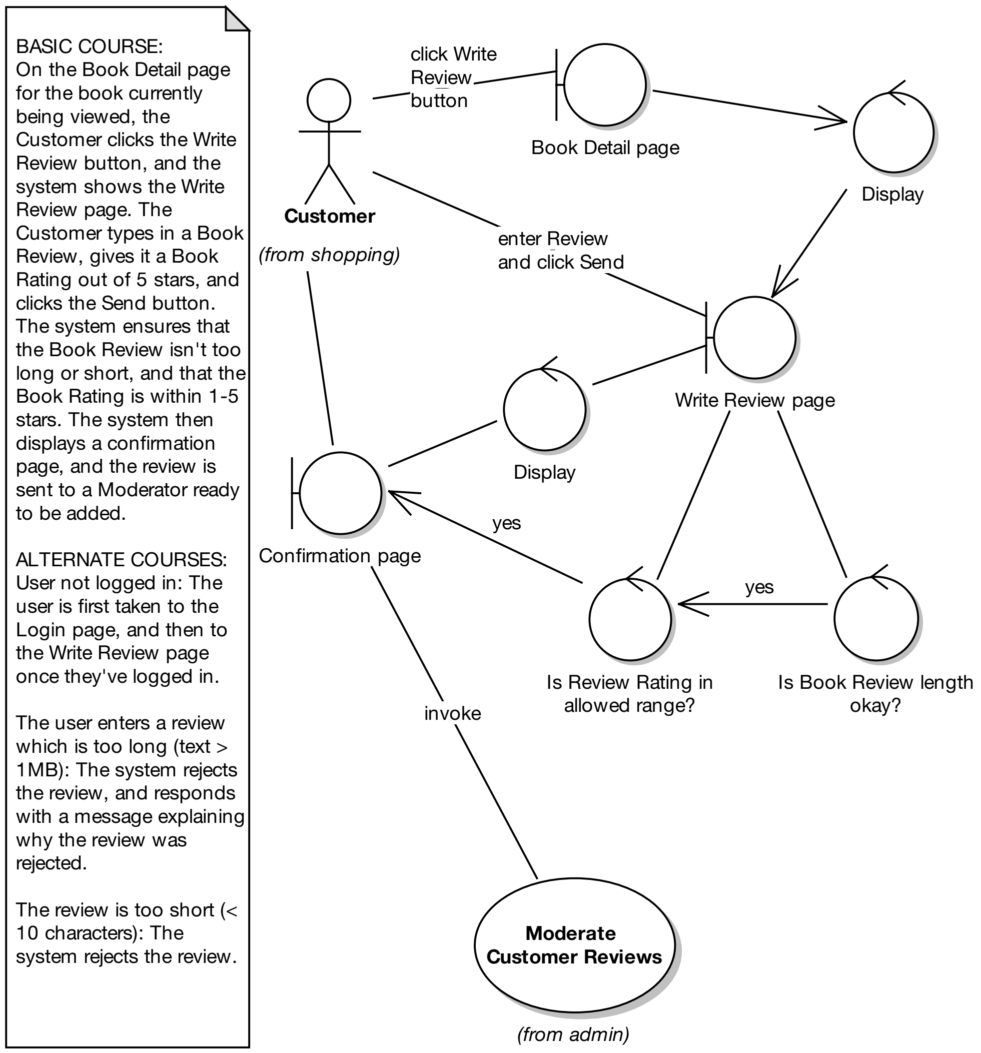
In Figure 5-15, we've corrected the diagram and also walked through the remainder of the basic course text. Notice that we've also corrected the use case text so that it doesn't just hint at a Book Detail screen, but instead refers to it explicitly.


BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button, and the system shows the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation page, and the review is sent to a Moderator ready to be added.

ALTERNATE COURSES:
User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.

Customer
(from shopping)
click Write Review button
Book Detail page
Display
enter Review and click Send
Write Review page
Display
Confirmation page
yes
yes
Is Review Rating in allowed range?
Is Book Review length okay?
invoke
Moderate Customer Reviews
(from admin)

Figure 5-15. *Step 4: The remainder of the basic course text in graphic form*

The text shown in the new part of the diagram is

The system then displays a confirmation screen, and the review is sent to a Moderator, ready to be added.

(As you'll discover later, this text also needs work!)

We've included a controller called Display, which "controls" the display of the Confirmation Screen boundary. We don't attempt to draw the review being sent to the Moderator, because this is handled by a separate use case, *Moderate Customer Reviews* (from the admin package). Instead, we simply drag that use case onto the diagram as a link, and indicate that we'll invoke it directly.

You'd be forgiven for thinking that we've finished at this stage; the robustness diagram looks pretty complete. However, we still haven't modeled the alternate courses, and this is often where robustness diagrams provide the most value.

Let's take a look at the first alternate course:

> ***User not logged in****: The user is first taken to the Login screen and then to the Write Review screen once he is logged in.*

To model this course, we'll add a new controller: "Is user logged in?" If the user isn't logged in, we'll invoke the *Login* use case; otherwise, control passes to the Display controller as it did previously. Because the use case text specifies that the Write Review screen is displayed once the user has logged in, we also need to show this on the diagram, so there should be a line between the *Login* use case and the Display controller. As we do this on the diagram, we should also modify the use case to match:

> ***User not logged in****: Invoke* Login. *Then display the Write Review screen.*

Exercise Which object should the system ask whether the user is logged in? Currently there isn't a `Customer Session` class, but it looks as if we're going to need one. (If you're wondering what "object discovery" means, this is it. We just discovered that we're missing an object.) Looking at Figure 5-15, where should this go on the diagram? You'd also need to update the use case text to refer to `Customer Session`. Try doing this first, and then compare the result with the updated diagram in Figure 5-16.

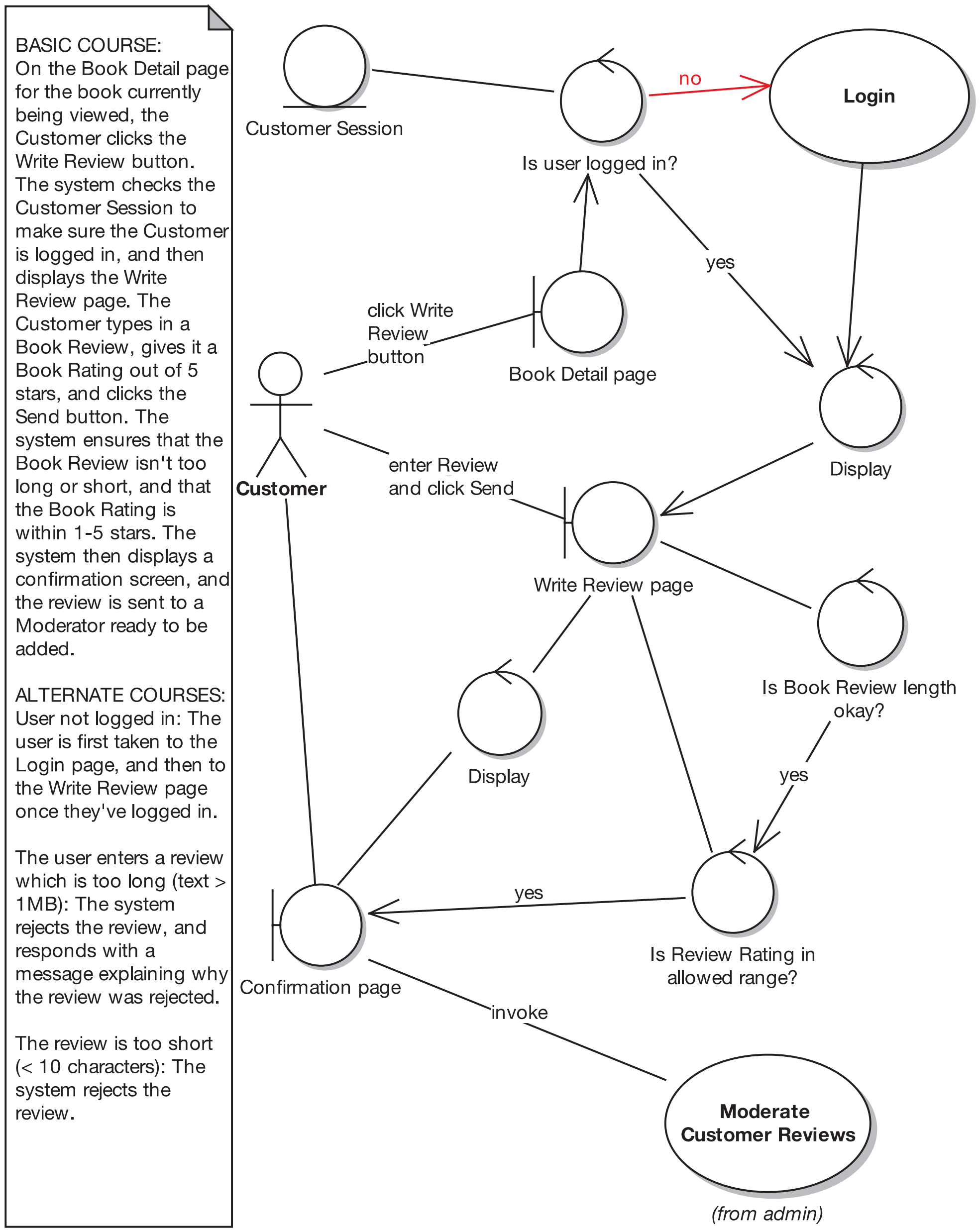
The updated diagram and use case text are shown in Figure 5-16. Notice that we've shaded the arrow pointing to the *Login* use case in red, to indicate that it's part of an alternate course. While not essential, this is a useful visual aid. If you print out the diagrams, you can use highlighter pens to do this part.

