Forewords by Manfred Broy, Technical University Munich,
and Erik Simmons, Intel Corporation

Software & Systems Requirements Engineering In Practice

BRIAN BERENBACH
DANIEL J. PAULISH
JUERGEN KAZMEIER
ARNOLD RUDORFER

Software & Systems Requirements Engineering: In Practice

About the Authors

Brian Berenbach is the technical manager of the requirements engineering competency center at Siemens Corporate Research in Princeton, NJ. Prior to joining Siemens, he consulted for many of the Fortune 100 companies on large projects. For several years he was an architect at ABB Corporation and oversaw the installation of large software-based systems in power companies. Mr. Berenbach has graduate degrees from Emory University and the U.S. Air Force, and he is an ACM Distinguished Engineer.

Daniel J. Paulish is a Distinguished Member of Technical Staff at Siemens Corporate Research in Princeton, NJ, responsible for the Siemens Software Initiative in the Americas. He is a co-author of *Software Metrics: A Practitioner's Guide to Improved Product Development*, the author of *Architecture-Centric Software Project Management: A Practical Guide*, and a co-author of *Global Software Development Handbook*. He is formerly an industrial resident affiliate at the Software Engineering Institute (SEI), and he has done research on software measurement at Siemens Corporate Technology in Europe. He holds a Ph.D. in Electrical Engineering from the Polytechnic Institute of New York.

Juergen Kazmeier holds a major degree in Mathematics and a Ph.D. in Computer Science from the Technical University of Munich. He has worked at Siemens on software development processes, methods, and tools, and he has been a researcher and consultant on modeling languages and visualization methods. As a member of the Corporate Development Audit Unit, he analyzed and supported large product development and IT projects. Within the Intelligent Transportation Systems Division, he headed a global development group, as Vice President of R&D. Dr. Kazmeier has been responsible for the Software and Engineering Research Department at Siemens Corporate Research, where he started the Siemens Requirements Engineering Global Technology Field. Currently, he is Vice President of the Software Engineering Services Division of Siemens IT Solutions and Services, headquartered in Vienna, Austria.

Arnold Rudorfer holds an M.S. in Telematics degree from the University of Technology, Graz. Prior to joining Siemens, he worked as a developer, process consultant, and manager for user interface design and usability engineering at the European Software Institute (Spain), the Institute of Production Engineering Research (Sweden), and Meta4 (Spain), a French software multinational. At Siemens, he was responsible for building up Corporate Technology's first regional business unit in the United States, the User Interface Design Center. Since 2004, he is heading the Requirements Engineering (RE) Global Technology Field with Centers of Competence in Princeton (NJ, USA), Munich and Erlangen (Europe), as well as Beijing (China).

Software & Systems Requirements Engineering: In Practice

Brian Berenbach
Daniel J. Paulish
Juergen Kazmeier
Arnold Rudorfer

New York Chicago San Francisco
Lisbon London Madrid Mexico City
Milan New Delhi San Juan
Seoul Singapore Sydney Toronto

ISBN: 978-0-07-160548-9

MHID: 0-07-160548-7

The material in this eBook also appears in the print version of this title: ISBN: 978-0-07-160547-2, MHID: 0-07-160547-9.

Contents at a Glance

This page intentionally left blank

Contents

Industrial Foreword

The last decade has seen a great deal of attention paid to requirements engineering by researchers, teachers, consultants, managers, and practitioners. Increasingly, people within information technology, commercial product development, services industries, nonprofits, government, and beyond regard good requirements as a key to project and product success. Requirements methods and practices are common subject matter for conferences, books, and classes. The business case for requirements is clear. It is in a sense a golden age for requirements.

So why then another book on the topic?

There is evidence from many sources to suggest that requirements engineering is not gaining much ground on the underlying problems of excessive rework, persistent scope creep, and finished products that fail to meet user expectations. So, despite the large investment made and the hard work done to this point, challenges still exist with regard to ever-increasing product complexity, time-to-market pressures, market segmentation, and globally diverse users.

It is here that books from practitioners, such as *Software & Systems Requirements Engineering: In Practice,* make a valuable contribution. Unlike most consultants and researchers, practitioners are deeply involved with individual projects. Moreover, they are present throughout the project and into the next one. In books from practitioners, we can see a set of requirements practices *and* the underlying setting; a detailed description of the philosophy and environment in which those practices work.

So, rather than being a compendium of possible practices, or a generic reference book, *Software & Systems Requirements Engineering: In Practice* provides readers a particular view into the world of product development and applied requirements engineering. Such windows provide a coherent and useful picture of requirements engineering.

For most practitioners, locating potential solutions to requirements engineering challenges is only part of the battle. When a method or practice is being considered for use, the question becomes "Will this work *for me*?" Understanding the experiences of

other practitioners can be an incredibly valuable shortcut to the answer, and books like *Software & Systems Requirements Engineering: In Practice* are a great place to find that information.

Erik Simmons
Requirements Engineering Practice Lead
Corporate Platform Office
Intel Corporation

Academic Foreword

Requirements engineering has proven to be one of the most difficult and critical activities for the successful development of software and software-intensive systems. The reasons for that are obvious. If requirements are invalid, then even the most careful implementation of a system will not result in a product that is useful. Moreover, if requirements are included in the requirements specifications that are not actually valid, then the product or system becomes unnecessarily expensive. This shows that requirements engineering is important.

In fact, requirements engineering is also difficult. There are many reasons for this. One is that often software-intensive systems are innovative in providing new functionality. Then, learning curves have to be considered. It is often impossible to understand, in advance, what the requirements actually are. The people involved have quite different perspectives on their valid requirements. Therefore, it is difficult to arrive at an agreement. At the same time, important requirements might be overlooked and only discovered when gaining first experiences with the produced systems. Moreover, for large, long-term projects requirements may change due to changes in the environment, the market, or user needs.

