Requirements Engineering So Things Don't Get Ugly

Deb Jacobs
Focal Point Associates

Seasoned IT professionals remember those panicked moments when customers say, "That's not what we're looking for;" customer staff couldn't agree; requirements constantly changed; embarrassment when you couldn't develop requirements as promised; requirements are overlooked; estimates are skewed due to lack of understanding; and all the nice to have's *drove cost and schedule. Requirements engineering is a tough task for both the requirements receiver and the requirements giver, even if you know exactly what you want. How you see a requirement depends on what vantage point you're coming from. We must understand both points of view – giver and receiver – to truly be able to do effective requirements engineering.*

It was a cold, gloomy night in October. Outside the fog was so thick you could not see your hand in front of your face. Inside was even worse. Joe walked out of the conference room in a daze. He tried to remember what had happened but it all seemed like such a blur. The throbbing in his head grew even stronger. He hurried down the hall to the men's room. His thoughts were spinning. He asked himself, "What am I going to do?"

Joe had just learned the project he was working on would require more overtime. He kept thinking of how his wife had threatened to leave him just last week if he could not spend more time with her and the kids; in fact, he had not seen his kids in weeks. By the time he got home, they were in bed. Even when he did see them, he was so tired and frustrated he did not enjoy them. In fact, he had not really enjoyed life in a long time. It had actually started about a year ago when he had begun working on this project.

All of a sudden, the meeting came back to him. Voices screamed out in his head: "What do you mean that's not what you want? That's what the requirements say."

"That's not what we meant though. Don't you people understand anything?"

"We understand what you wrote down in the statement of work."

"But did you even bother to ask what we meant?"

"Well, we thought it was pretty clear."

"You missed the basic functionality we were looking for; in fact, this is so bad we're going to have to start completely over. And, by the way, we can't give you any slack on the schedule either."

For Joe, things were definitely getting ugly! This scenario may sound familiar to many of you. It happens time after time on project after project. So, how does it get like this?

How Does a Project Get Like This?

Sadly, for the information technology (IT) industry as well as their customers, studies show that the majority of systems are delivered with only about 42 percent to 67 percent of requirements. The Standish Group has found that even though projects are being delivered on time and within budget, the statistics for delivering requirements and meeting customer expectations are decreasing significantly [1].

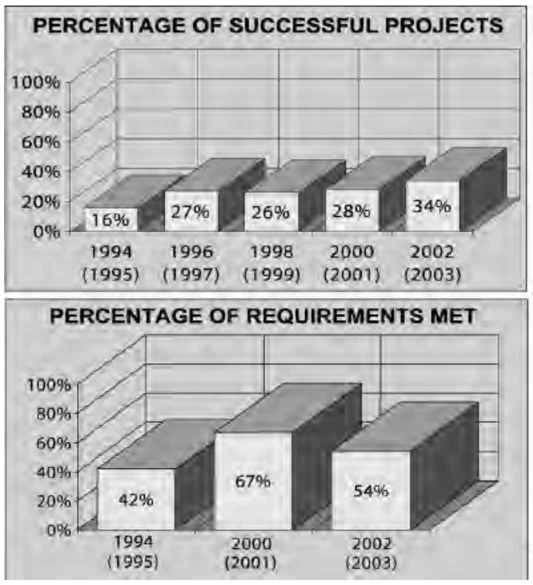
Figure 1 shows a summary of The Standish Group's reports concerning project success as well as the top 10 most important elements for successful projects. The Standish Group stated,

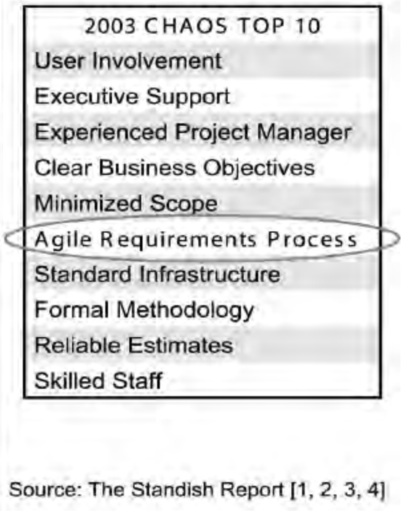
> We find that on average only 54 percent, down from 67 percent in 2001, of the originally defined features of a project are delivered. Even more troubling is the realization that of those features that are delivered – a full 45 percent are NEVER used. [4]

This article does not contain any new or eye-opening information; much of the information discussed is well known to requirements engineering experts. Best practices in requirements engineering have been honed since about 1968, and people have been writing about and teaching requirements engineering for several years; however, statistics continue to show that many IT practitioners and project managers still are not listening or have not been exposed to good requirements engineering.