In Figure 5-16, the new `Customer Session` object has been added to the robustness diagram (note that you'll also need to update the domain model anytime you identify a new domain object). The use case text shown to the left of the diagram has also been updated to refer to the `Customer Session`: "The system checks the Customer Session to make sure the Customer is logged in."

Finally, let's add the last two alternate courses:

> ***The user enters a review that is too long (text > 1MB)****: The system rejects the review and responds with a message explaining why the review was rejected.*

> ***The review is too short (< 10 characters)****: The system rejects the review and displays an error message.*


BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button. The system checks the Customer Session to make sure the Customer is logged in, and then displays the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation screen, and the review is sent to a Moderator ready to be added.

ALTERNATE COURSES:
User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

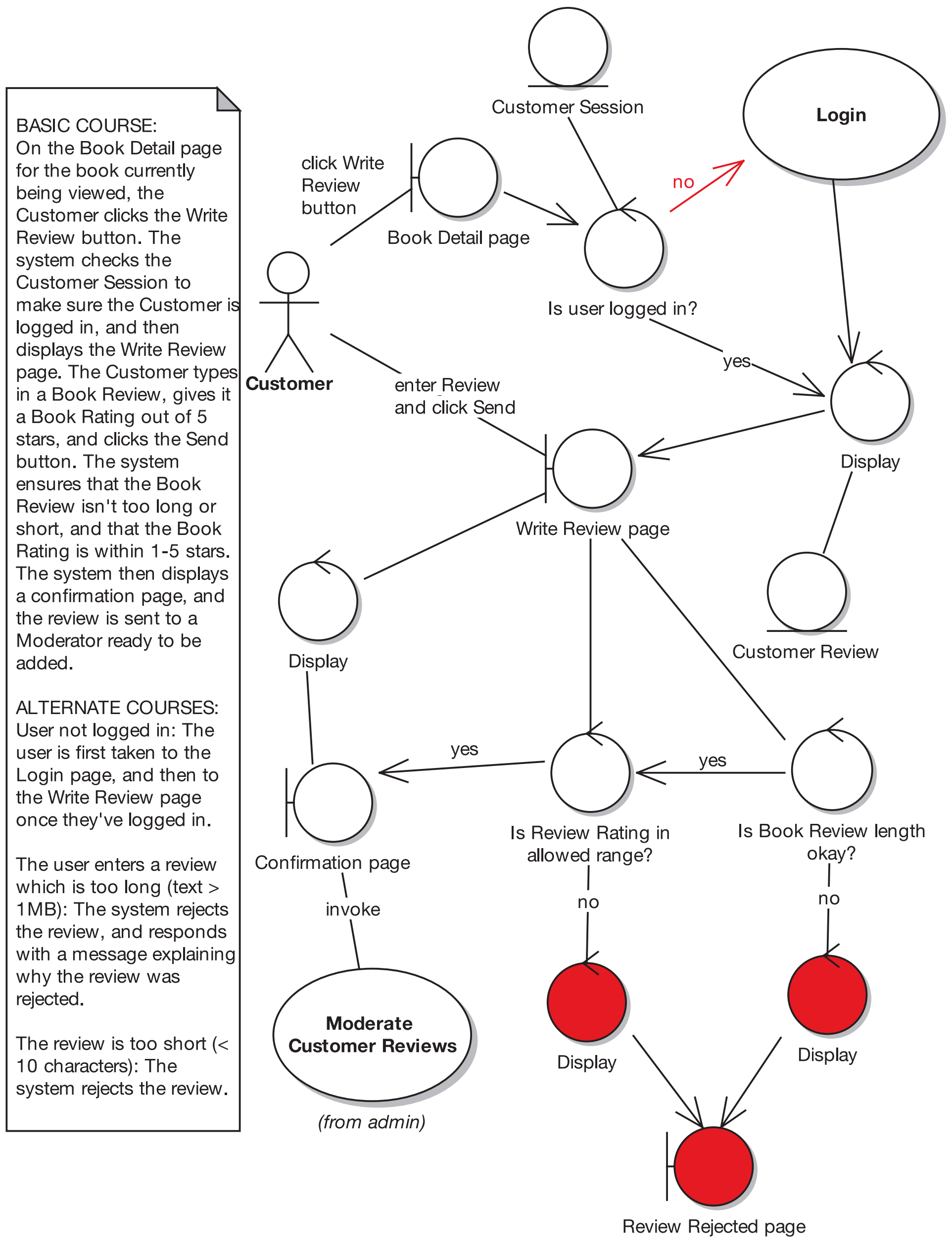
The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.

Figure 5-16. *Step 5: The first alternate course has been added.*

Note the two implied requirements: the book review length shall not exceed 1MB, and the book review length shall not be fewer than ten characters.

The updated diagram is shown in Figure 5-17 (again, the objects for the alternate courses are shown in red).


BASIC COURSE:
On the Book Detail page for the book currently being viewed, the Customer clicks the Write Review button. The system checks the Customer Session to make sure the Customer is logged in, and then displays the Write Review page. The Customer types in a Book Review, gives it a Book Rating out of 5 stars, and clicks the Send button. The system ensures that the Book Review isn't too long or short, and that the Book Rating is within 1-5 stars. The system then displays a confirmation page, and the review is sent to a Moderator ready to be added.

ALTERNATE COURSES:
User not logged in: The user is first taken to the Login page, and then to the Write Review page once they've logged in.

The user enters a review which is too long (text > 1MB): The system rejects the review, and responds with a message explaining why the review was rejected.

The review is too short (< 10 characters): The system rejects the review.

Figure 5-17. *Step 6: Robustness diagram with the last two alternate courses added*

■**Exercise** In Figure 5-17, following the “enter Review and click Send” arrow, it’s difficult to say which controller we go to after Write Review page. How could the diagram be improved to make this clearer? We reveal the answer during the PDR in Chapter 6.

■**Exercise** In Figure 5-17, the diagram invokes the *Moderate Customer Reviews* use case, but this doesn't match up with the text on the left. What should be done to make the diagram and the text match up? (Again, check the PDR in Chapter 6 for the answer.)

This diagram is now "functionally" complete, and you could reasonably happily draw a sequence diagram from it. But there are still a few ambiguities that should be ironed out, a couple of which are mentioned in the Exercise elements here. Ironing out these ambiguities will make our lives that much easier when we draw the sequence diagram.

This fresh in from the Department of Cliffhanger Endings: We'll cover those issues (and improve this diagram some more) in the Preliminary Design Review (PDR) in Chapter 6.

