Interpreting Requirements in a He Said/She Said World

Deb Jacobs
Software Engineering Services

Interpreting what someone else really wants can be one of the most difficult elements of software development – at times it can be like talking to your teenager. That includes both the customer and the contractor. We must each walk in the other's shoes to see it from the other's vantage point. Requirements interpretation must be a two-way street with collaboration and communication the keys to success.

Burnt popcorn permeated the room as Mary walked down to Bob's cubicle. She couldn't believe that someone had burnt popcorn again; she was going to have to put instructions on the microwave since this was the third time this week someone had left it in too long. But right now she was too frustrated to even think about the popcorn for long. As she came to Bob's cubicle, she was starting to feel a little better about the calculations on the look-up report for the MDPSR database she was working on.

"Bob, do you have a minute?" she asked as she saw him huddled over his computer.

"Sure Mary, just a sec, while I finish my thought here." he replied.

As Mary sat down and waited she took another look at the words in the requirements matrix and drawings she had brought with her to show Bob. *I remember they talked a lot about what they wanted to see, but I'm not sure they decided anything,* she thought. Looking at the requirement again it just wasn't specific enough. It left a lot open to interpretation. *What was it they said about…?*

Bob surprised Mary a few minutes later when he said, "So Mary, what's up?" Mary jumped but recovered quickly. "Bob, I'm working on the look-up report and I'm not sure I understand what fields are used in this calculation."

Bob looked at the requirements matrix and Mary's notes and drawings.

"I remember this," said Bob. "They threw out several things at the requirements meeting a few months ago, but I can't remember what they finally decided or if they finally decided what they wanted."

"So what do you think I should do?" Mary asked. "I talked to Mike when I saw him in the hallway the other day and he told me which ones he used. Since he's one of the users, don't you think that should be right?"

"I don't know Mary, maybe we should ask Don," Bob suggested.

"Last time I asked him about a requirement he treated me like I was stupid or something. Besides I'm not sure we have time – this thing is scheduled for testing in 28 days, and I still have a lot to do," Mary exclaimed, exasperated.

"I know. He did that to me too and with this thing due soon we don't want to get burned again for not making the schedule. I can't take much more overtime – I'm getting burned out." Bob replied, "By the way, who burnt the popcorn this time?"

"I don't know, but it makes you never want to eat popcorn again in your life." Mary said as she walked away to finish her look-up report. She wondered if she was doing the right thing or if she should get clarification, but the time crunch brought her back to reality and she decided to follow the user's advice.

One month later at the meeting following testing, Mary knew that was the wrong decision. "Where did these figures come from?" Don ranted, as he went over the reports generated from the tests. "We told you what fields to use at the requirements meeting last April. In fact, there are several of these reports that aren't right. What's going on here?"

Mary grimaced, as Jack, the Project Manager (PM) – who had been named PM since he was good at placating people like Don – got up and started to explain that they had gone by the documented requirements. Before Jack could get two words out of his mouth, Don stood up and threw the reports on the table.

"This will all have to be fixed. I want to go to retest in one week."

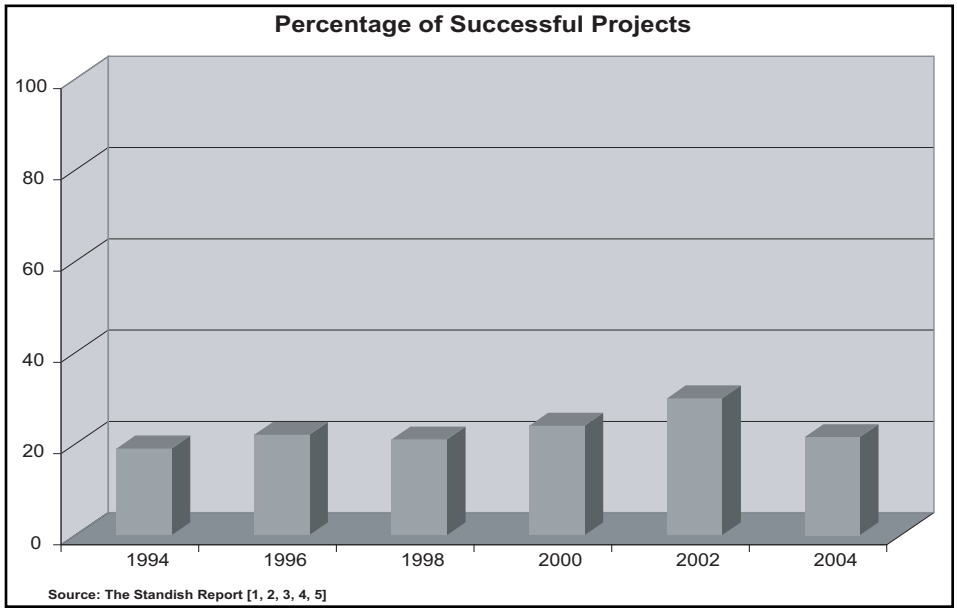
Sound familiar? Who is at fault here: the contractor receiving the requirements or the customer who gave the requirements? The answer is both. This scene may sound familiar to you since it happens time and again on project after project.

