Business Rules Rule

originally published in QSS (now Telelogic) online newsletter

by Ellen Gottesdiener

To encourage our sons to save their money and teach them how the exchange of money works in our society, my husband and I give them an allowance. The rule is that they must use the spending portion (not the charity or saving portion) to buy a toy. They also must have their money with them at the time of the purchase.

On a recent trip to the toy store, the eight-year old picked out a Lego Ninja's Fire Fortress. But when we checked the price, there was a problem. He had the right amount to cover the toy, but not the tax. He was short by 50 cents. He looked up at me, tears starting to form, disappointment and anguish in his face. He didn't ask me for the money, but I couldn't help breaking the rule. I fronted him the cash from next week's allowance.

If you're like me, small, underage people can sometimes get you to loosen up on some of your rules. But in the larger world, businesses can't afford the costs associated with failing to enforce business rules. For software teams, the time to know what's needed to implement business rules--such as the long reach of the tax collector into the pocket of an eight-year-old--is during the first major activity of the development process: capturing the requirements.

RULES, REQUIREMENTS: WHAT'S THE DIFFERENCE?

But wait a minute. Aren't business rules the same thing as requirements? Aren't they just the descriptions your users give you of what they want the software to do?

No – there is an important difference. Business rules are artifacts of the business and not of IT, and not every business rule is implemented in software. Business rules are what the functional requirements "know": the decisions, guidelines, assumptions, and controls that underlie the functionality. For example, requirements such as "determine product" or "offer discount" imply performance of actions, but they also embody knowledge -- the underlying business rules -- that is needed to perform those actions. An insurance claim in a "pending" stage means that certain things (variables, links to other things) must be true (invariants, or business rules). This means that

the behavior of a claim in a pending life cycle stage is dependent on business rules that govern and guide behaviors.

Business rules operationalize broader business policies. For example, a policy such as "provide discounts to repeat customers" is further clarified by stating a more discrete, atomic business rule: "If a customer orders products totaling $1000 or more in any calendar year, then offer a 10% discount on each non-sale product item." This business rule in turn is dependent on another, more fundamental business rule: "Each customer orders one or more products."

Suppose your requirements are captured in the form of use cases, a popular methodology. You have a use case called Place Order, and one of the steps is Request Total: "The customer requests a total, and the system calculates the total and applicable discounts." That's fine, as far as it goes. But it leaves a host of unanswered questions. For example, what are "applicable discounts"? Whom do I go to find out what they are? Which customers get discounts? When does a customer actually become a customer? What do you call these discounted customers? Are there other use cases that need to do something special with these discounted customers? What causes them to initially become discounted? When would they no longer get a discount? How much discount? What about sales tax? Is tax applied if it's an Internet order? Do you care about the customer's payment history in calculating the total? What if the customer has outstanding orders? What if the customer has unpaid invoices?

The answers to these questions are business rules.

WHY CAPTURE BUSINESS RULES IN REQUIREMENTS?

Each business policy, regulation, and guideline has a purpose: to use resources efficiently, boost revenue, reduce risk, conform to applicable laws, increase customer satisfaction and loyalty, and so on. If an Internet store treats its repeat customers well, for example, they will continue to buy from it. If a personnel agency does not get signed employment contracts from job candidates, they will probably continue to search for another job. If an insurance company does not follow state insurance regulations, it may have to pay fine, it may lose market share due to bad press, or it may even be shut down. Organizations need policies to act proactively, defensively, and efficiently. Each business rule aligns with one or more of these larger policies. It's easy to see that if a development team fails to capture the business rules in the first place, software projects will suffer from rework and other inefficiencies. It's a case of pay me now, or pay me (much more) later.

Because business rules lurk behind functional requirements, it's easy to miss them. Even when they're explicit, the business rules may be vague, unclear, or even contradictory. That's why software teams need formal guidance in uncovering, analyzing, and capturing business rules. Otherwise, developers will simply make whatever assumptions are needed to write the code, building their assumptions into the software with little regard for business consequences. Inevitably, they will guess wrong, and only during the latter phases will it be discovered that essential business rules have not been implemented. These late defects could have been avoided if the rules had been baselined during requirements analysis. The lack of explicit focus on capturing the business rules creates rework and other inefficiencies.

Rather than just mention business rules as "notes" in your models, you should capture and trace them as requirements in and of themselves. Focusing on business rules as the core functional requirements not only promotes validation and verification, but it can also speed the requirements analysis process.

In the end, it isn't only the unanswered questions that are costly. It's also the questions you didn't know you needed to ask.

Ellen Gottesdiener is principal consultant at EBG Consulting (www.ebgconsulting.com or ellen@ebgconsulting.com). Ellen consults, facilitates, and trains business and software teams, specializing in requirements modeling and collaborative team processes. She is author of: *Requirements by Collaboration: Workshops for Defining Needs*, contributor to *Scenarios, Stories, and Use Cases: Through the System Development Life-Cycle* and the forthcoming book *Software Requirements Memory Jogger*.

