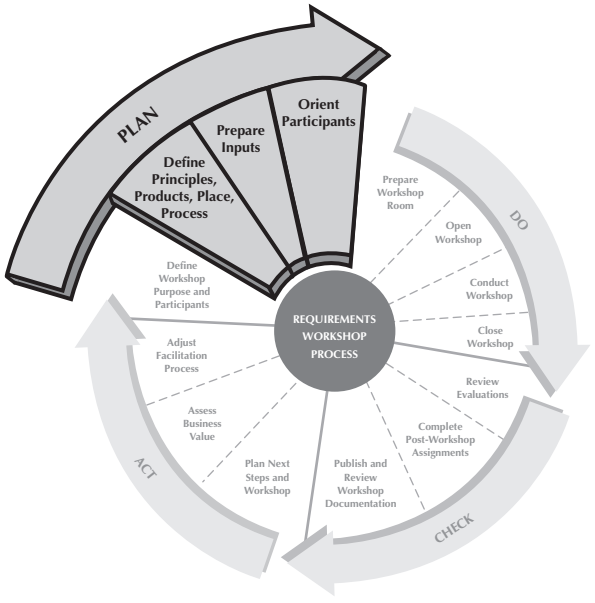
7

Products: Ending with the Beginning

"You can't have everything. Where would you put it?"
—Steven Wright

Your workshop *products* include inputs and outputs. The outputs include requirements and supplemental deliverables such as statements of issues and follow-up actions. The inputs are any materials that jump-start workshop activities and prepare participants: posters, templates, documents, workshop activity instructions, draft or predecessor requirements models, and the results of participant pre-work (see Figure 7-1).



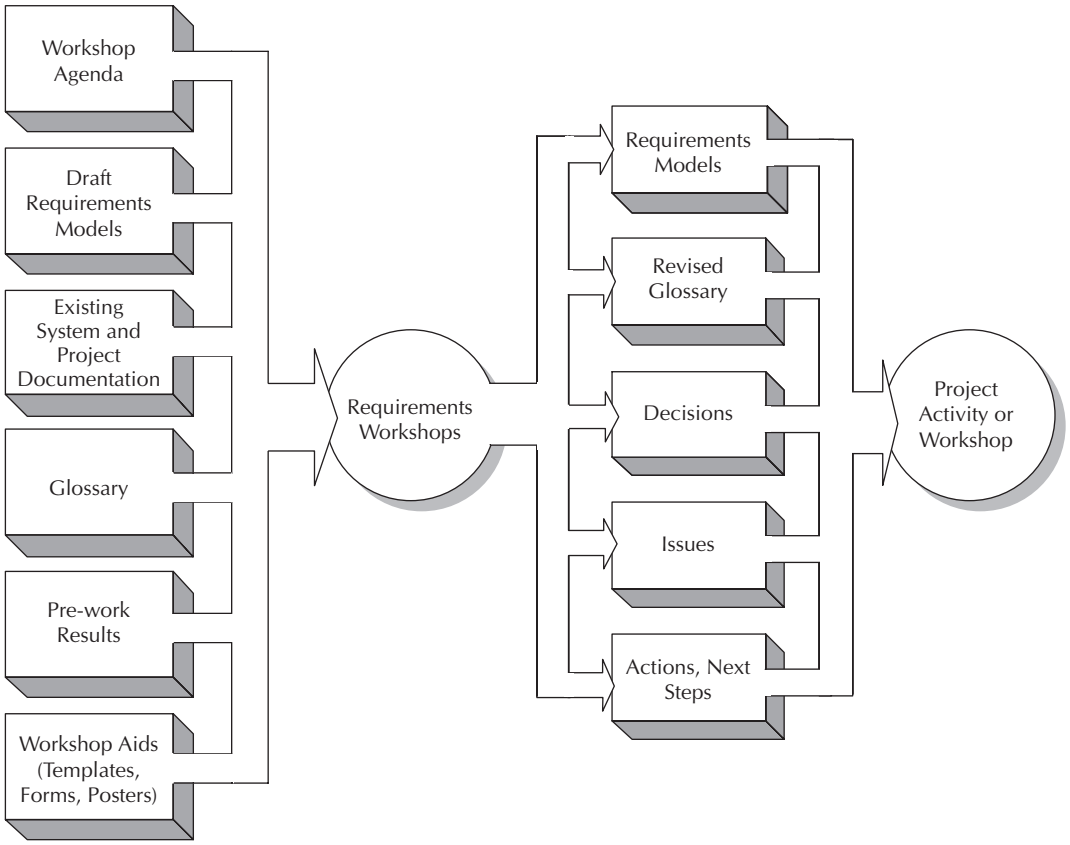


FIGURE 7-1 WORKSHOP INPUTS AND OUTPUTS

I discuss output products first because they are the basis for deciding which inputs you should use for your workshop.

OUTPUT PRODUCTS

Your primary output products are the requirements models created by the participants. (Later in this chapter I discuss intangible workshop products, such as decisions about the state of the requirements.) Along with your core requirements deliverables, you'll create supplemental deliverables, such as posters or documents that list actions or next steps, issues, and decisions.

In some cases, you may want to create supplemental workshop products to accelerate follow-up activities. In one workshop, the participants created a commu-

nication plan for organizations to be affected by the software. In another workshop, the participants identified risks and then created a risk mitigation plan and a post-workshop implementation plan. As part of an activity to jump-start their requirements modeling work, another group created a set of posters depicting a vision of what work would be like after the system was in place.

Although this book focuses on the core deliverables—the requirements models—you should consider how supplemental deliverables can dovetail with project activities and help the team to communicate effectively. Creating visually rich supplemental deliverables is fast and efficient.

MAKING DELIVERABLES VISUALLY RICH

To add visual value to deliverables, either the facilitator or the participants can use a variety of visual tools in addition to the diagram-oriented requirements models. Table 7-1 shows some formats for framing information and ideas graphically.

TABLE 7-1 VISUAL DELIVERABLES

Visual Product	Uses	Limitations
Poster	Display visions, draw themes or concepts, show agenda	Describes single point or concept
List	Brainstorm steps, define issues and expectations, identify parking lot items, name next steps	Hard to compare listed items
Clusters, affinity groups	Group related items, find themes, analyze options, associate items	Doesn't show dynamics; categories may be unclear
Arrows, flows	Cause and effect, sequence, logical progression	Implies sequence that could be incorrect
Grids, matrices	Define missing elements, clarify choices, compare choices	Can compare only a few items at a time
Drawings	Visions, stories, road maps, history, plans, business concepts	Fear of not being skilled enough to draw
Mind map	Ideas, categories, text and image groups, hierarchies	Complex to read
Circles, wheels, mandalas (circles symbolizing unity)	Unifying concepts, unstructured relationships, layers of relationships	Doesn't show details or sequence

Note that a *mind map* is an unstructured diagram that shows groupings of ideas and concepts associated with a central theme or topic (Buzan, 1989). A mind map starts with a central image or idea, which forms a focus for both eye and brain. It then branches out organically, with each branch representing a grouping or category.

