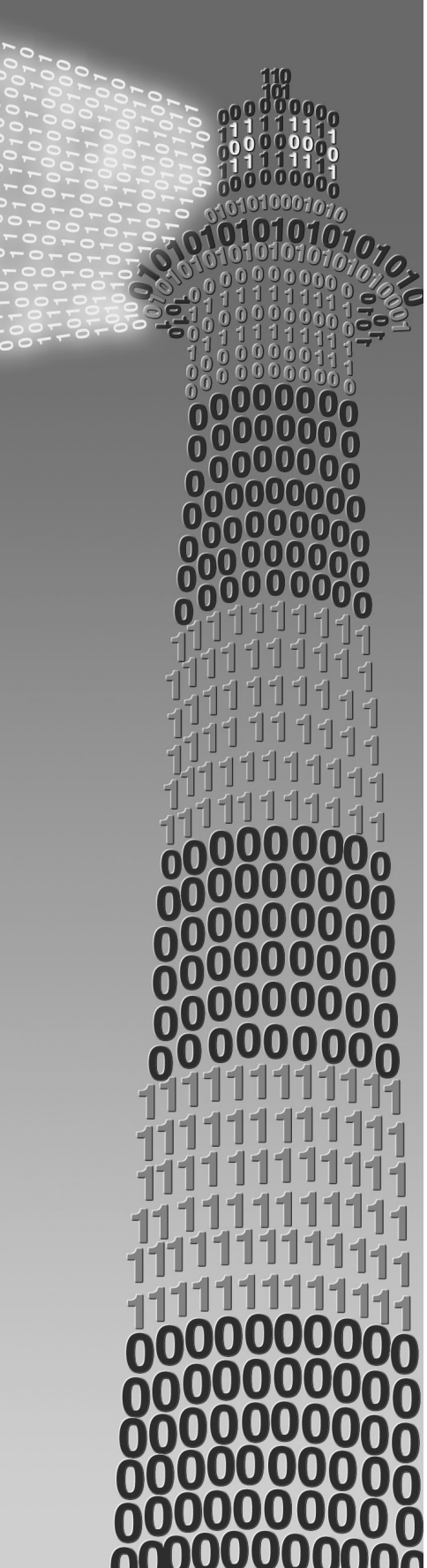
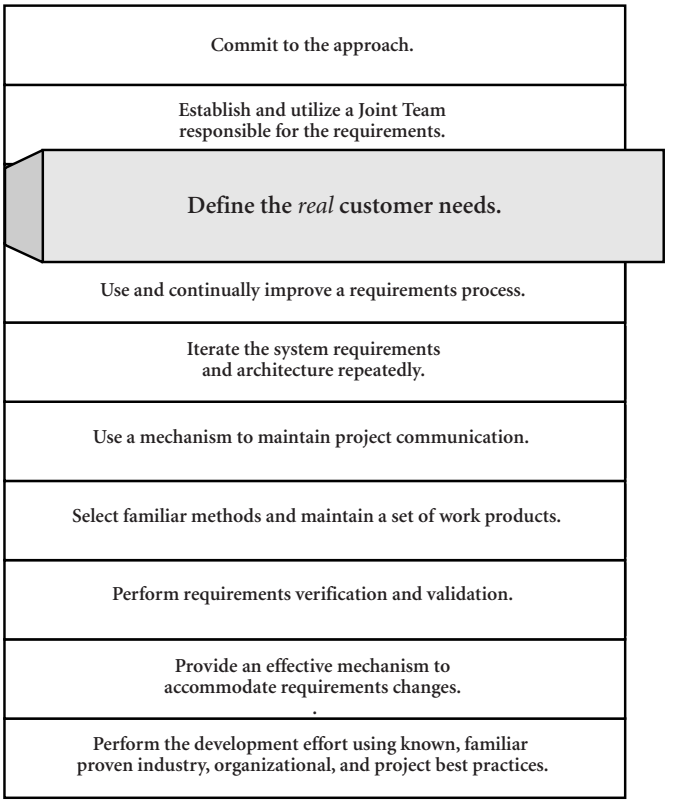


CHAPTER FOUR

Define the *Real* Customer Needs



This chapter provides specific recommendations to determine the real requirements for a planned system.

Most any supplier that has designed, developed, and implemented a system for a buyer would assert that it had "performed extensive effort to define the real customer needs." All suppliers go to extensive efforts to meet the needs of buyers of systems. Why, then, this chapter?

As discussed in Chapter 1, industry experience indicates that systems provided by suppliers often do not meet customer needs. In spite of extensive efforts, suppliers fail to measure up to expectations with the delivered systems.

I distinguish between *real* customer requirements and needs and *stated* requirements and needs. There is a huge difference between the two, and this difference accounts for many of our requirements-related problems. Historically, clients have not been able to articulate their real customer requirements and needs. Accordingly, an effective requirements process must provide for the time, resources, mechanisms, methods, techniques, tools, and trained requirements engineers familiar with the application domain to define the real customer requirements and needs.

This problem is not limited to large systems. Small projects[1] also experience the failure to identify the real requirements. My experience is that the practices presented in this book are applicable to projects of all sizes.[2] The differences are in the **tailoring** of the implementation approach. I provide some suggestions in the following chapters.

---

[1]There is no industry agreement on the definition of a "small project." One could consider it a "team." Often it is considered a project involving one to six professionals operating for as long as three to six months, but this definition is arbitrary. Consideration has been given in the industry literature to whether "small projects" are really all that different from "medium-size" or even "large" projects. See Mark Paulk, *Using the Software CMM with Judgment: Small Projects & Small Organizations;* Rita Hadden, *Now What Do We Do?;* and Louise Williams, *SPI Best Practices for Small Projects*. Members of small projects should be encouraged to take what they can from the experiences of larger projects by tailoring the approach, rather than using smallness as an excuse for not taking advantage of industry lessons. For a perspective giving careful attention and focus to "smallness," see Brodman and Johnson, *The LOGOS Tailored CMM for Small Businesses, Small Organizations, and Small Projects*. The changes tailor the Capability Maturity Model for Software (SW-CMM) for a small project environment. Participants in small projects or organizations may find this reference helpful.

[2]Rita Hadden's view based on observations and experience with more than 50 small projects is that professional judgment can be used to scale down and apply key practices appropriately to achieve positive outcomes for small projects. See "How Scalable Are CMM Key Practices?"

Industry consultant Karl Wiegers expresses the problem this way:

> Requirements exist in the minds of users, visionaries, and developers, from which they must be gently extracted and massaged into a usable form. They need to be discovered with guidance from a talented requirements engineer who helps users understand what they really need to meet their business needs and helps developers satisfy those needs. Few project roles are more difficult than that of the requirements engineer. Few are more critical.[3]

This chapter provides several recommendations to facilitate getting to the real requirements. Obviously, if we're not using a base of the *real requirements* to perform our system development work, huge amounts of resources are being misspent. These recommendations will help you to redirect these resources in ways that will produce better results.

Recommendations to Facilitate Getting to the *Real* Requirements

The following recommendations help to explain and perform an improved approach and are discussed in turn in the following subsections:

1. Invest 8% to 14% of total program costs on the requirements process. Spend additional time and effort near the beginning of a project to work to identify the real requirements. Ensure joint user and supplier responsibility for requirements. Facilitate clarification of the real requirements. Control changes to requirements.
2. Train program and project managers (PMs) to pay more attention to the requirements process.
3. Identify a **project champion**. A project champion is an *advocate* for the effort, is very familiar with the set of real customer needs for a system, and provides an active role in the development activities, facilitating the tasks of the development team.
4. Develop a definition of the project vision and scope.
5. Identify a requirements engineer and utilize **domain expert**s to perform requirements engineering tasks.
6. Train developers not to make requirements decisions and not to **gold plate**.

[3]Karl Wiegers, "Habits of Effective Analysts," p. 65.

7. Utilize a variety of techniques to elicit user requirements and expectations. Use a common set of techniques and tools among all parties involved in a particular project.
8. Train requirements engineers to write good requirements.
9. Document the rationale for each requirement.
10. Utilize methods and automated tools to analyze, prioritize, and track requirements.
11. Utilize peer reviews and inspections.
12. Consider the use of formal methods when appropriate.

The quantity of high-level system requirements for a large system should be on the order of 50 to 200 requirements, not in the thousands (based on Ivy Hooks's experience in supporting requirements efforts at the National Aeronautics and Space Administration for several years). Requirements should be documented graphically and textually and should be made visible to all stakeholders. One way to accomplish this is to invite stakeholders to participate in requirements reviews. A requirements review is a workshop involving the key stakeholders of a project for a short, intensive session that focuses on the definition or review of requirements for the project. Ideally, it is facilitated by an experienced outside facilitator or by a team member who can objectively process inputs and feedback.

Let's review each of these recommendations in turn.

Invest More in the Requirements Process

Many people think of the requirements process as being primarily limited to requirements management, that is, tracking the status and change activity associated with requirements and tracing requirements to the various activities and products of the development effort. Projects expend an estimated 2% to 3% of total project costs on this activity.[4] It is advantageous to define the requirements process more broadly and to expend 8% to 14% of total program costs on it. Special emphasis should be placed on joint user and supplier responsibility for requirements, getting to the real requirements, and controlling changes to requirements.