Finally, requirements engineering is often underestimated or even neglected by project management. The core of requirements engineering is devoted to understand and work on the problem statement and not so much the solution. However, management may think that only when a team of developers starts to work on the solution will the project begin to show real progress. Therefore, both for management and even for experienced developers, there is always a tendency to rush too early into the solution domain. As a result, solutions are produced that miss requirements or do not explore the full range of possible solutions.

However, even having accepted that requirements engineering is difficult, error-prone, costly, but nevertheless important, a lot more has to be understood to be able to do professional requirements engineering. For most projects, the overall development process can be easily standardized after the requirements have been captured.

What is most difficult is to standardize the process of requirements engineering, since requirements engineering is at the very beginning of a project when so much is unclear. Therefore, in industrial software development, it is important to come up with a requirements engineering approach that is on the one hand flexible but on the other hand gives enough methodological guidance.

In scientific research, exploring requirements engineering has been an active field for many years. However, at least in the beginning, requirements engineering was sometimes misunderstood as a discipline, which only has to document and specify requirements but neglects the necessary decision making. This ignores the difficulty of coming up with a requirements specification that takes into account all issues from functionality to quality and cost. There are even process development issues to consider, such as certification requirements or product constraints dealing with given operating systems or software reuse.

As a result of all these considerations, the software engineering group of Siemens Corporate Research in Princeton, New Jersey, decided a few years ago to concentrate their research on a broad spectrum of requirements engineering themes. I had the privilege to work extensively with this group of engineers and researchers, who gained a lot of experience in requirements engineering on coaching, teaching, and consulting methods in ongoing Siemens projects. Some of the projects are very large scale. It is helpful that the software engineering group in Princeton is not just focused on the core topics of requirements engineering but also covers closely related aspects such as architectural design, quality assurance, testing, model-based software development, and prototyping. Doing so, the group is looking at a systematic foundation to requirements engineering by creating a requirements engineering reference model, which helps to list all the necessary content in the requirements engineering process while at the same time providing flexibility by tailoring and by a choice of methods.

It is a pleasure to see the results of the requirements engineering research and practice at Siemens Corporate Research documented in this book. It describes a lot of precious experiences, principles, and the state of the practice in industry. As such, it is quite unique and complements existing academic books on requirements engineering, which look more at the basic terminology and approaches.

I hope that this book will help in many respects development teams around the world to improve their industrial requirements engineering. It is a pleasure for me to thank the authors and the members of Siemens Corporate Research for a scientifically fruitful cooperation over the last six years and to congratulate them on this book, which is a milestone in the field of industrial requirements engineering.

Manfred Broy
Professor of Software and Systems Engineering
Technical University of Munich

Preface

Today's software and systems engineers are facing an increasing number of challenges as they attempt to develop new products and systems faster, with higher quality and rich feature content. Part of these challenges are created by advances in computing technology, as processors and memory become faster and less expensive. Along with increased processing capability, there is an expectation that today's systems will do more. As more features are being defined for a product or system, the discipline of *requirements engineering* has increased in importance to help manage the development of the features throughout the product life cycle.

This book was written to help provide an understanding of the challenges in requirements engineering (RE) that are facing industrial practitioners and to present some best practices for coping with those challenges. Many texts on RE generally do a good job covering the basics of RE, but they may not adequately discuss the real-world problems that can make requirements elicitation, analysis, and management difficult. For example, Siemens products are typically defined with at least several thousand recorded requirements. Complex Department of Defense projects are sometimes reported as having 100,000 requirements or more in their project database. Managing projects of this size is very difficult, and managing the requirements on such a project can be quite daunting. The trend is toward defining more requirements, but developers often struggle with managing them, especially as requirements are added or changed during the development life cycle. Unfortunately, problems of scale often do not always appear on a project until it is too late to easily change process, tooling, or infrastructure. It is hoped that some of the techniques described in this book will be of use to industrial practitioners for helping to make project managers aware of potential problems before they happen, and providing techniques and guidance for successfully navigating the many pitfalls associated with large, complex projects.

Background

The Software and Systems Engineering Department of Siemens Corporate Research is involved with many software development projects with Siemens organizations working across a broad spectrum of application domains in the business sectors of industrial, health care, and energy. In our dual role of an industrial research and development laboratory, we have many opportunities for observing how requirements engineers do their work. Over time we can classify certain requirements engineering practices as "best practices," and we also learn from the not-so-best practices that were not as effective in achieving project goals.

This book was written to summarize our requirements engineering experiences, and to describe them in a form that would be useful to software and systems engineering practitioners; i.e., methods, processes, and rules of thumb that can be applied to new development projects. We are not so naïve as to believe that engineers who follow what is described in this book will work only on successful projects. We know too well that a practice that worked well in Princeton may not work so well in Poland, and much like our children, engineers sometimes learn best from their own mistakes. But, if software and systems engineers can learn from our experiences and increase the probability of a successful project outcome, our efforts will be worthwhile.

Requirements engineering is most critically applied in the early phases of a systems development project, but it is a decision-making process that is applied across the entire product development life cycle. Thus, the requirements engineer must work effectively with software and systems engineers working on other tasks such as architecture design and test procedures. Indeed, our research in requirements engineering was initiated based on the observation that the first task for an architect on a new project is to understand the product requirements.

