CHAPTER 4 *Use-Cases*

IN THIS CHAPTER

Having defined the project vision, the actors, and the events of interest in the system, we next move to assigning the events to use-cases. The use-case is the one UML artifact that focuses on what *the system will be contracted to do, not* how *it will do it. Use-cases are the hub from which all requirements are derived.*

*Every event identified in Chapter 3 as part of the Inception phase must be satisfied by a use-case. One use-case can satisfy many events. As a result, a use-case may have more than one pathway through it. A **pathway** is the set of steps that must be carried out to satisfy the goal of the actor. This chapter examines how to identify these pathways, and it describes in detail the primary pathway through each use-case.*

The chapter also investigates a preliminary software architecture. This architecture is based on what is known about the application's execution domain and is represented by the UML component and deployment diagrams.

The Inception phase will also produce an estimate of both the number of iterations and increments in which the system will be realized and the time and costs incurred for deliverables. This grouping of functionality is visualized with the UML package diagram.

GOALS

- To add to our information about Remulak Productions, especially pertaining to preliminary technology needs and goals.
- To explore the concept of the use-case and the use-case diagram.
- To review a sample use-case template.
- To define the various pathways through use-cases: primary, alternate, and exception.
- To learn how to give a detailed description of the most common pathway through a use-case.
- To discuss a preliminary software and hardware architecture.
- To review the planned increments and implementation schedule.

The Sample Project

Recall from Chapter 3 that Remulak Productions, which specializes in locating hard-to-find musical instruments, primarily guitars, wants to replace its legacy order entry application. Company founder, owner, and president Jeffery Homes has contacted us to address his concern about his company's ability to keep up with technology.

Homes is not an IT expert, but he knows that success depends on the replacement system's extensibility into the future. His initial concern is the applicability of the Internet to his business. He wants to minimize cost outlays for the technology used while maintaining flexibility to change platforms later. He makes the following observations about the company:

- Many of its products are very expensive and require quite a bit of hand-holding and selling by the order clerk.
- Many, if not all, of its customers want to be able to get information on orders that have been placed and shipped without interacting with a company representative.
- Many of its customers want to order other products from Remulak that are not as unique as its instruments, including recording and mixing technology, microphones, and recordable CD players.

On the basis of these observations, Homes decides that an Internet-based inquiry capability could benefit Remulak. However, some cases still require customized, personal interaction, at least initially to enter the order in the system. So it appears that all but the most complicated orders could utilize an Internet-based order solution. These facts are crucial to consider as we devise our initial software architecture for Remulak.

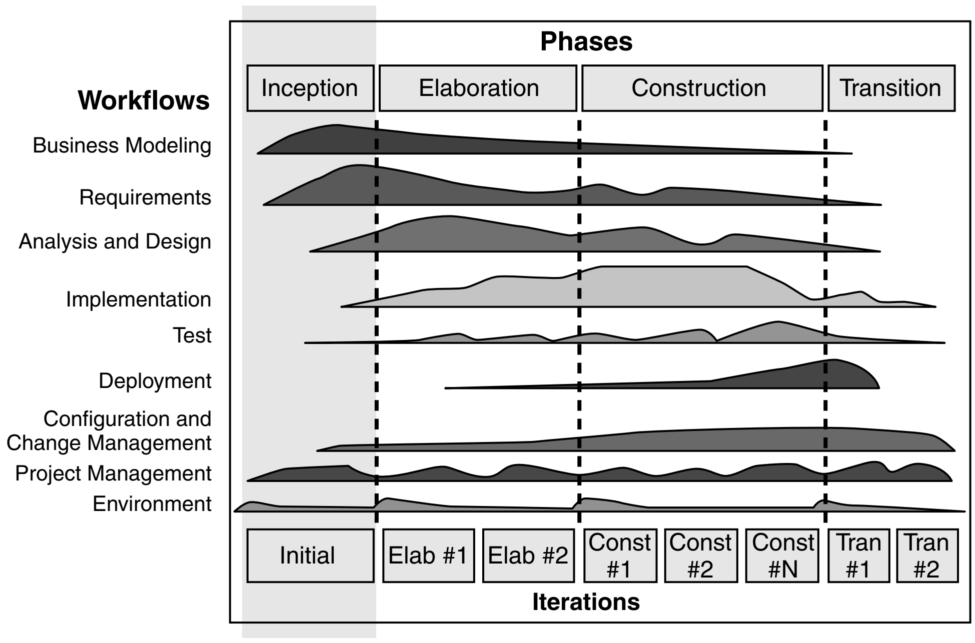
The Process Model

Once again, the Unified Process model is spotlighted, again with emphasis on the Inception phase (Figure 4-1).

In this chapter, the following Unified Process workflows and activity sets are emphasized:

- Requirements: Analyze the Problem
- Requirements: Define the System

FIGURE 4-1 *Unified Process model: Inception phase*



- Requirements: Manage the Scope of the System
- Test: Plan Test

Use-Cases

The project's success depends largely on defining requirements in a manner that is intuitive to both the IT staff and the project sponsors. The requirements documentation not only serves as a key artifact, but also must be a living artifact. Requirements cannot be expected to remain stable throughout the project effort, so they must be in a format that can be easily assimilated by the current project team, as well as by new team members. The key vehicle to capturing requirements is the **use-case**. Many people I work with ask, "But when do you do functional requirements?" Use-cases are the functional requirements of the application.

The concept of the use-case sprang from Ivar Jacobson's early work on large telecommunications switching systems at Ericsson. This was his primary contribution to UML. As mentioned in earlier chapters, Jacobson's Objectory Process was transformed into Rational Software's popular process model, the Unified Process.

Use-cases are goal-oriented and serve as containers of sorts for the interactions that they satisfy. Jacobson's definition of a use-case will serve as a baseline as we begin our exploration of this key UML artifact:

> *A behaviorally related sequence of interactions performed by an actor in a dialog with the system to provide some measurable value to the actor.*

Let's examine this definition in more detail.

1. *Behaviorally related* means that the interactions should, as a group, constitute a self-contained unit that is an end in itself, with no intervening time delays imposed by the business.
2. The use-case must be initiated by an actor and seen through to completion by an actor.
3. *Measurable value* means that the use-case must achieve a particular business goal. If we cannot find a business-related objective for the use-case, we should rethink it.
4. The use-case must leave the system in a stable state; it cannot be half done.

Use-cases are goal-oriented. Remembering this is key to using them effectively. They represent the *what* of the system, not the *how.* Use-cases are also technology neutral, so they can apply to any application architecture or process. Even if you were to throw out all that UML offers and use only use-cases, a project's requirements would be defined in a much clearer and more coherent fashion than if use-cases were not used. Figure 4-2 identifies the sequence that we follow to arrive at the use-cases.

The process of identifying use-cases is easier if the event table is grouped by actor, as in Table 4-1. Often a use-case will be associated with only one actor. However, some types of use-cases—for example, those that provide information such as reports—often have more than one actor.

Notice that certain events in the table tend to cluster together, such as those dealing with order entry. Also some events deal with maintaining an order, as well as inquiring about an existing order. We take these natural groupings and write a short descriptive phrase (one or two words) for each, asking these questions:

- What do these events have in common?
- Do these events have the same ultimate goal? If so, what is it?