We know from experience that buyers most often provide suppliers of systems a definition of their requirements ("stated requirements"). This definition

[4]Rob Sabourin notes from his extensive consulting work that it is amazing how many organizations and companies do not have *any* requirements process (comment included in Sabourin's review of this manuscript).

may be provided in the form of a statement of work, a Request for Proposal, a **requirements document**, a background description of a problem or need, and in other formats or combinations thereof. Buyers often have strong beliefs about their requirements documents and are strongly committed to their accuracy and validity. The reasons for this are easy to understand: Our customers have a lot of experience in their work and much expertise concerning it. They have spent a lot of time and money developing these **artifacts**. Often, the time spent in internal meetings discussing requirements and working out details about them clarifies, in the minds of the involved individuals, the specific details and characteristics. However, we note that almost always there are differences of opinion *within a customer organization* concerning important aspects of some of these details. It may be that the person writing the requirements in the customer's organization is not the person who is the intended user. Also, experience has shown that people with strong technical skills are not always effective communicators or writers.

Perhaps a valid criticism of work in our industry is that we often accept these artifacts as being complete and accurate and proceed with the task of responding to these requirements—that is, of designing and developing an approach to meet the *stated requirements.*

Experience suggests that we would be well advised to conduct partnering workshops and requirements reviews; to apply other mechanisms, methods, techniques, and tools; and to undertake a concerted effort in partnership with our customers to discover and evolve the *real requirements*.[5] A typical e-commerce application is required to be compatible with things that do not yet exist, implying that developers must be able to **hot swap** software.[6]

---

[5]Goguen regards requirements as "emergent, in the sense that they do not already exist, but rather *emerge* [emphasis added] from interactions between the analyst and the client organization." This is useful because conventional methods of requirements elicitation often assume that users know (1) exactly what they want from a future system and (2) how this system, once implemented, will affect the way they work. Common sense tells us these are not known in the early stages of any effort. See Jirotka and Goguen, *Requirements Engineering: Social and Technical Issues,* p. 194.

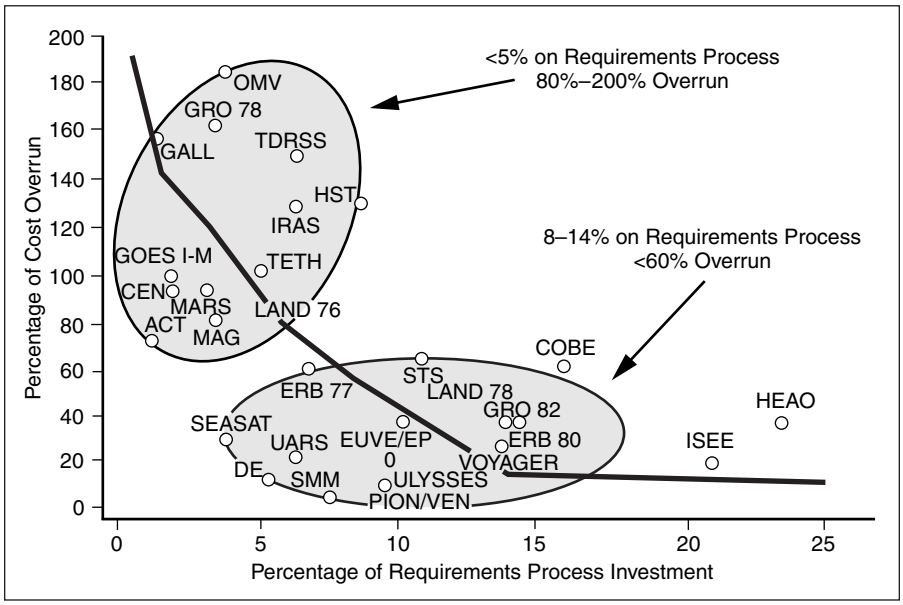
[6]*Hot swap* is a term taken from the **hardware** world. It means one can take out a board or component while the system is running and replace it with a new one without shutting down the system. For e-commerce systems we often use multiple servlets/server applications. To "hot swap" is to replace one of them without shutting down the e-commerce site and without losing a transaction. Some software engineering applications should ensure that components are designed to permit "hot swapping." This allows for reaction to new and evolving requirements without shutting down a system.

Some data from industry experience will clarify this point. It's preferable to utilize available data whenever possible to make decisions rather than to rely on intuition, experience, or the suggestions of others. We should "manage by fact." Figure 4-1 shows the effect of investment in a requirements process on total program costs. These data were provided by Werner M. Gruehl, Chief, Cost & Economic Analysis Branch, National Aeronautics and Space Administration headquarters, and were reported by Ivy Hooks.[7] Note that projects that spent less than 5% of total project or program costs on the requirements process experienced an 80% to 200% cost overrun, whereas those that invested 8% to 14% experienced less than a 60% overrun. *These data provide a powerful message to PMs and requirements practitioners: An expenditure of 8% to 14% of total program costs on the requirements process results in the best outcomes as measured by total program costs.*

Train PMs to Pay More Attention to the Requirements Process

Hooks addresses another key issue in her paper "Why Don't Program and Project Managers (PMs) Pay More Attention to the Requirements and the Requirements

Figure 4-1 Effect of Requirements Process Investment on Program Costs



[7]Hooks, *Managing Requirements,* pp. 1–2.

Process?" From her 30 years of experience in consulting concerning requirements, she concludes the following:

- PMs assume that everyone knows how to write good requirements, thus the requirements process "will take care of itself."
- PMs tend to come from a technical background and tend to focus on the nontechnical aspects of the program because these are new and alien.
- PMs know they do not fully understand budgets, so more attention goes to budgets.
- The PM's boss is focused on the budget, so the PM places more attention on what interests the boss.

This analysis is consistent with my experience. It is my sincere hope that one use of this book will be to provide practitioners the experience and data to encourage PMs to provide adequate funding for requirements-related activities and to pay close attention to the requirements and the requirements process.[8] This is obviously an issue that needs to be addressed in corporate and organizational training programs for PMs.

Steve McConnell[9] advocates that technical managers should have tools for five kinds of work: estimating, planning, tracking, managing risk, and measuring. He also observes that management skills have at least as much influence on development success as technical skills.

Identify a Project Champion

Among the industry experts in requirements engineering are Dean Leffingwell and Don Widrig of Rational Corporation. Their recent book, *Managing Software Requirements,* is highly recommended. It presents a very useful approach that is focused on utilizing trained teams to perform systems development activities. Chapter 18 of *Managing Software Requirements,* The Champion, provides an excellent discussion of the need for and role of a champion. In their experience

[8]SECAT LLC publishes a set of four pocket guides, each designed for a person with specific job responsibilities: the PM, an organizational leader, a system engineer, and one who facilitates performing microassessments (measuring projects against a framework). Each pocket guide provides a series of questions designed to help keep the project on track. A scorecard is provided for each pocket guide to facilitate the tracking progress of improvement activities. The questions in the pocket guides are a distillation of the practices found in the **Systems Engineering** Capability Maturity Model (SE-CMM), an industry framework for systems engineering improvement and measurement. See http://www.secat.com; e-mail, secat@secat.com.

[9]"The Software Manager's Toolkit," *IEEE Software*.

during the past 20 years, a champion was identified in virtually every successful project in which they were involved.[10] Figure 4-2 summarizes this role.

Define the Project Vision and Scope

The project vision and scope document describes the background leading to the decision to develop a new or modified system and provides a description of the system that will be extended by the work of the project. (In Canada, the terms ***manifest*** and ***rules of engagement*** are used. *Manifest* is in used in place of the project charter or the project vision document. The *rules of engagement* are a description of the roles and responsibilities for project decision makers, including

Figure 4-2 The Role of the Champion

- Manage the elicitation process and become comfortable when enough requirements are discovered.
- Manage the conflicting inputs from all stakeholders.
- Make the trade-offs necessary to find the set of features that delivers the highest value to the greatest number of stakeholders.
- Own the product vision.
- Advocate for the product.
- Negotiate with management, users, and developers.
- Defend against feature creep.
- Maintain a "healthy tension" between what the customer desires and what the development team can deliver in the release time frame.
- Be the representative of the official channel between the customer and the development team.
- Manage the expectations of customers, executive management, and the internal marketing and engineering departments.
- Communicate the features of the release to all stakeholders.
- Review the software specifications to ensure that they conform to the true vision represented by the features.
- Manage the changing priorities and the addition and deletion of features.

[10]Leffingwell and Widrig, *Managing Software Requirements*, p. 179.

requirement prioritization and escalation procedures.) It is based on the business requirements, and it specifies objectives and priorities. This facilitates a common understanding and communication of the scope of the system that is critical for success. The executive sponsor of the project owns the document.

Figure 4-3 provides a suggested table of contents for an operational concept definition (OCD) document, taken from J-STD-016, the successor standard to DoD-STD-2167A (1988) and MIL-STD-498 (1994). The idea is to create documentation that follows a similar format to facilitate gathering information concerning a planned development effort. Don't feel that you must address every topic in this template. Rather, tailor it for your project environment and needs. Note that the OCD (or whatever you choose to call it) addresses

- The *scope* of the planned effort by providing a system overview
- Documents (references) that provide *background* and related information
- The *current system* or situation; in other words, how the planned need is being met (or not) currently
- The *justification* for the planned development effort. What is it that requires an investment in developing a new system?
- The concept or *vision* for a new or modified system
- *Anticipated impacts* of the new system. How will having a new system affect operations and the organization?
- *Advantages and limitations of the new system* and alternative approaches that were considered

Other excellent references that provide guidance for this work include books by Leffingwell and Widrig[11] and by Wiegers.[12] Figure 4-4 is a template for a vision and scope document provided by Wiegers. This template is simpler than the DoD standard and may be sufficient for your needs.