The requirements models created by participants are sources for other project activities (such as design, testing, and coding) and for additional workshops (if you decide to produce requirements iteratively in a series of workshops). For example, outputs such as detailed use cases, business rules, and sketches of screens would be inputs to design and coding. You can use high-level requirements models—such as context diagrams, event tables, stakeholder classes, and use cases produced in a workshop that follows the horizontal, or "width first," workshop strategy (see Chapter 10)—as inputs to another workshop that adds more depth to those models.

To determine which requirements models to deliver, you need to do the following:

- Select model views that are aligned with the business problem domain.
- Use multiple models to increase the quality of your requirements.
- Mix text models with diagrammatic models to increase the speed of requirements definition and promote mutual learning.
- Pick multiple views and focuses (see "Model Focus" in Chapter 2) to enhance the quality of your requirements.
- Define the appropriate level of detail for each selected model.
- Iteratively deliver the requirements.
- Prioritize the requirements deliverables.
- Decide whether you should partition requirements across multiple workshops.
- Clarify your doneness tests for delivering "good enough" requirements in the workshop.

The following subsections explore these topics.

SELECT MODELS ALIGNED WITH THE BUSINESS PROBLEM

Model views—behavior, structural, dynamic, and control—provide differing perspectives of user requirements (see "Model Views" in Chapter 2). Even though

each business problem is unique, the generic set of heuristics presented in Table 7-2 illustrates how the business domain influences the types of requirements models that most strongly express user requirements.

This table is intended to be representative, not inclusive. No one view can express user requirements completely, so it's important to draw from several views. For example, if users are handling the ordering task, behavior models are useful.

TABLE 7-2 HEURISTICS FOR SELECTING REQUIREMENTS MODELS

Model View	Sample Domains	Suggested Models
Behavior	Operations, administration, inventory management, payroll, hiring, order processing, billing, accounting	Actor map and table Context diagram Domain model Event table Glossary Process map Prototype Scenarios Use cases Use case map
Structural	Data query and analysis, data extraction and reporting, customer relationship management	Business policies Business rules Context diagram Domain model Glossary Prototype Scenarios
Dynamics	Workflow problems such as document management, materials management, procurement, logistics, configuration management, imaging, loan management, credit application management, demand management, contract negotiation	Event table Glossary Process map Prototype Scenarios Statechart diagrams Use case map
Control	Logistics, claim adjudication, mortgage loan underwriting, financial risk, fraud detection, product configuration, power usage, clinical diagnosis, credit checking	Business policies Business rules Decision table/decision tree Event table Glossary Scenarios

At the same time, they're interacting with business concepts and domains such as orders, cancellations, and customers, which are best captured in structural models. As you can see from the table, a glossary should always be part of your requirements deliverables; in fact, a draft glossary should always be an input product (see "Draft Models" later in this chapter).

As you learn more about the models and use them in workshops, you'll begin to recognize which ones are more useful for the business problem you're trying to solve.

One project team asked me to review its requirements workshop plan. The business problem was to create a flexible hierarchy of salespeople and commission schemes to be used globally. The group's plan called for using use cases almost exclusively. I asked them a few questions, such as, "Who will directly interact with the system?" "How frequently will they interact with it?" "What kinds of decisions do you want the system to make?" "What information does the system use?" Their answers told me that little human interaction was needed and that the core characteristic of their business problem was to establish and manage commissions, salespeople, and zones (global groups). The problem was best expressed not with behavioral models but rather with structural and control models. Consequently, we refocused the model orientation from use cases to a data model, business policies, and business rules.

USE MULTIPLE MODELS

No single user requirements model can fully express the functional requirements for your project. A solution is to use multiple models (see the Multi-Model collaboration pattern in the Appendix).

Delivering multiple models increases the comprehensiveness of your requirements because each model reveals aspects of another. In addition, you can use one model to test the correctness of another. (Chapter 9 describes how to weave testing into the workshop flow.) This testing is aided with the use of a list of model quality assurance (QA) questions devised before the workshop (see "QA Checklist " later in this chapter). Possible questions include the following:

- For each event in our event table, is there at least one associated use case?
- Which use case would handle this scenario?
- For each use case step, have all the business rules been defined?
- What data is needed to support this business rule?

The specific questions depend on which models you deliver and their degrees of detail, but you can see how one model acts to trigger elements of another model. As illustrated in Figure 7-2, using multiple models allows you thread one model to another within a single workshop.

A structural model (such a context diagram or the glossary) relates to a dynamic model (such as the event table); a behavioral model (such as a process map) provides clues for a control model (such as business policies). Figure 7-2 also illustrates the concept of "chunking" the workshop deliverables into iterations (see "Iteratively Delivering Requirements" later in this chapter).

You can arrange the sequence of your models differently, depending on what you have as a starting point (see "Draft Models" later in this chapter). An example is presented later of a variety of sequences for arriving at business rules.

MIX TEXT AND DIAGRAMMATIC MODELS

Plan for participants to deliver a combination of both text and diagrammatic requirements models. Weave both styles of products into your process design (see Chapter 9). Text models are more precise and contribute to accuracy; diagrammatic models are fast to create and understand, something that promotes overall speed in the process. The two styles also work well with regard to our two-sided brains, with the right side being more adept at dealing with things that are visual and random and the left side being stronger at linear and analytical tasks.

To put it a different way, a picture may be worth a thousand words, but the question is, Which thousand, and what do you mean by them? You need words to answer that.

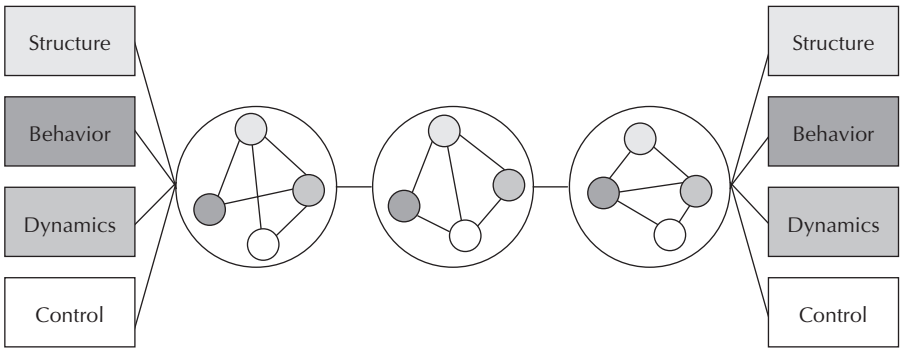


FIGURE 7-2 THREADING MULTIPLE MODELS THROUGH A WORKSHOP

Consider mixing text with diagrams for each view you deliver. For example:

- Use case text and a use case map
- An actor table and an actor map
- A glossary definition and a data model

Some models are strictly text-based and require visual models in order to elicit them. Business rules and business policies are good examples. You won't get too far asking business people, "What business rules do you need?" Instead, you need visual models that will call up the business rules. One way to overcome this problem is to represent business rules using a decision table or decision tree. Another way is to start harvesting business rules using other models. The BestClaims case study in Chapter 11 provides an example of using a visual model (a statechart diagram) to drive the specification of business rules.