FIGURE 4-2 *Getting to use-cases*

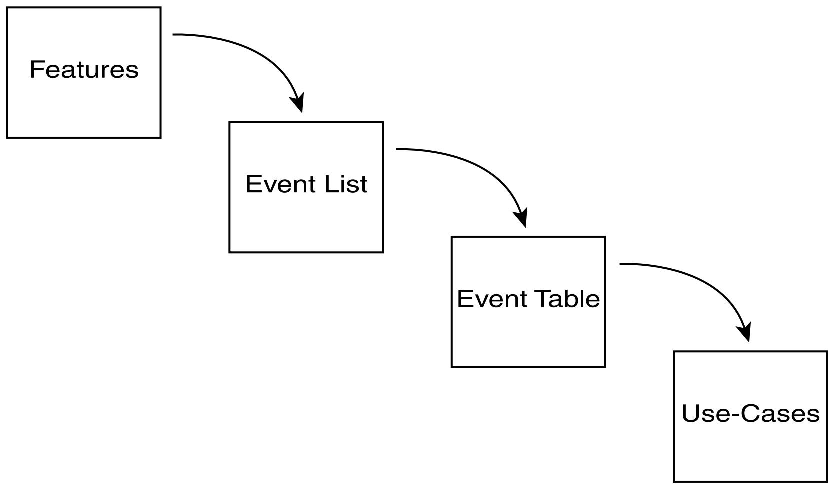


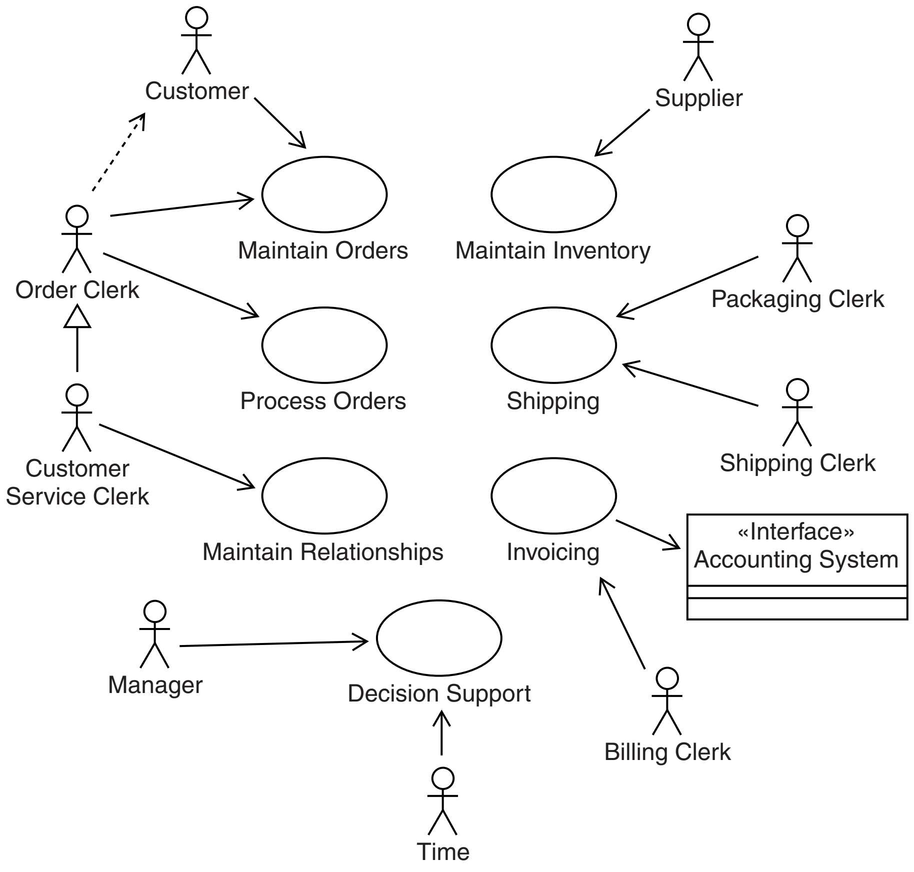
TABLE 4-1 *Event Table with Events Grouped by Actor*

Subject	Verb	Object	Frequency	Arrival Pattern	Response
Customer	Places	Order	1,000/day	Episodic	Order is edited and saved in the system.
Customer	Buys	Warranty	60/day	Episodic	Order is validated as to terms and then recorded.
Customer	Changes	Order	5/day	Episodic	Order is edited to make the change and then recorded.
Customer	Cancels	Order	1/week	Episodic	Order is removed from the system.
Customer	Inquires about	Order	200/day	Episodic	Order information is provided.
Customer service clerk	Changes	Address	5/week	Episodic	Address is changed.
Shipping clerk	Sends	Order	700/day	Episodic	Order is packaged and shipped according to the shipping terms.
Supplier	Sends	Inventory	5–10/day	Episodic	New inventory is checked in.
Time	Produces	Back-order report	3/week	Episodic	Report is produced.
Time	Produces	Accounting interface	1/week	Episodic	Interface is added to the system.
Packaging clerk	Prepares	Order	100/day	Episodic	Package is readied for shipment.
Manager	Inquires about	Orders	5/day	Episodic	Request is honored.
Billing clerk	Inquires about	Past-due invoice	10/day	Episodic	Past-due report is produced.

Next we place each of these descriptive phrases next to an oval (the designation for a use-case), along with the associated actors to produce our first attempt at a use-case diagram, shown in Figure 4-3.

The actors are connected to the use-cases by arrows, which indicate **association** relationships. Notice the accounting system actor. Because it is an external system and nonhuman, this actor is rendered by the *interface stereotype* and drawn as a box. Stereotypes are discussed in

FIGURE 4-3 *Remulak use-case diagram*



more detail later in the chapter. For now, think of a stereotype as a categorization or grouping mechanism. Using stereotypes is optional; there is nothing semantically incorrect about representing this actor as a stick figure. I consulted on a project whose management took a dim view of stick figures as part of the deliverables, so we went with the box notation. You can probably guess that this organization had quite a few other hurdles to clear before it became successful at using UML.

Notice also the line with the large triangle arrow connecting the customer service clerk and order clerk actors. This arrow means that a customer service clerk "is an" order clerk, thereby denoting a **generalization** relationship. That is, the two actors can be substituted for the same thing, and they perform the same logical function but happen to be physically one person (recall from Chapter 3 that we should focus on the role and not on the actor).

One last feature to notice is the dashed line from the order clerk to the customer, which denotes a **dependency association** relationship—that is, a relationship in which one actor depends on the other. In this case the fact that the arrow points from the order clerk to the customer (rather than vice versa) shows that the order clerk depends on the customer. Many practitioners would say that the actor involved in processing an order is the customer. This is debatable because the user interface will be geared toward the order clerk, not the customer. However, the order clerk is useless without the customer's input.

To check consistency, we determine whether any events identified during the Inception phase have not found a home in a use-case. Any that haven't are out of scope, worded incorrectly, or missing a use-case. It's a good practice to add a new column to the event table for recording which use-case satisfies each event.

This is a good time to talk about **scope creep**. Every project suffers from it. If you have experience with one that hasn't, then you have had the luxury of wonderful project sponsors, or you aren't telling the truth, in order to protect the guilty. Keep the following bit of wisdom close at all times:

> *All requirements originate as features in the project vision. These features lead to events that the use-cases then satisfy. If a particular feature isn't defined as an event and satisfied by a use-case, then it is a change request that requires some form of change control action.*

This doesn't mean that the project won't accept changes. Rather it says that to keep your schedule true to the increments to be implemented, changes must be tracked and assigned to a later-scheduled increment that will satisfy them. Do you need to update the event table and related use-case documentation? You bet. If the event table and use-cases are not kept up-to-date, you will lose all traceability.