Regardless of which report you read, the battle cry is loud and clear: *Projects are failing more often than they succeed. Something must be done.* But what? That is the million-dollar question. Even the oft-quoted Standish Report still shows that successful projects that meet schedule, budget, and requirements still only hover around the 30 percent mark, as shown in Figure 1.

A Great Start

A great start for fixing this long-standing software development crisis is with requirements. We must fully understand what we are developing before we can develop the right product for our cus-

Figure 1: *Standish Report Summary*



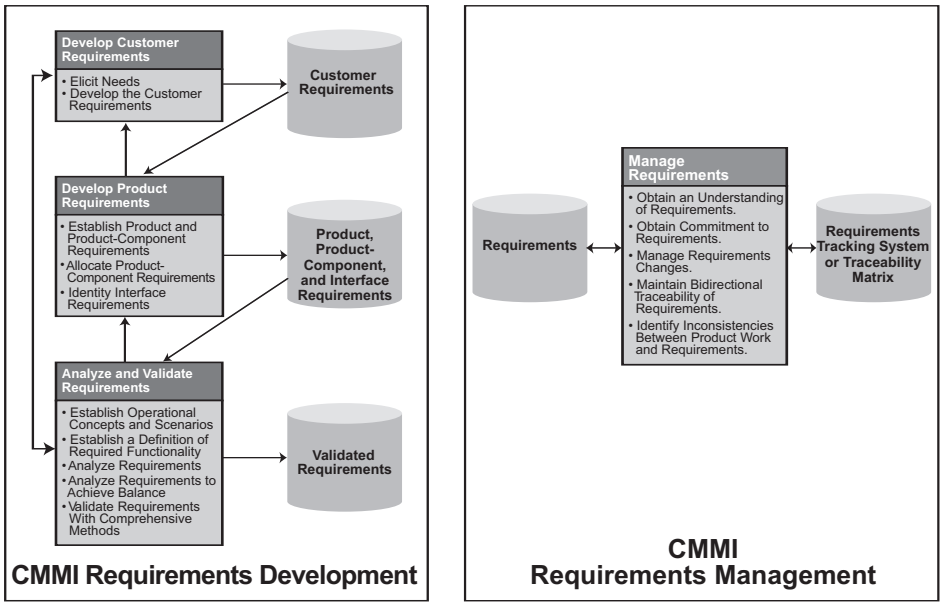


Figure 2: *Requirements Development and Requirements Management Process Areas* [7]

tomers. Readers will probably shake their head and say *of course we do*, but the statistics show that even though this seems to be a *no brainer*, we still are not doing it. This article will discuss some tried and true as well as some innovative methods that can help contractors and their customers through the quagmire of requirements elicitation.

There are many great sources for requirements engineering; I have provided some of the best sources at the end of this article. Based on my more than 25 years of experience in various software development and project management roles, I have found one of the best sources to be the Capability Maturity Model Integration℠ (CMMI®) [6], developed by the Software Engineering Institute (SEI℠). There are two process areas that are valuable resources for any project in defining and managing requirements: Requirements Development, and Requirements Management. Figure 2 illustrates these key process areas.