Models for Harvesting Business Rules

You can use all or part of a model to thread to another model. For example, a single step of a use case gives you a thread to the data attributes needed by a data model and also to the business rules that must be enforced within that step. To arrive at business rules, you can use a variety of sequences, including the following:

- Use case → events → business policies → business rules
- Use cases → actors → domain model (class model or data entities) → business rules
- Actor → decisions → business rules
- Actors → use cases → domain model → business rules
- Events → domain model → business rules
- Events → use cases → business rules
- Events → domain model → life cycles → business rules
- Domain model → events → life cycles → business rules

MIX FOCUSES AND VIEWS

Draw from various requirements model focuses (who, what, when, where, why, and how) as well as views. Here's a typical combination:

- Use cases are a behavior view focused on *how* work gets done.
- An actor map is also a behavior view, but it focuses on *who* does the work.

- A data model or class model is a structural view focused on *what* information is needed.
- Business rules are a control view focused on *why*—the decisions that software needs to make.

Eliciting a combination of models with different focuses helps you to detect requirements errors. In one of my workshops, for example, the use cases that were created made the customer realize that at least 20 business rules needed to be defined more precisely before the requirements could be closed. The model view can trigger creation of additional models that provide more focuses. For example, if you choose a behavioral view of your requirements (say, use cases), you can use those use cases to harvest related models.

Define the Level of Detail

Decide how precise each requirements deliverable should be (see Table 2-4 in Chapter 2).

Scope-level models are particularly important when there's a high risk of scope creep or conflict among users about requirements, or when the requirements aren't precise enough to support the start of design work. In that case, if users and developers expect to make the transition to design at the end of a workshop that delivers that level of detail, you'll have many unhappy project team members.

A useful strategy for moving from scope-level or high-level requirements down to detailed-level requirements is to use iterations. This involves working together to deliver a set of models at roughly the same level of precision, checking their quality, and then moving down to the next level of detail. This approach is a top-down horizontal strategy (see "Building a Horizontal Strategy: The Top-Down Approach" in Chapter 10). Iterating in a top-down manner can accelerate the group's mutual understanding of the requirements and reduce rework within the workshop.

But you may not always want to elicit your requirements models in that top-down order. If your project has a well-defined scope, the team should be ready to jump into high-level requirements. In some cases (particularly if you're replacing an existing system), you'll start at a lower level of detail, with prototype screens and use case steps. At other times, you'll iterate among the levels in a zigzag strategy (see "The Zigzag Strategy" in Chapter 10).

Each project is unique. To decide the best way to elicit user requirements in workshops, you'll need to consider factors such as team size, location, degree of

customer and user involvement, past history with software development, existing documentation and software, and team modeling experience.

ITERATIVELY DELIVERING REQUIREMENTS

As you consider each requirements model you want to deliver, begin to partition the model into its component parts. For example, divide a detailed use case into its name, header information (such as the triggering event, event response, and initiating actor), and use case steps. Next, group elements from your various models at about the same level of detail.

Figure 7-3 shows how a use case and its related requirements can be grouped in an iterative fashion.

Although this example shows four iterations, you can use two or three iterations. This example moves from the high level to the detailed level, following what I call the vertical, how-first strategy (see Chapter 10), in which you drill down within one focus. Note that if you choose this strategy, time will constrain how many use cases (and related requirements) you can deliver.

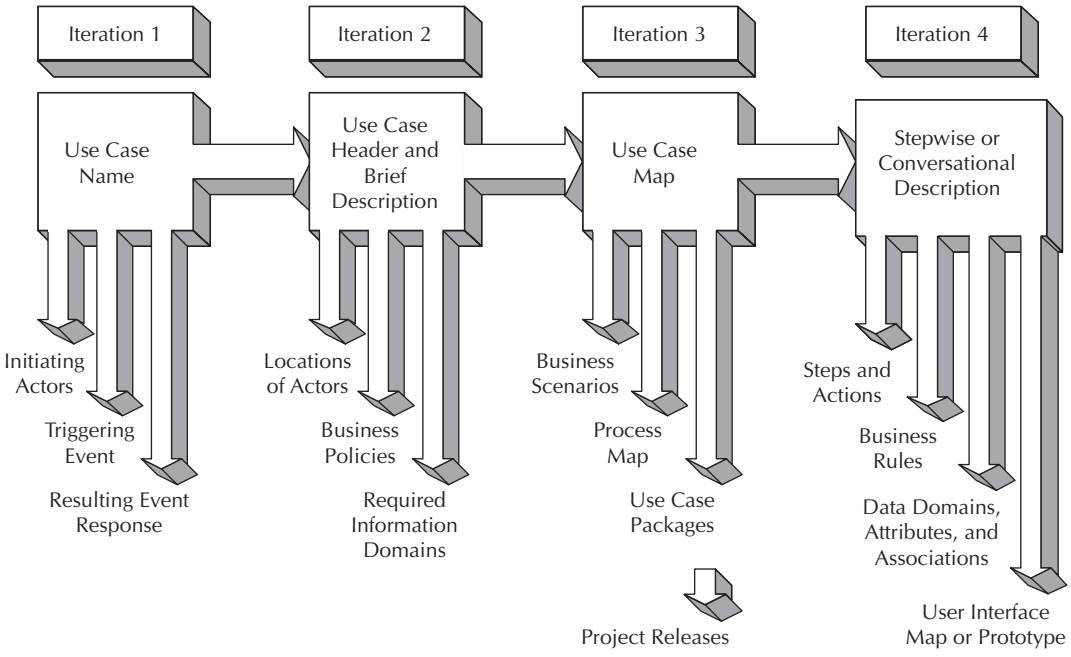


FIGURE 7-3 ITERATIVELY DELIVERING REQUIREMENTS IN A WORKSHOP

In iteration 1, participants begin with a named use case; they then define the initiating actor, the triggering event, and the event response. In iteration 2, using the prior set of requirements, participants complete the use case header using a use case template (see "Templates" later in this chapter). They do this by writing a one-paragraph description of the use case, adding the locations of the actors, listing the associated business policies, and modeling the high-level domain model (class model or data entities).

In iteration 3, participants create a *use case map* to visually lay out the predecessor-successor relationships among the use cases. (Creating a *process map* puts the use cases in the context of the workflow, giving the team the big picture of the set of requirements. This map also allows the users to understand how the use cases can fit into their workflow. This work is accelerated if a process map has been created before the workshop.) Participants logically partition use cases into packages, which in turn they'll use to define releases or increments for delivery.

In iteration 4, the participants add detailed use case steps, define business rules for each step, list data attributes needed by the steps and their rules, and sketch prototypes for each use case.

At the end of each iteration—which should take one to three hours, depending on the number of use cases—participants should test the quality of the models they've delivered (see "Define Doneness Tests" later in this chapter) in order to reach closure on that set of requirements before moving on to another set.