We have worked on projects for a broad range of application domains; e.g., medical equipment, factory automation, transportation, communications, automotive. The number of requirements that must be defined, analyzed, and managed in the projects may range from a few thousand to one hundred thousand. Many of our projects are distributed over multiple development sites, involving engineers living in many different countries. These software and systems engineers are often working under great pressure to deliver the product quickly, with good quality and a rich feature set. Most of the products contain both hardware and embedded software; thus, there are dependencies on electrical and mechanical characteristics, reliability, usability engineering, and requirements that must be considered by many different stakeholders. We often work within regulated domains such as medical devices where requirements must

be carefully documented, traced, reviewed, and tested. We have also had to develop expertise on subjects that are not commonly taught at universities, such as hazard analysis.

Requirements engineering has become more complicated over time as the complexity of the products we desire to develop has increased. Thus, the requirements engineer is continually challenged by issues of scale, unstable requirements, product complexity, and managing change. Our experience has resulted from the opportunities to work on, for example, a project that is defining the requirements for an automobile infotainment system and then a few months later a project that is defining the requirements for a medical imaging system.

How to Use This Book

Our experience is with requirements engineering for products, systems, and services; typically (but not always) with high software content. This book contains RE methods, processes, and rules of thumb that have been derived from observed best practices of RE across many such projects. Thus, this book is meant for software and systems engineering professionals who are interested in learning new or validating their current techniques for RE. Such professionals include practicing requirements engineers, who should benefit most from the best practices discussed. But, the book material may also be useful to other engineering professionals, such as system architects, testers, developers, and engineering managers. The book may be useful to "not quite yet" practitioners such as graduate students in software engineering, systems engineering, or computer science. We would also hope that product or marketing managers would receive valuable information from this book as they struggle with bringing new products to a competitive market.

In order to focus on best practices and techniques for the practitioner, there is very little introductory material presented, but pointers are given to reference books that cover basic software engineering concepts. Thus, users of this book typically would have at least an undergraduate degree in computer science, systems or software engineering and some experience developing systems.

This page intentionally left blank

Acknowledgments

Since requirements engineers work across the entire development life cycle, they must interface with engineers working on specialized project tasks. We're fortunate to have had the experience of working with many talented software and systems architects, testing experts, project managers, and requirements engineers. Some of these experts have also collaborated with us on this book project as contributing authors. We acknowledge the contributions of these authors here as well as in the chapters they have written: Sascha Konrad, Raghu Sangwan, Hans Ros, Xiping Song, Bea Hwong, Marlon Vieira, Bill Hasling, Gilberto Matos, Bob Schwanke, and Brad Wehrwein.

Like software system development, writing a book can be done using a very iterative process. Once the author puts the first words to paper, there is an iterative (seemingly endless) process of review and rewrite, until we either become comfortable with the work or run out of time. We'd like to acknowledge the contributions of our review team: Capers Jones, Manfred Broy, John Worl, John Nallon, Stephan Storck, and Mark Sampson.

We'd like to acknowledge the contributions of our cartoonist, Johnol Jones, who helped to insert some humor into a usually serious subject, and our intern, Lindsay Ivins, who developed the figures and helped keep us organized when the page counts started to grow.

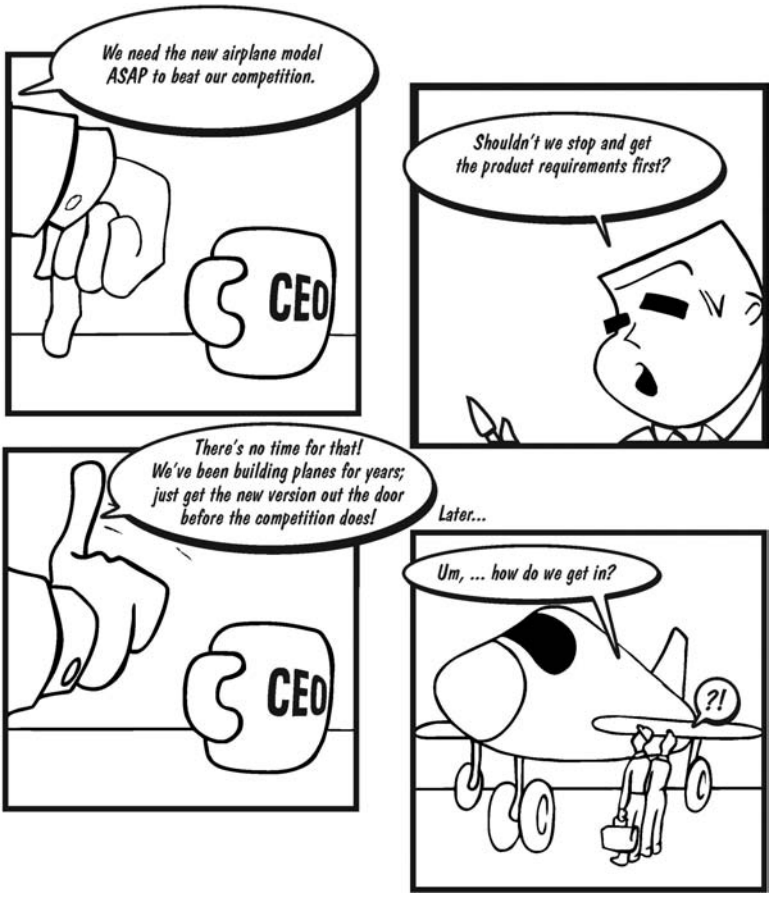
Finally, we'd like to acknowledge the support of our editor, Wendy Rinaldi, and the staff at McGraw-Hill and International Typesetting and Composition. Sometimes, just knowing that someone has confidence in us to complete the project is enough motivation to keep us working toward a successful completion.