We have now completed the first UML diagram: the use-case diagram (although it is subject to change). From the perspectives of the project sponsors, this is the most important diagram because it represents their domain; that is, it represents the boundaries of the project scope from a high-level perspective, and it is the basis for dividing the project into multiple increments.

Finding the Pathways through Use-Cases

Before we can complete the project vision, we need to do more work: We need to find the various pathways through each use-case. We will identify three levels of pathways—primary, alternate, and exception—as well as present a use-case template to be used across the entire project.

Unfortunately, the authors of UML provided us little, if any, input on what goes inside of a use-case. The official UML notation, as maintained by the Object Management Group, ventures no further than describing the diagrammatic components. This is little help to the practitioner. In defense of the UML authors, how we choose to capture the detail of the use-case is subjective to some degree. So it is crucial that a template be agreed upon for the project. The template presented here is what I consider to represent best practices after having been through many iterations of tweaking the process. The Unified Process also offers a use-case template, but it is a bit on the light side.

Other excellent use-case templates are available. A popular template is provided by Geri Schneider in her book *Applying Use Cases: A Practical Guide* (Addison-Wesley, 2001); another format is presented by Daryl Kulak and Eamonn Guiney in their excellent book *Use Cases: Requirements in Context* (Addison-Wesley, 2000).

We have assigned events to use-cases. Recall that use-cases are goal oriented and serve as containers for the interactions that they satisfy. Also recall from Ivar Jacobson that "the interactions should, as a group, be a self-contained unit that is an end in itself." To document the use-cases better, we employ a use-case template. The template is broken up into four sections that we apply to all use-cases.

Use-Case Template: Section 1

Section 1 of the template focuses on the high-level information gathered about the use-case. Most of what is contained here is informative and portrays the overall goal of the use-case. The "Trigger Event(s)" category is tied back to the event list produced in the previous section (see Table 4-1). It acts as a cross-check and balance to ensure that all the events are being assigned to a use-case. Trigger events are also important because if the team does not conduct an event generation exercise, at this point the events that kick the use-case into action should be identified.

Here's Section 1:

1. Use-Case Description Information

The following defines information that pertains to this particular use-case. Each piece of information is important in understanding the purpose behind the use-case.

1.1 Name

<Short, descriptive verb phrase naming the use-case.>

1.2 Goal

<A few sentences describing the ultimate goal of the use-case from the perspective of the user.>

1.3 Use-Case Team Leader/Members

<The person assigned the ultimate responsibility of completing the use-case, along with his/her team members.>

1.4 Precondition

<The state the system must be in before a use-case pathway can begin; may be specified further at the pathway level as well.>

1.5 Postcondition

<The state the system must be in after a use-case pathway has completed; may be specified further at the pathway level as well.>

1.6 Constraints/Issues/Risks

<Any items that may place a burden on the team that is describing the details of the use-case; may be specified further at the pathway level as well. It may be beneficial to assign each issue to a specific individual on the team.>

1.7 Trigger Event(s)

<The external event(s) or internal timer event(s) that stimulate a pathway through the use-case; may be defined at the individual pathway level as well.>

1.8 Primary Actor

<The key actor in the use-case. Typically this individual is the source of the event that stimulates the use-case pathway into action.>

1.9 Secondary Actor(s)

<Other actor(s) that play a part in the use-case.>

Use-Case Template: Section 2

Section 2 of the template focuses on the pathway names supported by the use-case. These are divided into three categories: primary, alternate, and exception. Note that these are only the names of the pathways, not the underlying tasks necessary to carry out the pathways. The details will be captured in the third section of the template.

Here's Section 2:

2. Use-Case Pathway Names

<The names of the pathways, serving only as a summary list to the subsequent detail of the pathways in Section 3.>

2.1 Primary Pathway (Happy Path)

<The most common pathway through the use-case. This path has no error conditions, with everything resulting in a positive outcome. More than one pathway may result in a positive outcome in the use-case; this one, however, occurs the most frequently.>

2.2 Alternate Pathway(s)

<Alternate pathway(s) through the use-case. Depending on the levels within the use-case, these may be just as detailed in content as the primary path.>

2.3 Exception Pathway(s)

<The primary exception(s) expected to occur in this use-case.>

Use-Case Template: Section 3

Section 3 of the template focuses on the detailed tasks, or steps, necessary to carry out a given use-case pathway. In many use-cases, completing this part of the template first for the primary (happy) pathway will satisfy a large majority of the effort in the use-case process. In addition, on the basis of the granularity of the project's use-cases, many of the alternate paths may be happy pathways as well—just not the most commonly occurring one.

Note that there will be a Section 3 of the template for each pathway. The "Business Rules" category of this section is a vital piece of information to the project at this point. Probably many business rules will have come up during the event generation exercise. If you recall, these rules are captured at that point, but here they are assigned ownership within a use-case.

Here's Section 3:

3. Use-Case Detail

<This section is completed for all pathways, regardless of the category of the pathway (i.e., primary, alternate, exception), which are usually documented in the three category groups. If the alternate and exception pathways are simple, they may refer back to a step within the main sequence of steps they are modifying or defining.>

3.1 Pathway Name

<The name of the pathway, as specified in Section 2.>

3.2 Trigger Event(s)

<Depending on the use-case granularity, events may be tied directly to a specific pathway.>

3.3 Main Sequence of Steps

<The detailed tasks, or steps, to be carried out in response to the event that started this pathway. The focus is on the what, *not the* how. *In addition, or in replacement, of the outline below, a UML activity diagram may be included. If a pathway step references a different use-case, the reference step is underscored. In addition, in the use-case diagram the included use-case is noted with the* <<includes>> *relationship.>*

Step	Description
<step #>	*<A one-sentence description of the step>*

optionally, for an <<*includes*>> relationship, underscore the step:

<step #>	*<A one-sentence description of the step>*

3.4 Variations (optional)

<Steps in an abbreviated alternate pathway that are documented as modifiers to one of the main-sequence steps. These may not be found for all pathways. If the alternative is not as simple as a variation to a previously defined main-sequence step, then provide a complete main sequence of steps for the alternative.>

Step	Description
<main sequence step #>	*<A one-sentence description of the step>*
<variation step #>	*<A one-sentence description of the step>*

3.5 Extensions (optional)
<Conditional steps that extend the use-case from a particular point. These may also be referred to as extension points. *They extend from a point within the main sequence of either the primary pathway or an alternate pathway within the use-case. The extension, if central to the overall understanding of the use-case, may be shown in the use-case diagram with an* <<extends>> *relationship to the extended use-case.>*

3.6 Business Rules (optional)
<Business rules that are pathway-specific. They may be global to the entire pathway, or they may be tied directly to a particular step within the pathway.>

3.7 Constraints/Issues/Risks (optional)
<Any items that may place a burden on the team that is describing the details of the use-case pathway. They may be global to the entire pathway, or they may be tied directly to a particular step within the pathway. It may be beneficial to assign each issue to a specific individual on the team.>

Use-Case Template: Section 4

Section 4 of the template focuses on more tactical elements of the use-case. Some practitioners label this section *Nonfunctional Requirements.* The Unified Process has a formal document to capture nonfunctional requirements, called the Supplementary Specification. Nevertheless, the items refer to many physical aspects of the use-case; some are even design focused. Some of you may be shuddering here because use-cases focus on the *what,* not the *how,* and are supposed to be technology neutral. However, I have found that to have a place to capture these aspects in the use-case process is very important for ensuring that the artifacts are not lost.