Using a "divide and conquer" approach helps you to avoid workshop scope creep—getting off-track during the workshop by moving up and down to different levels of detail. It also gives you a basis for ordering group activities. You also save time by eliminating the step of cleaning models up.

Chapter 9 describes ways to assign work to subgroups, iterating among individual, subgroup, and whole group activities for maximum efficiency.

Prioritize the Deliverables

It's not possible to predict exactly how long it will take to deliver each model. Knowing what's important *before* you begin the workshop will help you to adjust the agenda during the session. Decide with your sponsor and planning team which of the planned requirements models are the most critical and which can be skipped or skimmed if you run out of time.

Know whether you need to deliver *more* models with *less* precision or *fewer* models with *more* precision. This will influence the number of QA activities you build into the schedule. For one set of workshops I facilitated, the products included business rules, a data model, and life cycles for a well-defined scope. The work was to be done in four- to six-hour sessions within a four-week time frame. Anticipating that we might not complete all the deliverables, I needed to know which was more important: volume or quality. I asked the workshop sponsor what she wanted from the workshops: *more* business rules, or fewer, but *more correct*, business rules. She chose quality over volume. For that reason, I designed an agenda that added scenarios as a deliverable and also incorporated a process for testing each set of business rules with those scenarios before moving on to another set of rules.

Estimating Time

There is no magic formula for knowing how long it will take to deliver your requirements in a workshop. You must consider people factors and product factors.

People factors include how "formed" the group is as you begin (see "Forming, Storming, Norming, and Performing" in Chapter 6). Newly formed groups need more time before they can be productive, whereas a group that has worked together will be able to get down to business more quickly.

Product factors include how "done" you need your requirements to be and how much of a head start you have before the workshop. You'll need more time if your requirements must be very precise and well tested or if you don't have models to serve as a starting point. Chapter 9 has generic guidelines for workshop timing by model.

I can offer a general heuristic from my years of experience: List the deliverables you think the group can deliver in the workshop, and then divide your expectations by 3. The result will be a more realistic estimate. For example, if you think you will deliver a revised glossary, stepwise use case descriptions, prototype screens, and a high-level data model, you're likely to deliver one-third less content for each deliverable. This is why it's important to prioritize your requirements deliverables before the workshop.

Once participants have a good understanding of their requirements and are working well together, you should consult them about which ones are most important to work on together. Well-formed groups are very wise: The participants know what to do together and how to compensate for time pressures. For example, one group I facilitated decided to work through four high-priority use cases and then trust two of the participants to draft the remaining ones and return for a workshop to review, revise, and approve them.

PARTITION REQUIREMENTS ACROSS WORKSHOPS

If you're under tight time constraints or if your group is new and will need time to form, consider delivering your requirements iteratively across multiple workshops. An advantage of this approach is the efficient use of group time; participants take on post-workshop tasks, and the group uses that work as input to later workshops.

After one workshop in which we created high-level requirements, the participants went back to their business areas to ask questions of their colleagues and management so that they'd be prepared to provide details about the use cases and assess their priority. In another workshop, a list of business policies became the basis for research by a business analyst to determine which policies could be changed along with the new system. In yet another workshop, we used the high-level data model created by the participants to conduct data mapping for two possible software packages; in that way, we were able to provide details for selecting a software package in the next workshop.

Figure 7-4 shows an example. Each workshop delivers a predefined set of related requirements. These requirements then serve as inputs to another workshop occurring soon thereafter. I like to schedule iterative workshops no more than five working days apart.

There are numerous ways to arrange your session. You can conduct daily morning sessions and leave afternoons for post-workshop tasks, as described in the HaveFunds example in Chapter 11; you can use multiday workshops within a one- or two-week period, and so on. Use a schedule that optimizes the availability of people without exhausting them. Try to include time off between tasks that you will use to jump-start the next workshop.

DEFINE DONENESS TESTS

A *doneness test* consists of one or more criteria that you use to determine whether a particular deliverable is "good enough" to reach closure on. Your doneness tests will be more or less precise depending on three factors:

- The project's size (the number of people being coordinated)
- The criticality of the systems being created
- The priorities of the project (whether, for example, human lives are at stake or simply human comfort)

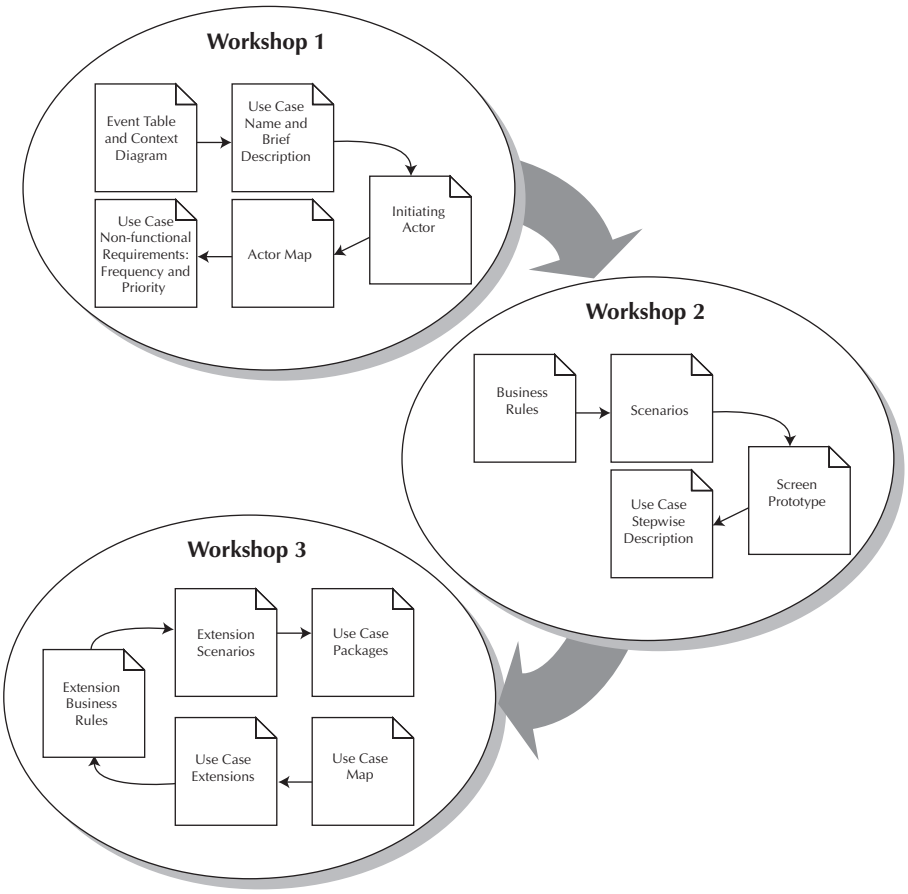


FIGURE 7-4 PARTITIONING REQUIREMENTS ACROSS WORKSHOPS

As Cockburn (2001) aptly points out, more correctness and documentation are needed by projects with a large number of team members producing critical systems using nondiscretionary funds. A "light" set of models would not do in that situation. If, however, you're building an application for internal consumption with a medium risk of monetary loss if you deliver defects, your doneness tests can be looser.