Eliciting Requirements

There are a myriad of methods for eliciting requirements. The key is interpreting the requirements correctly. The amount of time spent on eliciting requirements will depend on many factors such as team experience, management, level at which requirements have been pre-defined, and life-cycle methodology selected for development. The rule-of-thumb I have always used is approximately 15 percent of project time should be spent on identifying, defining, and clarifying requirements throughout the development life cycle with the majority of time spent up front. Some of the more popular methods for obtaining requirements include the following:

- Analysis of existing documentation.
- Statement of work/task definition.
- Interviews (structured and unstructured).
- Group brainstorming.
- Observation.
- Questionnaires and/or surveys.
- Prototyping.
- Modeling (enterprise, data, behavior, domain, non-functional).
- Rapid Application Development.
- Joint Application Development.
- Cognitive (examining usability).

A lot of resources have defined what a *good* requirement should look like. Most agree that it should be complete, consistent, unambiguous, verifiable/measurable, necessary, concise, implementation-free, and attainable. The additional reading at the end of this article provides great guidance for developing a *good* requirement.

There are some other methods that have been used successfully that are not quite as obvious as defining *good* requirements. These are the ones that this article will concentrate on.

Perspective Factor

Interpreting requirements correctly is the most prevalent problem in requirements engineering. We all look at things differently based on our background, education, experience, and simply from where we are standing at the moment. For requirements development, it will also depend on our responsibilities on the project.

MC Escher is famous for his optical illusion art (was and is well known for his impossible structures). A favorite is called *Relativity* that simply tells us that what you see is relative to where you are standing. When dealing with others realities, we have to see things from others' perspectives.

Each stakeholder is going to see something different. The customer or requirements giver sees the final products based on their outlook of either the manual methods used or a legacy system they have been using. Many times it is hard to articulate either in writing or verbally what they want even when they know exactly what they want. The contractor or requirements receiver sees the same thing based on their experiences in development, using like systems, etc. By looking at things from each others vantage point, we are able to better understand what needs to be developed. It is everyone's responsibility.

Proactive Requirements Management

Planning requirements management activities, open communications, and proactive resolution of even a simple *not sure* is crucial on a project in order to avoid nightmare scenarios later on. There is enough to panic about when developing a system without the added stress of misunderstandings or misconceptions.

Planning the requirements development activities can alleviate much of the typical *Keystone Kops* scenarios. Lack of planning has been oft-quoted as a top cause for failure in survey after survey. Lewis Carroll in *Alice in Wonderland* said,

> It sounded an excellent plan, no doubt, and very neatly and simply arranged. The only difficulty was, she had not the smallest idea how to set about it. [8]

More important than the planning process is ensuring the plan does not become shelfware: it must be useable and thus used by the project team. The level of planning will depend upon the size and scope of the project. Regardless of the project, a key word in planning is *flexibility*. Even the most closely planned activities can be unsuccessful without flexibility to accommodate unforeseen events and changing priorities.

Working closely with the customer throughout the development process can make the difference between success and failure. The key is for the customer to help ensure that the system's desired function-

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

℠ SEI is a service mark of Carnegie Mellon University.

ality is realized and the developer fully understands what the system's desired functionality is from the customer's point of view. The ever-growing-in-popularity Agile methodologies uses *active stakeholder participation* and *on-site customer* to describe this customer/contractor collaboration.

Who's Who

A graphical depiction for defining and keeping track of who's who provides a great tool for both the customer and the contractor. This ensures that there are no misunderstandings or misconceptions. For larger projects with numerous stakeholders, including subcontractors, this tool is essential in providing a means of averting potential issues resulting from wrong requirements.

So who is the best person to act as the decision maker? The customer (administrative or technical)? The contractor? The subcontractor? Or someone else? Typically, it is a combination of the administrative customer and the technical customer representatives. However, it depends upon the contract. Many times the contractor has the pertinent skills and knowledge to make the appropriate decisions concerning requirements so they may be tasked as the decision maker. Ultimately, the customer usually has the largest stake in the outcome of the project since they have to live with the results. They should be involved with the requirements decisions at all times.