Here's Section 4:

4. Use-Case Tactical Information
<Information about the use-case that deals with scheduling, priorities, frequency, user interface, and performance topics. These items are usually not known early in the use-case Inception phase but are uncovered later, during Elaboration.>

4.1 Priority

<The priority of the use-case relative to others, indicating how this use-case will be packaged and delivered. It is possible to attach priorities to individual pathways.>

4.2 Performance Target(s)

<Specific performance expectations of the use-case. It is possible to attach these to individual pathways.>

4.3 Frequency

<The frequency at which the use-case pathways occur will eventually indicate potential transaction loadings in the system. This is usually stated in a base frequency such as x/day, x/hour, x/week. It is possible to attach this information to individual pathways.>

4.4 User Interface

<User interface issues or requirements for the use-case. This information will be described in detail later, during the Elaboration phase of the project.>

4.5 Location of Source

<If the application has a geographically dispersed nature, it is valuable to identify the relevant locations.>

Finding the Happy Path

We need to know more about the interactions stimulated by events and now assigned to use-cases. Specifically, we need more information about what happens when a use-case responds to an event. We obtain this information by identifying the steps within the pathway that a use-case must enforce in response to an event.

The use-case template is initially used to define the primary pathway, called the **happy path**, or more formally, the Basic Course of Events (BCOE). The happy path (or, as one of my seminar attendees called it, the "sunny-day path") is the most common pathway through the use-case. It usually depicts the perfect situation, in which nothing goes wrong. If a use-case has a lot of happy pathways, we arbitrarily pick one as *the* happy path. However, I contend that with a little work, we'll likely find that one of these potential happy paths either happens more often than the others or is more interesting to the project sponsor. In addition, the existence of more than one happy path for a use-case

TABLE 4-2 *Happy Paths for the Remulak Use-Cases*

Use-Case	Happy Path
Maintain Orders	A customer calls to inquire about an order's status.
Maintain Inventory	The products arrive at the warehouse with a copy of the purchase order attached.
Process Orders	A customer calls and orders a guitar and supplies, and pays with a credit card.
Shipping	An entire order is shipped from stock on hand to a customer.
Invoicing	An order is invoiced and sent to the customer, indicating that payment was satisfied via credit card billing.
Maintain Relationships	A customer calls to change his/her mailing address.
Decision Support	The manager requests a back-order status report.

may indicate that the use-case is too coarse-grained and may, in fact, be two use-cases.

We want to identify the happy path for every use-case we have at this point. Table 4-2 does this for Remulak Productions.

Finding the Alternate Pathways

Having identified the happy pathway for each use-case, we next tackle finding the alternate pathways. An **alternate pathway** is a pathway that is still considered a good pathway; it's just not the most heavily traveled one. Another term often used for this type of pathway is Alternate Course of Events (ACOE). Table 4-3 describes the alternate pathways for the Remulak Productions use-cases.

Finding the Exception Pathways

Things don't always go as planned. An **exception pathway** is intended to capture an "unhappy" pathway or, as one of my seminar attendees called it, the "crappy path." An **exception** is an error condition that is important enough to the application to capture. In some application domains (such as failure analysis), the error conditions are more important than the success-oriented happy path. Some use-cases, however, might not have exceptions that are interesting enough to capture; don't be concerned about those.

TABLE 4-3 *Alternate Pathways for the Remulak Use-Cases*

Use-Case	Alternate Pathways
Maintain Orders	A customer calls to change a product quantity for one order item on an order.
	A customer calls to cancel an order.
	A customer calls to add a new item to an order.
	A customer calls to delete an item from an order.
	A customer calls to change the shipping terms of an order.
	A customer buys an extended warranty on an item.
	A customer calls to change the billing method on an order.
Maintain Inventory	A product arrives at the warehouse with a purchase order that is attached but incomplete as to the products ordered.
	A product is ordered to replenish stock on hand.
	A product is ordered to fill a special order.
	A product is ordered to fill a back order.
	Products are accounted for through a physical inventory count.
Process Orders	A customer calls and orders a guitar and supplies, and uses a purchase order.
	A customer calls and orders a guitar and supplies, and uses the Remulak easy finance plan to pay.
	A customer calls and orders an organ, and pays with a credit card.
	A customer calls and orders an organ, and uses a purchase order.
Shipping	A partial order is shipped from stock on hand to a customer.
	An entire order is shipped to a customer sourced directly from a third-party supplier.
Invoicing	An overdue notice is sent to a customer for a past-due account.
	Subledger transactions are interfaced to the accounting system.
Maintain Relationships	A customer calls to change his/her default payment terms and payment method.
	A new customer is added to the system.
	A prospective customer is added to the system.
	A new supplier is added to the system.
	A supplier calls to change its billing address.
Decision Support	It is time to print the back-order report.

Table 4-4 lists the exceptions for Remulak Productions.

TABLE 4-4 *Exception Pathways for the Remulak Use-Cases*

Use-Case	Exception Pathways
Maintain Orders	A customer calls to cancel an order that can't be found in the system.
	A customer calls to add a warranty that is no longer valid for the time that the product has been owned.
	A customer calls to add to an order, and the product to be added can't be found in the system.
Maintain Inventory	A product arrives with no purchase order or bill of lading.
Process Orders	A customer calls to place an order using a credit card, and the card is invalid.
	A customer calls with a purchase order but has not been approved to use the purchase order method.
	A customer calls to place an order, and the desired items are not in stock.
Shipping	An order is ready to ship, and there is no shipping address.
Invoicing	None.
Maintain Relationships	None.
Decision Support	None.

Common Use-Case Pitfalls

If a use-case has only one pathway, the granularity of the use-case is much too fine. The effort has probably produced a functional decomposition of the domain. For example, while on a consulting assignment at an international banking organization, I was introduced to its use-case diagram. I was awestruck to learn that the firm had identified almost 300 use-cases. Closer examination revealed that simple pathways of a use-case had been elevated to the rank of use-case. After a little work, we ended up with 17 use-cases. Now that's more like it.

Having too many use-cases that lack what I call *functional entitlement* is a very common mistake that projects make. **Functional entitlement** means there is a clear mission defined for the use-case. After you decide on what you think the use-cases are, ask yourself the question, Is this use-case an expression of a key goal of the application *in the eyes of the user?* Quite often I find projects that have defined use-cases with names like *Validate customer number.* Is this a key goal of the application? I don't think so. What is it then? It is merely a step within the detail of a pathway through a use-case.

All that aside, I'm afraid the granularity of use-case definition is very subjective. I must also be up-front and say that unless the use-case assignments have gone way overboard, the resulting software solutions will probably be the same. Where I find the key difference in getting the granularity right is the subsequent breakdown of the increments and how the resulting project is managed. A use-case that is too coarse-grained, such as *Process Transaction,* would be hard to break down into perhaps multiple increments of development. There isn't clear functional entitlement for this use-case. There are lots of transactions to process, but what is the goal? A little more work might find that there are transactions that deal with placing orders, paying invoices, and ordering supplies. These, I suspect, are the use-cases; they clearly have functional entitlement.

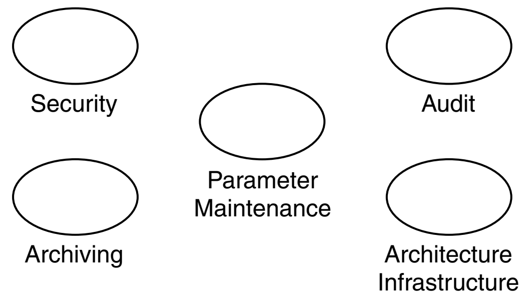
Shadow Use-Cases

Traditionally, use-cases have been viewed from the eyes of the business—that is, the user. In many application domains, however, some use-cases are never properly accounted for. These represent areas of functionality that meet all of the criteria of use-cases but that often have more meaning to the IT staff, which is their "user." The business sponsor might acknowledge them but often underestimates their impact on the application and the estimated time to completion. These use-cases often end up being budget busters.