In my positions over the last 20-plus years as project manager, software/technical project manager, software developer, systems engineer, process improvement engineer, new business proposal manager, and IT instructor, I have had the opportunity to be both the requirements giver and the requirements receiver. I have seen some very good examples of require-

Figure 1: *Standish Group Findings Summary*




2003 CHAOS TOP 10

- User Involvement
- Executive Support
- Experienced Project Manager
- Clear Business Objectives
- Minimized Scope
- Agile Requirements Process
- Standard Infrastructure
- Formal Methodology
- Reliable Estimates
- Skilled Staff

Source: The Standish Report [1, 2, 3, 4]

If Architects Had to Work Like Programmers

Dear Mr. Architect:

Please design and build me a house. I am not quite sure what I need, so you should use your discretion.

My house should have between two and 45 bedrooms. Just make sure the plans are such that the bedrooms can be easily added or deleted. When you bring the blueprints to me, I will make the final decision of what I want. Also, bring me the cost breakdown for each configuration so that I can arbitrarily pick one.

Keep in mind that the house I ultimately choose must cost less than the one I am currently living in. Make sure, however, that you correct all the deficiencies that exist in my current house (the floor of my kitchen vibrates when I walk across it, and the walls do not have nearly enough insulation in them).

As you design, also keep in mind that I want to keep yearly maintenance costs as low as possible. This should mean incorporating extra-cost features like aluminum, vinyl, or composite siding. (If you choose not to specify aluminum, be prepared to explain your decision in detail.) Please take care that modern design practices and the latest materials are used in constructing the house, as I want it to be a showplace for the most up-to-date ideas and methods. Be alerted, however, that the kitchen should be designed to accommodate, among other things, my 1952 Gibson refrigerator.

To ensure that you are building the correct house for our entire family, make certain that you contact each of our children, and also our in-laws. My mother-in-law will have very strong feelings about how the house should be designed, since she visits us at least once a year. Make sure that you weigh all of these options carefully and come to the right decision. I, however, retain the right to overrule any choices that you make.

Please do not bother me with small details right now. Your job is to develop the overall plans for the house: Get the big picture. At this time, for example, it is not appropriate to be choosing the color of the carpet. However, keep in mind that my wife likes blue.

Also, do not worry at this time about acquiring the resources to build the house itself. Your first priority is to develop detailed plans and specifications. Once I approve these plans, however, I would expect the house to be under roof within 48 hours.

While you are designing this house specifically for me, keep in mind that sooner or later I will have to sell it to someone else. It therefore should have appeal to a wide variety of potential buyers. Please make sure before you finalize the plans that there is a consensus of the population in my area that they like the features this house has.

I advise you to run up and look at my neighbor's house he constructed last year. We like it a great deal. It has many features that we would also like in our new home, particularly the 75-foot swimming pool. With careful engineering, I believe that you can design this into our new house without impacting the final cost.

Please prepare a complete set of blueprints. It is not necessary at this time to do the real design since it will be used only for construction bids. Be advised, however, that you will be held accountable for any increase of construction costs as a result of later design changes.

To be able to use the latest techniques and materials and to be given such freedom in your designs is something that cannot happen too often. Contact me as soon as possible with your complete ideas and plans.

Respectfully,

J.P. Anonymous

P.S.
My wife has just told me that she disagrees with many of the instructions I have given you in this letter. It is your responsibility as the architect to resolve these differences. I have tried in the past and have been unable to accomplish this. If you cannot handle this responsibility, I will have to find another architect.

P.P.S.
Perhaps what I need is not a house at all, but a travel trailer. Please advise me as soon as possible if this is the case.

ments engineering, and I have seen very tragic requirements engineering. I have become passionate about this topic based on these good and bad requirements experiences.

While working off and on as a process improvement engineer and process manager over the last several years with various organizations, I have had the distinct opportunity to learn about and use many exceptional tools. Some of the best tools I have found include the ever-popular, very effective Software Engineering Institute's Capability Maturity Model® (CMM®) Integration (CMMI®) [5] and the ISO [International Organization for Standardization] 9001.

I have developed processes for accomplishing requirements engineering based on numerous resources throughout my career. These processes have been developed based on the IT industry best practices documented in the CMM and CMMI as well as my own personal lessons learned. The techniques and process discussed in this article are a culmination of these process development efforts. Each organization must tailor a process to fit its particular needs but this process will provide an idea of the various aspects of requirements engineering that should be addressed in a successful requirements engineering process.

