Customer Rights and Responsibilities[1]

Karl E. Wiegers
Process Impact
www.processimpact.com

Software success depends on developing a collaborative partnership between software developers and their customers. Too often, though, the customer-developer relationship becomes strained or even adversarial. Problems arise partly because people don't share a clear understanding of what requirements are and who the customers are. To clarify key aspects of the customer-developer partnership, I propose a Requirements Bill of Rights for software customers and a corresponding customer's Requirements Bill of Responsibilities. And, because it's impossible to identify every requirement early in a project, the commonly used—and sometimes abused—practice of requirements sign-off bears further examination.

Who Is the Customer?

Anyone who derives direct or indirect benefit from a product is a customer. This includes people who request, pay for, select, specify, or use a software product, as well as those who receive the product's outputs. Customers who initiate or fund a software project supply the high-level product concept and the project's business rationale. These business requirements describe the value that the users, developing organization, or other stakeholders want to receive from the system. Business requirements establish a guiding framework for the rest of the project; everything else that's specified as a requirement should help satisfy the business requirements.

The next level of requirements detail comes from the customers who will actually use the product. Users can describe both the tasks they need to perform—the use cases—with the product and the product's desired quality attributes. Analysts (individuals who interact with customers to gather and document requirements) derive specific software functional requirements from the user requirements.

Unfortunately, customers often feel they don't have time to participate in requirements elicitation. Sometimes customers expect developers to figure out what the users need without a lot of discussion and documentation. Development groups sometimes exclude customers from requirements activities, believing that they already know best, will save time, or might lose control of the project by involving others. It's not enough to use customers just to answer questions or to provide selected feedback after something is developed. Your organization must accept that the days of shoving vague requirements and pizzas under the door to the programming department are over. Proactive customers will insist on being partners in the venture.

For commercial software development, the customer and user are often the same person. Customer surrogates, such as the marketing department, attempt to determine what the actual customers will find appealing. Even for commercial software, though, you should get actual users involved in the requirements-gathering process, perhaps through focus groups or by building on your existing beta testing relationships.

[1] This paper was originally published in *Software Development*, December 1999. It is reprinted (with modifications) with permission from *Software Development* magazine.

The Customer-Development Partnership

Quality software is the product of a well-executed design based on accurate requirements, which are in turn the result of effective communication and collaboration—a partnership—between developers and customers. Collaborative efforts only work when all parties involved know what they need to be successful, and when they understand and respect what their collaborators need to succeed. As project pressures rise, it's easy to forget that everyone shares a common objective: to build a successful product that provides business value, user satisfaction, and developer fulfillment.

Figure 1 presents a Requirements Bill of Rights for Software Customers, ten expectations that customers can place on their interactions with analysts and developers during requirements engineering. Each of these rights implies a corresponding software developer's responsibility. Figure 2 proposes ten responsibilities the customer has to the developer during the requirements process. These rights and responsibilities apply to actual user representatives for internal corporate software development. For mass-market product development, they apply more to customer surrogates, such as the marketing department.

Early in the project, customer and development representatives should review these two lists and reach a meeting of the minds. If you encounter some sticking points, negotiate to reach a clear understanding regarding your responsibilities to each other. This understanding can reduce friction later, when one party expects something the other isn't willing or able to provide. These lists aren't all-inclusive, so feel free to change them to meet your specific needs.

Figure 1. Requirements Bill of Rights for Software Customers

As a software customer, you have the right to:

1. Expect analysts to speak your language.
2. Expect analysts to learn about your business and your objectives for the system.
3. Expect analysts to structure the requirements information you present into a software requirements specification.
4. Have developers explain requirements work products.
5. Expect developers to treat you with respect and to maintain a collaborative and professional attitude.
6. Have analysts present ideas and alternatives both for your requirements and for implementation.
7. Describe characteristics that will make the product easy and enjoyable to use.
8. Be presented with opportunities to adjust your requirements to permit reuse of existing software components.
9. Be given good-faith estimates of the costs, impacts, and trade-offs when you request a requirement change.
10. Receive a system that meets your functional and quality needs, to the extent that those needs have been communicated to the developers and agreed upon.

Requirements Bill of Rights for Software Customers

Right #1: To expect analysts to speak your language. Requirements discussions should center on your business needs and tasks, using your business vocabulary (which you might have to convey to the analysts). You shouldn't have to wade through computer jargon.

Right #2: To expect analysts to learn about your business. By interacting with users while eliciting requirements, the analysts can better understand your business tasks and how the product fits into your world. This will help developers design software that truly meets your needs. Consider inviting developers or analysts to observe what you do on the job. If the new system is replacing an existing application, the developers should use the system as you do to see how it works, how it fits into your workflow, and where it can be improved.