I call these **shadow use-cases** because they are not given their due respect in most applications. Figure 4-4 shows the most common shadow use-cases found across all application domains: *Security, Audit, Parameter Maintenance, Archiving,* and *Architecture Infrastructure.*

Often both *Security* and *Audit* will show up in "includes" relationships to other use-cases (*Process Orders* "includes" *Security*). However, both are usually much more complicated than just logging onto a system (e.g., maintaining users and profiles, application functionality,

FIGURE 4-4 *Shadow use-cases*



value-level security, and field-level security). Both should be represented on the business view of the use-case diagram. I worked on one project whose security package alone contained 15 distinct classes and consumed about 500 person-hours to build. All this grew from an innocuous statement found in a use-case pathway: "Clerk is validated by the system."

Parameter Maintenance is a use-case that tends to consume too many resource cycles in development and also leads to project cost overruns. Items satisfied by this use-case are setting up and maintaining things like code tables and system parameters. These items always end up being hacked together at the last minute. Worse yet, they are maintained by a database administrator through hand-edited SQL statements (nothing against database administrators; I married one so they can't be all that bad!). Tell me that isn't a disaster waiting to happen.

Archiving also belongs on the business view of the use-case diagram, but it typically is given little consideration. Archiving is not as easy as just backing something up and deleting objects. For example, for a project that is complicated by effective dating schemes, what and when to archive something isn't all that straightforward.

Architecture Infrastructure relates to the plumbing that must be in place to allow the layers of the application to communicate—for example, the components that allow a user interface to communicate to the business rules of the application (Remote Method Invocation, RMI; Common Object Request Broker Architecture, CORBA; Component Object Model, COM+). The same applies to communication from the business rules to the data management component (JDBC, Java

Database Connectivity). This use-case should not be on the business view of the use-case diagram; rather it should be a use-case specifically for the IT staff.

Although some might argue that these shadow use-cases are simply "nonfunctional requirements" and not use-cases, I contend that they have all the properties of a use-case (i.e., functional entitlement, goal-orientedness, many pathways). Furthermore, if the project doesn't acknowledge them, the estimates for the project will be very skewed. Some of my colleagues come around to my way of thinking after they view the issues from the perspective of the actor, which in many cases is the IT department. In practice, if these items are brought to the surface and treated as first-class use-cases, they will be given the attention they demand and deserve, along with a representative portion of the project budget.

Describing Details of the Happy Path

Now the use-cases have been defined, along with their primary, alternate, and exception pathways. For the Inception phase, we have one more task to do regarding use-cases: describe the details of the happy path. We do this for the happy path (or any other pathway) by outlining the necessary steps to implement the pathway's functionality. As with the previous caveat regarding use-cases, use-case pathways derive from a *what* perspective, not a *how* perspective.

A detailed description is necessary so that we can better understand the complexity that might be involved in realizing the use-cases. We need this level of understanding also to estimate both the incremental release strategy and the accompanying time and cost components. The detailed steps of the happy pathway for the *Process Orders* use-case (*A customer calls and orders a guitar and supplies, and pays with a credit card*) are identified as follows:

1. Customer supplies customer number.
2. Customer is acknowledged as current.
3. For each product the customer desires:
   - 3.1 Product ID or description is requested.
   - 3.2 Product description is resolved with its ID if necessary.

  3.3 Quantity is requested.
  3.4 Item price is calculated.
4. Extended order total is calculated.
5. Tax is applied.
6. Shipping charges are applied.
7. Extended price is quoted to the customer.
8. Customer supplies credit card number.
9. Customer's credit card is validated.
10. Inventory is reduced.
11. Sale is finalized.

The detailed steps for the pathway are meant to be at a relatively high level. Notice that there are no specific references to technology in the form of, for example, "button clicks" or "scanning." The details we describe for the pathway will be determined by both the features identified in the charter and any assumptions made about the use-case. Section 4 of the use-case template is a place to document some of the user-centric requirements, as well as throughput requirements. However, it is best to hold off on the user interface portion until the project is closer to producing more design-oriented artifacts. Appendix D contains a complete listing of the detailed steps for the happy path of each use-case.

Most projects seem to work well with the outline format of the use-case details. One reason for this, as cognitive psychologists have known for years, may be that people remember things as outlines in their brains. The processes of driving to the store or fixing your car, for example, are series of predetermined outlines stored away for recall. Another option for documenting Section 3 of the use-case template is the UML activity diagram. We will explore this diagram in Chapter 7; for now, suffice it to say that it is as close to a flowchart as you can get.

The Completed Process Orders *Use-Case Template*

Now that we have completed most of the framework for the use-cases, a sample of a completed template is in order. What follows is the use-case template for *Process Orders.*

1. Use-Case Description Information

1.1 Name
Process Orders.

1.2 Goal
This use-case satisfies all of the goals of setting up a new order. This applies for both new and existing customers. All aspects of the order entry process are covered, from initial entry to ultimate pricing.

1.3 Use-Case Team Leader/Members
Rene Becnel (team lead), Stan Young, Todd, Klock, Jose Aponte.

1.4 Precondition
Order clerk has logged onto the system.

1.5 Postcondition
Order is placed, inventory is reduced.

1.6 Constraints/Issues/Risks
The new system might not be ready in time for the summer product promotions.

1.7 Trigger Event(s)
All events dealing with new and existing customers calling and placing orders.

1.8 Primary Actor
Order clerk.

1.9 Secondary Actor(s)
Customer.

2. Use-Case Pathway Names

2.1 Primary Pathway (Happy Path)
Customer calls and orders a guitar and supplies, and pays with a credit card.

2.2 Alternate Pathway(s)
- *Customer calls and orders a guitar and supplies, and uses a purchase order.*

- *Customer calls and orders a guitar and supplies, and uses the Remulak easy finance plan.*
- *Customer calls and orders an organ, and pays with a credit card.*
- *Customer calls and orders an organ, and pays with a purchase order.*

2.3 Exception Pathway(s)

- *Customer calls to place an order using a credit card, and the card is invalid.*
- *Customer calls with a purchase order but has not been approved to use the purchase order method.*
- *Customer calls to place an order, and the desired items are not in stock.*

3. Use-Case Detail

A Section 3 will exist for all use-case pathways that are detailed enough to warrant their own unique set of steps. In this case only the happy path and the payment variation are shown.

3.1 Pathway Name

Customer calls and orders a guitar and supplies, and pays with a credit card.

3.2 Trigger Event(s)

All events dealing with new and existing customers calling and placing an order.

3.3 Main Sequence of Steps

Step	Description
1	*Customer supplies customer number.*
2	*Customer is acknowledged as current.*
3	*For each product the customer desires:*
3.1	*- Product ID or description is requested.*
3.2	*- Product description is resolved with its ID if necessary.*
3.3	*- Quantity is requested.*
3.4	*- Item price is calculated.*
4	*Extended order total is calculated.*
5	*Tax is applied.*
6	*Shipping charges are applied.*

7	*Extended price is quoted to the customer.*
8	*Customer supplies credit card number.*
9	*Customer's credit card is validated.*
10	*Inventory is reduced.*
11	*Sale is finalized.*

3.4 Variations

Step	Description
8.1	*Customer may pay with purchase order or easy-finance plan.*

3.5 Extensions (optional)

None.