Brian Berenbach
Daniel J. Paulish
Juergen Kazmeier
Arnold Rudorfer
Princeton, NJ 08540 USA

This page intentionally left blank

CHAPTER 1
Introduction

by Brian Berenbach, Arnold Rudorfer



Studies such as the CHAOS report [Johnson 2000] indicate that about half of the factors associated with project or product success are requirements related. Recently, researchers have reported on studies showing that project success is directly tied to requirements quality [Kamata et al. 2007]. With such overwhelming evidence that requirements engineering is a cornerstone of software systems engineering, one could ask, why is it still a relatively neglected topic in university training? It is quite rare, for example, that a new Computer Science (CS) university graduate might be asked to participate in the development of a compiler or operating system, yet nearly every graduate working in the industry will, sooner or later, be asked to participate in creating the requirements specifications for a product or service.

1.1 Why Has Requirements Engineering Become So Important?

For years, many products were successfully created without the participation of professionals who specialized in requirements creation or management. So, why is requirements engineering (RE) so important today? The answer lies in the changing nature of industry and society in general. First, the pace of product development has picked up drastically. Whereas just a few decades ago, product improvements would be a slow process, today customers often demand new versions of a product in less than one year. For example, Siemens estimates that approximately 20 years ago, 55 percent of sales were from products that were less than 5 years old. Today, 75 percent of sales are from products that were developed less than 5 years ago (Figure 1.1). Second, turnover and technology change have impacted the experience levels of professionals engaged in the development of products. Just a few short years ago, engineers might expect to spend their entire careers with a single company, whereas today job change is more common. Finally, outsourcing and offshoring have dramatically changed the product life cycle. Specifications must now be created for implementation or manufacturing by organizations with potentially limited or no domain expertise. Imagine, for example, having to create a product specification for a washing machine, dishwasher, or luxury automobile to be built by staff who may have never even seen one! Under such circumstances specifications must be exact and detailed.

Software development is highly coupled to the domain; e.g., cell phone software and avionics software tend to be designed, built, and managed with processes that are heavily domain specific. Furthermore, industries have begun to use software as product differentiators. Product innovations can be more easily implemented in software than hardware because of the lower engineering

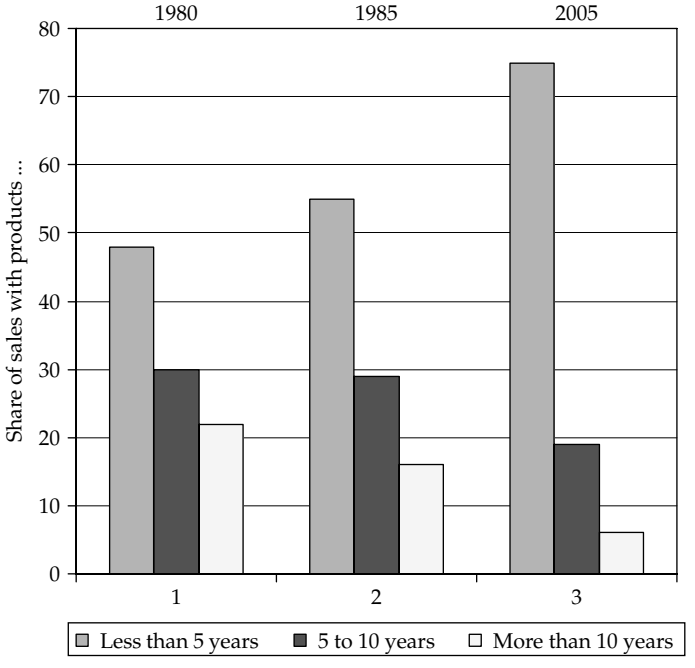


FIGURE 1.1 Acceleration of new product creation

investment and modification costs. This results in domain-specific, complex software for which high-quality requirements specifications are essential.

Requirements engineering is extremely important when a product, service, or industry is regulated. For example, the U.S. government's Food and Drug Administration (FDA) and Federal Aviation Administration (FAA) both mandate specific activities and work products (e.g., hazard analysis) where there is the potential for injury or death. Sarbanes-Oxley regulations mandate traceability for certain types of financial software used by companies doing business in the United States. The European Union and Japan have regulations for their respective businesses. Good requirements engineering practices are essential for companies that must comply with government regulations.

1.2 Misconceptions about Requirements Engineering

Misconceptions about requirements engineering can strongly influence a company's processes. Many companies and organizations have a solid understanding of requirements processes, but some do not. Some of the more common misconceptions are listed under the headings that follow.

Misconception 1: Any Subject Matter Expert Can Become a Requirements Engineer after a Week or Two of Training

Requirements engineers need strong communication and knowledge of engineering skills, the ability to organize and manage a data set of requirements, high-quality written and visual presentation skills, and the ability to extract and model business processes using both text and graphical (e.g., Integration DEFinition [IDEF], Unified Modeling Language [UML]) techniques. First and foremost, to elicit requirements from stakeholders requires the ability to interact with a variety of roles and skill levels, from subject matter experts (detailed product requirements) to corporate officers (elicitation of business goals).

Moreover, people have to be trained to write good specifications. High school and university training tends to teach a style of writing that is antithetical to the techniques needed to create unambiguous and complete documents. Requirements analysts typically need significant training, both classroom and on the job, before they can create high-quality specifications.

Misconception 2: Nonfunctional and Functional Requirements Can Be Elicited Using Separate Teams and Processes

The subject domains for nonfunctional and functional requirements are related, may impact each other, and may result in iterative changes as work progresses (see Chapter 5). Team isolation may do more harm than good.

Misconception 3: Processes That Work for a Small Number of Requirements Will Scale

Requirements engineering processes do not scale well unless crafted carefully. For example, a trace matrix is an $N \times N$ matrix, where N is the number of requirements of interest. In each cell, a mark or arrow indicates that there is a trace from requirement R_i (row i) to requirement R_j (column j). It is relatively easy to inspect, say, a 50-requirement matrix, but what happens when five to ten thousand requirements are needed to define a product? Filtering and prioritization become important in order to retrieve results that can be better understood, but the requirement annotations necessary to provide such filtering are often neglected up front because the database is initially small.