Identify a Requirements Engineer and Utilize Domain Experts to Perform Requirements Engineering Tasks

My experience is that a project of any size requires an individual assigned as the requirements engineer. Depending on the size of the project, this may be a part-time assignment or may require the full-time effort of several people. *It's valuable*

---

[11]Leffingwell and Widrig, *Managing Software Requirements,* pp. 187–222.

[12]Wiegers, *Software Requirements,* pp. 95–108.

Figure 4-3 Suggested Table of Contents for an OCD

Operational Concept Description (OCD)
Contents

1. **Scope.** This clause should be divided into the following subclauses:
   - 1.1 **Identification.** This subclause shall contain a full identification of the system to which this document applies, including, as applicable, identification number(s), title(s), abbreviations(s), version number(s), and release number(s).
   - 1.2 **System overview.** This subclause shall briefly state the purpose of the system to which this document applies. It shall describe the general nature of the system; summarize the history of system development, operation, and maintenance; identify the project sponsor, acquirer, user, developer, and maintenance organizations; identify current and planned operating sites; and list other relevant documents.
   - 1.3 **Document overview.** This subclause shall summarize the purpose and contents of this document and shall describe any security or privacy protection considerations associated with its use.
2. **Referenced documents.** This clause shall list the number, title, revision, date, and source of all documents referenced in this manual.
3. **Current system or situation.** This clause should be divided into the following subclauses to describe the system or situation as it currently exists.
   - 3.1 **Background, objectives, and scope.** This subclause shall describe the background, mission or objectives, and scope of the current system or situation.
   - 3.2 **Operational policies and constraints.** This subclause shall describe any operational policies and constraints that apply to the current system or situation.
   - 3.3 **Description of current system or situation.** This subclause shall provide a description of the current system or situation, identifying differences associated with different states or modes of operation (for example, regular, maintenance, training, degraded, emergency, alternative-site, wartime, peacetime). The distinction between states and modes is arbitrary. A system may be described in terms of states only, modes only, states within modes, modes within states, or any other scheme that is useful. If the system operates without states or modes, this subclause shall so state,

(*continued*)

Figure 4-3 Suggested Table of Contents for an OCD (*continued*)

without the need to create artificial distinctions. The description shall include, as applicable:

a. The operational environment and its characteristics
b. Major system components and the interconnections among these components
c. Interfaces to external systems or procedures
d. Capabilities/functions of the current system
e. Charts and accompanying descriptions depicting input, output, data flow, and manual and automated processes sufficient to understand the current system or situation from the user's point of view
f. Performance characteristics, such as speed, throughput, volume, frequency
g. Quality attributes, such as reliability, maintainability, availability, flexibility, portability, usability, efficiency
h. Provisions for safety, security, privacy protection, and continuity of operations in emergencies

3.4 **Users or involved personnel.** This subclause shall describe the types of users of the system, or personnel involved in the current situation, including, as applicable, organizational structures, training/skills, responsibilities, activities, and interactions with one another.

3.5 **Support strategy.** This subclause shall provide an overview of the support strategy for the current system, including, as applicable to this document, maintenance organization(s); facilities; equipment; maintenance software; repair/replacement criteria; maintenance levels and cycles; and storage, distribution, and supply methods.

4. **Justification for and nature of changes.** This clause should be divided into the following subclauses:

4.1 **Justification for change.** This subclause shall

a. Describe new or modified aspects of user needs, threats, missions, objectives, environment, interfaces, personnel, or other factors that require a new or modified system

b. Summarize deficiencies or limitations in the current system or situation that make it unable to respond to these factors

4.2 **Description of needed changes.** This subclause shall summarize new or modified capabilities/functions, processes, interfaces, or other changes needed to respond to the factors identified in 4.1.

4.3 **Priorities among the changes.** This subclause shall identify priorities among the needed changes. It shall, for example, identify each change as essential, desirable, or optional, and prioritize the desirable and optional changes.

4.4 **Changes considered but not included.** This subclause shall identify changes considered but not included in 4.2, and rationale for not including them.

4.5 **Assumptions and constraints.** This subclause shall identify any assumptions and constraints applicable to the changes identified in this clause.

5. **Concept for a new or modified system.** This clause should be divided into the following subclauses to describe a new or modified system:

5.1 **Background, objectives, and scope.** This subclause shall describe the background, mission or objectives, and scope of the new or modified system.

5.2 **Operational policies and constraints.** This subclause shall describe any operational policies and constraints that apply to the new or modified system.

5.3 **Description of the new or modified system.** This subclause shall provide a description of the new or modified system, identifying differences associated with different states or modes of operation (for example, regular, maintenance, training, degraded, emergency, alternative-site, wartime, peacetime). The distinction between states and modes is arbitrary. A system may be described in terms of states only, modes only, states within modes, modes within states, or any other scheme that is useful. If the system operates without states or modes, this subclause shall so state, without the need to create artificial distinctions. The description shall include, as applicable:

(*continued*)

Figure 4-3 Suggested Table of Contents for an OCD (*continued*)

a. The operational environment and its characteristics
b. Major system components and the interconnections among these components
c. Interfaces to external systems or procedures
d. Capabilities/functions of the new or modified system
e. Charts and accompanying descriptions depicting input, output, data flow, and manual and automated processes sufficient to understand the new or modified system or situation from the user's point of view
f. Performance characteristics, such as speed, throughput, volume, frequency
g. Quality attributes, such as reliability, maintainability, availability, flexibility, portability, usability, efficiency
h. Provisions for safety, security, privacy protection, and continuity of operations in emergencies

5.4 **Users/affected personnel.** This subclause shall describe the types of users of the new or modified system, including, as applicable, organizational structures, training/skills, responsibilities, and interactions with one another.

5.5 **Support strategy.** This subclause shall provide an overview of the support strategy for the new or modified system, including, as applicable, maintenance organization(s); facilities; equipment; maintenance software; repair/replacement criteria; maintenance levels and cycles; and storage, distribution, and supply methods.

6. **Operational scenarios.** This clause shall describe one or more operational scenarios that illustrate the role of the new or modified system, its interaction with users, its interface to other systems, and all states or modes identified for the system. The scenarios shall include events, actions, stimuli, information, interactions, etc., as applicable. References may be made to other media, such as videos, to provide part or all of this information.
7. **Summary of impacts.** This clause should be divided into the following subclauses:

7.1 **Operational impacts.** This subclause shall describe anticipated operational impacts on the user, acquirer, developer, and maintenance organizations. These impacts may include changes in interfaces with computer operating centers; change in procedures; use of new data sources; changes in quantity, type, and timing of data to be input to the system; changes in data retention requirements; and new modes of operation based on peacetime, alert, wartime, or emergency conditions.

7.2 **Organizational impacts.** This subclause shall describe anticipated organizational impacts on the user, acquirer, developer, and maintenance organizations. These impacts may include modification of responsibilities; addition or elimination of responsibilities or positions; need for training or retraining; and changes in number, skill levels, position identifiers, or location of personnel in various modes of operation.

7.3 **Impacts during development.** This subclause shall describe anticipated impacts on the user, acquirer, developer, and maintenance organizations during the development effort. These impacts may include meetings/discussions regarding the new system; development or modification of databases; training; parallel operation of the new and existing systems; impacts during testing of the new system; and other activities needed to aid or monitor development.

8. **Analysis of the proposed system.** This clause should be divided into the following subclauses:

8.1 **Summary of advantages.** This subclause shall provide a qualitative and quantitative summary of the advantages to be obtained from the new or modified system. This summary shall include new capabilities, enhanced capabilities, and improved performance, as applicable, and their relationship to deficiencies identified in 4.1.

8.2 **Summary of disadvantages/limitations.** This subclause shall provide a qualitative and quantitative summary of disadvantages or limitations of the new or modified system. These disadvantages and limitations shall include, as applicable, degraded or missing

(*continued*)

Figure 4-3 Suggested Table of Contents for an OCD (*continued*)

capabilities, degraded or less-than-desired performance, greater-than-desired use of computer hardware resources, undesirable operational impacts, conflicts with user assumptions, and other constraints.

8.3 **Alternatives and trade-offs considered.** This subclause shall identify and describe major alternatives considered to the system or its characteristics, the trade-offs among them, and rationale for the decisions reached.

9. **Notes.** This clause shall contain any general information that aids in understanding this document (e.g., background information, glossary, rationale). This clause shall include an alpabetical listing of all acronyms, abbreviations, and their meanings as used in this document and a list of any terms and definitions needed to understand this document.

A. **Annexes.** Annexes may be used to provide information published separately for convenience in document maintenance (e.g., charts, classified data). As applicable, each annex shall be referenced in the main body of the document where the data would normally have been provided. Annexes may be bound as separate documents for ease in handling. Annexes shall be lettered alphabetically (A, B, etc.).

for those assigned in this role to have had extensive experience and expertise in the functional area being addressed by the planned system (domain experts or ***subject matter experts*** *[SMEs])*. The reason for utilizing domain experts as requirements engineers is that the requirements need to be understood in the customer's context. This is an extremely important issue. Unfortunately, many projects cripple their requirements efforts by not providing domain experts. This is a false economy. It may be that a project can have the domain expert assume the role of the project champion.