3.6 Business Rules (optional)

- *Customers may not order more than ten products at one time.*
- *Any sale over $50,000 requires supervisor approval.*
- *Any sale over $20,000 receives a five-percent discount.*

3.7 Constraints/Issues/Risks (optional)

Timeliness of the product is key to the next sales promotion.

4. Use-Case Tactical Information

4.1 Priority

Highest (#1).

4.2 Performance Target(s)

None indicated.

4.3 Frequency

- *Customer calls and orders a guitar and supplies, and pays with a credit card (800/day).*
- *Customer calls and orders a guitar and supplies, and uses a purchase order (120/day).*
- *Customer calls and orders a guitar and supplies, and uses the Remulak easy finance plan (25/day).*
- *Customer calls and orders an organ, and pays with a credit card (40/day).*
- *Customer calls and orders an organ, and pays with a purchase order (15/day).*

4.4 User Interface
This portion of the application will not use the Web as a form of entry because of the need for clerk assistance.

4.5 Location of Source
- *Newport Hills, Washington.*
- *Portland, Maine (in the future).*

The detailed use-case pathways are then specified for all of the individual pathways in each category (primary, alternate, and exception). Each section can be documented at different times. Actually, each section can be done as a mini-iteration. I think one of the most important sections is Use-Case Pathway Names (Section 3) because it gives the analyst and user a succinct look at what the pathways are anticipated to be without documenting all details of each pathway up front. Doing this for all use-cases is crucial for producing an overall estimate for the project. Some practitioners collapse Sections 2 and 3 together; this approach is fine as long as you can first identify the pathway names, and then just fill in the body detail as the use-case evolves.

The use-case detail is reflected in the Unified Process via the Software Requirements Specification (SRS). The nonfunctional elements are captured in the Supplementary Specification. As I pointed out earlier, I prefer to put the nonfunctional elements that relate directly to the use-case in Section 4 of the use-case template. Nonfunctional elements such as the database that will be used I place in the Supplementary Specification.

Preparing the Preliminary Architecture

We now know a lot more about Remulak Productions' requirements, as well as some of its technology needs and desires. The last artifact we need to begin, but not finish, before completing the Inception phase is the framework of the preliminary architecture. The project vision template contains some high-level architecture placeholders, but the primary resting place of the project architecture is the Software Architecture Document (SAD) of the Unified Process. Officially, the SAD isn't produced until the Elaboration phase, but I choose to begin fleshing it out with elements that we already know about when we're in

TABLE 4-5 *Preliminary Architecture of the Remulak Order-Processing Application*

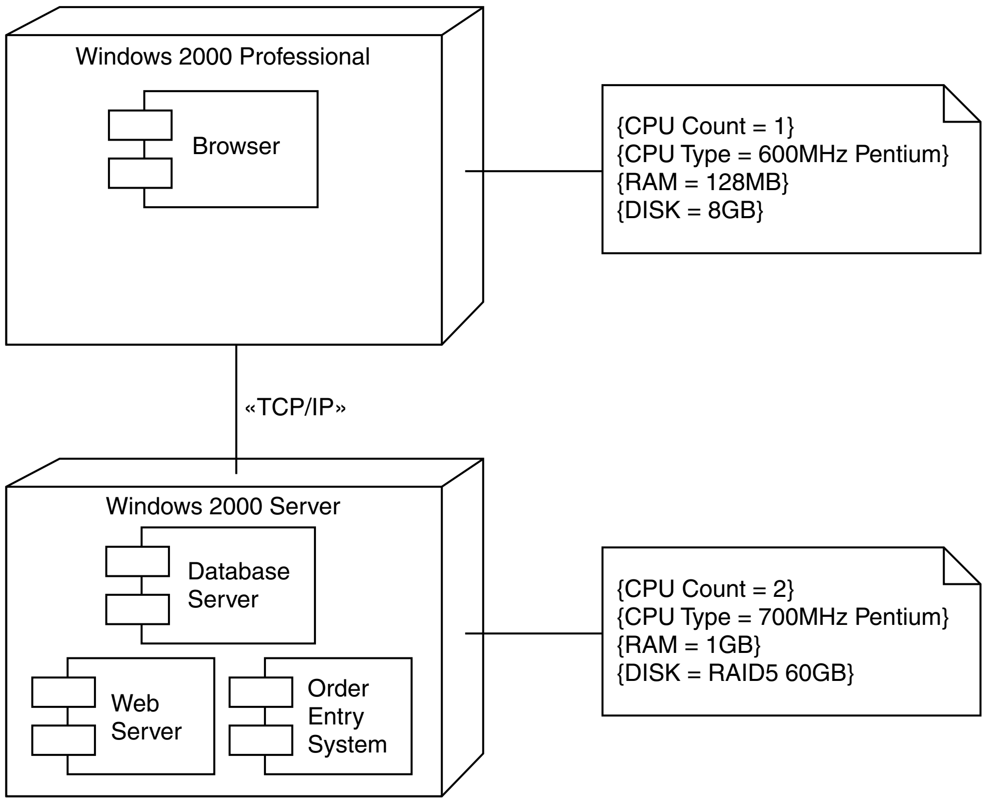
Component	Implementation
Hardware: Client	600MHz Pentium III–based clients with 128MB of RAM and an 8GB hard disk
Hardware: Server	Dual-CPU 700MHz Pentium III–based server with 1GB of RAM and RAID5 I/O subsystem supporting 60GB of storage
Software: Operating system (server)	Windows 2000 Server
Software: Operating system (client)	Windows 2000 Professional
Software: Application (client)	Any browser
Software: Database (server)	Microsoft SQL Server 2000 or Oracle 9i
Software: Transaction (server)	JavaBeans with JDBC transaction support, or Enterprise JavaBeans (where appropriate)
Software: Web (server)	Microsoft Internet Information Server, Apache Tomcat server, or commercial application server such as BEA WebLogic
Software: Web interface (server)	Servlets and JavaServer Pages
Software: Visual modeling	Rational Rose (Enterprise Edition), Together Control Center from TogetherSoft
Protocol: Network	TCP/IP
Protocol: Database	JDBC-ODBC Bridge

the Inception phase. Table 4-5 lists the technology components of the architecture.

To depict this architecture better, we use two different UML diagrams, which we combine to show both the software realization (component diagram) and hardware hosts (deployment diagram). Remember that this is a preliminary architecture. It is a snapshot based on what is known at this juncture of the project. Figure 4-5 shows the preliminary architecture model rendered in a hybrid UML component in the deployment diagram format.

For scalability, the architecture must allow various layers of the application to run on a processor other than the client's. In addition, we will want to take advantage of Enterprise JavaBeans to coordinate all of the various resources of the application. We will explore these and other technical considerations as the application evolves.

FIGURE 4-5 *Preliminary architecture with UML component and deployment diagrams*



Project Charter: Increments and Estimates

Increments

So far, our mantra has been to approach any application development effort with an eye toward incremental releases. Recall from Chapter 1 that risk is reduced exponentially if we tackle the project in stages. To align our terminology with the Unified Process, we will produce each of these increments by conducting many different iterations through the four phases (Inception, Elaboration, Construction, Transition). This approach will allow the project to focus first on the riskiest requirements. Toward that end, we propose the following release cycles as the staged increments for Remulak Productions:

Increment 1:

1.1 *Process Orders*

1.2 *Maintain Orders*

1.3 *Maintain Relationships* (customer pathways only)

1.4 *Architecture Infrastructure*

Increment 2:

2.1 *Maintain Inventory*

2.2 *Shipping*

2.3 *Invoicing*

2.4 *Maintain Relationships* (remaining pathways)

Increment 3:

3.1 *Decision Support*

3.2 *Security*

3.3 *Audit*

3.4 *Archiving*

Estimates: The Issues

For years, analysts and designers have been told in classroom settings never to provide estimates until all of the requirements are known. With the Unified Process, and any other process that is iterative and incremental, we use the learn-as-you-go approach. Our bases are somewhat covered because we have done a flyby of all of the events, use-cases, and pathways and we have detailed the happy pathways. However, we still don't know all of the supporting detail behind the requirements. Yet the project sponsors need an estimate; without it, they can't decide whether or not to give the go-ahead.

