

Process & Techniques

Collaborate for Quality

Using workshops to determine your project's requirements *by Ellen Gottesdiener*

Just how important is it to fully develop your project's requirements? After all, nailing down your requirements is usually only 8% to 15% of your overall project effort. Truth be told, it's not really something you'll want to spend your resources and energy on—*unless*, that is, you care at all about the quality of your product, your customers' level of satisfaction, and the amount of post-implementation repair you'll have to take care of down the road.

Why is it so important to get requirements right? For one thing, you're likely to introduce more defects into your software product in the requirements phase than in any other phase—and these defects account for as much as half of the product's total defects. Defects in requirements are harder to remove than defects originating in any other phase. But that's not all. Fixing them later

▶▶ QUICK LOOK

- The benefits of requirements workshops
- Applying proven patterns for effective collaboration
- Examples from successful efforts

Workshop participants, sometimes called a *task group*, are productive because collectively they have the right mix of skills and knowledge to create the work products. Members of the task group act interdependently, relying on one other's knowledge, experience, skills, and perspectives.

in the project will *cost* you more, too—as much as 100 times more after implementation than if you detected and corrected them in the requirements phase. It's no wonder that rework due to requirements defects can eat up as much as 50% of your overall budget.

One other aspect of low-quality requirements is harder to measure, but just as treacherous. It's called "scope creep," and it's often cited as the most vexing problem in software development. Unrestrained by carefully developed requirements and mutual IT-customer or product development agreement, the scope of the project keeps creeping—expanding as the work proceeds.

For all these reasons, project teams are searching for ways to develop requirements that are as free from defects as possible. One way to develop high-quality requirements is based on the use of collaborative workshops along with walkthroughs and QA checklists. As this article will illustrate, that combination of best practices gives you a powerful and efficient way to deliver quality user requirements—and, by extension, quality software products.

What Makes Workshops Work?

In collaborative workshops, participants share a common goal, and they agree to join together to create work products in the pursuit of that shared goal. Workshop participants, sometimes called a *task group*, are productive because collectively they have the right mix of skills and knowledge to create the work products. Members of the task group act interdependently, relying on one other's knowledge, experience, skills, and perspectives. They are cohesive, meaning that they are motivated to act together. With appropriate direction, such a group can be highly productive.

If you're familiar with the acronym JAD (joint application design), then you're already familiar with one type of *collaborative workshop*. JAD workshops are structured events in which a carefully selected group of stakeholders and content experts works together to create, correct, and document a predefined *deliverable*, or *work product*. The group agrees ahead of time on the deliverables, and the participants often produce some of them before the workshop; this sort of timeboxing enables the group to focus on what's really important. The most successful workshops are composed of a healthy mix of business experts (or product development people representing those experts) and IT people, led by a neutral facilitator and a scribe who documents the group's work as it proceeds.

How are JADs and other collaborative workshops different from *other* kinds of business meetings?

On the surface, collaborative workshops are like "meetings" in that both types of gatherings involve people meeting together at the same time, and both (presumably) follow a logical flow. But there are significant differences. Among them is the presence of two people who fulfill two specific process roles: facilitator and scribe. The *facilitator* is responsible for managing the group's activities, dynamics, and work products. The *scribe* documents the group's work as it proceeds. Neither facilitator nor scribe operates as a content expert; nor does either collaborate in product creation. As a result, they are free to focus on the process. As the group becomes more familiar with the process, the facilitator's role in controlling the process can be relaxed.

Another important difference between a workshop and a meeting is that a workshop is authorized by a sponsor, who ensures that the right participants are present, verifies the workshop's

purpose, and ensures that the workshop outcomes are implemented.

In effective collaborative groups, energy is high, individuals respect one another, skills are complementary, and responsibilities and roles are clear both inside and outside the group setting. To maintain energy, creativity, and motivation, the facilitator uses interactive as well as parallel group activities. For example, in a user requirements workshop, subgroups can be formed to work on portions of a single deliverable, such as business rules. Alternatively, subgroups might be assigned to work on entire work products—one group may work on use cases while another drafts a prototype and yet another creates a high-level class model. The subgroups then reconvene in a plenary (whole group) activity to share and critique their work.