SMEs can be found by recruiting experienced developers from other projects within your organization. Another source is professional staff departing customer organizations for reasons of retirement or a desire for a new opportunity. SMEs function as a critical part of the team by understanding and explaining the

Figure 4-4 Template for a Vision and Scope Document

1. Business Requirements
   - 1.1 Background
   - 1.2 Business Opportunity
   - 1.3 Business Objectives
   - 1.4 Customer or Market Requirements
   - 1.5 Value Provided to Customers
   - 1.6 Business Risks
2. Vision of the Solution
   - 2.1 Vision Statement
   - 2.2 Major Features
   - 2.3 Assumptions and Dependencies
3. Scope and Limitations
   - 3.1 Scope of Initial Release
   - 3.2 Scope of Subsequent Releases
   - 3.3 Limitations and Exclusions
4. Business Context
   - 4.1 Customer Profiles
   - 4.2 Project Priorities
5. Product Success Factors

context of the requirements for the planned system.[13] SMEs can determine, based on their experience, whether the requirements are reasonable, how they extend the existing system, how the proposed architecture should be designed, and the impacts on users, among other areas. This approach enables the requirements engineering tasks to be performed more effectively.

A pitfall for which to watch is an SME whose approach is inflexible. An SME who can assist most effectively is one who is open to new ideas, approaches, and technologies.

[13]Sabourin notes that it is very difficult in some domains to locate knowledge experts who are able to express requirements clearly. In these situations, a role of the requirements engineer is to map domain expert input to clear requirements. (Comment included in Sabourin's review of this manuscript.)

Train Developers Not to Make Requirements Decisions and Not to Gold Plate

On a small project, the requirements engineer may also be a programmer (developer). On larger projects, we typically have individuals assigned in the developer role. Developers often find themselves in the situation of being required to design and code capabilities for systems when the requirements are not well defined (look ahead to Figure 4-9 for the criteria for a good requirement). Faced with this decision, the easier action is to make some assumptions and keep working, particularly in the face of tight deadlines and unpaid overtime. A better choice would be to interrupt work and get the requirement clarified. Developers need to be trained that this choice is best (and expected). Such "training" needs to be conveyed with good judgment so that technical performers do not feel that they are being overly constrained. Developers who are accustomed to an undisciplined environment may take exception to having to conform to rules. A related problem is a developer who adds features and capabilities that are not required by the specification (gold plating). This may be done because the developer sincerely believes it is appropriate and "best" for all concerned. However, gold plating adds to costs and extends the schedule and may complicate other areas of the system. If a user noticed this feature or capability in one area of the system, he might decide that it should be provided throughout the system! This contributes to requirements creep and results in added costs.

Utilize a Variety of Techniques to Elicit Customer and User Requirements and Expectations

There is extensive information available in the system and software engineering literature concerning requirements elicitation—that is, the effort undertaken by systems and software requirements engineers to understand customer needs and expectations.[14]

Leffingwell and Widrig[15] provide an insightful discussion of useful techniques and tools to elicit user requirements and expectations in their book. These techniques and tools include interviewing, questionnaires, requirements

---

[14]See Sommerville and Sawyer, *Requirements Engineering: A Good Practice Guide*. Another source is Gause and Weinberg's *Exploring Requirements: Quality Before Design,* which provides a thorough discussion of the issues related to elicitation of user needs from customers and users.

[15]See Leffingwell and Widrig, *Managing Software Requirements,* Chapters 7 through 15, which provide guidelines for understanding user needs.

workshops, brainstorming and idea reduction, storyboards, **use cases**, role playing, and **prototyping**.[16]

Requirements checklists provide a way to evaluate the content, completeness, and quality of the requirements prior to development. McConnell[17] provides a good checklist in *Code Complete,* and Wiegers[18] provides another for inspection of software requirements specifications at his Web site. If the requirements are explicit, the users can review them and agree to them. If they're not, the developers will end up making requirements decisions during coding, a sure-fire recipe for problems, as discussed earlier. Weinberg, in *The Secrets of Consulting,* provides helpful advice concerning giving and getting advice successfully.

Use Cases

One requirements technique is the use case. Schneider and Winters[19] provide a practical approach. See Figure 4-5 for an example of a use case diagram utilizing the **Unified Modeling Language** (UML) notation. UML is a graphical language for visualizing, specifying, constructing, and documenting the artifacts of a software-intensive system that was adopted by the Object Management Group in late 1997. UML has become a vendor-independent standard for expressing the design of software systems and is being rapidly adopted throughout industry. UML incorporates use cases as the standard means of capturing and representing requirements.

Many developers believe that use cases and scenarios facilitate team communication. They provide a context for the requirements by expressing sequences of events and a common language for end users and the technical team. They identify system interfaces, enable modeling the system graphically and textually, and are reusable in test and user documentation. Rumbaugh[20] also provides a helpful approach in "Getting Started: Using Use Cases to Capture Requirements."

---

[16]See Connell and Shafer, *Structured Rapid Prototyping,* for a discussion of the benefits of rapid prototyping, tools, and techniques that can be used, and other practical aspects of building prototypes and evolving them into production systems. See also Kaplan et al., *Secrets of Software Quality,* pp. 265–269.

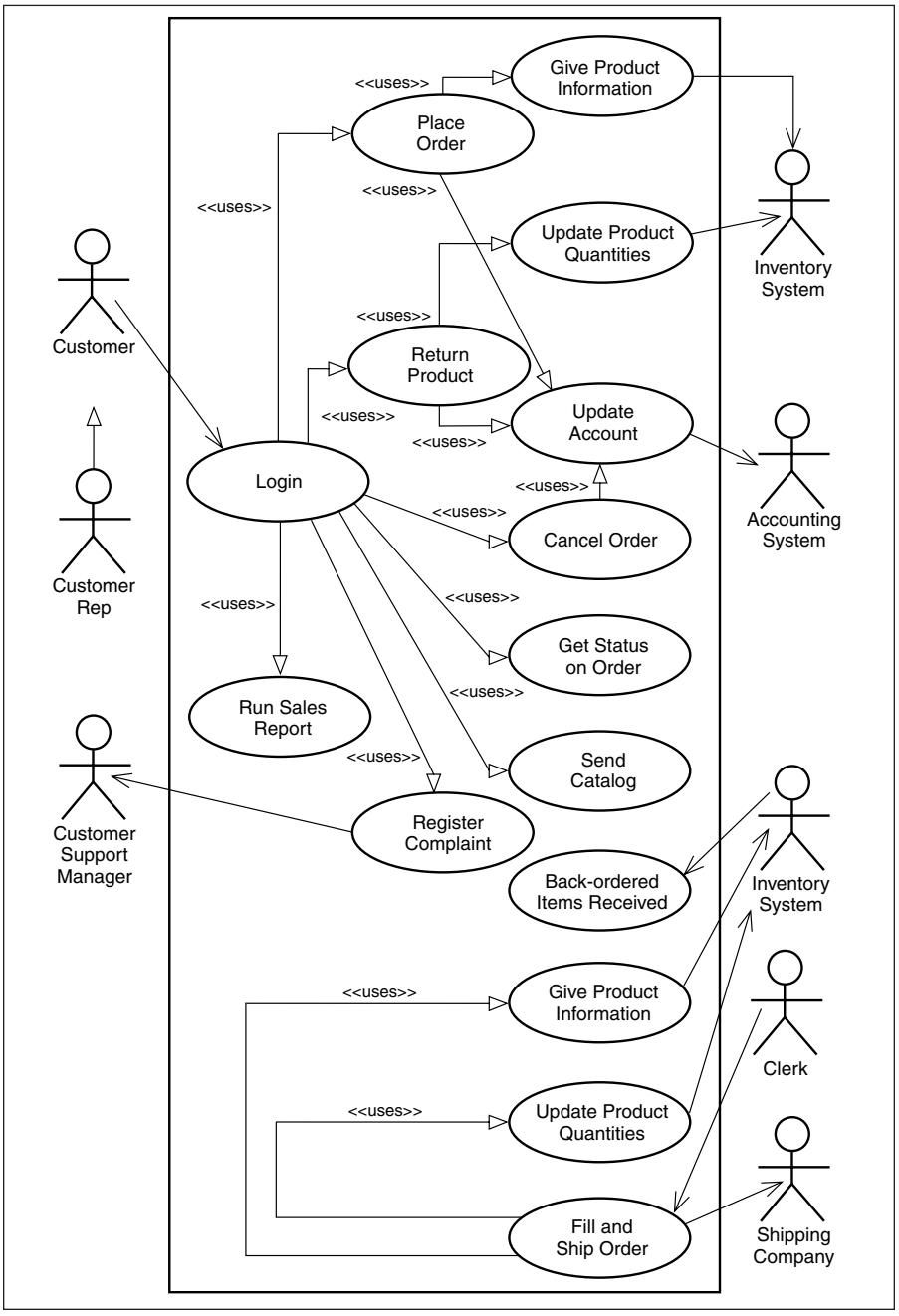
[17]McConnell, *Code Complete,* pp. 32–34.