Different stakeholders have different views of doneness:

- A customer might want to deliver the requirements and end product as quickly and cheaply as possible. Perhaps you need only do some scope requirements and a prototype. The desire is to deliver relatively little detail for fewer requirements models.

- A user might be concerned with usability, so you'd focus on translating requirements associated with *who* and *how* focuses—such as actors, actor maps, prototypes, interface navigation diagrams, use cases, and use case maps—into more precise requirements.
- A software designer or architect might be most interested in building a robust product. She would want requirements models that cross multiple views and focuses, thus offering a broader understanding of the whole project rather than only the current release.

Each of these perspectives presents a special challenge to the requirements facilitator, who must help the team determine the most appropriate doneness tests for the short term while also considering the long-term goals of the project and the needs of the various stakeholders.

Your doneness tests can involve the use of a tool or a process. Available tools and processes include the following:

- A QA checklist for testing the models (see the next section)
- Scenario- or prototype-based reviews integrated into your workshop process (see Chapter 9)
- Matrices for analyzing one model or model element against another
- Metaphors for testing doneness

QA CHECKLIST A *quality assurance (QA) checklist* is a series of questions, usually stated in binary format, about a requirements model or its elements, along with questions about how one model cross-checks another. For example, if you're producing an actor table, use cases, and the glossary, these questions would apply for each use case:

- Is the use case named as an actor's goal?
- Does the use case name start with a strong action verb?
- Does the use case name include a meaningful object?
- Is the object in the use case name defined in the glossary?

The Web site for this book offers other QA checklist questions.

Using QA checklists provides more benefits than may be immediately obvious. The very process of creating and agreeing on the checklist helps the team—users,

software team, perhaps even sponsors—to clarify and define expectations for each deliverable clearly and precisely. As with using reviews during a workshop (see Chapter 9), checklists push participants to create high-quality requirements in the first place. The checklist forces you to begin with the end in mind.

In one workshop I facilitated, the group created scope-level requirements in the form of an event table and a context diagram. I divided the group of 14 people into subgroups. Each subgroup received a copy of the same checklist.

As a facilitator, I've discovered that participants give you what you ask for. My experience is that taking a testing attitude toward deliverables helps workshop participants to find more defects and find them earlier. So I told them, "Find errors in what we created."

Each subgroup indeed found defects, which were shared with the larger group and corrected. For example, the group forgot that it had to get periodic updates from an employee database, and it realized that it would need someone to play the role of approving certain types of queries to sensitive data. After that, the group continued the workshop by defining detailed requirements for each event within the scope.

MATRICES *Matrices* can help you to detect incomplete and extraneous requirements; they also serve as a useful tool for checking doneness. Participants can also create complete matrices to detect missing or extraneous requirements.

In the matrix in Table 7-3, participants fill in the cells to indicate which actors initiate which use cases. An actor without an associated use case (such as Actor02) indicates either an extraneous actor or a missing use case. Perhaps having two initiating actors might be fine (as in UseCase01); or perhaps it indicates that the actors are truly the same and that you might benefit from having a more generic name for an actor.

TABLE 7-3 SAMPLE MATRIX FOR DONENESS TESTING

	Actor01	Actor02	Actor03	Actor04
UseCase01	x		x	
UseCase02				x
UseCase03			x	
UseCase04	x			

Metaphors

A *metaphor* is a symbol, image, or figure of speech. You can use a metaphor as a loose form of doneness testing. In one workshop, we used a bull's-eye. I created a poster with a bull's-eye showing concentric circles with the label "100%" in the center. The sponsor and planning team declared before the workshop that the goal was to achieve 80 percent doneness for the models. At the end of each workshop day, I gave each participant a colored sticky dot and asked each of them to place the dot on the bull's-eye to represent where he believed our requirements deliverables were at the moment.

Each day, over four days, participants placed different dots on the bull's-eye. Each day, they got closer to the center. I also used each day's bull's-eye in leading a brief discussion about what they needed to do to get closer the next day.

Metaphors can also include wishes in the form of lists, scenarios, or visions. In one of my workshops, participants wrote stories of an ideal future, describing their work environment after all their requirements were met. In another, participants provided a list of ideal reports they could get if their requirements were met. For yet another, participants drew posters of their ideal operational environment.

Each of these metaphors serves as a doneness test because you can ask participants to return to their original metaphors or visions after they create their requirements to see whether their vision has been achieved.

Doneness Testing and Decision Making

No matter which doneness tests you use, they're not a substitute for your decision rule and decision-making process (see "Decision Rules" in Chapter 6). Your doneness tests check only the desired level of quality of your requirements, so you must follow up with your agreed-upon decision-making process. If a doneness test tells you that you have "uncooked" requirements, you can still make a decision on how to proceed.

In one of my workshops, the decision maker was the business project manager; he was also a subject matter expert. He decided to reach closure on a set of use cases, prototype screens, and business rules even though some of the QA answers indicated that the models were not complete. By following the decision-making process, he learned that the participants, including the software team, supported the current states of the models despite the flaws. He made the decision to declare them good enough, and we moved on to another set of requirements.

Combining Pre-Work with Doneness Tests: An Example

For one workshop I facilitated, plant managers needed to provide high-level data requirements for financial analysis. There were many complaints about the existing data: It was inconsistent, incorrect, late, and more. The primary deliverable for the workshop would be a data model, supplemented by the questions that managers would need to ask to run their plants, streamline operations, and meet operating targets.

Using a spreadsheet, I created a template in which they would enter their plant-management questions along with the business reasons for the questions, the decisions they made based on the answers, and a list of specific data attributes they would need. During the workshop, a review of those questions led to the discovery of high-level data entities and attributes.

We also asked the plant managers to bring to the workshop a list of the top three reports they used to help run the business, and a list of their top five "wish list" reports.

Their pre-work served as doneness tests for their data requirements, which in the workshop were listed on wall posters. In subgroups, the managers responded to these questions:

- Can you get the top three reports using the questions you've created? (If not, write a new question.)
- Can you get the top three reports with the data shown on the walls? (If not, add all the missing data to the entity—we called them data groupings in the workshop—to which it belongs.)
- Will your wish list reports answer the questions you've created? (If not, add one or more new questions to get answers.)
- Will your wish list reports be answered with the data shown on the walls? (If not, add all of the data to the appropriate data grouping.)

INTANGIBLE OUTPUT PRODUCTS

In addition to the tangible deliverables, your workshop should deliver *intangible* outcomes, including decisions, enhanced knowledge, and increased motivation.

Stopping to test for closure—in other words, to make decisions about the state of your requirements—sounds, on the surface, as if it would take valuable workshop time. Actually, it speeds up a workshop. Forcing an explicit decision creates tension as you talk about what is missing, wrong, or important. It requires you to analyze related details. It emboldens communication.

Making decisions explicit requires the group to define ground rules for achieving closure (see "Basic Ground Rules" in Chapter 6). Reaching closure requires that you combine your doneness tests with the use of a decision rule process.