The one thing I have learned well, and heard many times from other IT professionals, is that requirements engineering is tough work! For the requirements giver, it is very hard to articulate requirements either in writing or verbally, even if you know exactly what you want. It is just as difficult for the requirements receiver to understand what others are trying to articulate. We tend to overlook seeing things from others' points of view. When engineers and clients start working together and understanding each other's points of view, we will truly be able to do effective requirements engineering.

The bottom line is this: Development teams must understand what they are building, or they cannot build it. This is only achievable through teamwork – developer and client teamwork.

Whose Responsibility Is Understanding Requirements?

To correct this prevalent problem, the IT industry as a group has to depart from the them-and-us attitude that permeates the industry: it must be just *us*. The finger pointing must stop, and we must start

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

working as a team, both the requirements receivers and requirements givers. The current trend toward agile/eXtreme programming (XP) [6] consists of several practices that lend themselves to accomplishing better requirements engineering. One significant aspect of agile/XP is working closely with the client throughout the development process, which is called Active Stakeholder Participation.

The following are some ideas that have worked well for others:

- Bill of rights or stakeholder contract.
- Approval process for all requirements.
- Win-win negotiations meetings that negotiate requirements based on technology, environment, time, effort, and budget constraints.
- Requirements team training; i.e., same training for all team members.

It is the development team's responsibility to learn to balance the stakeholder needs and expectations. The needs are the identified requirements and the expectations are the unidentified requirements. Sometimes the expectations drive the full understanding of the identified requirements. If you understand what the customer is looking for in terms of their expectations, you gain insight into what they have identified as the real requirements. It is the customer's responsibility to articulate their expectations so that the development team fully understands what they are looking for in the resulting product.

For both the development team and the customer, there must be a clear understanding of who are the decision makers or final authorities for requirements. This includes someone who can do the following:

- Add or approve a new requirement.
- Change an existing requirement.
- Accept changes to requirements.
- Direct the developer or their manager.
- Determine if a requirement has or has not been met.
- Accept requirements as met or not met.

A graphical depiction can help immensely in defining and keeping track of who's who. These 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.

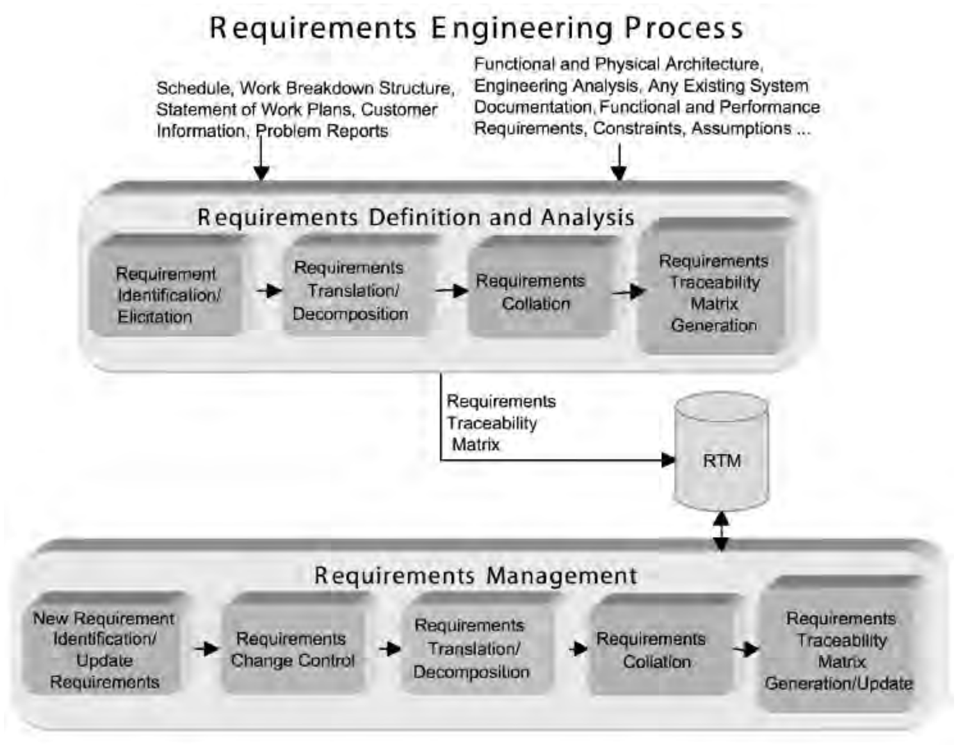

Requirements Engineering Process
Schedule, Work Breakdown Structure, Statement of Work Plans, Customer Information, Problem Reports
Functional and Physical Architecture, Engineering Analysis, Any Existing System Documentation, Functional and Performance Requirements, Constraints, Assumptions ...
Requirements Definition and Analysis
Requirement Identification/ Elicitation
Requirements Translation/ Decomposition
Requirements Collation
Requirements Traceability Matrix Generation
Requirements Traceability Matrix
RTM
Requirements Management
New Requirement Identification/ Update Requirements
Requirements Change Control
Requirements Translation/ Decomposition
Requirements Collation
Requirements Traceability Matrix Generation/Update

Figure 2: *Requirements Engineering Process*

Why Is Requirements Engineering So Important?