Updating Your Domain (Static) Model

While you're drawing robustness diagrams, it's a good idea to also be updating the domain model incrementally, as you go along. You'll almost inevitably discover new domain classes. You'll also identify attributes to be added to classes. This all needs to go on the domain model (aka analysis-level static model), and it's best do it ***now***, as soon as you identify these changes or new details, before they're forgotten about.

The new attributes may be discovered from the use case text, or from the UI prototypes, or even from the functional requirements.

■**Caution** Try not to get sidetracked by assigning **operations** to your classes just yet—that's a detailed design activity. If you spend time doing that now, you'll probably just need to redo it later.

The feedback cycle between the robustness model and the static model is shown in Figure 5-18.

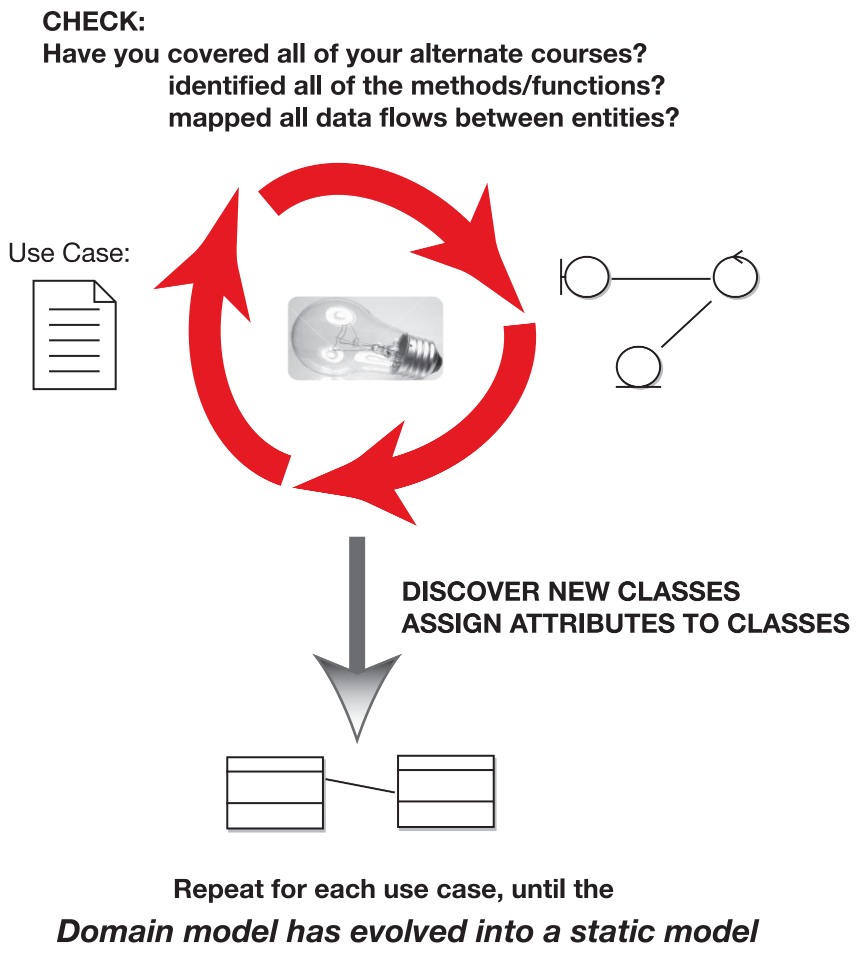

CHECK:
Have you covered all of your alternate courses?
identified all of the methods/functions?
mapped all data flows between entities?
Use Case:
DISCOVER NEW CLASSES
ASSIGN ATTRIBUTES TO CLASSES
Repeat for each use case, until the
Domain model has evolved into a static model

Figure 5-18. *Robustness model/static model feedback loop*

Figure 5-19 shows the updated static model for the Internet Bookstore, following robustness analysis for the *Show Book Details* and *Write Customer Review* use cases. The added or updated classes are shown in red.

The changes that we've made to this diagram are as follows.

After doing robustness analysis for the *Show Book Details* use case (see Figure 5-11), we

- Added attributes to Book: title and synopsis (these details weren't in the use case text, but were found from looking at the screen UI mock-ups)

After doing robustness analysis for the *Write Customer Review* use case (see Figure 5-17), we

- Added `CustomerSession`, gave it an attribute called `loggedIn` (a Boolean), and linked it to `CustomerAccount` (which will later, in Chapter 8, become simply `Customer`)
- Deleted `CustomerRating`, because it turned into an attribute on `BookReview`
- Added new attributes to `BookReview` that were mentioned in the use case text

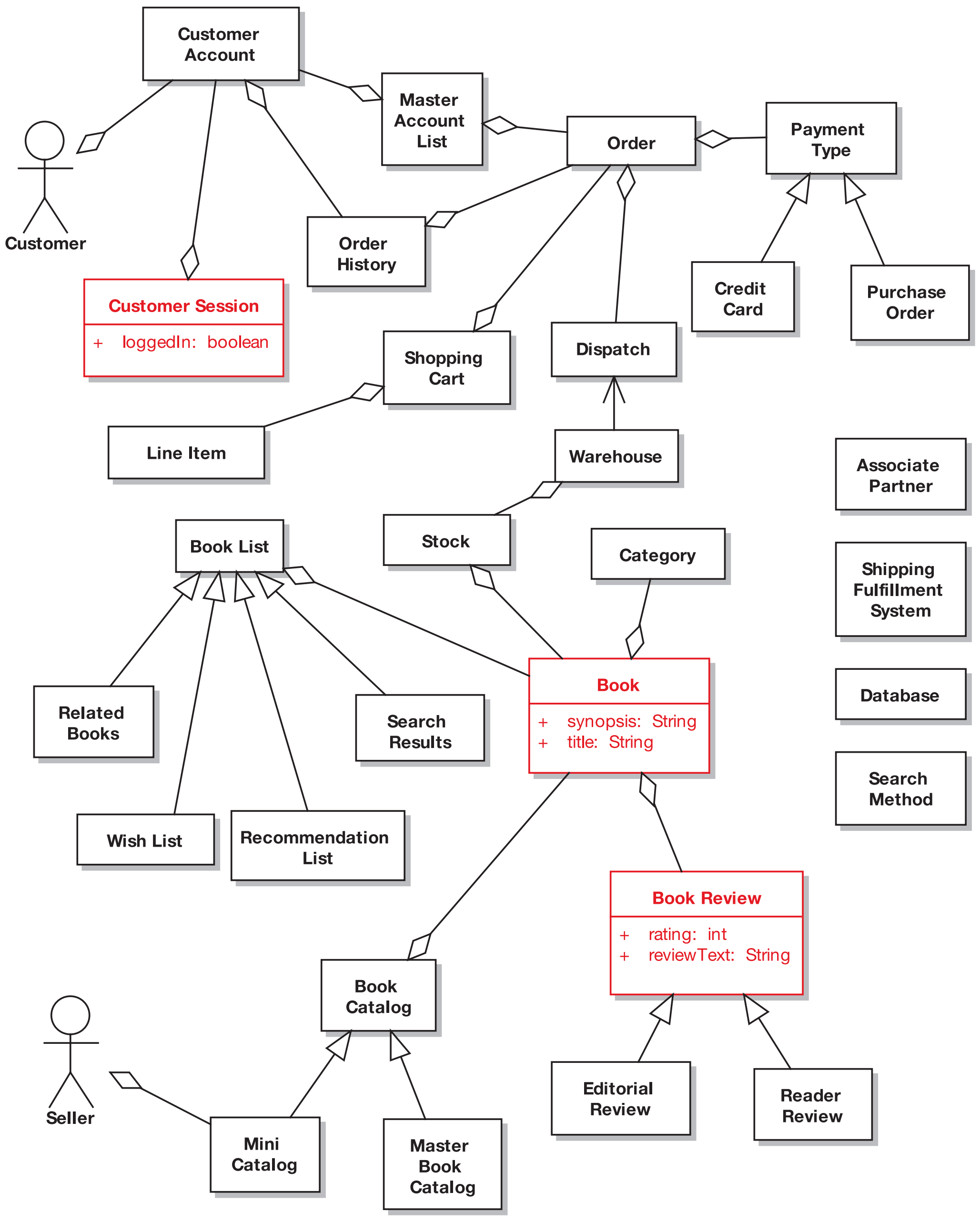


Figure 5-19. *Static model for the Internet Bookstore, after robustness analysis for two use cases*

There's one last step before you finish robustness analysis. Once all of your robustness diagrams are drawn, you must **finish updating the analysis-level class diagram**. Take a sweep through all of the robustness diagrams that you've drawn for this release, and make sure that all of the relevant details have been fed back into the static model.

