Microsoft

Software Requirements

Third Edition

Karl Wiegers and Joy Beatty

Praise for this book

"*Software Requirements, Third Edition*, is the most valuable requirements guidance you will find. Wiegers and Beatty cover the entire landscape of practices that today's business analyst is expected to know. Whether you are a veteran of requirements specification or a novice on your first project, this is the book that needs to be on your desk or in your hands."

—Gary K. Evans, Agile Coach and Use Case Expert, Evanetics, Inc.

"It's a three-peat: Karl Wiegers and Joy Beatty score again with this third edition. From the first edition in 1999 through each successive edition, the guidance that *Software Requirements* provides has been the foundation of my requirements consulting practice. To beginning and experienced practitioners alike, I cannot recommend this book highly enough."

—Roxanne Miller, President, Requirements Quest

"The best book on requirements just got better! The third edition's range of new topics expands the project circumstances it covers. Using requirements in agile environments is perhaps the most significant, because everyone involved still needs to understand what a new system must do—and agile developers are now an audience who ought to have a good grasp of what's in this book."

—Stephen Withall, author of Software Requirement Patterns

"The third edition of *Software Requirements* is finally available—and it was worth waiting so long. Full of practical guidance, it helps readers identify many useful practices for their work. I particularly enjoy the examples and many hands-on solutions that can be easily implemented in real-life scenarios. A must-read, not only for requirements engineers and analysts but also for project managers."

—Dr. Christof Ebert, Managing Director, Vector Consulting Services

"Karl and Joy have updated one of the seminal works on software requirements, taking what was good and improving on it. This edition retains what made the previous versions must-have references for anyone working in this space and extends it to tackle the challenges faced in today's complex business and technology environment. Irrespective of the technology, business domain, methodology, or project type you are working in, this book will help you deliver better outcomes for your customers."

—Shane Hastie, Chief Knowledge Engineer, Software Education

"Karl Wiegers's and Joy Beatty's new book on requirements is an excellent addition to the literature. Requirements for large software applications are one of the most difficult business topics of the century. This new book will help to smooth out a very rough topic."

—T. Capers Jones, VP and CTO, Namcook Analytics LLC

a user gets before his account is locked, and the like. Applications that the organization develops should apply these policies—these business rules—consistently. Tracing each rule into the code that implements it makes it easier to update systems to comply with changes in the rules, such as altering the required frequency of password changes. It also facilitates code reuse across projects.

A business rules taxonomy

The Business Rules Group (2012) provides definitions for business rules from the perspectives of both the business and its information systems:

- From the business perspective: "A business rule is guidance that there is an obligation concerning conduct, action, practice, or procedure within a particular activity or sphere." (There ought to be an explicit motivation for the rule, as well as enforcement methods and an understanding of what the consequences would be if the rule were broken.)
- From the information system perspective: "A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure or to control or influence the behavior of the business."

Whole methodologies have been developed based on the discovery and documentation of business rules and their implementation in automated business rules systems (von Halle 2002; Ross 1997; Ross and Lam 2011). Unless you're building a system that is heavily rules-driven, you don't need an elaborate methodology. Simply identify and document the rules that pertain to your system and link them to the specific requirements that implement them.

Numerous classification schemes have been proposed for organizing business rules (Ross 2001; Morgan 2002; von Halle 2002; von Halle and Goldberg 2010). The simple taxonomy shown in Figure 9-1, with five types of rules, will work for most situations. A sixth category is *terms*, defined words, phrases, and abbreviations that are important to the business. You could group terms with factual business rules. A glossary is another convenient place to define terms.

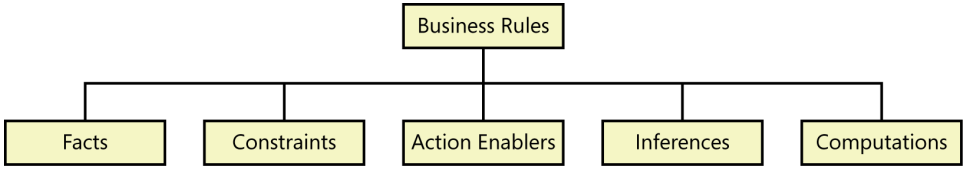

FIGURE 9-1 A simple business rule taxonomy.

Recording the business rules in a consistent way is more important than having heated arguments about precisely how to classify each one. However, a taxonomy is helpful to identify business rules you might not have thought of otherwise. Classifying the rules also gives you an idea of how you might apply them in a software application. For instance, constraints often lead to system functionality that enforces the restrictions, and action enablers lead to functionality to make something happen under certain conditions. Let's see some examples of these five kinds of business rules.