We, as an industry, cannot afford the consequences of not doing requirements engineering effectively. The cost of incorrect, misunderstood, and not agreed upon requirements affects all of us in terms of time, money, and lost opportunities. The results can be confusion, distrust, misdirection, frustration, lack of quality, higher cost, overtime, a general lack of understanding, and incapability due to being ill-equipped to handle issues.

Requirements engineering is a means of providing the functions and related characteristics of systems by providing the tools, concepts, and methods that mediate between the providers of information technology services and products, and the users or markets for the services and products. It is a means of providing the necessary communications to define needed products. Misunderstood, wrong, or even slightly skewed requirements propagate as the project moves forward until you get to the testing phase and scenarios like those discussed earlier occur.

Like dominoes, once problems start, they proliferate throughout the project – requirements problems at the beginning proliferate through design, development, and, finally, into test. Many times it gets to the point where starting over takes less time than trying to fix what you have already done. The sidebar "If Architects Had to Work Like Programmers" illustrates this point very well.

What Are Requirements?

Requirements tell the development team what the customer is contracting the team to build. As a whole, they provide a means of determining the functionality and attributes of the resulting product. The Institute of Electrical and Electronics Engineers [7] defines a requirement as the following: (1) a condition or capability needed by a user to solve a problem or achieve an objective; (2) a condition or capability that must be met or possessed by a system or system component to satisfy a contract, standard, specification, or other formally imposed documents; and (3) a documented representation of a condition or capability as in (1) or (2).

Proven Requirements Engineering Process

CMMI provides a good foundation for requirements engineering. It describes what should be included in an effective requirements engineering process. CMMI is based on best practices and lessons learned from the IT community, including both government-related and private industry. There are, in fact, several good taxonomies and methodologies that have been defined for requirements engineering. The requirements engineering process illustrated in Figure 2 and described in this article has proven effective on numerous successful projects and includes these basic best practices.

There are two major phases that are essential in defining and controlling requirements: Requirements Definition and Analysis, and Requirements Management.

Requirements Definition and Analysis

Requirements Definition and Analysis sets the stage for all subsequent tasks in developing the resulting product. Getting this right is key to the success of the overall project. A domino effect will begin here due to wrong, misunderstood, or slightly skewed requirements. The rule of thumb I have learned during my career is approximately 15 percent of project time should be spent on identifying, defining, and clarifying the requirements. This will vary depending upon the life-cycle methodology selected but it works well for most projects.

The following list includes some examples of typical inputs to the Requirements Definition and Analysis task. The inputs will depend upon an organization's processes, customer's processes, and whether the system being developed is an upgrade, major refurbishing of an existing system, or a new system.

- **Functional and performance requirements.** This information can be obtained using many methods but could simply be a list or a verbal exchange.
- **Statement of work.** Typically provided by the customer to the development team. Can be a key document for the customer and development team.
- **Plans.** Projects produce numerous plans that may drive some requirements such as the project plan, configuration management plan, logistics plan, communications plan, development plan, or engineering plans.
- **Customer information.** Various customer information can be derived that drives requirements such as customer standards, including user interface standards and security standards.
- **Problem reports.** Many times problem reports drive major system upgrades or refurbishment. The original problem report contains a good deal of information pertinent to understanding requirements.
- **Schedule.** The schedule may contain some information needed to understand what the customer is looking for and the complexity expected, especially if the customer provides the overall schedule.
- **Work Breakdown Structure (WBS).** This provides information concerning the breakdown of requirements if it is developed using a WBS method that breaks the project down by product functionality.
- **Architecture (physical and functional).** For existing systems, this can be key to understanding requirements such as communications protocols, existing functionality, interfaces, etc. For new systems, the customer may provide this information in a statement of work and existing system documentation may help understand interfacing systems and data.
- **Engineering analysis.** Many times one or more trade studies or prototypes are developed that will help in understanding requirements.
- **Constraints.** There are many forms that constraints can take, including time, cost, and technical.
- **Assumptions.** Development team assumptions will drive the requirements and understanding of requirements. These should always be documented and discussed with the customer. Conversely, customers have assumptions that also must be communicated.
- **Existing system documentation.** Existing system documentation even when not up to date can provide invaluable information for understanding requirements.