In short, the key is to know the *who*, *what*, *when*, *where*, *why*, and *how*. There should be a clear understanding of who are the decision makers for requirements. The *who's who* diagrams should be approved by the appropriate managers and distributed to the entire team. If a project has a communications plan, this is a good place to include these diagrams and should include who can accomplish the following:

- Add a new requirement.
- Change existing requirements.
- Clarify a requirement.
- Accept changes to requirements.
- Direct the various teams.
- Determine if a requirement has or has not been met.
- Accept requirements as met or not met.

Distribution lists for the various project elements should be developed to ensure all stakeholders are included as needed.

Win-Win Stakeholder Negotiations

A process should be in place that is fair to both the customer and the contractor, thus *win-win*. The customer is responsible for ensuring they have conveyed the correct requirements, and the contractor is responsible for ensuring they understand the requirements correctly, and an approved requirements method can be developed. Some of the most effective methods of documenting approval and tracking requirements are the use of requirements matrices and requirements databases depending upon the size of the effort. Whichever method is used, the key is *approved*.

An effective method of providing a win-win is through the use of win-win stakeholder negotiations. These negotiations can be in the form of meetings that provide a forum for open discussions between stakeholders at the worker-bee level (the engineers developing the systems and the ultimate users of the system). Requirements can be negotiated based on technology, environment, time, effort, and budget constraints. They provide a way of fully understanding the contractual requirements and discussing derived requirements. Someone with the power to make decisions on the project must be present in order to make the meetings effective.

A process should be in place that is fair to both the customer and the contractor, thus win-win.

Sometimes a trained negotiator can help alleviate some of the pain involved in these meetings. Role-playing is a great method used where the customer and contractor trade places and look at things from each others perspective. The goal is for everyone to walk away satisfied, or at least *break-even*.

Many times, either the contractor or the customer will have tasks that need to be accomplished as part of the requirements elicitation; the negotiations meetings can include discussion and negotiation of these commitments for both the customer and the contractor. To make adherence to the agreed-upon commitments more effective, some organizations use what they call a stakeholder's contract that puts these particular commitments in writing.

Test Early, Test Often

Test early, test often. This should be the mantra for all projects. If we think of a project in terms of how we can test it, we will more readily be able to see what needs to be developed. This should again be a collaborative effort between the requirements giver and receiver.

On one project in which I was staff engineer, I was responsible for a replanning effort since the project was doomed to go over both schedule and budget. After much discussion, we decided to try an incremental approach where requirements were *chunked* into categories and the system was built incrementally. Starting with the system communications as the foundation, we tested this component and then built requirements into a solid framework. This allowed us to avoid the inherent problem of finding bugs when a large system is fully integrated. By working closely with the client, it also avoided finger-pointing at the end of the project since they were involved throughout the testing and the requirements were fully fleshed out. This project finished within the original schedule and budget.

If an incremental approach does not work or the project is not large enough for that level of testing, early development of test cases, use cases, or test scenarios can enable the developer to more easily visualize the finished product. It provides a chance to work with the customer to ensure a full understanding of the requirements. It also weeds out or helps clarify requirements that are not testable or in some way measurable.

Another proven method of fleshing out requirements is prototyping to simulate the final product or a complex component of the final product. There are many benefits to be gleaned from prototyping, including exposure of misunderstandings between customers and developers, detection of missing functionality or services, identification of confusing functionality or services, development of a simplified working system early in development cycle for the user, incremental delivery of a system starting with the prototype, identification of risks, and a basis for derivation of requirements.

If a project can afford the extra time and expense of prototyping or there are safety or security issues involved, prototyping is key to building a successful system. Depending upon the system under development, the time and expense of doing a prototype may actually cost less in time and ultimately be less expensive. There are many factors that must be

weighed in making a decision to prototype. Sometimes a simple paper prototype makes the difference between day and night.