[18]Available at http://www.processimpact.com/goodies.shtml.

[19]Geri Schneider and Jason P. Winters, *Applying Use Cases: A Practical Guide*.

[20]James Rumbaugh, "Getting Started: Using Use Cases to Capture Requirements." *Journal of Object-Oriented Programming*.

Figure 4-5 Example of a Use Case Diagram



Leffingwell and Widrig[21] provide checklists concerning use cases. Eman Nasr[22] has provided an easy-to-understand basic introduction in his *Use Case Technique for Requirements Engineering*. Wiegers's[23] view is that use cases alone often don't provide enough detail for developers to know just what to build.

Consideration should be given to using use cases to describe the outwardly visible requirements of a system.[24] Use cases allow analysts to identify the required features of a system. They describe the things users of a system want the system to do (sometimes referred to as *scenarios*). Use cases are especially helpful for processes that are iterative and risk driven (which helps identify and address risks early in the program). The high-level use cases should be developed to help determine the scope of the project. What should be included? What can we realistically accomplish given our schedule and budget?[25] The developed use cases can also be utilized as test cases.

As with any method, there are both advantages and disadvantages of using use cases as a method. Among the advantages is that because of the thread of behavior characteristics and the fact that UML includes certain specialized modeling elements and notations (for example, "use case realization"), use cases provide additional value to their role of linking the requirements activities to design and implementation. Among the disadvantages is that use cases are not good

---

[21]*Managing Software Requirements,* pp. 289–292.

[22]Nasr is associated with the Computer Science Department, University of York, in the United Kingdom. E-mail: Eman.Nasr@cs.york.ac.uk.

[23]Karl Wiegers, "10 Requirements Traps to Avoid." *Software Testing and Quality Engineering Magazine*. In addition to describing ten important requirements traps, Wiegers provides keys to excellent requirements, including a collaborative customer-developer partnership for requirements development and management, and prioritizing requirements.

[24]Another good reference is by Daryl Kulak and Eamonn Guiney, *Use Cases: Requirements in Context,* which explains and provides examples of the nine diagrams of the UML (use case diagram, sequence diagram, collaboration diagram, statechart diagram, activity diagram, class diagram, object diagram, component diagram, and deployment diagram). They also provide a comprehensive and thoughtful list of problems related to using use cases (pp. 154–165). See also Korson, *The Misuse of Use Cases*. Korson notes that projects can expend a lot of time and effort on use cases without much benefit when they are not used correctly. Root causes of the misuse of use cases are (1) a requirements process that is neither understood nor properly managed, (2) poor-quality requirements, and (3) poor-quality designs. Analysts sometimes neglect fundamental principles of requirements gathering in the name of use cases.

[25]A reference point based on industry data is that systems and software projects are overpromised by an average of 100% to 200% (The Standish Group, 8,000 projects, 1996).

containers for nonfunctional requirements (such as the-ilities and attributes of the system environment) and design constraints. Dean Leffingwell's book, *Managing Software Requirements,* recommends alternative approaches based on the experience of the project team. In the situation in which the team's experience with the requirements process is limited and the object-oriented (OO) paradigm has not been adopted and used, a conventional software requirements specification approach is recommended.[26] If the team's experience with the requirements process is limited but the team is in the process of adopting the OO paradigm, the recommendation is to work with the use case method but to master it fully before depending on it to represent the requirements.

As noted earlier, the developed use cases can also be utilized as test cases. Bob Poston, Director of Quality Assurance Technology at Aonix, Inc., advocates front-end testing or specification testing to achieve **defect prevention** in a requirements process.[27] Poston asserts that project time and resources allocated to testing (typically 30% or more[28]) can be dramatically reduced, and he recommends adding formality to the requirements phase using a requirements modeling tool and use case notation and scenarios. Provide system-level use cases and then object-level use cases for the design. Add sufficient information to the use case to make it test ready. Poston notes (based on data from Capers Jones) that typically 16% of the test cases are redundant and 10% are irrelevant; therefore, in a typical project, 26% of the test effort is wasted. We need to develop requirements specifications that have in them the data that allow primary specification based test design. Poston cited two examples in his presentation, one in which the defect count dropped 94%[29] and another in which productivity increased 100 fold from 100 test cases in 20 days to 1,000 test cases in 2 days.[30]

---

[26]See the Institute of Electrical and Electronics Engineers' (IEEE) Standard 830, *IEEE Recommended Practice for Requirements Specifications*.

[27]See his presentation from the National SEPG99 Conference, *Generating Test Cases from Use Cases Automatically,* March 1999.

[28]Capers Jones, *Software Quality: Analysis and Guidelines for Success,* p. xxiv. Watts Humphrey's experience is that testing typically removes only 50% of the errors present. You must have quality code going into testing to have quality code coming out (personal e-mail communication with Humphrey, April 17, 2000).

[29]Robert M. Poston. "Counting Down to Zero Software Failures," p. 230.

[30]Richard Adhikari, "Development Process Is a Mixed-Bag Effort."

In summary, my experience is that use of a common set of techniques and tools among all parties involved in a particular project is a much bigger help than one would imagine, because this enables the entire development team to share the same concepts and language. This is more easily recommended than accomplished, however. Each system and software engineer has her/his own experience and familiarity with a set of tools. It's human nature to like to use that with which each of us is most familiar. Getting consensus on the use of a specific set of methods and tools is difficult, and providing the training and the opportunity to use them and become very familiar with their capabilities is expensive and time-consuming. (Also, recall the comments I provided at the end of Chapter 1 concerning systems and software engineers and the recommended context for readers of this book.)

Train Requirements Engineers to Write Good Requirements

There is strong evidence of the value of utilizing trained requirements engineers. Trained requirements engineers correlate with

- Well-written, unambiguous requirements statements
- The ability to utilize an effective automated requirements tool
- More effective use of project resources because of reduced rework

The Impact of Requirements Errors

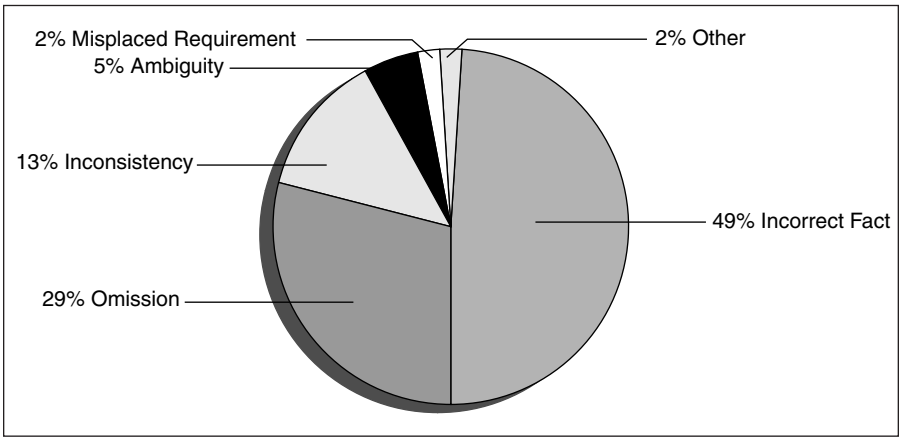
Industry research shows that requirements errors are both the most common and also the most expensive defects in the technical work. Figure 4-6 quantifies the typical types of requirements errors.

Hooks and Farry[31] report that more than 80% of all product defects are inserted in the requirements definition stage of product development. This means that we can save money! If we provide good requirements, we can eliminate 80% of the rework problems. Rework costs are estimated at 45% of total project costs.[32] *Thus, by taking 80% of 45%, we learn that 36% (more than one third) of total project costs (based on industry data) potentially can be avoided by driving requirements errors out of the work products*. I'll acknowledge that it would

[31]*Customer-Centered Products,* p. 3.

[32]Leffingwell, "Calculating Your Return Investment from More Effective Requirements Management," p. 3. Available at http://www.rational.com/products/whitepapers/300.jsp. Rational Corporation. Available at http://www.rational.com/index.jtmpl.

Figure 4-6 Types of Nonclerical Requirements Errors



be difficult to achieve this amount of savings. However, *clearly a significant portion of this waste should be redirected by any and every development effort* through use of the practices recommended in this book and other process improvements. *From the perspective of the PM, the savings achieved by employing effective requirements practices should be redirected to pay for the needed effort and any associated training, methods, techniques, and tools required.*

The Importance of Requirements to Program Costs

Managers would be well advised to take careful note of the relative cost to fix an error. Barry Boehm[33] analyzed 63 software development projects in corporations such as IBM, GTE, and TRW and determined the ranges in cost for the error types described earlier that were created by false assumptions in the requirements phase but not detected until later phases (Figure 4-7).