Table 1: *Requirements Identification Techniques*

Technique	Brief Description
Interviews (structured and unstructured)	Meetings can be structured, unstructured, or a combination of both. • Structured: Interviewer has predetermined questions. • Unstructured: Interviewer may have some preliminary questions, but much less formal with random discussions.
Group Brainstorming	These are informal meetings that generate discussion of requirements. Basically used to generate ideas. Should have defined rules of conduct; avoid a *free-for-all*.
Observation	When upgrading systems, it is extremely helpful to observe a system in its operational environment. When the automated system is not available, observing the manual functions helps understanding.
Analysis of Existing Documentation	When upgrading systems, it is extremely helpful to analyze existing documentation (development and user documents). When the automated system is not available, some documentation may be available describing the functionality.
Questionnaires and/or Surveys	Questionnaires and surveys can give the development team insight into expectations (a word of caution: appropriate customer representative should always approve each requirement and derived requirement).
Prototyping	Prototyping is a great tool to use to simulate the final product especially for large systems by providing a visualization of the system or parts of a system. It is very effective in exposing any misunderstandings, risks, or missing functionality; clarifying confusing functionality; defining derived requirements; and avoiding being trapped in loops of refinement during design and coding. The two types of prototypes include discovery (aka throw-away) and production (aka evolutionary).
Conceptualization/ Modeling	Conceptualization/modeling helps to clarify and prove requirements, weed requirements, clarify gaps in requirements, generate derived requirements, discover rules and procedures and functions/algorithms needed, ferret out data needs, and ensure correctness of the defined requirements. Effective in meetings (interviews and brainstorming) to describe how a system will behave, to describe interfaces (internal and external), and to discuss the consequences of each requirement using a whiteboard or paper.
Cognitive	Examining usability of requirements based upon cognitive characterizations of user interaction. This technique incorporates the human factors into requirements definition. Some methods include protocol analysis, laddering, card sorting, and repertory grids.

Requirements Identification/ Elicitation

The Requirements Identification/Elicitation step provides an in-depth description of the desired resulting product. Some of the techniques used to identify and analyze requirements include those shown in Table 1.

I always recommend that one or more of these techniques be used to fully understand and communicate requirements throughout the project. The better the requirements are understood, the more likely the resulting system will be effective for the customer. During Requirements Identification/Elicitation, several questions need to be addressed that are shown in Table 2.

There is no simple formula for writing good, useable requirements; however, sources for writing good, useable requirements can be found on the Internet.

Requirements Translation/ Decomposition

Once requirements have been identified, each requirement must be examined for a full understanding. Just as important is getting agreement between all project stakeholders, especially the identified decision makers. This step will be where implied or derived features/issues are uncovered. This is an iterative process where the interviewing and brainstorming sessions discussed in Table 1 are critical tools. These tools should continue to be used until all requirements are fully flushed out and beyond.

Drill down should be used to decompose requirements by starting at the basic high-level requirement and drilling down to the details of each requirement only after each level of detail is fully understood. Hence, only after the high-level requirements are fully understood and agreed upon should a development team move to finer 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. Drill down ensures that time and money are not wasted on detailing requirements that are misunderstood from the beginning.

During this step, requirements should be associated with a particular subsystem.

Requirements Collation

Grouping and prioritizing requirements are key to managing them. Requirements should be grouped for easier understanding, assignment, allocation, and tracking. The categories should be based upon the project needs; some suggestions include the following: function, effect of result, cause, impact and priority, timing, exception handling, and performance criteria. Function is the most prevalent and understandable categorization method.

Prioritization should be given to each requirement to understand its importance. Prioritization will help determine the sequence of tasking as well as weed out the essential versus the desirable versus the optional requirements.

Each requirement should be examined to determine any technical, cost, or schedule implications or risks. Any risks associated with each requirement should be recorded and tracked using the project's risk management process. The impact and the potential for occurrence will be key factors in managing risk.

Any impractical and excessive requirements should be weeded out since they drive cost and schedule. Multiple requirements pertaining to the same functional/performance feature should be examined to ensure they are coherent and consistent. Finally, requirements should be allocated to system components for assignment. This can be done using a WBS if it has been designed using that method.

Requirements Traceability Matrix Generation

A very useful tool in managing requirements is a Requirements Traceability Matrix that is generated with the complete set of requirements. The matrix provides an authorized record of the requirements. The tool selected for the matrix will depend upon the size and scope of the project. A database is the optimal method for managing requirements but a simple spreadsheet can also be very effective. Several very good commercial tools are available; see <www.incose.org/tools/tooltax/reqtrace_tools.html> [8].