The process works. Collaborative workshops have proven to be remarkably successful as a means to reduce risk, enhance quality, and increase productivity. They can reduce requirements creep by almost half. But that's only a few of their benefits. They can also

- commit users to the requirements definition process
- promote ownership for the deliverables and, ultimately, the system
- shorten the requirements phase
- eliminate nonessential requirements
- form or reinforce effective communication patterns
- build trust among project participants

By actively involving users in eliciting and testing requirements, successful workshops help you reduce defects in requirements—and en-

hance team communications along the way.

If you've ever experienced a poorly run meeting or workshop, you know how unproductive and negative an experience that can be. Successful ones, on the other hand, have a different feeling and definite flow. How can you learn from these successes, and reproduce them for your requirements workshops? One way, discussed next, is to apply proven "patterns" for effective collaborative work.

Collaboration Patterns

A pattern describes a known solution to a specific type of problem, documenting core insights or instructive information. A pattern is a best practice that can be applied in new, similar situations. Our software community has used patterns to solve problems related to software analysis, architecture, and design. More recently, patterns have been documented and applied to software development processes and organizations.

Similarly, *collaboration patterns* are recurring activities used successfully by collaborating groups. They are high-level blueprints for the behavior that these groups undertake to accomplish results together. When groups collaborate in a facilitated workshop setting, the facilitator often establishes the collaboration patterns. With experience, the group learns to incorporate these patterns, reducing the need for the third-party facilitator.

"Divide, Conquer, Correct, Collect" (see Table 1,) is one pattern that groups can use to elicit and test a requirement deliverable in a group. If you were applying this to a use case, the first step—*dividing* the use case—

Name	Divide, Conquer, Correct, Collect
Context	A complex product needs to be created quickly by people with differing expertise. Individuals need to be familiar with the content to reduce the learning curve later in the project.
Problem	How do you make sure all participants have input and verify the product? How do you permit participants to work on aspects of a product with which they are familiar and also aspects with which they are unfamiliar?
Solution	Divide —partition each product into component parts and categorize them Divide —map out the relationships between the parts into logical order, ensuring that concurrency as well as dependency is present Conquer —allocate partitions to subgroups with expertise in that category of the product Correct —conduct whole group QA activities Collect —synchronize the product elements
Consequences	Closure on decisions; since participants have created the end product by portioning it, working in parallel, and sharing details of each partition, greater in-depth knowledge of product is obtained.
Entry Criteria	<ul style="list-style-type: none"> ■ Shared time and space of knowledgeable participants ■ Ability to subdivide participants by deliverable or by experience ■ Ability to logically partition the deliverable ■ List of quality criteria for each deliverable ■ Active involvement of knowledgeable users
Exit Criteria	Creation of the end product that is agreed upon by all participants and to which all participants have had input in both creation and quality checking
Uses	Statement of business goals and objectives, creation of context-level use case, use case text, business rules, state chart diagram, project plan, migration strategy, communication plan, actor catalog

TABLE 1 The Divide, Conquer, Correct, Collect collaboration pattern

would be to partition it into its component parts, such as its name, header information, a brief description, a step-wise description, and exceptions. To *conquer* use cases in workshops in large groups (seven or more participants), each subteam focuses on one part of a single use case. Alternatively, multiple subteams can conquer different use cases at the same time and then reconvene for the next step.

In the *correct* part of the collaboration pattern, you test the use cases

with other requirements, such as scenarios and business rules, and a QA checklist. Next, you *collect* all the parts of a use case along with all the use cases for the release. These, too, are tested as a whole. This final testing can be driven by scenarios and a use case navigation diagram (a visual diagram showing the relationships among all the use cases).

In the "Divide, Conquer, Correct, Collect" pattern, the group follows these general steps:

The *facilitator* is responsible for managing the group's activities, dynamics, and work products. The *scribe* documents the group's work as it proceeds. Neither facilitator nor scribe operates as a content expert; nor does either collaborate in product creation. As a result, they are free to focus on the process.