Before I stray into the semantics of estimating using use-cases, let me say a few words about estimates and the project sponsor. When I am consulting or teaching seminars, one question that always comes up is, "Well, this is all fine and good, but what do you do when your sponsor wants all the functionality you've specified for the same price, but in half the time?" My initial answer is that I feel sorry for the project team. At the same time, I usually sketch a little diagram and discuss its merits and meaning (see Figure 4-6).

I contend that all projects must face the reality of the equilateral triangle creed. Once the information technology group has estimated the project and calculated a delivery schedule, the ratios generated form the basis of the equilateral triangle. The edict is: *All sides must remain in the same proportion as to the initial ratios.* Quality is *never* negotiable.

This creed implies that if a sponsor wants more functionality, the time and cost factors will increase in proportion to the increase in functionality. Typically, however, the request of the project sponsor yields a picture of these factors that looks like Figure 4-7.

FIGURE 4-6 *Realistic time, cost, functionality, and quality relationships*

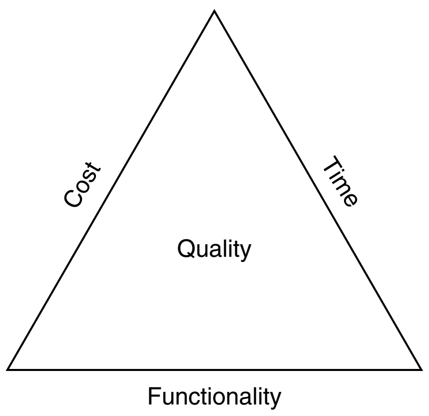
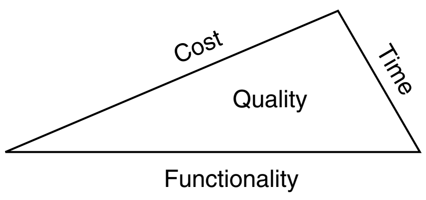


FIGURE 4-7 *Project sponsor's preferred time, cost, functionality, and quality relationships*



This situation is not feasible. The sponsors want the project done in half the time, but at the same cost and of course the same level of functionality. These types of no-win situations are worth walking away from. What ends up being sacrificed is quality. If the discussion about keeping time, cost, and functionality ratios equal doesn't help the project sponsor see the light, then I usually launch into a lecture pointing out that a building contractor would laugh in our face if we suggested such foolishness. And how about a plastic surgeon practicing his/her craft on an accident victim. What would the response be in that situation? Would we want either one of these professionals to sacrifice quality or to somehow rush the job while still attaining all the original goals?

The solution is to look at the problem and break it down into even more elemental pieces—that is, more increments. As mentioned in the preface and in Chapter 1, we must avoid risk if we are going to build software that meets the needs of a business but can stand the test of time. Working 100-hour workweeks is a perceived temporary solution that has absolutely no long-term benefit.

Estimates: The Process

Varying levels of success have been realized with structured approaches to project estimating. Estimating still is a combination of mystic art, the school of hard knocks, and plain luck. However, some very interesting research has been done at Rational Software by Gustav Karner (which he begain initially while at Objectory AB, which was later purchased by Rational Software). The result is a modification of work originally done by Allan Albrecht on estimating by using function point analysis. Appendix C provides an overview of that estimating technique, as well as how the estimates were reached for Remulak Productions.

Remulak Productions' deliverable will be realized by implementation of three different increments, staged as three different release cycles. This will enable Remulak to manage the risk of the project, as well as ease it into the new millennium without too much new-system shock. The estimates for each increment are as follows:

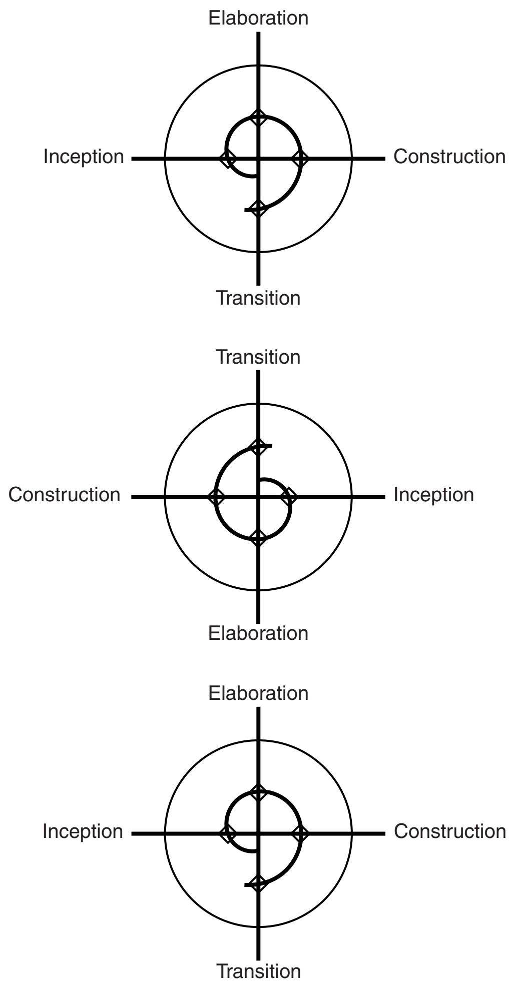
Increment 1: 670 person-hours

Increment 2: 950 person-hours

Increment 3: 950 person-hours

Figure 4-8 depicts the project with all of the increments in process. This figure does a good job of showing the iterative, incremental approach that we will take for the Remulak Productions application. The middle spiral is flipped to indicate that many increments or deliverables can be active at any one time, each in its own phase.

FIGURE 4-8 *Increments for the Remulak application*

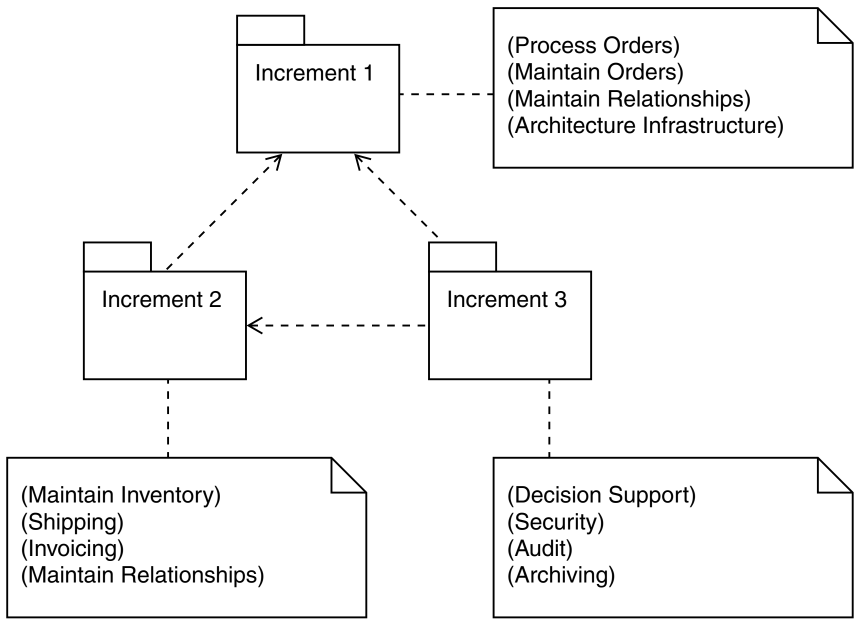


Increment 1:
- Process Orders
- Maintain Orders
- Maintain Relationships (customer paths)
- Architecture Infrastructure