Robustness Analysis in Practice

The following exercises, taken from the preliminary design activities for the Internet Bookstore, are designed to test your ability to spot the most common mistakes that people make during robustness analysis.

Exercises

Each of the diagrams in Figures 5-20 to 5-23 contains one or more typical modeling errors. For each one, try to figure out the errors and then draw the corrected diagram. The answers are in the next section.

Exercise 5-1

Figure 5-20 shows an excerpt from a robustness diagram for the *Create New Customer Account* use case. It violates one of the rules of robustness analysis that we described earlier in this chapter—but which one?

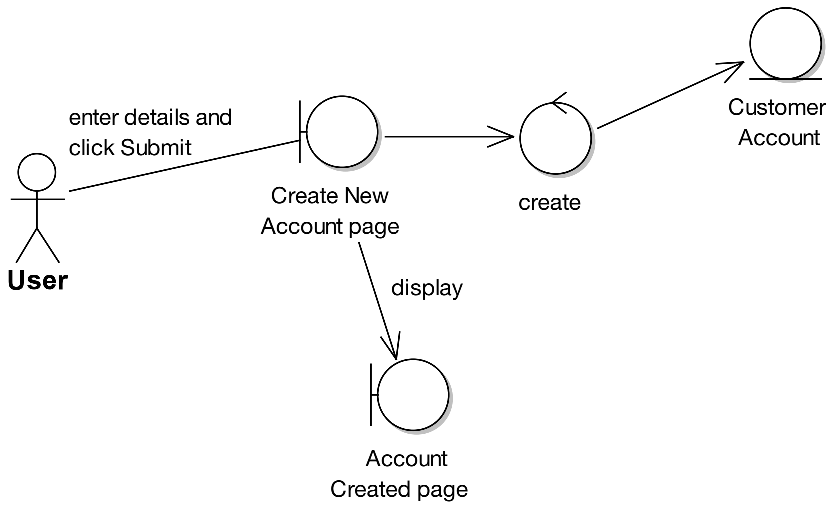
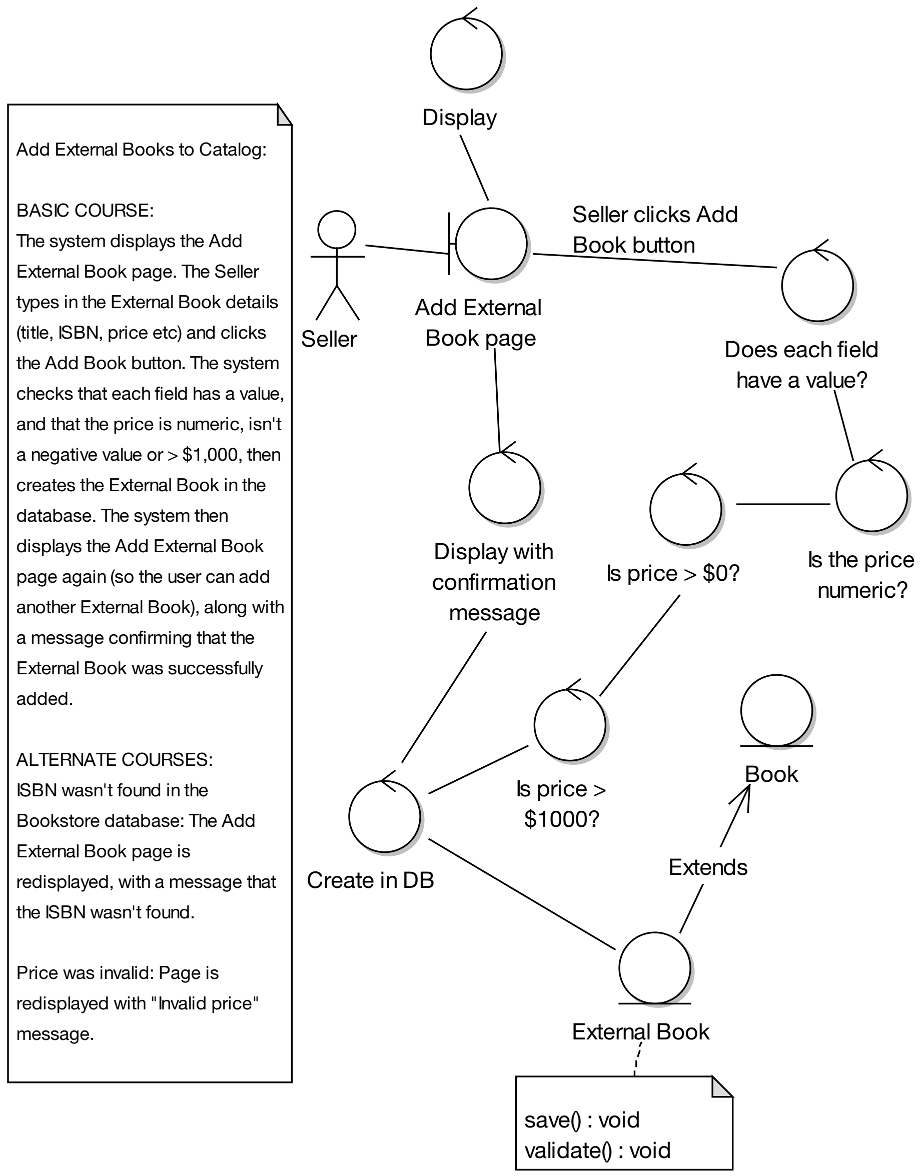


Figure 5-20. *Excerpt from a robustness diagram showing an invalid relationship*

Exercise 5-2

Figure 5-21 shows a robustness diagram for the *Add External Books to Catalog* use case (in which bookseller partners may add their own titles to the Internet Bookstore website). It contains a couple of errors to do with the sort of detail you should put on a robustness diagram (Hint: Look at External Book) and an error where the alternate course doesn't have any relation to the events in the basic course. Speaking of alternate courses, there's also one other error related to alternate courses.



Add External Books to Catalog:

BASIC COURSE:
The system displays the Add External Book page. The Seller types in the External Book details (title, ISBN, price etc) and clicks the Add Book button. The system checks that each field has a value, and that the price is numeric, isn't a negative value or > $1,000, then creates the External Book in the database. The system then displays the Add External Book page again (so the user can add another External Book), along with a message confirming that the External Book was successfully added.

ALTERNATE COURSES:
ISBN wasn't found in the Bookstore database: The Add External Book page is redisplayed, with a message that the ISBN wasn't found.

Price was invalid: Page is redisplayed with "Invalid price" message.

Figure 5-21. *Robustness diagram for the* Add External Books to Catalog *use case, with four modeling errors*

Exercise 5-3

The robustness diagram excerpt in Figure 5-22 shows at least five modeling errors, including at least one in the use case text, in the last sentence of the basic course, and two where the text and the diagram don't match up. Have fun finding them all!