Creation of a homegrown requirements database will give users exactly what they are looking for if the expertise and time are available. Requirements tools have many advantages over a simple listing, including easy search, smooth requirements management, requirements change control, requirements metrics collection with minimal effort, and any needed documentation. The key to selecting the right tool is to ensure that you are getting the bang for the buck.

In [4], The Standish Group states, "Only 5 percent of new and changing applications will use a requirements management tool." That could be why we have many of our requirements problems.

Requirements Management

The requirements management phase consists of monitoring and controlling the requirements throughout the remaining development life cycle. Monitoring and controlling requirements ensures that the resulting system has all of the agreed upon or authorized requirements. It helps to avoid the widespread requirements epidemic known as requirements creep.

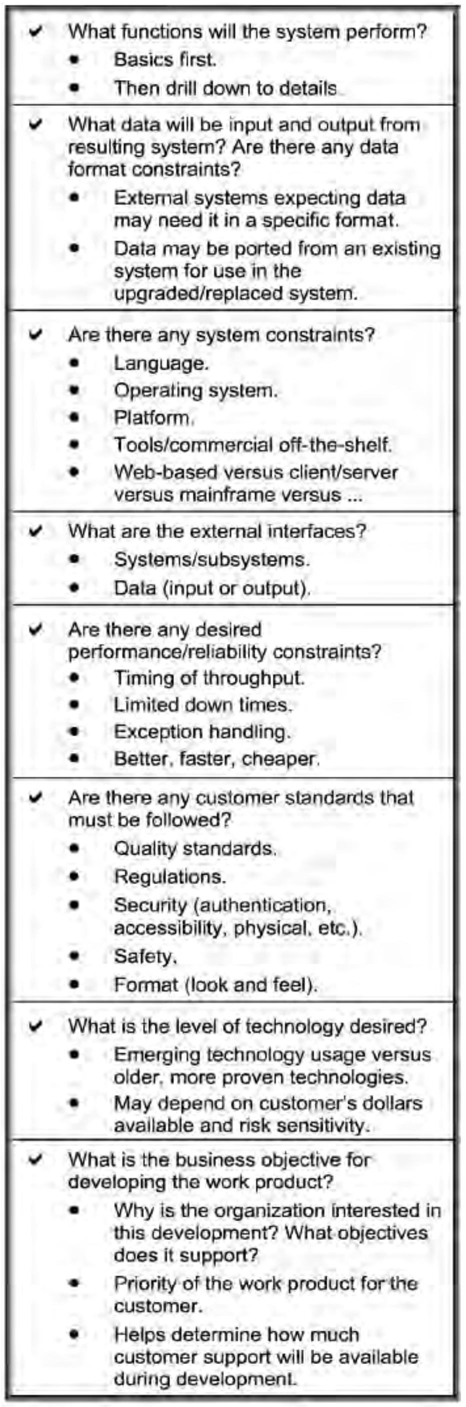
Requirements creep can drive both cost and schedule significantly. When a new or upgraded requirement is identified, the development team must go through the same process as defined for the Requirements Definition and Analysis phase with the appropriate decision makers.

New Requirement Identification/ Update Requirements

As new requirements are identified or existing requirements change, they must be updated in the requirements baseline. Requirements should be maintained and baselined using the same type of configuration management controls as software such as the four elements of configuration management: identification, change management (the key), status accounting, and verification and audit.

Some configuration management tools have built-in requirements management features. This ensures the integrity of the

Table 2: *Requirements Engineering Process*



✓ What functions will the system perform? • Basics first. • Then drill down to details.
✓ What data will be input and output from resulting system? Are there any data format constraints? • External systems expecting data may need it in a specific format. • Data may be ported from an existing system for use in the upgraded/replaced system.
✓ Are there any system constraints? • Language. • Operating system. • Platform. • Tools/commercial off-the-shelf. • Web-based versus client/server versus mainframe versus ...
✓ What are the external interfaces? • Systems/subsystems. • Data (input or output).
✓ Are there any desired performance/reliability constraints? • Timing of throughput. • Limited down times. • Exception handling. • Better, faster, cheaper.
✓ Are there any customer standards that must be followed? • Quality standards. • Regulations. • Security (authentication, accessibility, physical, etc.). • Safety. • Format (look and feel).
✓ What is the level of technology desired? • Emerging technology usage versus older, more proven technologies. • May depend on customer's dollars available and risk sensitivity.
✓ What is the business objective for developing the work product? • Why is the organization interested in this development? What objectives does it support? • Priority of the work product for the customer. • Helps determine how much customer support will be available during development.

requirements. As requirements change – and they will – the changes must be controlled.