Right #3: To expect analysts to write a software requirements specification (SRS). The analyst will sort through the customer-provided information, separating actual user needs from other items such as business requirements and rules, functional requirements, quality goals, and solution ideas. The analyst will then write a structured SRS, which constitutes an agreement between developers and customers about the proposed product. Review these specifications to make sure they accurately and completely represent your requirements.

Right #4: To have developers explain requirements work products. The analyst might represent the requirements using various diagrams that complement the written SRS. These graphical views of the requirements express certain aspects of system behavior more clearly than words can. Although unfamiliar, the diagrams aren't difficult to understand. Analysts should explain the purpose of each diagram, describe the notations used, and demonstrate how to examine it for errors.

Right #5: To expect developers to treat you with respect. Requirements discussions can be frustrating if users and developers don't understand each other. Working together can open each group's eyes to the problems the other faces. Customers who participate in requirements development have the right to have developers treat them with respect and to appreciate the time they are investing in project success. Similarly, demonstrate respect for the developers as they work with you toward your common objective of a successful project.

Right #6: To have analysts present ideas and alternatives for requirements and implementation. Analysts should explore ways your existing systems don't fit well with your current business processes, to make sure the new product doesn't automate ineffective or inefficient processes. Analysts who thoroughly understand the application domain can sometimes suggest improvements in your business processes. An experienced and creative analyst also adds value by proposing valuable capabilities the new software could provide that the users haven't even envisioned.

Right #7: To describe characteristics that will make the product easy and enjoyable to use. The analyst should ask you about characteristics of the software that go beyond your functional needs. These "quality attributes" make the software easier or more pleasant to use, letting you accomplish your tasks accurately and efficiently. For example, customers sometimes state that the product must be "user-friendly" or "robust" or "efficient," but these terms are both subjective and vague. The analyst should explore and document the specific characteristics that signify "user-friendly," "robust," or "efficient" to the users.

Right #8: To be presented with opportunities to adjust your requirements to permit reuse. The analyst might know of existing software components that come close to addressing some need you described. In such a case, the analyst should give you a chance to modify your requirements to allow the developers to reuse existing software. Adjusting your requirements when sensible reuse opportunities are available can save time that would otherwise be needed to build precisely what the original requirements specified.

Right #9: To be given good-faith estimates of the costs of changes. People sometimes make different choices when they know one alternative is more expensive than another. Estimates of the impact and cost of a proposed requirement change are necessary to make good business decisions about which requested changes to approve. Developers should present their best estimates of impact, cost, and trade-offs, which won't always be what you want to hear. Developers must not inflate the estimated cost of a proposed change just because they don't want to implement it.

Right #10: To receive a system that meets your functional and quality needs. This desired project outcome is achievable only if you clearly communicate all the information that will let developers build the product that satisfies your needs, and if developers communicate options and constraints. State any assumptions or implicit expectations you might hold; otherwise, the developers probably can't address them to your satisfaction.

Figure 2. Requirements Bill of Responsibilities for Software Customers

As a software customer, you have the responsibility to:

1. Educate analysts about your business and define jargon.
2. Spend the time to provide requirements, clarify them, and iteratively flesh them out.
3. Be specific and precise about the system's requirements.
4. Make timely decisions about requirements when requested to do so.
5. Respect developers' assessments of cost and feasibility.
6. Set priorities for individual requirements, system features, or use cases.
7. Review requirements documents and prototypes.
8. Promptly communicate changes to the product's requirements.
9. Follow the development organization's defined requirements change process.
10. Respect the requirements engineering processes the developers use.

Requirements Bill of Responsibilities for Software Customers

Responsibility #1: To educate analysts about your business. Analysts depend on you to educate them about your business concepts and terminology. The intent is not to transform analysts into domain experts, but to help them understand your problems and objectives. Don't expect analysts to have knowledge you and your peers take for granted.

Responsibility #2: To spend the time to provide and clarify requirements. You have a responsibility to invest time in workshops, interviews, and other requirements elicitation activities. Sometimes the analyst might think she understands a point you made, only to realize later that she needs further clarification. Please be patient with this iterative approach to developing and refining the requirements, as it is the nature of complex human communication and essential to software success.

Responsibility #3: To be specific and precise about requirements. Writing clear, precise requirements is hard. It's tempting to leave the requirements vague because pinning down details is tedious and time-consuming. At some point during development, though, someone must resolve the ambiguities and imprecisions. You are most likely the best person to make those decisions; otherwise, you're relying on the developers to guess correctly. Do your best to clarify the intent of each requirement, so the analyst can express it accurately in the SRS. If you can't be precise, agree to a process to generate the necessary precision, perhaps through some type of prototyping.

Responsibility #4: To make timely decisions. The analyst will ask you to make many choices and decisions. These decisions include resolving inconsistent requests received from multiple users and making trade-offs between conflicting quality attributes. Customers who are authorized to make such decisions must do so promptly when asked. The developers often can't proceed until you render your decision, so time spent waiting for an answer can delay progress. If customer decisions aren't forthcoming, the developers might make the decisions for you and charge ahead, which often won't lead to the outcome you prefer.