The technique is simple: You explicitly define what needs to be decided during your workshop, and then add those decisions to the list of output products. Ask the workshop sponsor or project sponsor to remind people about it during the workshop kickoff. Typical decisions include the following:

- The scope of the project as represented by the requirements
- Which requirements, as represented by use cases, will be delivered for the current release
- The detailed requirements and their priorities for the next release

You also need to know whether other intangible outcomes are desired, such as people leaving the workshop with increased knowledge or motivation. These can be the healing hidden agendas discussed in Chapter 6. If you specify these outcomes, you must design workshop activities to incorporate ways of addressing them (see Chapter 9).

Note that this might prompt you to revise the list of workshop observers (see "Observers" in Chapter 5). When one intangible outcome is to increase the software team's knowledge of the business, for example, include developers and testers as observers. For one project, one desirable intangible outcome was for analysts and the project leader to become knowledgeable about how to conduct a requirements workshop. We included them as observers; I gave them a list of questions to keep handy while they observed. After the requirements workshop, we also conducted a debrief and an action planning workshop to help them incorporate, in their own projects, what they learned from observing.

When you ask questions about intangible outcomes, you might uncover hidden agendas. When I asked one project leader about intangibles, he told me that the team members were discouraged by prior requirements efforts and were concerned that they wouldn't learn anything new. He wanted to enhance the team's motivation. We decided that it was important for the project sponsor to kick off the workshop and for subject matter experts, not surrogates, to attend.

In another project, the sponsor wanted to increase team members' motivation. In addition to having him kick off the workshop, I facilitated a brief activity in which each person shared his or her professional development goals.

INPUT PRODUCTS

Once you've planned your workshop outputs, it's time to determine what input products you need. Using a set of well-thought-out input products speeds up your workshop and helps groups reach closure.

Input products include a variety of materials that prepare participants for the mental work they'll do, provide tools for jump-starting workshop activities, give guidance for creating higher-quality requirements models, and serve as doneness test resources. Input products can include the following:

- The workshop agenda
- Draft requirements models
- Systems and user documentation
- The results of pre-work
- Templates
- Workshop aids

The following subsections discuss each of these kinds of input products.

THE WORKSHOP AGENDA

The *workshop agenda* is an ordered list of activities planned for the session. The agenda, which you send to all the participants before the workshop, should also include your other P's: purpose, participants, principles, products, and place. (See Chapter 9 for tips on designing your agenda; an agenda template is available on the book's Web site.)

DRAFT MODELS

Use predecessor requirements models or draft output models to jump-start your modeling work. For example, if you're creating use cases in the workshop, you can use an actor table or an actor map as a mechanism to help you create a list of use cases. By using focus questions about the predecessor model (for example, "What goals does this actor have for interacting with the system?"), you can elicit another related model. (See "Using Focus Questions" in Chapter 9 for more about focus questions.)

You can also use predecessor and draft requirements in performing doneness tests on models. For example, a list of scenarios created before the workshop can

be used in testing use cases. To test for doneness during the workshop, you can use pre-work such as user-created scenarios and prototype screens created by the software team by walking through them in parallel with use case steps.

A draft version of a model provides the basis for an activity to fix or finish it. For example, you can use a draft event table or context diagram during an activity in which the group reaches closure on scope. A rough version of a statechart diagram gives you a starting point for generating more states, which in turn triggers events and business rules.

At a minimum, *you should create a draft glossary before a requirements workshop*. The terms in the glossary are the semantic basis for all your user requirements; the definitions are themselves fundamental business rules.

I like to ask one person to be the *glossary guardian,* the person in charge of keeping the glossary up-to-date during the workshop. The job of the glossary guardian is to listen for business terms that haven't been defined or that might need to be clarified. Allow everyone to tune in to terms; challenge each participant to also play the role of glossary guardian. (To make it fun, I sometimes announce a reward for anyone who stops the group when a term needs to be defined or clarified. Reverse rewards can work, too, if you make them humorous—for example, getting "banged" with a rubber hammer if you use a term without explaining it.)

The glossary guardian keeps the glossary up-to-date during the workshop.

Data modelers make excellent glossary guardians because they naturally think in terms of words and their connections, and they often seek precise definitions of business terms.

UNIVERSAL MODELS An alternative or supplemental draft model you can use as input is a *universal model*. This generic business model is documented in books or available for purchase from consulting companies.

Universal models tend to be abstract. They use generic business terms (*business party, transaction, event, agreement*) that you can use as starting points for modeling activities. For example, if the universal model is represented as a data model, participants can list examples of each concept in the model or list subtypes to jump-start the modeling of their own domain model.

SYSTEMS AND USER DOCUMENTATION

It's a good idea to have documentation available that might be useful during the workshop. This includes the project charter or vision statement, organization charts, operating procedures and guidelines, reference manuals, user documentation, help desk documentation, user job aids, training manuals and documentation, and systems manuals and documentation. Identify each document that might be useful, and be sure that someone is responsible for bringing it.

PRE-WORK

Pre-work involves specific assignments for participants to complete before the workshop. Examples of pre-work include filling out a template that lists steps to complete a task, naming business rules in a specific format for a set of use cases, listing stakeholders and their functional areas, reviewing and commenting on a set of draft use cases, reading the project charter, listing key inputs needed by users, and drawing a poster that depicts an image of the future after the requirements have been met.

Asking participants to complete pre-work has one or more of these benefits:

- It mentally prepares participants for the type of thinking they'll do in the workshop.
- It provides research information and starting points to one or more workshop activities.
- It forces participants to seek information from other experts who might fill gaps in their subject matter expertise.

Specifying some kind of pre-work signals that the requirements workshop isn't an ordinary meeting.

Pre-work is essential when you're operating under a tight time frame, when users are surrogates and not real subject matter experts, when participants hail from different business areas, or when you'll be working with detailed requirements. Your workshop schedule must take into account the time for preparing pre-work. For example, you may need to create a template with instructions (see "Templates" next).

Designing pre-work is one thing, but how do you get participants to complete it? Your sponsor and your planning team can help. For example:

- Have the pre-work assignments made by the sponsor or an influential planning team member.
- Provide information at least one week before the workshop.
- Conduct an orientation meeting to brief participants on how to complete the pre-work, particularly if their assignments are complex.
- Ask the workshop sponsor or project sponsor to send e-mail or voice mail to the participants requesting that they complete their pre-work.
- Have the pre-work responses sent to the workshop sponsor or a planning team member rather than to the facilitator.

Workshop Orientation Meetings

Under certain circumstances I like to arrange orientation meetings before workshops.

In one workshop I facilitated, participants came from around the globe. They had pre-work assignments and would be gathering in a central location for the session. Many of them had never participated in a requirements workshop, and few of them knew one another. We arranged a videoconference orientation meeting kicked off by the project sponsor. In this meeting, we reviewed the workshop agenda and clarified how to complete the pre-work. We also discussed important issues such as what to wear and how they would access their voice mail on breaks during the workshop. This 45-minute meeting helped to break the ice before the live workshop.