1.3 Industrial Challenges in Requirements Engineering

Over the last few years, the requirements engineering R&D focus program at Siemens Corporate Research has been involved with a substantial number of requirements engineering (RE) projects with Siemens development organizations. Many RE challenges have been identified as potentially impacting project performance. We have

observed that problems tend to be exacerbated by three critical factors, the first being a decision to outsource the implementation, the second being a significant change in technology, and the third being the introduction of new products (e.g., entering a market where the company has minimal prior experience).

When a decision is made to outsource, changes must take place in all processes, especially in the area of requirements engineering. The implementation may be done by staff with minimal domain knowledge and, because of customs, logistics, time, or distance, with limited access to subject matter experts. Attempts to use the same processes and techniques used for in-house development for the development of specifications for subcontracting or outsourcing may lead to significant delays in delivery, sometimes even resulting in project cancellation.

When technology changes rapidly, domain experts may no longer be "experts." Techniques and solutions that worked for many years may become obsolete or irrelevant. Such technological discontinuities may require substantial new training, or the experts in the older technologies may make poor decisions for new product designs. A set of key success factors for identifying potential requirements engineering problems early has been developed at SCR and is described in the next section.

1.4 Key Success Factors in Requirements Engineering

This section contains a checklist describing key factors for success in requirements engineering. Most of the factors can be evaluated prior to project initiation. Although project success cannot be guaranteed, it is likely that if several of the success factors are not in place there may be significant project difficulties.

The Project Has a Full-Time, Qualified Chief Architect

On many large projects the only senior technical role that spans the requirements process through delivery is that of the chief architect. He provides technical continuity and vision, and is responsible for the management of the nonfunctional requirements (e.g., scalability, quality, performance, environmental, etc.) and for the implementation of the functional requirements. In our experience having an experienced, full-time architect on a project contributes significantly to its success [Hofmeister et al. 1999], [Paulish 2002].

A Qualified Full-Time Architect Manages Nonfunctional Requirements

The architect is responsible for managing nonfunctional requirements and the relationships among requirements analysis, development, and management.

An Effective Requirements Management Process Is in Place

The critical success factors in a requirements management process are well defined by the Capability Maturity Model Integration (CMMI), specifically those addressing change management and traceability. A *change control board (CCB)* performs an impact analysis and conducts cost/benefit studies when feature changes are requested. The CCB acts as a gatekeeper to prevent unwanted "scope creep" and ensures properly defined product releases.

Requirements Elicitation Starts with Marketing and Sales

The marketing and sales organizations and the project's requirements engineering staff must establish strong bonds to enable accurate definition of product and/or product line features. Incorrect features and requirements may be carried over into the requirements development activities and create downstream problems.

Requirements Reviews Are Conducted for All New or Changed Requirements or Features

Requirements must be reviewed, and the review must occur at the right level. Since it typically takes one hour to review four to ten requirements (e.g., for the first review—followup reviews may go faster), reviews must be conducted at a high enough level to avoid "analysis paralysis" and yet low enough to catch significant feature-level defects.

Requirements Engineers Are Trained and Experienced

Requirements engineering is like any other scientific or engineering endeavor in that the basic skills can be learned through training. But without experienced staff, the project may "stall" or "churn" in the requirements definition stage. If the staff is new, and the team has more than four members, RE mentors should be used to improve the skills of the team.

Requirements Processes Are Proven and Scalable

When processes are defined at the start of a project, they should be bootstrapped from prior successful efforts, not just based on "textbook" examples. As the size of a project increases, or the number or size of work products increases, the methodologies must be scaled to match.

Subject Matter Experts Are Available as Needed

Arrangements must be made early on to access the experts needed to assist in defining requirements. For example, during tax season, tax

accountants and attorneys may be unavailable. Schedules cannot be defined unless the experts are available during requirements development.

All Stakeholders Are Identified

All the relevant stakeholders must be identified if requirements are to be properly defined and prioritized. The later key requirements are identified during the project, the greater the risk that major changes to the in-progress implementation will be necessary. Furthermore, the success of a product may be jeopardized by failure to validate key requirements.

The Customer Is Properly Managed

Customer management includes rapid feedback during prototyping, minimizing the number of points of contact between project staff and stakeholders, and maintaining strict control of feature change requests. It also includes using good techniques to elicit product features that are correct and unambiguous.

Progress and Quality Indicators Are Defined

The CMMI has a measurement and analysis practice area that overlaps with both requirements development and requirements management. Sometimes, a methodology (such as the Rational Unified Process [RUP] techniques for capturing text use cases) doesn't include progress or work product quality measures. These indicators must be defined in advance, or project management will find it difficult to gauge project progress and make appropriate corrections.

The RE Tools Increase Productivity and Quality

Any software tools used must enable a process (increasing productivity and CMMI compliance), rather than hinder it. Positive outcomes may require tool integration, customization, or, in rare cases where there is a justifiable cost benefit, creating a new tool from scratch.

The Core Project Team Is Full Time and Reports into a Single Chain of Command

Studies have shown that a full-time core team is essential to the success of a large project [Ebert 2005]. Without the continuity provided by a committed full-time core of people, issues may "fall through the cracks" or not show up until problems are revealed at integration testing time.

1.5 Definition of Requirements Engineering

"*Requirements engineering* [DoD 1991] involves all lifecycle activities devoted to identification of user requirements, analysis of the requirements to derive additional requirements, documentation of the requirements as a specification, and validation of the documented requirements against user needs, as well as processes that support these activities." Note that requirements engineering is a domain-neutral discipline; e.g., it can be used for software, hardware, and electromechanical systems. As an engineering discipline, it incorporates the use of quantitative methods, some of which will be described in later chapters of this book.

