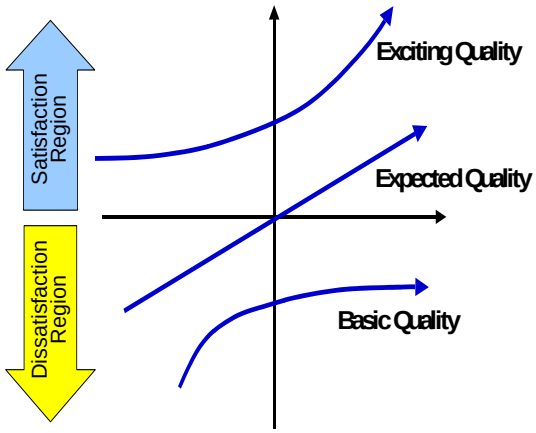

Figure 1: Kano model for quality requirements

If the requirements are incomplete, we end up building a software product that does not meet all of the customer's and user's needs. As illustrated in Figure 1, Noritaki Kano developed a model of the relationship between customer satisfaction and quality requirements. [Pyzdek-00] 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 if this level of quality is not met and increasingly satisfied as this quality level increases. There is a basic level of quality 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 quality does not satisfy the customer. Note that the entire basic quality line is in the dissatisfaction region. However, absence of this level of quality will increase a customer's dissatisfaction. Exciting quality is the innovative quality level and represents unexpected quality items. These are items that the customer doesn't even know they want, but they love them when they see them. For example, remember the first time you had a cup holder in your car. 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 we can elicit fairly easily if we talk to all of the product's stakeholders. However, it's easy to miss both the basic and exciting quality requirements if we don't do a thorough job of detailed requirements elicitation and analysis. On the other hand if we miss a stakeholder group or if we don't get the users involved in the requirements process, we can end up with gaps even in our 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 our understanding as we develop the software. Changes also occur because of changes in the environment or the user's needs over time that are a natural part of a project of any significant duration. However, if requirements are poorly defined, 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 to 85 percent of the rework costs on a software project. [Leffingwell in Wiegers-03] If we find a requirements defect during the requirements phase and it costs one unit to fix (e.g., 3 engineering hours, $500) that same defect will typically increase in cost to fix as it is found later and later in the life cycle. In fact, studies show that it can cost 100+ times more to fix a requirements defect if 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 takes place when a developer adds functionality to the software that wasn't in the requirements specification but that they believe " the user will just love" without putting that functionality through the requirements engineering process. The users may or may not want the new functionality. If they don't, the cost of developing it is a waste.

A second kind of gold plating comes from the users. For example, if we ask the users "what they want" rather than "what they need to be able to do with the system", we may end up with a wish list of nice to haves or things that they might want some time in the future but don't 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 isn't 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 inaccurate.

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. To quote Karl Wiegers, "Nowhere more than in the requirements process do the interests of all the stakeholders in a software or system project intersect." [Wiegers-03] There are three main categories of stakeholders: the suppliers of the software product, the acquirers of the software product and other stakeholders.

The acquirer type stakeholders can be broken down 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 (e.g., 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 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 & 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 we are creating a payroll system and we don't consider charities as one of the stakeholders, we 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 knows. 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.

Identifying the different 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-03]

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 the transition of the software product into the operational environment.

So the first step in identifying the stakeholder is to make sure that you are considering all of the potential stakeholders. The following checklist can help in identify potential stakeholders:

- What types of people will use the software product?
- What business activities are supported by the software product & who performs, is involved in, or manages those activities?
- Whose job will be impacted by the introduction of the new software product?
- Who will the reports, outputs or other information from the software product go to?
- 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 & maintaining the software product?
- Who knows about the hardware, other software or databases that this software product has to interface with?
- 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 doesn't want the software product to be successful?

It is almost impossible for the design of a software product to take into consideration the needs of all of the potential stakeholders. The needs of stakeholders may contradict each other. For example, the need to keep unfriendly hackers from breaking into the payroll software conflict with the accountant's need for quick and easy access to the software. Therefore, the second step is to decide how we are going to

determine which stakeholders have higher priorities because they are key to the success of the software product so that we can do appropriate trade-offs.

Gause & Weinberg [Gause-89] discuss what they call the user-inclusion strategy, which I have expanded to the stakeholder-inclusion strategy. In this strategy, we look at the list of all of the potential stakeholders and determine if:

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

For each stakeholder group that we have designated as friendly or unfriendly, the third step is to decide who will represent that stakeholder group in the requirements engineering activities. 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 a single customer or a small set of customers it might be important to obtain input from each of the customers.

We 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, we may want the stakeholder champion for the accountants to be a member of the requirements team and to participate throughout the requirements engineering process. We may not need the human resource stakeholder representative to participate on the requirements team but we need them on a continuous, on-call basis in case we have questions. However, we may only need to sample the employees during the requirements elicitation activities.

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

The final decision is to establish the priority of the stakeholder team based on their relative importance to the success of the software product. As conflicts arise between the needs of various stakeholders, priorities need to be established so we can determine whose voice to listen to.