Facts

Facts are simply statements that are true about the business at a specified point in time. A fact describes associations or relationships between important business terms. Facts about data entities that are important to the system might appear in data models. (See Chapter 13, "Specifying data requirements," for more about data modeling.) Examples of facts include the following:

- Every chemical container has a unique bar code identifier.
- Every order has a shipping charge.
- Sales tax is not computed on shipping charges.
- Nonrefundable airline tickets incur a fee when the purchaser changes the itinerary.
- Books taller than 16 inches are shelved in the library's Oversize section.

Of course, there are countless facts floating around about businesses. Collecting irrelevant facts can bog down business analysis. Even if they're true, it might not be obvious how the development team is to use the information. Focus on facts that are in scope for the project, rather than trying to amass a complete collection of business knowledge. Try to connect each fact to the context diagram's inputs and outputs, to system events, to known data objects, or to specific user requirements.

Constraints

A constraint is a statement that restricts the actions that the system or its users are allowed to perform. Someone describing a constraining business rule might say that certain actions *must* or *must not* or *may not* be performed, or that *only* certain people or roles can perform particular actions. Following are some examples of constraints with various origins.

Organizational policies

- A loan applicant who is less than 18 years old must have a parent or a legal guardian as cosigner on the loan.
- A library patron may have a maximum of 10 items on hold at any time.
- Insurance correspondence may not display more than four digits of the policyholder's Social Security number.

Government regulations

- All software applications must comply with government regulations for usage by visually impaired persons.
- Airline pilots must receive at least 8 continuous hours of rest in every 24-hour period.
- Individual federal income tax returns must be postmarked by midnight on the first business day after April 14 unless an extension has been granted.

Industry standards

- Mortgage loan applicants must satisfy the Federal Housing Authority qualification standards.
- Web applications may not contain any HTML tags or attributes that are deprecated according to the HTML 5 standard.

So many constraints

Software projects have many kinds of constraints. Project managers must work within schedule, staff, and budget limitations. Such project-level constraints belong in the project management plan. Product design and implementation constraints represent imposed conditions that one might otherwise expect to be left to the discretion of the people building the solution. Such restrictions on the developer's choices belong in the SRS or design specification. Certain business rules impose constraints on the way the business operates; these should be stored in a business rules repository. Whenever these constraints are reflected in the software requirements, indicate the pertinent rule as the rationale for each such derived requirement.

Constraining business rules can convey implications for software development even if they don't translate directly into functionality. Consider a retail store's policy that only supervisors and managers are allowed to issue cash refunds larger than $50. If you're developing a point-of-sale application for use by store employees, this rule implies that each user must have a privilege level. The software must check to see if the current user is of sufficiently high privilege level to perform certain actions, such as opening the cash register drawer so a cashier can issue a refund to a customer.

Because many constraint-type business rules deal with which types of users can perform which functions, a concise way to document such rules is with a roles and permissions matrix (Beatty and Chen 2012). Figure 9-2 illustrates such a matrix for various users of a public library's information system. The roles have been separated into employees and non-employees. The system functions are grouped into system operations, operations dealing with patron records, and operations involving individual library items. An X in a cell indicates that the role named in the column has permission to perform the operation shown in the row.

Action enablers

A rule that triggers some activity if specific conditions are true is an *action enabler*. A person could perform the activity in a manual process. Alternatively, the rule might lead to specifying software functionality that makes an application exhibit the correct behavior when the system detects the triggering event. The conditions that lead to the action could be a complex combination of true and false values for multiple individual conditions. A decision table (described in Chapter 12, "A picture is worth 1024 words") provides a concise way to document action-enabling business rules that involve extensive logic. A statement in the form "If <some condition is true or some event takes place>, then

<something happens>" is a clue that someone might be describing an action enabler. Following are some examples of action-enabling business rules for the Chemical Tracking System:

- If the chemical stockroom has containers of a requested chemical in stock, then offer existing containers to the requester.
- On the last day of a calendar quarter, generate the mandated OSHA and EPA reports on chemical handling and disposal for that quarter.
- If the expiration date for a chemical container has been reached, then notify the person who currently possesses that container.

Businesses often develop policies that are intended to enhance their commercial success. Consider how an online bookstore might use the following business rules to try to stimulate impulse purchases after a customer has asked to buy a specific product:

- If the customer ordered a book by an author who has written multiple books, then offer the author's other books to the customer before completing the order.
- After a customer places a book into the shopping cart, display related books that other customers also bought when they bought this one.