Whereas requirements analysis deals with the elicitation and examination of requirements, requirements engineering deals with all phases of a project or product life cycle from innovation to obsolescence. Because of the rapid product life cycle (i.e., innovation→development→release→maintenance→obsolescence) that software has enabled, requirements engineering has further specializations for software. Thayer and Dorfman [Thayer et al. 1997], for example, define software requirements engineering as "the science and discipline concerned with establishing and documenting software requirements."

1.6 Requirements Engineering's Relationship to Traditional Business Processes

It is extremely important to tie requirements activities and artifacts to business goals. For example, two competing goals are "high quality" and "low cost." While these goals are not mutually exclusive, higher quality often means higher cost. Customers would generally accept the higher cost associated with a car, known for luxury and high quality, but would likely balk at paying luxury car prices for a car expected to compete in the low-cost automotive market.

Unfortunately, some organizations may tend to decouple business and requirements activities. For example, business goals may drive marketing activities that result in the definition of a new product and its features. However, the business goals may have no clearly defined relationship to the artifacts used and produced during requirements analysis and definition. RE activities start at the very beginning of product definition with business goals and innovation. Requirements engineering techniques can add an element of formality to product definition that can improve communication and reduce the downstream implementation effort.

1.7 Characteristics of a Good Requirement

Requirements characteristics are sometimes overlooked when defining requirements processes. They can be an excellent source of metrics for gauging project progress and quality. One question we typically ask organizations when discussing their quality processes is, "Given two requirements specifications, how would you quantitatively determine that one is better than the other?" This question may be answered by looking at the IEEE 830 Standard [IEEE 1998]. The characteristics of a good requirement, as defined by the IEEE, are listed next, with several additional useful ones.

It is important to distinguish between the characteristics of a requirement and the characteristics of a requirements specification (a set of related requirements). In some cases a characteristic can apply to a single requirement, in some cases to a requirements specification, and in other cases to the relationship of two or more requirements. Furthermore, the meaning may be slightly different when referring to a requirement or a specification. Care must be taken, therefore, when discussing the characteristics described here to define the context of the attributes.

Feasible

A requirement is *feasible* if an implementation of it on the planned platform is possible within the constraints of the program or project. For example, the requirement to handle 10,000 transactions per second might be feasible given current technologies, but it might not be feasible with the selected platform or database manager. So a requirement is feasible if and only if it can be accomplished given the resources, budget, skills, schedule, and technology available to the project team.

Valid

A requirement is *valid* if and only if the requirement is one that the system shall (must) meet. Determination of validity is normally accomplished by review with the stakeholders who will be directly responsible for the success or failure of the product in the marketplace. There can be a fine line between "must" and "nice to have." Because the staff of a development team may be mainly focused on technology, it is important to differentiate between stakeholder requests that are wishful thinking and those that are actually needed to make the project or product a success. The inclusion of requirements that are nice but not valid is called "gold plating." As the name implies, having requirements on a project that are not valid will almost certainly add cost without adding value, possibly delaying project completion.

The Use of the Terms "Valid" and "Correct"
The IEEE Standard 830 uses the term "correct." We use the term "valid" instead because "correct" can be misleading. Something that is "correct" is said to be "without error," or mathematically provable. However, in the context of a requirement, "valid" is more appropriate, as the requirement may be exactly what the customer wants, but it may still contain errors or be an inappropriate solution.

Unambiguous

A requirement is *unambiguous* if it has only one interpretation. Natural language tends toward ambiguity. When learning writing skills in school, ambiguity can be considered a plus. However, ambiguity is not appropriate for writing the requirements for a product, and care must be taken to ensure that there is no ambiguity in a requirements specification. For example, consider this statement:

"The data complex shall withstand a catastrophe (fire, flood)."

This statement is ambiguous because it could mean "The data complex shall withstand a catastrophe of type fire or flood," or it could mean "The data complex shall withstand any catastrophe, two examples being fire and flood." A person skilled in writing requirement specifications would rephrase as

"The data complex shall be capable of withstanding a severe fire. It shall also be capable of withstanding a flood."
An example of an ambiguous statement is "The watch shall be water resistant." An unambiguous restatement is "The watch shall be waterproof to an underwater depth of 12 meters."

A measure of the quality of a requirements specification is the percent of requirements that are unambiguous. A high level of ambiguity could mean that the authors of the specification likely need additional training. Ambiguity often causes a project to be late, over budget, or both, because ambiguity allows freedom of interpretation. It is sometimes necessary to take a holistic view of ambiguity; e.g., a requirement may be ambiguous, but when placed in the context of the background, domain, or other related requirements, it may be unambiguous. Product features found in marketing literature (e.g., shock resistant) are typically ambiguous. However, when placed in the context of the detailed specifications used by manufacturing, the ambiguity is no longer present. On the other hand, a requirement may be unambiguous, but when placed in the context of related requirements, there may be ambiguity.

When two requirements conflict with each other or create contextual ambiguity, they are said to be *inconsistent* (see the later section "Consistent").

Verifiable

A requirement is *verifiable* if the finished product or system can be tested to ensure that it meets the requirement. Product features are almost always abstract and thus not verifiable. Analysis must be done to create testable requirements from the product features. For example, the requirement "The car shall have power brakes" is not testable, because it does not have sufficient detail. However, the more detailed requirement "The car shall come to a full stop from 60 miles per hour within 5 seconds" is testable, as is the requirement "The power brake shall fully engage with 4 lbs. of pressure applied to the brake pedal." As we have noted, product features lack detail and tend to be somewhat vague and not verifiable. However, the analysis of those features and the derived requirements should result in a specification from which full coverage test cases can be created.