Requirements Change Control

Either formal or informal change control methods can be used. Formal change mechanisms include using a Configuration Change Board and appropriate formal authorizations. Less formal methods can also be effective, especially for smaller projects.

Whether a formal or informal change control method is selected, it is important that the identified decision makers finalize and authorize all requirements changes. This includes management of even the smallest detail since even a slight change can alter elements of the product. The changes must be coordinated with all stakeholders since they may have an impact on the tasks they are assigned either directly or indirectly.

Recycle Definition and Analysis

Once the new or upgraded requirements are approved, they must undergo the same process as the initial requirements.

- Requirements Translation/Decomposition.
- Requirements Collation.
- Requirements Traceability Matrix Generation/Update.

It is important to ensure that changes to requirements do not impact other requirements. Many times even simple changes will have a ripple effect on other requirements. The development team must be prepared to handle these changes. The Requirements Traceability Matrix should always reflect the current requirements as they are at any point in the project. They will be the authorized record for the resulting product.

Requirements Volatility Metric

Several metrics will help determine the status of a project but a key metric is requirements volatility, which is considered a key project success indicator. It indicates the stability of the baselined requirements.

How much change is too much and at what stage of the development cycle will it have a significant impact? There are some rules of thumb for how much requirements change is too much. "A Gentle Introduction to Software Engineering" indicates,

> The accepted requirements volatility metric is 1 percent of requirements per month. If it is much less, one should ask oneself if the system would be desirable to its intended audience. If it is much more than 2 percent a month, development chaos is all but assured. [9]

Other sources also use that rule of thumb; however, a study accomplished at the Colorado State University, Department of Computer Science concluded the following:

> All the results show that changes have more influence on defect density when they occur closer to the end of the testing effort. This temporal dependence is generally exponential. Changes made very early can be relatively inconsequential, but those occurring later can raise defect density quite significantly. [10]

I have found in my experience that frequent changes to requirements are expected during the early stages of the project; however, a high volume of changes late in the development life cycle can have a significant impact to functionality, interfaces, cost, and schedule. The amount of acceptable requirements change can depend upon many factors, including the project phase, development team, requirements complexity, system complexity, system size, customer expectations, schedule, technology, methodologies, tools, etc.

If frequent changes are expected, it may be beneficial to use either an iterative build life cycle such as the spiral or incremental build or an agile/XP approach. There is an upside and a downside to all methods; the key is to select the method that is right for that development team, the customer, the system being developed, and the environment.

The Bottom Line

Time after time, projects experience nightmare scenarios similar to the one described at the beginning of this article. It is key to a project's success in delivering the customer's needed functionality that development teams and customers work as a team to develop effective requirements in order to develop effective products. If we look at things from each other's vantage point, the chance of success grows by leaps and bounds. We all look at things differently based on our background, education, experience, and simply from where we are standing at the moment. Open communications and respect for each other's position is crucial.

There is enough to panic about when developing a system without the added stress of misunderstandings. Products must be delivered with better numbers than 42 percent to 67 percent of the required functionality. Nobody can afford the consequences of wrong, misunderstood, or even slightly skewed requirements.

The bottom line is this: Always remember that what you see is relative to where you are standing. We must all work as a team and select the best methodologies, techniques, and tools that keep things simple so things do not get ugly.◆

References

1. The Standish Group. CHAOS: A Recipe for Success. West Yarmouth, MA: The Standish Group International, Inc., 1999 <www.standishgroup.com/sample_research/PDFpages/chaos1999.pdf>.
2. CHAOS Reports <www.standishgroup.com>.
3. The Standish Group. The CHAOS Report (1994). West Yarmouth, MA: The Standish Group International, Inc., 1995.
4. The Standish Group. What Are Your Requirements? West Yarmouth, MA: The Standish Group International, Inc., 2003, Standish Group (based on 2002 CHAOS Report).
5. CMMI Product Team. CMMI^{SM} for Systems Engineering/Software Engineering, Vers. 1.1, Staged Representation. Pittsburgh, PA: Software Engineering Institute, Dec. 2001.
6. The Official Agile Modeling (AM) Site <www.agilemodeling.com>.
7. Institute of Electrical and Electronic Engineers. IEEE Software Engineering Standards Collection: 1994 Edition. Washington, DC: IEEE 1994.
8. International Council on Systems Engineering <www.incose.org>.
9. Cook, David A., Leslie Dupaix, and Larry Smith. "A Gentle Introduction to Software Engineering." Rev. 3.0. Hill Air Force Base, UT: Software Technology Support Center, 31 Mar. 1999.
10. Gotel, Orlena C.Z., and Anthony C.W. Finkelstein. "An Analysis of the Requirements Traceability Problem." London, England: Imperial College of Science, Technology, and Medicine <http://csis.pace.edu/~ogotel/papers/RT_PAP.pdf>.