Responsibility #5: To respect a developer's assessment of cost and feasibility. All software functions have a price and developers are in the best position to estimate those costs. Some features you would like might not be technically feasible or might be surprisingly expensive to implement. The developer can be the bearer of bad news about feasibility or cost, and you should respect that judgment. Sometimes you can rewrite requirements in a way that makes them feasible or cheaper. For example, asking for an action to take place "instantaneously" isn't feasible, but a more specific timing requirement ("within 50 milliseconds") might be achievable.

Responsibility #6: To set requirement priorities. Most projects don't have the time or resources to implement every desirable bit of functionality, so you must determine which features are essential, which are important to incorporate eventually, and which would just be nice extras. Developers usually can't determine priorities from your perspective, but they should estimate the cost and technical risk of each feature, use case, or requirement to help you make the decision.

When you prioritize, you help the developers deliver the greatest value at the lowest cost. No one likes to hear that something he or she wants can't be completed within the project bounds, but that's just a reality. A business decision must then be made to reduce project scope based on priorities or to extend the schedule, provide additional resources, or compromise on quality.

Responsibility #7: To review requirements documents and prototypes. Having customers participate in formal and informal reviews is a valuable quality control activity—indeed, it's the only way to evaluate whether the requirements are complete, correct, and necessary.

It's difficult to envision how the software will actually work by reading a specification. To better understand your needs and explore the best ways to satisfy them, developers often build prototypes. Your feedback on these preliminary, partial, or possible implementations helps ensure that everyone understands the requirements. Recognize, however, that a prototype is not a final product; allow developers to build fully functioning systems based on the prototype.

Responsibility #8: To promptly communicate changes to the product's requirements. Continually changing requirements pose a serious risk to the development team's ability to deliver a high-quality product within the planned schedule. Change is inevitable, but the later in the development cycle a change is introduced, the greater its impact. Extensive requirements changes often indicate that the original requirements elicitation process wasn't adequate.

Changes can cause expensive rework and schedules can slip if new functionality is demanded after construction is well underway. Notify the analyst with whom you're working as soon as you become aware of any change needed in the requirements. Key customers should also participate in the process of deciding whether to approve or reject change requests.

Responsibility #9: To follow the development organization's requirements change process. To minimize the negative impact of change, all participants must follow the project's change control process. This ensures that requested changes are not lost, the impact of each requested change is evaluated, and all proposed changes are considered in a consistent way. As a result, you can make good business decisions to incorporate certain changes into the product.

Responsibility #10: To respect the requirements engineering processes the developers use. Gathering requirements and verifying their accuracy are among the greatest challenges in software development. Although you might become frustrated with the process, it's an excellent investment that will be less painful if you understand and respect the techniques analysts use for gathering, documenting, and assuring the quality of the software requirements. Customers should be educated about the requirements process, ideally attending classes together with developers. I've often presented seminars to audiences that included developers, users, managers, and requirements specialists. People can collaborate more effectively when they learn together.

What About Sign-Off?

Agreeing on a new product's requirements is a critical part of the customer-developer partnership. Many organizations use the act of signing off on the requirements document to indicate customer approval. All participants in the requirements approval process need to know exactly what sign-off means.

One potential problem is the customer representative who regards signing off on the requirements as a meaningless ritual: "I was given a piece of paper that had my name printed beneath a line, so I signed on the line because otherwise the developers wouldn't start coding." This attitude can lead to future conflicts when that customer wants to change the requirements or when he's surprised by what is delivered: "Sure, I signed off on the requirements, but I didn't have time to read them all. I trusted you guys—you let me down!"

Equally problematic is the development manager who views sign-off as a way to freeze the requirements. Whenever a change request is presented, he can point to the SRS and protest, "You signed off on these requirements, so that's what we're building. If you wanted something else, you should have said so."

Both of these attitudes fail to acknowledge the reality that it's impossible to know all the requirements early in the project and that requirements will undoubtedly change over time. Requirements sign-off is an appropriate action that brings closure to the requirements development process. However, the participants have to agree on precisely what they're saying with their signatures.

More important than the sign-off ritual is the concept of establishing a "baseline" of the requirements agreement—a snapshot at some point in time. The subtext of a signature on an SRS sign-off page should therefore read something like: "I agree that this document represents our best understanding of the requirements for the project today. Future changes to this baseline can be made through the project's defined change process. I realize that approved changes might require us to renegotiate the project's costs, resources, and schedule commitments."

A shared understanding of the requirements approval process should alleviate the friction that can arise as the project progresses and requirements oversights are revealed, or as marketplace and business demands evolve. Sealing the initial requirements development activities with such an explicit agreement helps you forge a continuing customer-developer partnership on the way to project success.

Copyright © 1999 by Karl E. Wiegers