Here are some circumstances under which you should consider holding an orientation meeting:

- Participants have pre-work assignments that may involve completing templates or doing complex research.
- Participants don't know one another.
- Participants would benefit from hearing suggestions for what materials to bring to the workshop.
- You are meeting over a period of days or weeks.

Keep the meeting short, and be sure you have the project or workshop sponsor kick it off to set the stage.

Templates

Templates are standardized formats that you use during workshop activities to structure the contents of an output product. Templates can be used for building the requirements models, prioritizing requirements, creating related deliverables

(such as an action plan or a communication plan), debriefing the workshop, and assessing the team climate (if you incorporate team-building activities into the workshop).

Creating templates forces your workshop planning team to think deeply about project-specific deliverables. It also helps participants to make the best use of their workshop time, and it helps you to provide precise, clear instructions about workshop tasks.

Business Rules Templates

Business rules come in many forms and categories, including terms, facts, constraints, factor clauses, and action clauses. There's no agreement on a standard set of categories for business rules—nor should there be, because the rules should fit the business problem. Some business problems are more business rule-*based* (for example, insurance underwriting), whereas others are more business rule-*constrained* (such as payroll). Business rules are vital for high-quality requirements, and they also provide useful links to other requirements models.

You can capture your business rules as free-form statements generated by business people, but these statements tend to be ambiguous and not rigorous. Each free-form business rule may decompose into numerous rules. You must untangle each statement and resolve its internal inconsistencies. Also, it's hard to trace such unstructured statements to other requirements models, and they create change control headaches. If the rules are smaller and more discrete, you can more easily manage change. To capture business rules atomically, it helps to have a business rule template designed for the purpose.

A *business rule template* presents standard, precise syntax for writing business rules. The template should be designed so that the resulting business rules are declarative, atomic, distinct, independent statements. The very process of tailoring a template to a particular business problem is beneficial because it requires you to understand the problem in great depth. This means working directly with business customers to design and tailor the template.

Examples of useful business rule templates include the following:

- Each <business term> may <verb phrase> <business term>.
- When <event|condition is true>, then <action>.
- If <condition true>, then <condition true/conclusion>.

You can use your templates during the workshop to guide participants in writing business rules.

For one workshop, I worked with the planning team to settle on a template for the use cases. For another, I created a business requirements matrix template for use during the workshop. For yet another, I designed a simple form for capturing scenarios. In another case, I devised a template for scenario testing our use cases and data model. In several workshops in which the subsequent software product had cross-functional impact, I used a communication plan template. For most workshops delivering precise business rules, a business rule template is also critical.

Your template should include instructions (or it should be supplemented by a document work aid that has instructions) and perhaps examples of how to properly complete the template. Templates can be created as word processing documents or can be drawn on posters. If your recorder acts as a technographer (see Chapter 5), she can input the group's work into a word processor. You can then display or print the contents of all completed templates for participants to refer to or review during the session.

Templates help to ensure that you get high-quality, consistent information from participants. This is especially critical when you use multiple subgroups working on different requirements for the same model at the same time during the session, a technique discussed in Chapter 9. Templates also force you to clarify doneness tests because you must describe what each deliverable should look like. If you can visualize what the group members must deliver, it's easier to design the collaborative processes to get them there. Templates also help you to give precise instructions to the group during the workshop.

You can supplement the template with a completed example (see "Examples" later in this chapter), which helps participants understand the models they need to create.

Workshop Aids

Workshop aids are tools for conducting activities in the session. These aids include static posters for participants to reference, sample models, instructions or tips to use while working on the models, worksheets for documenting changes to models, and materials and supplies.

Posters *Posters* are charts that are visible in the room. Create your posters before the workshop, and place them on the workshop room walls using tape, pins, or tacks or by using sticky poster paper (posters on a pad with the top back

prepared with repositionable glue). You should prepare posters with the following titles:

- Workshop Purpose
- Workshop Products
- Workshop Agenda (the flow of activities)
- Issues or Parking Lot (titled and left blank)
- Input Products
- Actions (titled and left blank)
- Decisions (titled and left blank)

Tips for Working on Posters

- Print in thick, CAPITAL letters.
- Write straight up and down.
- Use plain block letters (not script).
- Avoid black, except for numbering charts.
- Use these colors for text: blue, brown, purple, green.
- Use these colors for highlighting: orange, red, yellow, pink.
- Be faithful to people's own words.
- Use white space liberally.
- Consider using the symbols shown in Table 7-4.

EXAMPLES Sample models use a "begin with the end in mind" philosophy: They show participants what the end product should look like by providing simple but correct examples. I've found these examples useful for almost every re-

TABLE 7-4 POSTER SYMBOLS

•	Bullets (centered dot) to help items stand apart
☆	Stars for noteworthy items
○	Circles to connect ideas and draw attention to items
▭	Borders to frame a page or blocks of text
→	Arrows for sequences, cycles, and merging

quirements model that employs a template, including use cases, business rules, and scenarios. To create examples that are relevant to the problem, consult with your planning team or other subject matter experts. Even an example that's wrong, but is correctly written or drawn, is useful for participants.

INSTRUCTIONS AND TIPS Prefabricated instructions or guidelines can help you save time during the workshop. Tell participants what they need to do for each workshop activity, and also give them written instructions (paper handouts or on posters). This method gives everyone a reference point for completing activities.

Provide instructions, especially early in a workshop, for subgroup activities (see Chapter 9). For example, instructions might stipulate that there should be a timekeeper, a recorder, and someone to make sure that the content of the work follows a template format.

It's a good idea to include tips for creating high-quality models. In one workshop, I gave participants a list of good verbs to use in their use case names. In another, I provided a list of verbs for them to use for data model relationships. These kinds of tips help save workshop time.

WORKSHEETS Your recorder can use worksheets for tracking the disposition of and changes to requirements models. When your recorder uses a laptop with a word processor or spreadsheet program, you can project the changes on a screen or print them for reference. In several workshops that delivered use cases, I found a use case completion worksheet helpful (see the Web site for an example).

A note of caution: Don't assume that your readers will be able to understand your workshop aids. Test any input products you create—instructions, examples, tips, worksheets— before you give them to participants. Ask planning team members and a few of the participants to review the instructions and see whether they understand them. Then, during the workshop, review the instructions and ask for clarifying questions.

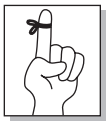
MATERIALS AND SUPPLIES Materials and supplies include all the physical things you'll need to run the workshop: items such as name cards, posters, cards or sticky notes, color markers, paper cutters, side-hole punches, a printer, and binders. (A generic list is available on the Web site.)

Set up binders with dividers in which participants can store all the paper documents they'll work with during the session. I like to supply tab dividers already labeled for things such as purpose, principles, and agenda. The binders can also hold copies of modeling tips, activity instructions, templates, and examples.

After the workshop, participants can use the binders for reference and for storing hard copies of requirements and workshop and project information. Binders save people time by helping them to avoid searching through large sets of documents. They also help participants stay organized during the workshop, and using them sends a message that their work warrants a special storage place.