The Value of Pictures

Conceptual modeling provides a great method for visualizing the final system. Fred R. Barnard said, "One picture is worth a thousand words [9]." Use paper, use a whiteboard, or use graphics applications, but by all means draw pictures. Draw as many pictures as it takes to fully understand and agree on each requirement. Some of the most successful projects are accomplished by spending as much time in front of the white board as in front of the computer. This is by far the easiest and best method for interpreting and agreeing upon requirements. It can also be effective in weeding out requirements not considered, identifying potential or real bottlenecks, or deriving requirements. Some effective popular methods of graphically depicting requirements include the following:

- Data-flow diagrams.
- Flowcharting.
- Cross-functional flowchart (charted by responsibility).
- Information mapping.
- Entity relationship diagram.
- Unified Modeling Language (UML) use case.
- UML collaboration/communications diagram.
- UML state chart diagram.
- UML sequence diagram.
- UML activity chart.
- UML component diagram.
- UML deployment diagram.
- Structured analysis and structured design.
- Structured analysis and design technique.
- Integrated computer-aided manufacturing definition family of methods.
- State transition diagram.
- Object role modeling.
- Decision trees.
- Role activity diagrams.
- Petri net.

There are numerous methods for helping developers and customers to visualize the final product. The use of several of these diagrams is optimal when interpreting requirements in order to fully understand and communicate. They can be used with various levels of detail depending upon what the person is trying to convey. I have found that by keeping each diagram/picture as simple as possible, the greatest value can be derived so remember the *KISS* principle: Keep it short and simple. The level of detail in the initial diagrams should be flexible enough to leave room for analysis and optimization. There are a number of tools available to make using these methods easier.

Drilling Down/Chunking

Drilling down and chunking are ways of examining each requirement for full understanding. Drill down can be used to decompose requirements by starting at the basic high-level requirement and drilling down to the details of each requirement. After the high-level requirements are fully understood and agreed upon, the development team should move to fine-tune the details. Drill down should be iterative; as more details of higher-level requirements are understood, they are drilled down to lower levels for a complete understanding of what the customer is looking for. This is where the derived and implied requirements are defined. This should be accomplished until all requirements are fully fleshed out.

Drill down ensures that time and money is not wasted on detailing requirements that are misunderstood from the beginning. The decomposition must be approved in order to move forward to the next iteration, thus avoiding further misunderstandings. This step can be used to associate each requirement with a particular subsystem or component. It can also include verification/testing strategy and method generation.

Chunking is a method used to organize and arrange information so that it is easier to read, understand, access, and retrieve. It is a method of subdividing and organizing into short *chunks* of information in a uniform format. The simpler the better is key for certain types of information, with clutter being the villain. The premise is to group information and provide white space to break information into manageable components. This is sometimes called *visual chunking* and results in greater readability and accessibility. Chunking can be used effectively in understanding requirements especially if they are obtained from a statement of work or other typical task definition document. When you look at elements in logical groupings, you are able to remember and understand them much easier. There are things to avoid with chunking, including over-generalization and being overspecific. Keep it simple, but do not take away from content and run the risk of not being fully understood.

Balancing Act

One of the most difficult things encountered during requirements definition is balancing the customers and, ultimately, the user's needs versus their expectations. The needs are the identified requirements, which many times differ from their actual expectations, which are the unidentified requirements. Sometimes understanding their expectations can drive the understanding of the needs for defined requirements and gain insight into the *real* requirements. This can be accomplished by spending some time with the users, but with the caution that the designated decision maker is the one with the authority to add, change, delete, or alter the defined requirements in any way. It is easy for developers to get caught up in the *nice to have's* or *this is the way I do it* from users.

The second – and perhaps more important – balancing act is typically called *managing requirements creep*. Getting feedback from the customer is key, but at the same time a close eye must be kept on out-of-scope requirements that drive the project: schedule, cost, and risk. Flexibility is important to ensure that all the customer's requirements are met, which is the ultimate goal of any development project. In fact, for the Agile programmer, changing requirements are a way of life; they are expected and embraced. But the key is to balance these changing requirements with feasibility, applicability, and impactability to the system under development.

Iterative Requirements Interpretation

Interpreting requirements means that each requirement must be examined to ensure a full understanding. This is an iterative process of meetings for interviewing or brainstorming until all of the requirements have been fleshed out and beyond. Where did meetings get such a bad reputation? Used properly, meetings are a great tool for ensuring everyone is dancing to the same beat.

Many of the Agile methodologies use meetings on a regular basis. These meetings are called *stand-ups* or *scrums*. They are short, quick meetings first thing every morning to briefly discuss what actions are in progress. These meetings go a long way in ensuring no misunderstandings or misconceptions throughout the development process. The key is to make the meetings applicable and quick. They must include the appropriate project stakeholders. If side issues need to be resolved, this should be done in a separate meeting with only the necessary stakeholders. Remember that there are

designated decision makers who must be kept involved or at least informed at each step, especially when requirements change.