Modifiable

The characteristic *modifiable* refers to two or more interrelated requirements or a complete requirements specification. A requirements specification is *modifiable* if its structure and style are such that any changes to a requirement can be made easily, completely, and consistently while retaining the structure and style. Modifiability dictates that the requirements specification has a coherent, easy-to-follow organization and has no redundancy (e.g., the same text appearing more than once), and that it keeps requirements distinct rather than intermixed. A general rule is that information in a set of requirements should be in one and only one place so that a change to a requirement does not require cascading changes to other requirements.

A typical way of ensuring modifiability is to have a requirement either reference other requirements specifically or use a trace mechanism to connect interrelated requirements.

Consistent

In general, consistency is a relationship among at least two requirements. A requirement is *consistent* if it does not contradict or is not in conflict with any external corporate documents or standards or other product or project requirements. Contradiction occurs when the set of external documents, standards, and other requirements result in ambiguity or a product is no longer feasible to build. For example, a corporate standard might require that all user interface forms have a corporate logo in the upper-right corner of the screen,

whereas a user interface requirement might specify that the logo be at the bottom center of the screen. There are now two conflicting requirements, and even though a requirements specification may be *internally* consistent, the specification would still be inconsistent because of conflict with corporate standards. Creating documentation that is both internally and externally consistent requires careful attention to detail during reviews.

Complete

A requirements specification is *complete* if it includes all relevant correct requirements, and sufficient information is available for the product to be built. When dealing with a high-level requirement, the completeness characteristic applies holistically to the complete set of lower-level requirements associated with the high-level feature or requirement. Completeness also dictates that

- Requirements be ranked for importance and stability.
- Requirements and test plans mirror each other.

A requirements specification is *complete* if it includes the following elements [IEEE 1998]:

1. Definition of the responses of the system or product to all realizable classes of input data in all realizable classes of situations. Note that it is important to specify the responses to both valid and invalid input values and to use them in test cases.
2. Full labels and references to all figures, tables, and diagrams in the specification and definitions of all terms and units of measure.
3. Quantification of the nonfunctional requirements. That is, testable, agreed-on criteria must be established for each nonfunctional requirement.

Nonfunctional requirements are usually managed by the project's chief architect. In order for the completed product to be *correct and complete,* it must include the testable requirements that have been derived from the high-level nonfunctional requirements.

It is difficult to create complete specifications, yet complete specifications are mandatory under certain circumstances; e.g., where the implementation team has no domain knowledge, or where communication between subject experts and developers will be problematic. We have seen projects where the requirements definition phase was shortened for schedule reasons. The general consensus

was that "the developers will finish writing the requirements." But when doing a risk analysis, it was nearly always quite clear that having the developers complete the requirements was not an appropriate process, due to

- Limited access to subject matter experts
- Lack of experience or bias when defining product requirements

At the back end of the project, the failure to properly define the requirements almost always caused a greater delay than would have happened by allowing the requirements specification to be completed with the appropriate level of detail up front.

Traceable

Requirements *traceability* is the ability to describe and follow the life of a requirement, in both a forward and backward direction, i.e., from its origins, through its development and specification, to its subsequent deployment and use, and through periods of ongoing refinement and iteration in any of these phases" [Gotel et al. 1994]. Traceability is required for proper requirements management and project tracking.

A requirement is *traceable* if the source of the requirement can be identified, any product components that implement the requirement can easily be identified, and any test cases for checking that the requirement has been implemented can easily be identified.

Tracing is sometimes mandated by a regulatory body such as the Federal Aviation Administration (FAA) or Food and Drug Administration (FDA) for product safety. Furthermore, there are some rare situations where failure to create the appropriate traces between requirements can have legal repercussions. Traceability is discussed in more detail in Chapter 7.

Other Project- or Product-Specific Characteristics

Occasionally, the requirements for a specific project or product have characteristics that do not apply to all the projects or products. While it can be argued that an attribute that crosscuts all other requirements is just another requirement, when treated as a characteristic it is more likely that the requirement will be fulfilled. For example, if a new system is being built that must be downward compatible with an older system, it could be argued that the need for downward compatibility is just a nonfunctional requirement. However, we have found that having such all-encompassing requirements converted to characteristics makes it more likely that the completed system will be

in compliance. A similar approach can be used for other "umbrella" requirements such as

- Compliance with Sarbanes-Oxley regulations
- Meeting all corporate security requirements
- Meeting electrical safety requirements

Characteristics of a Good Requirements Specification

As was stated in the definition of *consistency*, the definition of a characteristic may be different when applied to requirements and to a specification. A requirements specification is a filtered compendium of requirements. Having the requirements in a document rather than a database permits holistic views and allows the addition of history, a rationale, etc. There are certain characteristics that apply to specifications as opposed to individual requirements as listed here:

- A requirements specification is *feasible* if building the product specified is feasible given the state of technology, the budget, and the allotted time.
- A requirements specification is *unambiguous* if there is no pair-wise ambiguity in the specification.
- A requirements specification is *valid* if every requirement in it is valid.
- A requirements specification is *verifiable* if every requirement in it is verifiable.[1]
- A requirements specification is *modifiable* if there is no redundancy and changes to requirements are easily and consistently made; e.g., a change to one requirement does not require cascading changes to other requirements.
- A requirements specification is *consistent* if the requirement set is internally consistent.
- A requirements specification is *complete* if it provides sufficient information for complete coverage testing of the product or system.
- A requirements specification is *traceable* if every requirement in it can be traced back to its source and forward to test cases.
- A requirements specification is *concise* if the removal of any requirement changes the definition of the product or system.

[1] Product or business requirements specifications typically describe features, and as such there may be ambiguity and a lack of testability.