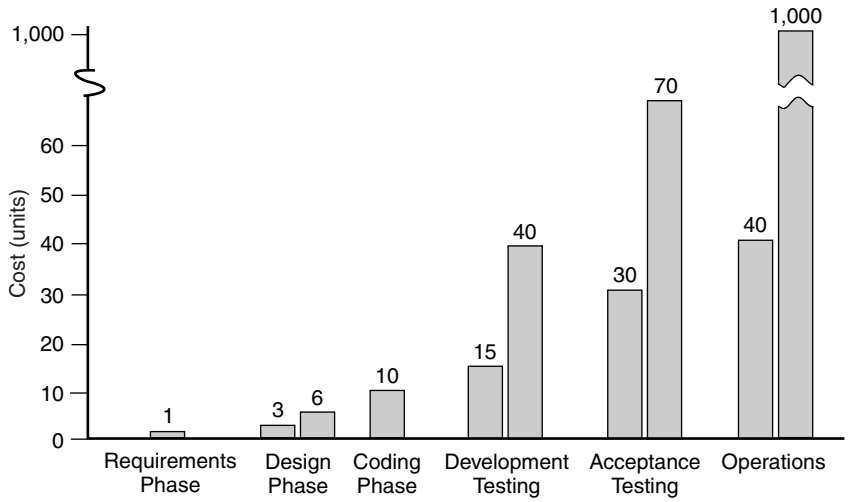
Figure 4-8 shows the value of investing in an effective requirements process in which the real requirements are identified and in which requirements errors are driven out of the requirements work products during the earliest possible phase of system development. The cost to repair a requirements defect costs

[33]See Barry W. Boehm, *Software Engineering Economics.* These figures actually may be conservative because Boehm studied only those projects that were completed. See Gause and Weinberg, *Exploring Requirements: Quality Before Design,* for a discussion of the cost of ambiguity and how to remove it (pp. 17–21).

Figure 4-7 Relative Cost to Fix an Error

Phase in Which the Error Is Found	Cost Ratio
Requirements	1
Design	3–6
Coding	10
Development Testing	15–40
Acceptance Testing	30–70
Operation	40–1,000

Figure 4-8 Relative Cost to Fix Requirements Defects When Discovered in Later Stages



more the later in the project life cycle the error is discovered. For example, it costs 15 to 40 times as much to correct a requirements error during development testing than if we resolve the error earlier. This is a very strong argument for investing more in the requirements process.

What Is a Good Requirement?

There are several good articles and white papers on what is considered a good requirement.[34] Figure 4-9 presents a summary checklist of the criteria for a good requirement, providing criteria and a description of each.

Figure 4-9 Criteria of a Good Requirement

Criterion	Description
Necessary	Can the system meet prioritized, real needs without it? If yes, the requirement isn't necessary.
Verifiable	Can one ensure that the requirement is met in the system? If not, the requirement should be removed or revised. Note: The verification method and level at which the requirement can be verified should be determined explicitly as part of the development for each of the requirements. (The verification level is the location in the system where the requirement is met (for example, the "system level," the "segment level," and the "subsystem level).[35]
Attainable	Can the requirement be met in the system under development?
Unambiguous	Can the requirement be interpreted in more than one way? If yes, the requirement should be clarified or removed. Ambiguous or poorly worded writing can lead to serious misunderstandings and

[34]See the Compliance Automation Web site at http://www.complianceautomation.com/ to access excellent papers concerning requirements. Several are required reading for anyone seriously involved with requirements: *Guide for Managing and Writing Requirements,* which is a thorough treatment; *Writing Good Requirements,* which provides helpful hints to avoid many of the most common requirements writing problems; *Characteristics of Good Requirements,* which describes major characteristics of well-defined requirements, and *Managing Requirements,* which provides important insights into the requirements process. The greeting at this Web site reflects the wisdom of extensive experience: "People who write bad requirements should not be surprised when they get bad products, but they always are!"

[35]See Grady, *System Validation and Verification,* pp. 101–102, for a discussion of verification levels.

Criterion	Description
	needless rework. Note: Specifications should include a list of acronyms and a glossary of terms to improve clarity.
Complete	Are all conditions under which the requirement applies stated? Also, does the specification document all known requirements? (Requirements are typically classified as functional, performance, interface, constraints, and environment.)
Consistent	Can the requirement be met without conflicting with all other requirements? If not, the requirement should be revised or removed.
Traceable	Is the origin (source) of the requirement known, and can the requirement be referenced (located) throughout the system? The automated requirements tool should enable finding the location in the system where each requirement is met.
Allocated	Can the requirement be allocated to an element of the system design where it can be implemented? If not, the requirement needs to be revised or eliminated.[36]
Concise	Is the requirement stated simply and clearly?
Implementation free	The requirement should state what must be done without indicating how. The treatment of interface requirements is generally an exception.
Standard constructs	Requirements are stated as imperative needs using "shall." Statements indicating "goals" or using the word "will" are not imperatives.
Unique identifier	Each requirement should have a unique identifying number that assists in identification, maintaining change history, and providing traceability.

[36]The alternative is to risk a major costly change in the system or software architecture.

A "good" requirement is not necessarily a "real" requirement. The requirement may meet our criteria for a good requirement, but the requirement may not meet a real need of the users of the planned system. We discover the real requirements by following the recommendations provided in this chapter.

Oliver and colleagues[37] provide a good requirements taxonomy and believe that the engineering effort and costs associated with assessing requirements can be reduced substantially with modeling.

Document the Rationale for Each Requirement

Industry sources indicate that by taking the effort to document why each requirement is needed, as many as half of the "requirements" can be eliminated. The documentation step reduces the life cycle cost of system development significantly by obviating the need for follow-on work for unnecessary requirements. The rationale describes some or all of the following related information:[38]

- Assumptions
- Why it is needed
- How it is related to expected operations
- Design decisions

An example of documenting the rationale for a requirement is the following: Requirement 101 is needed in the system to enable the users of the system to receive feedback that their request was transmitted. In documenting the rationale for requirements, the requirements engineer may

- Gather data to enable a projection of how the activity involved may vary depending on different circumstances and uses of the system
- Perform a **trade study** to determine alternative ways to address the requirement
- Consider alternatives and provide the basis for the selected alternative

The easiest way to capture rationale is as each requirement is written. No requirement should be put into the specification until its rationale is well understood.

---

[37]Oliver et al., *Engineering Complex Systems with Models and Objects,* pp. 104–115. VITECH's automated tool, CORE, has behavioral modeling capabilities. See http://www.vtcorp.com.

[38]Ivy Hooks, *Guide for Managing and Writing Requirements,* p. 5–4. See pp. 5–4 through 5–6 for a more extensive discussion of why the documentation step is critical and how to do it.

Utilize Methods and Automated Tools to Analyze, Prioritize, and Track Requirements

As suggested previously, the broader term *requirements process* involves many aspects of the project throughout its entire life cycle, not just "requirements management." However, the automated tools available today are often described as requirements management tools. See Figure 4-10 for a list of several of the available tools and their related Web sites. Note that the International Council on Systems Engineering's Tools Working Group provides information concerning a large set of tools at its Web site, http://www.incose.org/tools/tooltax.html. Many projects have been supported by office automation tools such as Microsoft Word or Microsoft Excel and database applications such as Informix to manage requirements, but these tools are relatively limited in their capabilities (although they can provide some of the capabilities needed for a particular project). Many organizations have developed their own requirements tools (some have developed several), but this approach is not cost-effective, given the tools available on the market today.

A sophisticated requirements tool is able to do much more than requirements management. It should be able to facilitate requirements elicitation, help with prioritization of requirements, provide traceability[39] of requirements throughout the development effort (to design, implementation, and test verification, for example) and allow for assignment of requirements to subsequent releases of system products. It should allow assignment of an unlimited number of attributes (characteristics of requirements) to any and all requirements. See Figure 4-11 for a sample requirements matrix that shows attributes. Attributes allow users to associate data with objects, table markers, table cells, modules, and projects. For example, there are two kinds of attributes in DOORS, user-defined attributes and system-defined attributes. User-defined attributes may be built from specific attribute types such as text, integer, date, and so forth and are instantiated by users for their own needs. System-defined attributes, however, are predefined by DOORS and automatically record essential and highly useful information in the background. Attributes allow you to associate information with individual or related groups of requirements and often facilitate analysis of requirements data

[39]Traceability gives essential assistance in understanding the relationships that exist within and across software requirements, design, and implementation, and it is critical to the development process. See James D. Palmer, "Traceability." See also the definition and guidelines for requirements traceability in Figure 9-5.

Figure 4-10 Commercial Requirements Tools, Vendors, and Web Sites

Tool	Vendor	Web Site
Caliber RM	Technology Builders, Inc., Atlanta, Georgia	http://www.tbi.com
C.A.R.E. 2.0	SOPHIST Group, Nuremberg, Germany	http://www.sophist.de
CORE	VITECH Corporation, Vienna, Virginia	http://www.vtcorp.com
DOORS	Telelogic, Malmo, Sweden	http://www.telelogic.com/doors
RDD ISEE	Holagent Corporation, Gilroy, California	http://www.holagent.com
Requisite Pro (ReqPro)	Rational Software Corporation, Lexington, Massachusetts	http://www.rational.com
RTM Workshop	Integrated Chipware, Inc., Reston, Virginia	http://www.chipware.com
SLATE	TD Technologies, Richardson, Texas	http://www.tdtech.com
SynergyRM	CMD Corporation, Dallas, Texas	http://www.cmdcorp.com
Vital Link	Compliance Automation, Inc., Boerne, Texas	http://www.complianceautomation.com
Xtie-RT Requirements Tracer	Teledyne Brown Engineering, Los Angeles, California	http://www.tbe.com

via filtering and sorting based on attribute values. System-defined attributes may also be used for filtering and sorting. Although they are, for the most part, read-only and are not user modifiable, they perform essential and automatic information gathering.