Roles and Permissions Matrix	Employee	Administrator	Circulation Staff	Library Aide	Non-Employee	Volunteer	Patron
System Operations							
Log in to library system		X	X	X			
Set up new staff members		X					
Print hold pick list		X	X	X			
Patron Records							
View a patron record		X	X				
Edit a patron record		X	X				
View your own patron record		X	X	X		X	X
Issue a library card		X	X				
Accept a fine payment		X	X				
Item Operations							
Search the library catalog		X	X	X		X	X
Check out an item		X	X				
Check in an item		X	X	X		X	
Route an item to another branch		X	X	X		X	
Put an item on hold		X	X	X		X	X

FIGURE 9-2 Constraining business rules sometimes can be represented in a roles and permissions matrix.

Overruled by constraints

I recently redeemed some of my frequent-flyer miles on Blue Yonder Airlines to buy a ticket for my wife, Chris. When I attempted to finalize the purchase, BlueYonder.com said that it had encountered an error and couldn't issue the ticket. It told me to call the airline immediately. The reservation agent I (finally!) spoke with told me that the airline couldn't issue a mileage award ticket through the mail or by email because Chris and I have different last names. I had to go to the airport ticket counter and show identification to have the ticket issued.

This incident resulted from a constraining business rule that probably went something like this: "If the passenger has a different last name from the mileage redeemer, then the redeemer must pick up the ticket in person." This is probably for fraud prevention. The software driving the Blue Yonder website enforces the rule, but in a way that resulted in usability shortcomings and customer inconvenience. Rather than simply telling me about the issue with different last names and what I needed to do, the system displayed an alarming error message. It wasted my time and the reservation agent's time with an unnecessary phone call. Poorly thought-out business rule implementations can adversely affect your customer and hence your business.

Inferences

Sometimes called *inferred knowledge* or a *derived fact*, an *inference* creates a new fact from other facts. Inferences are often written in the "if/then" pattern also found in action-enabling business rules, but the "then" clause of an inference simply provides a piece of knowledge, not an action to be taken. Some examples of inferences are:

- If a payment is not received within 30 calendar days after it is due, then the account is delinquent.
- If the vendor cannot ship an ordered item within five days of receiving the order, then the item is considered back-ordered.
- Chemicals with an LD_{50} toxicity lower than 5 mg/kg in mice are considered hazardous.

Computations

The fifth class of business rules defines *computations* that transform existing data into new data by using specific mathematical formulas or algorithms. Many computations follow rules that are external to the enterprise, such as income tax withholding formulas. Following are a few examples of computational business rules written in text form.

- The domestic ground shipping charge for an order that weighs more than two pounds is $4.75 plus 12 cents per ounce or fraction thereof.
- The total price for an order is the sum of the price of the items ordered, less any volume discounts, plus state and county sales taxes for the location to which the order is being shipped, plus the shipping charge, plus an optional insurance charge.

- The unit price is reduced by 10 percent for orders of 6 to 10 units, by 20 percent for orders of 11 to 20 units, and by 30 percent for orders of more than 20 units.

Representing the details of computations in natural language like this can be wordy and confusing. As an alternative, you could represent these in some symbolic form, such as a mathematical expression or in a table of rules that is clearer and easier to maintain. Table 9-2 represents the previous unit-price discount computation rule in a clearer fashion.

TABLE 9-2 Using a table to represent computational business rules

ID	Number of units purchased	Percent discount
DISC-1	1 through 5	0
DISC-2	6 through 10	10
DISC-3	11 through 20	20
DISC-4	More than 20	30

Trap Watch out for boundary value overlaps when you are writing a set of business rules or requirements that define ranges. It's easy to inadvertently define ranges like 1–5, 5–10, and 10–20, which introduces ambiguity about which range the values of exactly 5 and 10 fit into.

Atomic business rules

Suppose you walk up to your friendly local librarian with a question. "How long can I check out a DVD for?" you ask. The librarian replies, "You can check out a DVD or Blu-ray Disc for one week, and you may renew it up to two times for three days each, but only if another patron hasn't placed a hold on it." The librarian's answer is based on the library's business rules. However, her answer combines several rules into a single statement. Composite business rules like this can be hard to understand and maintain. It's also hard to confirm that all possible conditions are covered. If several functionality segments trace back to this complex rule, it can be time-consuming to find and modify the appropriate code when just one part of the rule changes in the future.

A better strategy is to write your business rules at the atomic level, rather than combining multiple details into a single rule. This keeps your rules short and simple. It also facilitates reusing the rules, modifying them, and combining them in various ways. To write inferred knowledge and action-enabling business rules in an atomic way, don't use "or" logic on the left-hand side of an "if/then" construct, and avoid "and" logic on the right-hand side (von Halle 2002). You might break that complex library rule down into several atomic business rules, as shown in Table 9-3. (Chapter 10, "Documenting the requirements," describes the hierarchical labeling notation illustrated in Table 9-3.) These business rules are called *atomic* because they can't be decomposed further. You will likely end up with many atomic business rules, and your functional requirements will depend on various combinations of them.