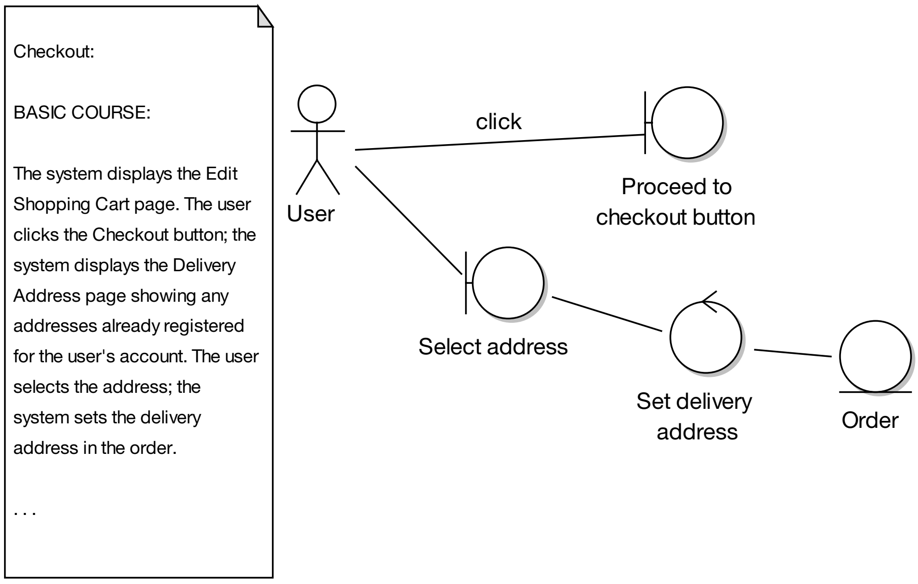
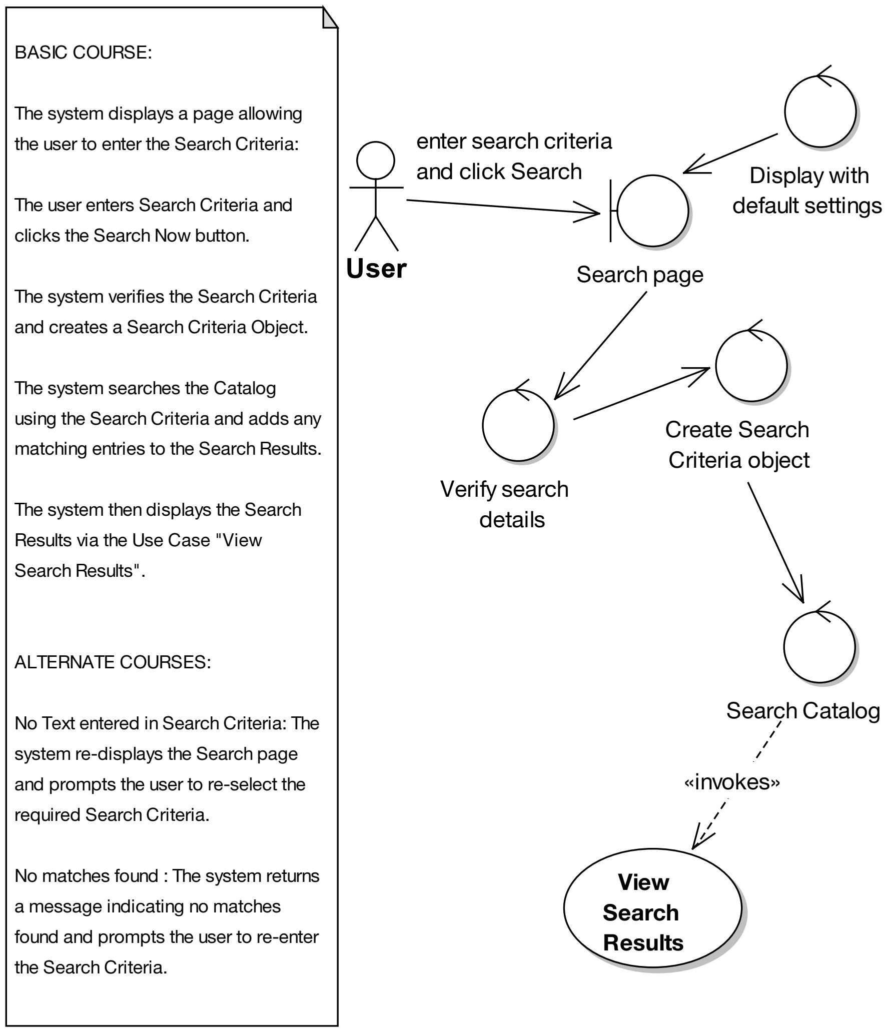

Checkout:
BASIC COURSE:
The system displays the Edit Shopping Cart page. The user clicks the Checkout button; the system displays the Delivery Address page showing any addresses already registered for the user's account. The user selects the address; the system sets the delivery address in the order.
. . .
User
click
Proceed to checkout button
Select address
Set delivery address
Order

Figure 5-22. *Excerpt from a robustness diagram containing five errors*

Exercise 5-4

The robustness diagram in Figure 5-23 is for the *Search for Books* use case. The diagram shows a total of eight modeling errors, including one in the first sentence of the use case text (Hint: Page name) and another in the fourth paragraph of the use case text. See how many errors you can find!



BASIC COURSE:

The system displays a page allowing the user to enter the Search Criteria:

The user enters Search Criteria and clicks the Search Now button.

The system verifies the Search Criteria and creates a Search Criteria Object.

The system searches the Catalog using the Search Criteria and adds any matching entries to the Search Results.

The system then displays the Search Results via the Use Case "View Search Results".

ALTERNATE COURSES:

No Text entered in Search Criteria: The system re-displays the Search page and prompts the user to re-select the required Search Criteria.

No matches found : The system returns a message indicating no matches found and prompts the user to re-enter the Search Criteria.

User
enter search criteria and click Search
Search page
Display with default settings
Verify search details
Create Search Criteria object
Search Catalog
«invokes»
View Search Results

Figure 5-23. *Robustness diagram for the* Search for Books *use case showing eight errors*

Exercise Solutions

Following are the solutions to the exercises.

Exercise 5-1 Solution: Noun-Noun Relationship

Figure 5-24 highlights the part of the diagram that violates a robustness diagramming relationship rule. The "Create New Account page" boundary object is talking to the "Account Created page" boundary object. These are both "nouns," and a noun-noun relationship isn't allowed. There must be a controller (a verb) between them, so that the relationship is noun-verb-noun.

Looking at Figure 5-24, the arrow is already labeled "display," so it makes sense simply to create a Display controller. Figure 5-25 shows the corrected diagram.

■ **Note** See Figures 5-8 and 5-9 for all of the possible valid and invalid relationships that you can show on a robustness diagram.

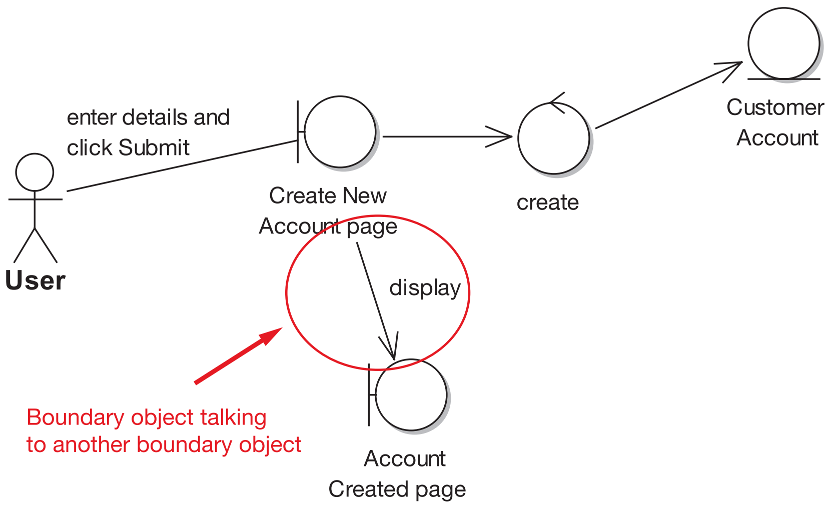

enter details and click Submit
Create New Account page
create
Customer Account
User
display
Boundary object talking to another boundary object
Account Created page

Figure 5-24. *The robustness diagram excerpt from Exercise 5-1, with the error highlighted*