Figure 4-11 Sample Requirements Attribute Matrix

NFAK0 Tag	Requirement text	Priority	Status	Cost	Difficulty	Stability	Assigned to	Unique ID	Location	Author	Revision	Date	Reason	Traced-from	Traced-to	RootTag#
NFAK0208	The time required for the equipment to warm up prior to operation shall not exceed one (1) minute from a cold start at –20 degrees C.	Medium	Approved		Medium	Medium		510	NFA Annex K section C	Chardon	1.0001	9/16/99 18:25	Marked trace to NFASSS289.		NFASSS289	208

Approaches, Tools, and Methods for Prioritizing Requirements

It's important to be able to prioritize the system and software requirements. An excellent discussion of this topic is provided by Karl Wiegers.[40] He suggests two scales, each with three-levels: (1) high/medium/low and (2) essential/conditional/optional. One can visualize how utilizing these scales at an appropriate level of abstraction (for example, the use case level, the feature level, or the functional requirement level) will facilitate dealing with the common problem of having a limited development budget for release 1.0! Wiegers discusses his semiquantitative analytical approach and provides an example for a sample project: "Any actions we take to move requirements prioritization from the political arena into an objective and analytical one will improve the project's ability to deliver the most important functionality in the most appropriate order" (p. 30). This is recommended reading for managers and requirements practitioners. Wiegers provides a set of useful tools at his Web site, including a Microsoft Excel requirements prioritization spreadsheet.[41]

Another method for prioritizing requirements was developed by Karlsson and Ryan.[42] Their concern was that there are usually more requirements than can

[40]Karl Wiegers, "First Things First: Prioritizing Requirements," pp. 24–30.

[41]Available at http://www.processimpact.com/goodies.shtml.

[42]Joachim Karlsson and Kevin Ryan, "A Cost-Value Approach for Prioritizing Requirements," pp. 67–74.

be implemented given stakeholders' time and resource constraints (sound familiar?). They sought a way to select a subset of the customers' requirements and still produce a system that met their needs. The process they developed is described well in the referenced article. It has been applied successfully to two commercial projects, and these are also described. The Analytic Hierarchy Process (AHP) is used to compare requirements pairwise according to their relative value and cost. The approach is considered simple, fast, and accurate and yields accurate results and holds stakeholder satisfaction as both the ultimate goal and the guiding theme. Stakeholder satisfaction addresses maximum quality, minimum costs, and short time-to-market. Karlsson and Ryan believe that this cost-value approach is a useful first step in addressing a criticism of software engineering for lacking the trade-off analysis that is a component of multidisciplinary systems engineering. They feel that this approach is similar to that of the Quality Attribute Requirements and Conflict Consultant tool within Barry Boehm's WinWin system.[43]

Boehm has continued to evolve the WinWin Spiral Model to develop system and software requirements and architectural solutions based on winning conditions negotiated among a project's stakeholders.[44] The WinWin negotiation tool is a UNIX workstation-based groupware support system that allows stakeholders to enter winning conditions, explore their interactions, and negotiate mutual agreements on the specifics of the project. The model and support system feature a central role for quantitative trade-off analysis tools such as COCOMO. This method is obviously more complex than the other two, but the research is a promising effort. Many publications are available at the Web site concerning the win-win approach.

These methods for prioritizing requirements offer a significant opportunity to strengthen and improve your requirements process further. See the discussion of the rationale for prioritizing requirements in Chapter 8. All requirements are not equal—some are more important to customers and users than others. It is the job of the system developers (the requirements engineers, specifically) in concert with the customer to figure out how to prioritize the requirements and how to size the development effort to meet the project budget and schedule. The good news is that proven methods are available to help. The challenge is to use them.

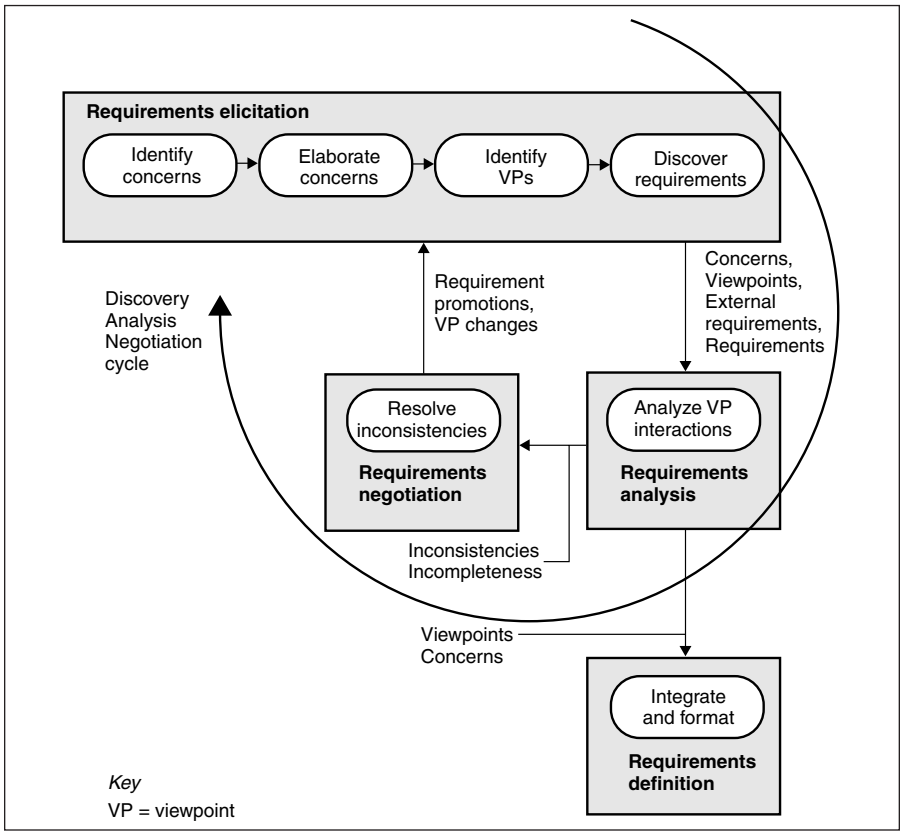
[43]See Boehm and H. In, "Identifying Quality-Requirements Conflicts," pp. 25–35.

[44]Available at http://sunset.usc.edu/research/WINWIN/index.html.

Collect Requirements from Multiple Viewpoints

From our experience, we know that information about the requirements for the planned system needs to be elicited from a variety of stakeholder perspectives. Sommerville and Sawyer[45] have provided a good discussion of this topic in their book *Requirements Engineering: A Good Practice Guide*. In Chapter 13 they describe the basic principle underlying various viewpoints. They recommend a systematic approach called PREview (which stands for process and requirements engineering viewpoints), developed from experience with large systems engineering projects. Figure 4-12 provides an overview of how PREview checklists

Figure 4-12 The PREview Process



[45]See pp. 90–93 and 359–388. See also the Web site for this book, http://www.comp.lancs.ac.uk/computing/resources/re-gpg/.

and tables are used when iterating requirements elicitation/discovery, requirements analysis, and requirements negotiation.

Viewpoint-oriented analysis is obviously more expensive than an unstructured, informal approach to requirements elicitation. However, it may prove to be a good investment. As with any process improvement, an organization may want to "pilot" it, using a relatively small project. As noted in a recent article by Sommerville and colleagues,[46] they believe PREview helps improve the quality of requirements specification by providing a framework for analysis based on the key business concerns that define the success or failure of a project. PREview does not define how priorities, inconsistencies, and redundancies are resolved. This is the task of the joint team.

Consider the Use of Formal Methods When Appropriate

A formal method in software development is a method that provides a formal language for describing a software artifact such as a specification, design, or source code. Formal proofs are possible, in principle, about properties of the artifact so expressed. Vienneau[47] recommends using formal methods to help adequately capture requirements and cautions that many software engineers have adopted new methodologies without understanding the root concepts. He believes formal methods promise to yield benefits in quality and productivity. He notes that formal methods are typically used in organizations at SW-CMM level 3 and above and asserts that an organization that can figure out how to integrate formal methods effectively into their current process will be able to gain a competitive advantage.

Pitfalls

We've captured a lot of experience and lessons learned in this chapter. Here are some stumbling blocks you may run into and suggestions for how to deal with them:

1. It's very difficult to find one person who has the qualities of a domain or an SME *and* is a trained requirements engineer. Often it's easier to place someone with *one* of these credentials in the role of the requirements engineer.

---

[46]Sommerville et al., "Viewpoints for Requirements Elicitation: A Practical Approach."

[47]Vienneau, "A Review of Formal Methods." The discipline of a formal specification can result in fewer specification errors. Using specifications written in a formal language to complement natural language descriptions can make the contract between a user and a developer more precise.

My experience is that it's worth the extra effort and cost to find that one person who has both qualities. The reason is that the domain expert who is also a trained requirements engineer will provide the project invaluable advice concerning the real requirements. If you can't find one person with both skills, my suggestion is to train someone who is a domain expert in requirements engineering. This training will help a domain expert balance her preconceived ideas concerning the solutions with the concept of eliciting the real requirements and being sensitive to implementation issues.