Requirements elicitation and analysis are typically done under project time constraints. Consequently, it is important to prioritize and identify risks when defining requirements. For example, "If this nonfunctional requirement is not completely analyzed, what are the risks to the project, the company, and/or the user?" By doing a risk analysis, the effort associated with fully defining a requirement set can usually be balanced against the needs of the project. Techniques for doing risk analysis of high-level requirements (e.g., balancing effort against need) will be discussed further in Chapter 5.

1.8 Requirements and Project Failure

It must be remembered that most systems under development are not new; i.e., only a fraction of the requirements in the product are new or unique [Jones 2007]. Yet issues of requirements maintenance and long-term support are often missing from project plans; e.g., the project plan is created as though the requirements will be discarded after project completion. When long-term requirements management is not planned, requirements creep can cause significant problems late in a project. Furthermore, Capers Jones reports that the defect rate increases significantly in requirements that are injected late over those that are created prior to the start of implementation, and the most egregious defects in requirements defined or modified late in a project can sometimes show up in litigation [Jones 2007].

1.9 Quality and Metrics in Requirements Engineering

As was mentioned in connection with the success factors for projects, project indicators need to be defined in order to have some measure of project transparency. It is important to be able to answer the questions "Am I making progress?" and "What is the quality of my work products?" How does one, for example, determine that a requirements specification is of high quality?

Requirement characteristics or quality indicators are extremely important for determining artifact quality. They can be measured by inspection (metrics), and the reported metrics can then be used to determine the quality of individual requirements and requirements specifications. Furthermore, metrics summaries tracked over time can be used to identify potential problems earlier to permit corrective actions, and provide guidance as to what type of corrective actions to take. For example, a high level of ambiguity in a requirement set might indicate that the analysts creating the requirements may need additional training in requirements writing. Some of the chapters in this book provide guidance on how to capture and use metrics to improve requirements processes.

Function Point Metrics as Leading Indicators

A function point is used to estimate the complexity and effort necessary to build a software product. Capers Jones has published extensively on this topic [Jones 2007, 2008]. Function point metrics are an excellent way of identifying potential problems with requirements prior to the implementation of a project. Furthermore, there is a clear correlation between function points and requirements; that is, function points can be used as an indicator of requirements creep and quality. Furthermore, it has been shown that function point analysis (FPA) can be effective in determining requirements completeness [Dekkers et al. 2001].

1.10 How to Read This Book

We suggest that you start by reading Chapters 1 and 2 before looking at any of the other chapters. They lay the groundwork for the remaining chapters by defining basic terminology that is used throughout the book.

Chapters 3 and 5 describe techniques for eliciting requirements. If you are interested in gathering requirements for software platforms or middleware, we also suggest that you read Chapter 6.

Chapter 4 describes modeling techniques that can be used for business or use case analysis. One specific method that has been used successfully at Siemens on several projects, the hierarchical decomposition of use cases, is described in detail.

Chapter 9 is devoted to rapid prototyping and describes a simple technique that has been found useful in the development of systems that are categorized by workflow and graphical user interfaces.

Chapter 7 describes techniques and best practices for requirements management. If you are interested in managing environments where the work may be distributed, then read Chapter 10 as well.

Chapter 8 describes advanced techniques for transforming requirements into test cases. It will be of interest to project and quality assurance staff. However, as Chapter 8 uses model-based methods, be sure to read Chapter 4 before reading Chapter 8.

Finally, Chapter 11 describes hazard and threat analysis and management in the context of a requirements engineering process. If you are an analyst working in a domain that is regulated or where there is the potential for physical or financial harm to an end user of a product, we recommend reading this chapter.

1.11 Summary

We've introduced some of the key challenges for requirements engineering and some of the success factors to achieve good RE. We've provided a definition of requirements engineering, and we've

described the characteristics of a good requirement and a good requirements specification.

1.12 Discussion Questions

1. Why is good requirements engineering more important to product development than it was ten years ago?
2. What are the differences between good requirements and a good requirements specification?
3. What are some of the key full-time roles necessary for a project to be successful?
4. What is the role of the chief architect?

References

Dekkers, C. and Aguiar, M., "Applying Function Point Analysis to Requirements Completeness," *Crosstalk,* February 2001.

DoD 91, U.S. Department of Defense, *Software Technology Strategy,* December 1991.

Ebert, C., "Requirements BEFORE the Requirements: Understanding the Upstream Impact," *Proceedings of the 13th IEEE International Conference on Requirements Engineering (RE'05),* 2005, pp. 117–124.

Gotel, O. and Finkelstein, A., "An Analysis of the Requirements Traceability Problem," *Proceedings of the First International Conference on Requirements Engineering*, Colorado Springs, CO, pp. 94–101, April 1994.

Hofmeister, C., Nord, R., and Soni, D., *Applied Software Architecture,* Addison-Wesley, Boston, MA, 1999.

IEEE Standard 830, *IEEE Recommended Practice for Software Requirements Specifications,* 1998.

Johnson, J., "Turning Chaos into Success," *Software Magazine,* Vol. 19, No. 3, December 1999/January 2000, pp. 30–39.

Jones, C., *Applied Software Measurement,* 3rd ed., McGraw-Hill, New York, 2008.

Jones, C., *Estimating Software Costs,* 2nd ed., McGraw-Hill, New York, 2007.

Kamata, M.I. and Tamai, T., "How Does Requirements Quality Relate to Project Success or Failure?" *Proceedings of the International Requirements Engineering Conference (RE'07),* 2007.

Paulish, D., *Architecture-Centric Software Project Management,* Addison-Wesley, Boston, MA, 2002.

Standish Group Report, "CHAOS," http://www.projectsmart.co.uk/docs/chaos_report.pdf, 1995.

Thayer, R. and Dorfman, M., *Software Requirements Engineering,* 2nd ed., Los Alamitos, CA: IEEE Computer Society Press, 1997.