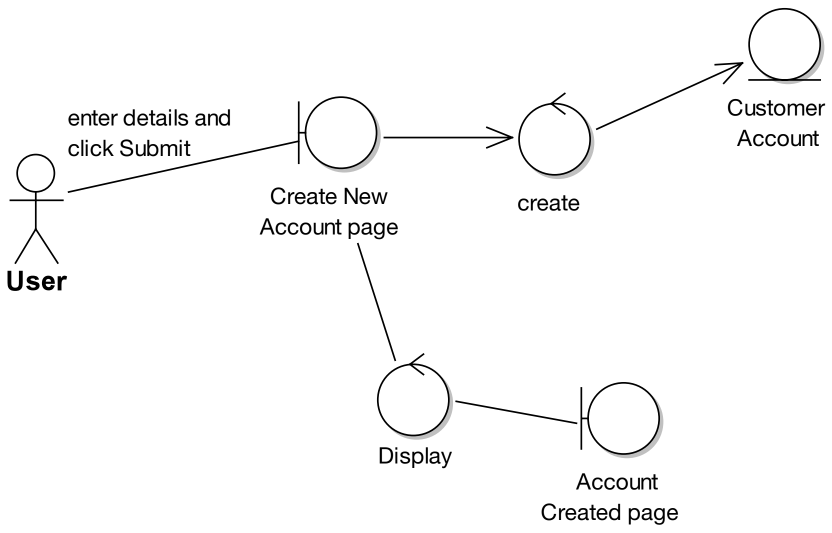


Figure 5-25. *The corrected robustness diagram excerpt for Exercise 5-1*

Exercise 5-2 Solution: Allocating Behavior

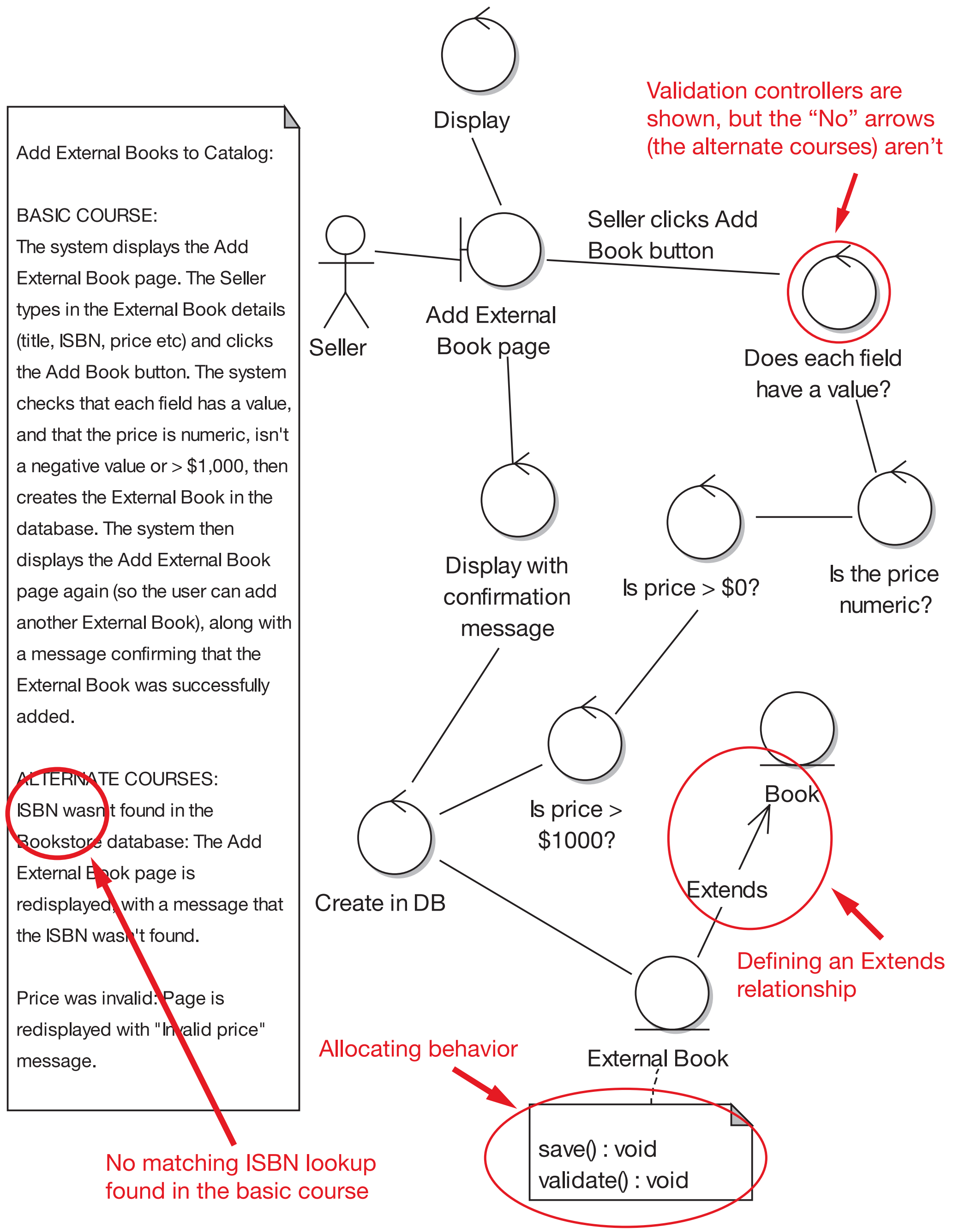
Figure 5-26 highlights the errors in Figure 5-21.

Let's start at the top of the diagram and work our way down. The diagram contains the validation controllers to check the incoming form values for the Add External Book page, but it doesn't show what happens if any of the validation checks fail. The solution is actually pretty straightforward—we just need a "Display with error message" controller pointing back to the Add External Book page—but it still needs to be shown on the diagram, so that the extra processing doesn't get forgotten about.

The next error, over on the left of Figure 5-26, is that there's an alternate course called "ISBN wasn't found in the Bookstore database," but there's no matching text in the basic course for this error condition to ever arise. The fact that it appeared in the alternate courses shows that the error condition was on the use case author's mind, but it needs to be stated explicitly so that the designers know that they're meant to deal with it.

Over to the right, Figure 5-26 shows an extends relationship between External Book and its parent Book. While this might be a valid relationship, the robustness diagram definitely isn't the right place to capture this sort of information. Extends relationships belong on the domain model diagram (and later, the class diagrams).

Finally, there's a note attached to External Book showing two methods (`save()` and `validate()`). Again, the robustness diagram just isn't the right place to allocate operations to classes; this information should go on the class diagrams and be captured during sequence diagramming (see Chapter 8).


Display
Validation controllers are shown, but the "No" arrows (the alternate courses) aren't
Seller clicks Add Book button
Seller
Add External Book page
Does each field have a value?
Display with confirmation message
Is price > $0?
Is the price numeric?
Is price > $1000?
Book
Create in DB
Extends
Defining an Extends relationship
Allocating behavior
External Book
save() : void
validate() : void
No matching ISBN lookup found in the basic course

Add External Books to Catalog:

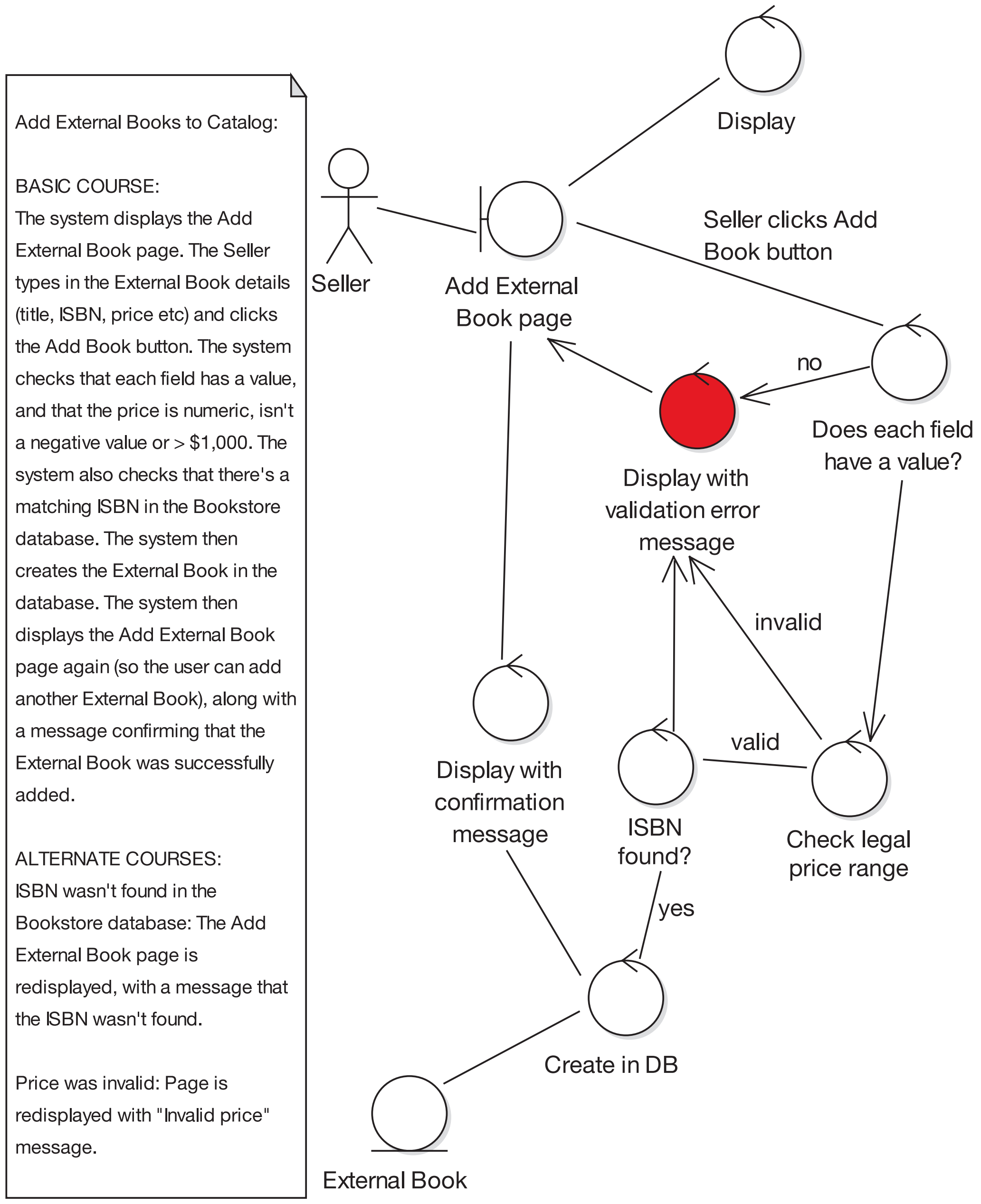
BASIC COURSE:
The system displays the Add External Book page. The Seller types in the External Book details (title, ISBN, price etc) and clicks the Add Book button. The system checks that each field has a value, and that the price is numeric, isn't a negative value or > $1,000, then creates the External Book in the database. The system then displays the Add External Book page again (so the user can add another External Book), along with a message confirming that the External Book was successfully added.

ALTERNATE COURSES:
ISBN wasn't found in the Bookstore database: The Add External Book page is redisplayed, with a message that the ISBN wasn't found.

Price was invalid: Page is redisplayed with "Invalid price" message.

Figure 5-26. *The robustness diagram from Exercise 5-2, with errors highlighted*

Figure 5-27 shows the corrected diagram. In this version, we've also collapsed three controllers into one and called the new controller `Check legal price range`.



Add External Books to Catalog:

BASIC COURSE:
The system displays the Add External Book page. The Seller types in the External Book details (title, ISBN, price etc) and clicks the Add Book button. The system checks that each field has a value, and that the price is numeric, isn't a negative value or > $1,000. The system also checks that there's a matching ISBN in the Bookstore database. The system then creates the External Book in the database. The system then displays the Add External Book page again (so the user can add another External Book), along with a message confirming that the External Book was successfully added.

ALTERNATE COURSES:
ISBN wasn't found in the Bookstore database: The Add External Book page is redisplayed, with a message that the ISBN wasn't found.

Price was invalid: Page is redisplayed with "Invalid price" message.

Display
Seller
Add External Book page
Seller clicks Add Book button
no
Does each field have a value?
Display with validation error message
invalid
valid
Display with confirmation message
ISBN found?
Check legal price range
yes
Create in DB
External Book

Figure 5-27. *The corrected robustness diagram from Exercise 5-2*

Exercise 5-3 Solution: Diagram Doesn't Match Up with the Description

Figure 5-28 highlights the errors in Figure 5-22. The first error in the example is that the text starts at an earlier point in time than the diagram (a common mistake). An easy way to spot this type of error is to use the "highlighter test" (see earlier in this chapter).

The second error—the GUI widget is shown as a boundary object—is another common mistake. A GUI widget such as a button is too fine-grained to be the boundary object; instead, the boundary object should be the screen or web page.

The third error is rather fundamental: an entire chunk of the use case text has been left off the diagram. It's surprising how often this happens. It's usually a sign that somebody isn't working through the use case one sentence at a time.

The fourth and final error is a direct consequence of ambiguous use case text: the text wasn't tied closely enough to the objects, so the modeler just sort of went off in a pseudo-random direction and, lacking the real boundary object to work with, used "Select address" as the boundary object, even though it's a verb and therefore is a controller masquerading as a boundary object. And as we're sure you know by now, actors can't talk directly to controllers.

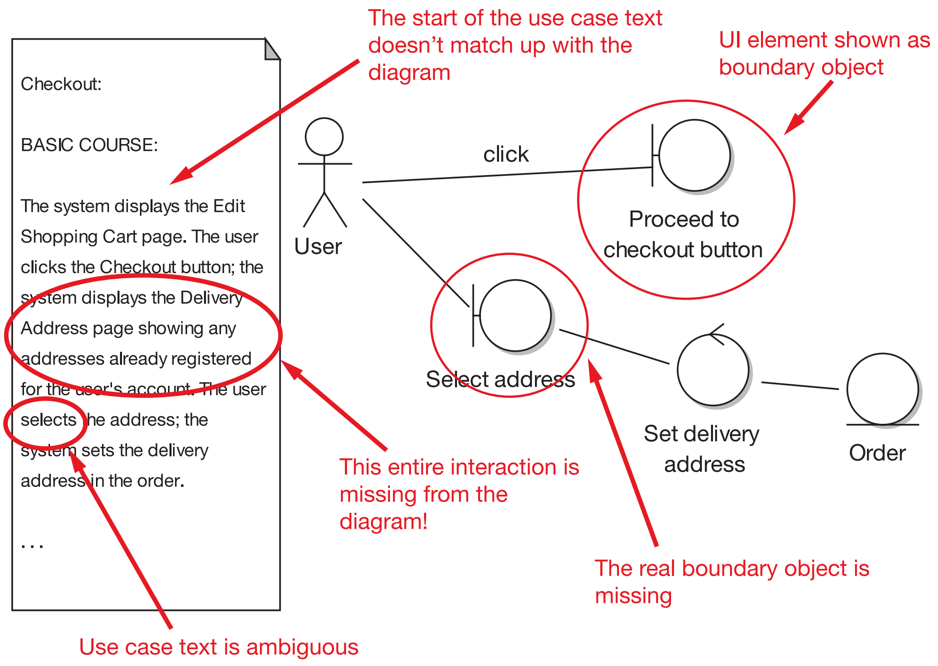

The start of the use case text doesn't match up with the diagram
UI element shown as boundary object
Checkout:
BASIC COURSE:
The system displays the Edit Shopping Cart page. The user clicks the Checkout button; the system displays the Delivery Address page showing any addresses already registered for the user's account. The user selects the address; the system sets the delivery address in the order.
. . .
User
click
Proceed to checkout button
Select address
Set delivery address
Order
This entire interaction is missing from the diagram!
The real boundary object is missing
Use case text is ambiguous

Figure 5-28. *The robustness diagram excerpt from Exercise 5-4, with errors highlighted*

Figure 5-29 shows the corrected diagram and use case text. In redrawing the diagram, we discovered another ambiguity in the text (funny how that happens!). The text "the system displays the Delivery Address page showing any addresses already registered for the user's account" implies some search and retrieval initialization behavior, which currently also doesn't appear anywhere on the diagram. So we rewrote this part of the use case and added it into the diagram, and in the process we discovered another domain class (Delivery Address List). This is exactly the kind of scenario where robustness analysis proves invaluable—discovering hidden functionality and missing domain classes, just in time to begin designing.