2. Customers will try to put much of the burden for defining requirements on developers. "You tell me; you're the expert," they'll say! Not really. Developers may be trained and proficient in developing systems, but they are not the ones who should decide on real customer needs and expectations. As I've emphasized, *it requires a joint effort to define the real requirements*. As noted earlier, *it is important to train requirements engineers and developers not to make assumptions, not to make requirements decisions, and not to gold plate*. You'll find that this investment in training and discipline is valuable.
3. Project start-up situations are often hectic. It's difficult to pay attention to all of the tasks that need to be addressed. This is certainly true with respect to initiating and installing effective requirements practices. Consider utilizing an internal or external expert to assist with needed activities.
4. Don't use "smallness" as an excuse for not taking advantage of the practices, recommendations, and suggestions provided in this book. These are proven practices—on small projects and on larger ones. Tailor your approach based on common sense. Make good use of the underlying ideas and concepts.

Summary

This chapter highlights that the customer's stated needs require careful scrutiny to determine the real needs. Several specific recommendations and suggestions are advocated to help you determine the real customer needs and requirements. You will find that you can save effort and money, as well as do a better job (improve customer satisfaction), by addressing these recommendations. Please don't ignore them because they are presented concisely. I've emphasized that a huge amount of waste (almost half the costs on a typical project) is caused by using the normal approach—relying on the customer's *stated* requirements. Informed PMs and requirements engineers can redirect resources that are typically wasted to the implementation of these recommendations and suggestions. Enlist the

support of your PM, and utilize a requirements specialist to implement these proven ideas. An organization should undertake implementation of these recommendations gradually and seek to evolve an approach that is continuously improved, based on your own experience and what works in your environment.

Key References and Suggested Readings

Barry Boehm. WinWin Spiral Model & Groupware Support System. 1998 Available at http://sunset.usc.edu/research/WINWIN/index.html. Dr. Boehm is Director of the University of Southern California Center for Software Engineering. The Center is under contract to the Defense Advanced Research Projects Agency via the Air Force Research Laboratory (formerly known as Rome Laboratories). It plans to develop (in collaboration with The Aerospace Corporation) a robust version of the WinWin System and to apply it to the domain of satellite ground stations.

Barry Boehm, Alexander Egyed, Julie Kwan, Dan Port, Archita Shah, and Ray Madachy. "Using the WinWin Spiral Model: A Case Study." *IEEE Computer* 1998:31 33–44. This University of Southern California Center for Software Engineering research project has three primary elements: (1) Theory W, a management theory and approach that says making winners of the system's key stakeholders is a necessary and sufficient condition for project success; (2) the WinWin Spiral Model, which extends the spiral software development model by adding Theory W activities to the front of each cycle; and (3) WinWin, a groupware tool that makes it easier for distributed stakeholders to negotiate mutually satisfactory system specifications. The authors found in this work that the most important outcome of product definition is not a rigorous specification but a team of stakeholders with enough trust and shared vision to adapt effectively to unexpected changes. The researchers believe that the approach will transition well to industry use.

Daniel P. Freedman and Gerald M. Weinberg. *Handbook of Walkthroughs, Inspections, and Technical Reviews.* 3rd ed. Chicago: Scott, Foresman and Co., 1990. This book provides a variety of examples of peer reviews and is a good source for organizations that want to consider alternative approaches.

Donald C. Gause and Gerald M. Weinberg. *Are Your Lights On? How to Know What the Problem REALLY Is.* 2nd ed. New York: Dorset House Publishing, 1989. As the title suggests, this book is interesting and light reading but offers

valuable insights concerning real needs. The authors' perspective is that customers need assistance in defining their real requirements. A good requirements process will (1) identify the real problem, (2) determine the problem's "owner," (3) identify its root cause, and (4) determine whether to solve it. This is recommended reading for requirements engineers and their customers.

Tom Gilb and Dorothy Graham. *Software Inspection.* Reading, MA: Addison-Wesley, 1993. This book is about inspections of any work product, not just software. The authors' approach is very rigorous and therefore requires more training and is more expensive than normal peer reviews. However, it results in more defects being removed earlier, thus saving costs later in the development cycle. This book is invaluable for an organization that is committed to using inspections of work products—a proven method with good payback. Note that Rob Sabourin offers an economical inspections training and implementation approach. Contact him at rsabourin@amibug.com.

Rita Hadden. "How Scalable Are CMM Key Practices?" *CROSSTALK* 1998: vol. 11(4) 18–23. Hadden provides process improvement consulting services for organizations of all sizes. She notes that many practitioners are convinced that models such as the SW-CMM are not practical for small organizations because the cost of applying the recommended practices outweighs benefits. Her experience with more than 50 small projects does not support this view. The article describes using a disciplined, repeatable approach for projects of short duration. She concludes that CMM key practices are scalable.

Ivy Hooks. *Guide for Managing and Writing Requirements.* 1994. Available at ivyh@complianceautomation.com. This is a concise, well-written guidebook based on extensive experience by a practicing requirements engineer and consultant. It addresses scoping a project, managing requirements, how systems are organized, and levels of requirements, writing good requirements, requirements attributes, and specifications.

Ivy Hooks. *Managing Requirements.* 1994. This white paper is available at the Compliance Automation Web site http://www.complianceautomation.com/. It provides a good analysis of how failure to invest in the requirements process affects projects, and it describes major problems based on Hooks's experience. Also, it describes some of the characteristics of good requirements.

Ivy Hooks. *Writing Good Requirements: A One-Day Tutorial.* McLean, VA, 1997 Compliance Automation, Inc. Sponsored by the Washington metropolitan

area chapter of the International Council on Systems Engineering, June 1997. This is an example of the types of briefings and courses that can be provided to facilitate a project or an organization in dealing with the requirements process. The pearl here is to ensure that you *have* a requirements process and that you take advantage of industry best practices in executing it. Don't find your own way and learn the errors of your ways at considerable financial, personal, project, and organizational costs.

Pradip Kar and Michelle Bailey. *Characteristics of Good Requirements.* 1996. Available at http://www.complianceautomation.com/. This document provides a valuable, readily available discussion of the characteristics of individual and aggregate requirements (note that characteristics of individual requirements are applicable to aggregates too). Kar and Bailey emphasize that writing good requirements is difficult, requires careful thinking and analysis, but is not magical. Time spent up front, carefully defining and articulating the requirements, is essential to ensuring a high-quality product. This is recommended reading for requirements engineers.

Joachim Karlsson and Kevin Ryan. "A Cost-Value Approach for Prioritizing Requirements." *IEEE Software* 14(5) 1997: 67–74. This is an excellent article that explains a method for prioritizing requirements (see the summary of their method provided in this chapter). Karlsson and Ryan provide a process for using the cost-value approach, utilizing the Analytic Hierarchy Process (AHP), which is also explained in their article.

Geri Schneider and Jason P. Winters. *Applying Use Cases: A Practical Guide.* Reading, MA: Addison-Wesley, 1998. This is a practical guide to developing and using use cases. Schneider and Winters provide examples from their experience and provide a case study that offers insight into common errors. An illustration of the UML notation for diagramming use cases is provided. Of particular use to requirements engineers is a "how-to" discussion on applying use cases to identify requirements.

I. Sommerville, P. Sawyer, and S. Viller. "Viewpoints for Requirements Elicitation: A Practical Approach." In: *Proceedings of the 1998 International Conference on Requirements Engineering* (ICRE'98), April 6–10, 1998, Colorado Springs, CO. New York: IEEE Computer Society, 1998: 74–81. Sommerville and colleagues introduce an approach called PREview to organize requirements derived from radically different sources. They show how concerns that are key

business drivers of the requirements elicitation process may be used to elicit and validate system requirements. They note that PREview has been designed to allow incremental requirements elicitation (see Figure 4-12 for a high-level view of the PREview process).

Gerald M. Weinberg. *The Secrets of Consulting.* New York: Dorset House Publishing, 1986. Weinberg defines consulting as the art of influencing people at their request. As noted by Virginia Satir in the foreword, this book actually advises people on how they can take charge of their own growth. The author provides a light-hearted view of the role of a consultant, sharing valuable insights about people. A fundamental tenet is that we all need to follow a personal learning program. Several sources for readings and other experiences are provided.

Karl Wiegers. "First Things First: Prioritizing Requirements." *Software Development Magazine* 1999: 7(10):24–30. This is a good explanation of why requirements need to be prioritized and a helpful description of how to do it. Wiegers provides a Microsoft Excel requirements prioritization spreadsheet and other requirements tools that can be downloaded from his Web site, http://www.processimpact.com.

Karl Wiegers. "Habits of Effective Analysts." *Software Development Magazine* 2000: 8(10):62–65. See also http://www.swd.mgazine.com. Wiegers provides thoughtful and provocative insights concerning the role of the requirements engineer (also called the requirements analyst, business analyst, systems analyst, or requirements manager), patterned after Steven Covey's acclaimed book *The Seven Habits of Highly Effective People* (Fireside, 1989). He emphasizes that requirements engineering has its own skill set and body of knowledge, which is given scant attention in most computer science educational curricula and even by most systems and software engineering organizations. Many organizations expect developers or project managers to handle this vital function on their own. A competent requirements engineer must combine communication, facilitation, and interpersonal skills with technical and business domain knowledge. Even a dynamite developer or a systems-savvy user needs suitable preparation before acting in this role. Wiegers recommends that every organization should develop an experienced cadre of requirements analysts, even though requirements engineering may not be a full-time function on every project. This article is recommended reading for all PMs and task leaders.