THE WORKSHOP REPOSITORY

The *workshop repository* is where you store the soft copies of your workshop products, pre-work, and post-workshop products. You might use a project Web site, a requirements management tool, a network folder, or a combination of these. Remind participants before, during, and after the session about the repository. Arrange access for them, allowing them to deposit any pre-work there.



TIPS

- For your workshop deliverables, use a combination of text and diagrammatic models.
- Select requirements models that align with the problem domain.
- Use multiple views and focuses.
- Partition each requirements model into parts; group parts at the same level of detail.
- Prioritize your expected deliverables.
- Be realistic about how much participants can deliver.
- Design or use templates for your requirements deliverables.
- Provide examples or draft models for participants to use as starters.
- Assign pre-work to participants.
- Define doneness tests for each requirements model you plan to deliver.
- Use multiple workshops integrating the best practices listed here (see the example in Figure 7-5).

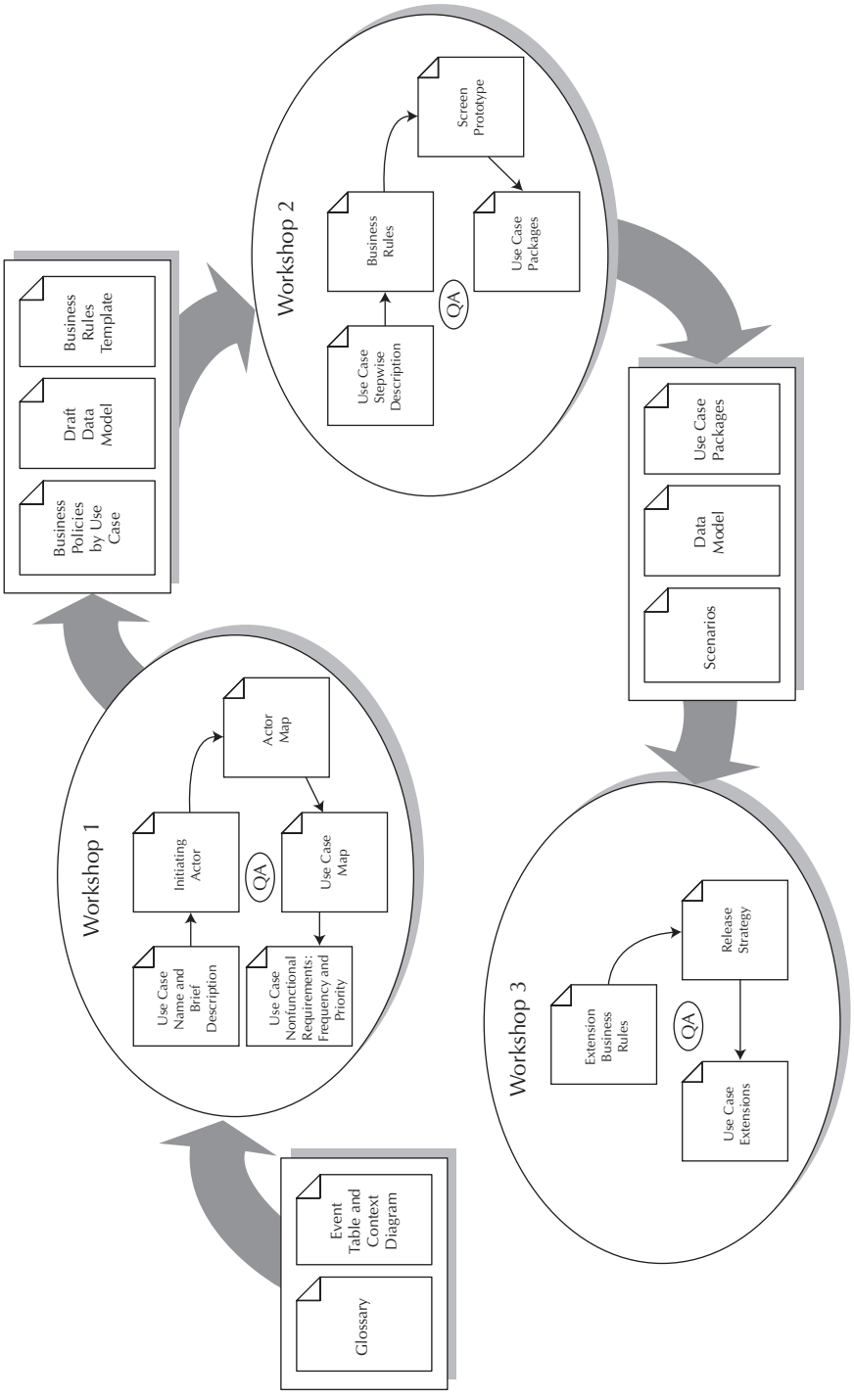


FIGURE 7-5 Workshop Iterations Integrating Pre-Work, Post-Work, and QA

QUESTIONS TO ASK STAKEHOLDERS ABOUT PRODUCTS

The following questions can help you determine workshop inputs and outputs. You can preface these questions by saying, "The next set of questions will help us determine what products to deliver during the workshop."

- What deliverables do you expect to produce in the workshop?
- How much detail should each deliverable have?
- What criteria will you use to determine that the deliverables are complete, clear, and correct? How will we know that they have the quality you expect and need?
- Would you rather have more requirements delivered at a higher level of detail, or fewer requirements with more detail?
- What products are needed as input to the workshop? How much lead time do we have to pull together input products?
- Who can review the instructions for the group activities that we'll create?
- Where will the workshop materials be stored (a workshop repository)? Do we need to do any work to create this repository?

See the Web site for the book for more questions about workshop products.

FOR MORE INFORMATION

Boehm and Basili (2001) summarize 10 ways to reduce defects in code, including peer reviews and perspective-based reviews (such as using scenarios to walk through your requirements).

Boehm and Hoh (1999) describe a technique for analyzing and negotiating conflicts in requirements.

Brooks (1995), a classic on the human aspects of software engineering, points out that eliciting requirements is the most difficult part of building software. The author emphasizes the need to continually refine and iterate requirements with customers.

Buzan (1989) provides insights and techniques, based on research on the human brain, for tapping into the right and left sides of our brains.

Cockburn (2001) presents wisdom on how software teams work best. Chapter 4, on methodologies, provides an excellent discussion on how you must tailor your methodology for each project, including the types of work products and their precision.

Cohen (1995) provides a step-by-step guide to quality function deployment (QFD), a systematic methodology for deriving and evaluating product features from both customer and designer points of view. The key technique, the "voice of the customer," is explained along with Kano's understanding of quality from the customer's perspective.

Chapter 9 of Highsmith (2000) discusses workstate lifecycle management, in which teams iteratively deliver components (from requirements to code and tests) in a predefined state, rather than focus on workflow.

Howell (1995) is a useful reference for a variety of visual tools you can adapt for your supplemental workshop deliverables. Included are tools for representing text information in provocative and interesting ways, such as circles, continuums, workflows, t-charts, and a variety of worksheets.