Additional Reading

1. Nuseibeh, Bashar, and Steve Easterbrook. "Requirements Engineering: A Road Map." 3rd International Symposium on Requirements Engineering, Toronto, Canada, 2000

<www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf>.
2. Requirements Working Group of the International Council on Systems Engineering. "Characteristics of Good Requirements." INCOSE Symposium.
3. Bernard, Frederick R. Printers' Ink. Mar. 1927.
4. Malaiya, Yashwant K., and Jason Denton. "Requirements Volatility and Defect Density." Ft. Collins, CO: Colorado State University.
5. Bamford, Robert, and Bill Deibler. SSQC. Requirements Engineering Workshop <www.ssqc.com>.
6. Ambler, Scott. The Elements of UML Style. Cambridge University Press, 18 Nov. 2002.
7. Christel, M., and K. Kang. Issues in Requirements Elicitation. Pittsburgh, PA: Software Engineering Institute, 1992.
8. McConnell, Steve. Construx Software. <www.stevemcconnell.com> or <www.construx.com>.
9. Robertson, Suzanne, and James Robertson. Mastering the Requirements Process. Addison-Wesley, 1999.
10. Hooks, Ivy. Writing Good Requirements. Proc. of the Third International Symposium of the NCOSE. Vol. 2, 1993. Updated Sept 2003.
11. Wiegers, Karl. Writing Good Requirements. 2nd ed. Microsoft Press, 26 Feb. 2003.
12. KPMG. "What Went Wrong? Unsuccessful Information Technology Projects." KPMG Study <www.kpmg.ca>.

About the Author

Deb Jacobs is a professional consultant for Focal Point Associates specializing in process improvement and project management. She provides support to organizations in training, process improvement consulting, project management consulting, software engineering consulting, and proposal development. Jacobs has more than 25 years experience in system/software engineering, project management, process improvement, and proposal development. Her notable successes include leading a successful Capability Maturity Model® (CMM®) Level 3 effort in one year, successfully reorganizing struggling projects, mentoring new managers, and gaining new business for companies through proposal development. She is former *SPINOUT* editor/originator; former Computer Emergency Response Team conference chairperson, infotec deputy Software Tracks chair, and a Software Engineering Institute CMM Integration^{SM} contributor. She is currently working on a book to help organizations successfully achieve process maturity at minimal costs. Jacobs has a Bachelor of Science in computer science.

Focal Point Associates
c/o Priority Solutions
1508 JF Kennedy DR STE 100
Bellevue, NE 68005
Phone: (402) 932-5349
(402) 292-8660
E-mail: djacobsfpa@aol.com
djacobs@prioritytech.com
djacobs@sessolutions.com

Web Sites

Project Management Institute
www.pmi.org
The Project Management Institute (PMI) is a not-for-profit, project-management professional association with more than 100,000 members in 125 countries. PMI publishes "A Guide to the Project Management Body of Knowledge," offers Project Management Professional certification, and maintains ISO 9001 certification in Quality Management Systems.

Software Program Managers Network
www.spmn.com
The Software Program Managers Network (SPMN) is sponsored by the deputy under secretary of defense for Science and Technology, Software Intensive Systems Directorate. It seeks out proven industry and government software best practices and conveys them to managers of large-scale Department of Defense software-intensive acquisition programs. The SPMN provides consulting, on-site program assessments, project risk assessments, software tools, guidebooks, and hands-on training.

NASA Independent Verification and Validation Facility
www.ivv.nasa.gov
The NASA Independent Verification and Validation (IV&V) Facility was established in 1993 to provide the highest achievable levels of safety and cost-effectiveness for mission critical software. The IV&V Facility's efforts have contributed to the improved safety record of NASA since its inception. The IV&V Facility houses more than 150 full-time employees and more than 20 in-house partners and contractors.

Practical Software and Systems Measurement
www.psmsc.com
Practical Software and Systems Measurement (PSM) is sponsored by the Department of Defense and the U.S. Army. PSM is an information-driven measurement process that addresses the unique technical and business goals of an organization by providing objective information needed to successfully meet cost, schedule, and technical objectives.

International Society of Parametric Analysts
www.ispa-cost.org
The International Society of Parametric Analysts (ISPA) is a professional society dedicated to the improvement and promotion of parametric cost modeling techniques and methodologies and the related fields of risk analysis, econometrics, design-to-cost, technology forecasting, and management. ISPA provides a forum that encourages the professional development of its members through the interchange of ideas and perspectives. ISPA members represent government agencies, universities, and nearly 200 organizations in 12 countries.