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. However,

continued elaboration of the requirements can continue into the later phase of 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 we progress through the software development process. Requirements management is an ongoing activity throughout the software development life cycle

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

Requirements engineering is an iterative process. We 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, we may discover gaps in our business requirements that result in their further refinement. We can then use the information we gain from refining the business requirements for further update of the user level requirements. The business and user level requirements then 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 the need for further refinement of our business, user and product level requirements.

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 2, 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.

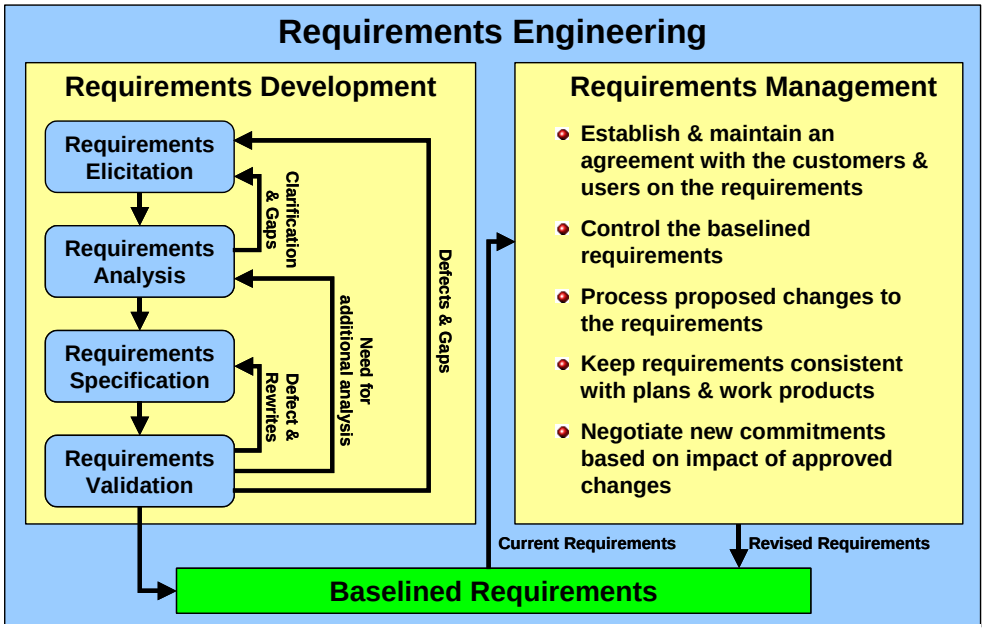

Figure 2: Requirements Engineering Process [based on Wiegers-03]

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 requirements analysis step includes taking the stakeholder needs and refining them into further levels of detail. It also includes representing the requirements in various forms including prototypes and models, 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.

During the specification step the requirements are formally documented so that they can be communicated to all the product stakeholders. The requirements specification can take one of many forms. For example, on small projects all of 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
- The software functional and nonfunctional requirements and the constraints may be documented in a Software Requirements Specification (SRS)
- External interfaces may be included in the SRS or in separate External Interface Requirements documents

Another way of specifying requirements is to document them in a requirements tool or database.

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 you 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.

Requirements development is an iterative process. Don't expect to go through the steps in the process in a one-shot, linear fashion. For example, the requirements analyst may talk to a user, then analyze what the user told them. 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 that "When is better the enemy of good enough?" The requirements will never be perfect -- you can always do more and more refinement and gather more and 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 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 we progress through the software development process.

CONCLUSION

To do a good job of requirements engineering we must understand the different types and levels of requirements. 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 stakeholders groups. It also requires expertise in the various skills of requirements engineering including requirements elicitation, requirements analysis, requirements specification, requirements validation and requirements management.

REFERENCES

Brooks-87 Fredrick Brooks, *Mythical Man-Month*, 1987.

Gause-89 Donald Gause & Gerald Weinberg, *Exploring Requirements, Quality Before Design*, Dorset House Publishing, New York, NY, 1989.

IEEE-610 IEEE Standards Software Engineering, Volume 1, *IEEE Standard Glossary of Software Engineering Terminology, IEEE Std. 610-1990*, The Institute of Electrical and Electronics Engineers, 1999, ISBN 0-7381-1559-2.

IEEE-830 IEEE Standards Software Engineering, Volume 4, *IEEE Recommended Practice for Software Requirements Specifications, IEEE Std. 828-1998*, The Institute of Electrical and Electronics Engineers, 1999, ISBN 0-7381-1562-2.

Pyzdek-00 Thomas Pyzdek, Quality Engineering Handbook, Marcel Dekker, New York, 2000, ISBN 0-8247-0365-0.

Robertson-99 Suzanne Robertson and James Robertson, *Mastering the Requirements Process*, Addison-Wesley, Harlow, England, 1999, ISBN 0-201-36046-2.

Wiegers-03 Karl E. Wiegers, *Software Requirements*, 2nd Edition, Microsoft Press, Redmond, WA, 2003, ISBN 0-7356-1879-8.

Wiegers-04 Karl E. Wiegers, *In Search of Excellent Requirements*, Process Impact website: www.processimpact.com, 2004.