Increment 2:
- Maintain Inventory
- Shipping
- Invoicing
- Maintain Relationships (remaining paths)

Increment 3:
- Decision Support
- Security
- Audit
- Archiving

FIGURE 4-9 *Remulak package diagram*

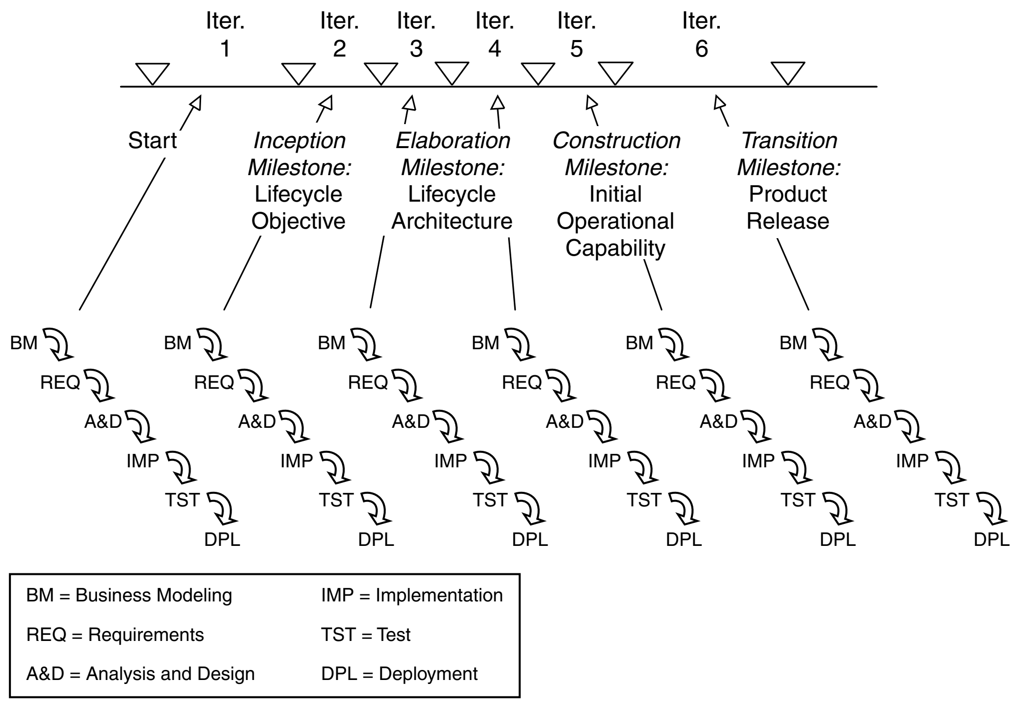


The UML package diagram can depict the same thing, while hiding much of the detail. Figure 4-9 is a package diagram reflecting the incremental deliverables.

The Inception phase is now complete. The road is laid out for us clearly and concisely. The diagrams produced, together with the project vision, are collectively called the **requirements model**, and we have reached the Lifecycle Objective milestone in the Unified Process. The next step is to create two project plans: One will be a high-level phase plan, the other a detailed plan for the first iteration in the Elaboration phase, where we will tackle Increment 1 of the Remulak Productions order entry application. The bases for these project plans are drawn from the Unified Process project plan templates that can be found in Appendix A.

The remainder of this book will explore the details of the first increment for Remulak Productions. The remaining two increments are not presented in this book, but they would be developed in the same fashion.

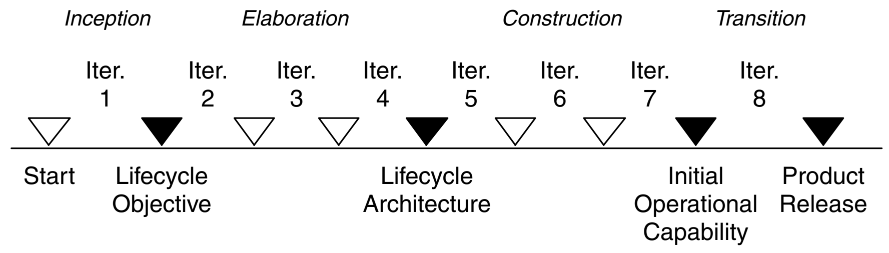
FIGURE 4-10 *Workflows and phases in the Unified Process*


Iter. 1
Iter. 2
Iter. 3
Iter. 4
Iter. 5
Iter. 6
Start
Inception Milestone: Lifecycle Objective
Elaboration Milestone: Lifecycle Architecture
Construction Milestone: Initial Operational Capability
Transition Milestone: Product Release
BM
REQ
A&D
IMP
TST
DPL
BM = Business Modeling
REQ = Requirements
A&D = Analysis and Design
IMP = Implementation
TST = Test
DPL = Deployment

Let's now look at how the project-planning process is laid out in the Unified Process. Figure 4-10 shows how each iteration cuts vertically through all the workflows offered in the Unified Process (Business Modeling, Requirements, and so on). Just remember that as the project moves farther to the right in its lifecycle, the project plan tasks will shift more toward design and construction activities.

Here the timeline shows multiple iterations: one in Inception, two in Elaboration, two in Construction, and one in Transition. For Remulak, because there are three packages or increments of delivery, the proposition is to stretch both the Elaboration and Construction phases into three iterations each. This approach maps nicely to the packages and spreads out the risk. Figure 4-11 shows the phase timeline with the proper number of iterations. So the increment packages shown in Figure 4-9 will map to Iterations 2, 3, and 4 for the Elaboration phase of the

FIGURE 4-11 *Remulak iteration/package mappings*



application. The same applies for Iterations 5, 6, and 7 in the Construction phase.

There we have the outline of our phase plan and the input necessary to create a detailed project plan for the first Elaboration iteration.

Checkpoint

Where We've Been

- Use-cases are technology neutral and applicable to any process or methodology used by a project team.
- Each use-case is a behaviorally related sequence of interactions performed by an actor in a dialog with the system to provide some measurable value to the actor. Use-cases are goal oriented and are significant to a business.
- The primary pathway, or Basic Course of Events, is considered the most common pathway through a use-case. It is also called the happy pathway.
- Alternate pathways are also good pathways, but they are not traveled as often.
- A detailed description of the pathway chronicles the steps that must be undertaken to satisfy the originating event. The steps should avoid, if possible, reference to *how* the event is being performed.

- All of the pieces of documentation produced up to this point, including the UML diagrams, are collectively called the requirements model.

Where We're Going Next

In the next chapter we:

- Identify more detail about the use-cases in the first iteration of the Elaboration phase.
- Explore how to derive classes from the use-cases.
- Explore how to derive associations.
- Review various UML diagramming constructs for various types of associations (generalization, composition, and aggregation) and how they relate to the Remulak Productions solution.
- Create a complete class diagram for Remulak Productions.
- Begin to identify attributes and operations for Remulak Productions' classes.