Summary

By always looking at things from each others vantage point, the software development world can start to overcome some of the nightmare scenarios so typical of software development projects. Most of these nightmares are directly attributable to requirements and the understanding of each requirement. By using some of the techniques discussed in this article, the typical Mary and Bob scenario can be avoided. Heed the battle cry: We can make projects more successful if we all work together. When all project stakeholders work from the perspective that this is a collaborative effort, then everyone wins.◆

References

1. The Standish Group. "CHAOS: A Recipe For Success, 1998." Standish Group International, 1999.
2. The Standish Group. "CHAOS Reports." Standish Group International <www.standishgroup.com>.
3. The Standish Group. "The Standish Group Report – CHAOS 1994." Standish Group International, 1995.
4. The Standish Group. "What Are Your Requirements?" Standish Group International, 2003.
5. Hayes, Frank. "Chaos Is Back." Computerworld Nov. 2004.
6. Carnegie Mellon University. "CMMI℠ for Systems Engineering/Software Engineering, Vers. 1.1, Staged Representation." CMU/SEI-2002-TR-002. Dec. 2001.
7. Jacobs, Deb. "Accelerating Process Improvement Using Agile Techniques." Auerbach Publications, 2006.
8. Carroll, Lewis. Alice's Adventures in Wonderland and Through the Looking Glass. Reissue edition. Signet Classics, 2000
9. Barnard, Frederick R. Printers' Ink. Mar. 1927.

Additional Reading

1. Bashar Nuseibeh, and Steve Easterbrook. "Requirements Engineering: A Road Map." 3rd International Symposium on Requirements Engineering <www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf>.
2. Scott Ambler. Ronin International, Inc. The Elements of UML Style <www.modelingstyle.info>.
3. McConnell, Steve. Construx Software <www.stevemcconnell.com> or <www.construx.com>.

About the Author

Deb Jacobs has more than 28 years of experience in information technology, including system and software engineering, project management, process improvement and proposal development and has helped many organizations be more successful in development and management. She has provided CMMI expertise to numerous organizations, including training, implementation, and assessments/appraisals. Jacobs is former *Spinout* newsletter editor/originator, former Computer Engineering Readiness Team conference chairperson, Infotec Deputy Software Tracks Chair, SEI CMMI contributor, and is a member of the CROSSTALK Editorial Board. Jacobs has authored several technical articles and the popular process improvement book *Accelerating Process Improvement Using Agile Techniques*. She has a bachelor of science in computer science.

Software Engineering Services
1508 JF Kennedy DR
STE 100
Bellevue, NE 68005
Phone: (402) 932-5349 or
(402) 292-8660
E-mail: djacobsfpa@aol.com or
djacobs@sessolutions.com

Letter to the Editor

Dear CROSSTALK Editor,

I just wanted to say that those were REALLY great articles (very detailed, nicely written). I got into *Star Wars* this summer, and I know how exciting it can be. See if you can beat this: I've watched every episode except *Revenge of the Sith*, I'm going to be Jango Fett the bounty hunter for Halloween, AND I'm going to be Luke Skywalker when my school has Spirit Week!

May the Force be with you.

— Katelyn Crombie, age 12

Coming Events

January 3-6
HICSS-40
Hawaii International Conference on System Sciences
Waikoloa, HI
/www.hicss.hawaii.edu/hicss_40/apahome40.htm

January 7-12
COMSWARE 2007
Second International Conference in Communication System Software and Middleware
Bangalore, India
www.comsware.org

January 8-12
CBIS '07 Chemical and Biological Information Systems Conference and Exhibition
Austin, TX
www.ndia.org/

January 17-19
CEA '07 Computer Engineering and Applications
Queensland, Australia
http://wseas.org/conferences/2007/australia/cea/

January 29-31
Enterprise Architecture Practitioners Conference
San Diego, CA
www.opengroup.org/sandiego2007

June 18-21, 2007
2007 Systems and Software Technology Conference
Tampa, FL
www.sstc-online.org