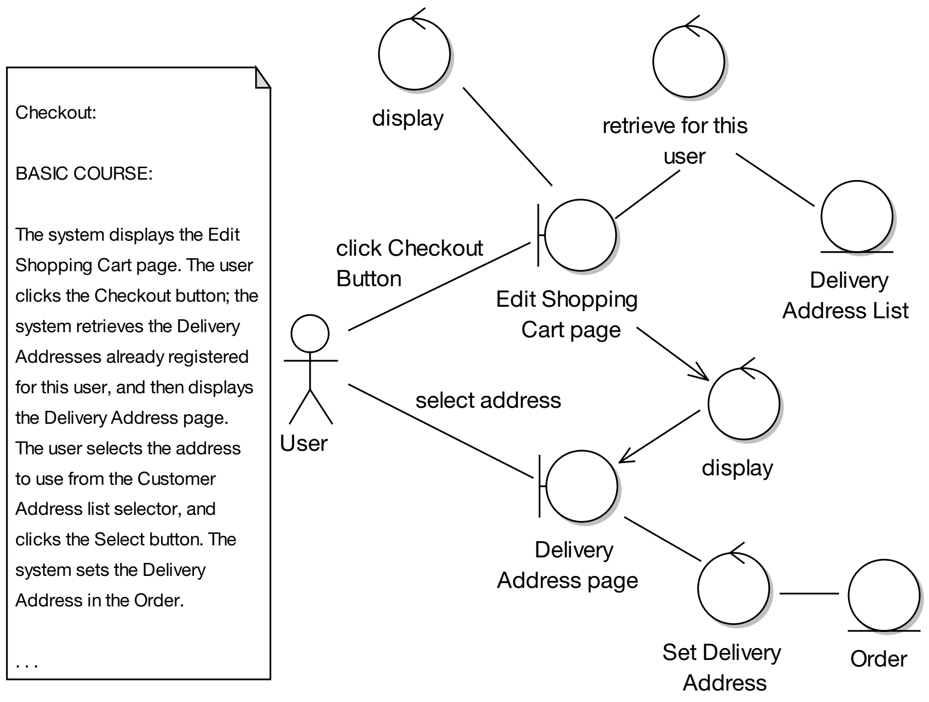

Checkout:
BASIC COURSE:
The system displays the Edit Shopping Cart page. The user clicks the Checkout button; the system retrieves the Delivery Addresses already registered for this user, and then displays the Delivery Address page. The user selects the address to use from the Customer Address list selector, and clicks the Select button. The system sets the Delivery Address in the Order.
. . .
display
retrieve for this user
click Checkout Button
Edit Shopping Cart page
Delivery Address List
User
select address
display
Delivery Address page
Set Delivery Address
Order

Figure 5-29. *The corrected robustness diagram excerpt for Exercise 5-4*

Exercise 5-4 Solution: Alternate Courses Not Shown

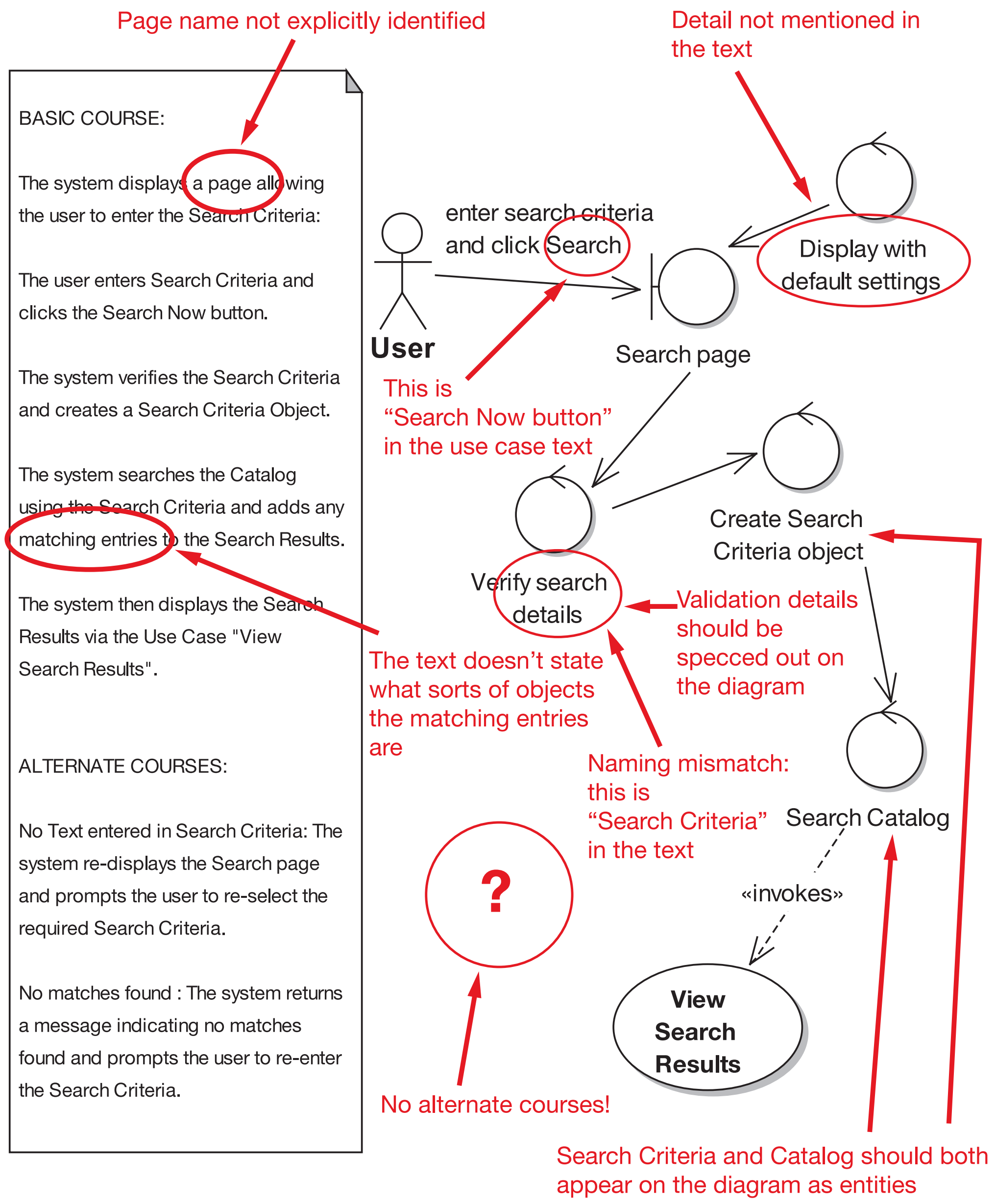
Figure 5-30 highlights the errors in Figure 5-23.

To kick off, the text "a page allowing the user to enter the Search Criteria" sounds like a roundabout way of saying "the Search Page," and it's always better to give your pages explicit names.

On the arrow between User and Search Page, the text "click Search" should be "click the Search Now button" to match it up with the use case text.

On the controller at the top right, some extra detail has been added to the diagram—"Display with default settings"—that doesn't appear in the use case text. Remember, the text and the diagram should be virtual carbon copies of each other. As "Display with default settings" is a bit like saying, "The system displays what it displays" (i.e., it doesn't add anything meaningful in this case since the screen doesn't have to fetch any data when it initializes), it can safely be removed.

Back in the use case description, the text "adds any matching entries to the Search Results" is rather vague. It doesn't describe exactly what the "entries" are. "Entries" is vague and ambiguous. This is another example of use case text that isn't tied closely enough to the domain model. Simply replacing "matching entries" with "matching Books" fixes this.



BASIC COURSE:

The system displays a page allowing the user to enter the Search Criteria:

The user enters Search Criteria and clicks the Search Now button.

The system verifies the Search Criteria and creates a Search Criteria Object.

The system searches the Catalog using the Search Criteria and adds any matching entries to the Search Results.

The system then displays the Search Results via the Use Case "View Search Results".

ALTERNATE COURSES:

No Text entered in Search Criteria: The system re-displays the Search page and prompts the user to re-select the required Search Criteria.

No matches found : The system returns a message indicating no matches found and prompts the user to re-enter the Search Criteria.

Page name not explicitly identified
Detail not mentioned in the text
User
enter search criteria and click Search
Display with default settings
Search page
This is "Search Now button" in the use case text
Create Search Criteria object
Verify search details
Validation details should be specced out on the diagram
The text doesn't state what sorts of objects the matching entries are
Naming mismatch: this is "Search Criteria" in the text
Search Catalog
?
«invokes»
View Search Results
No alternate courses!
Search Criteria and Catalog should both appear on the diagram as entities

Figure 5-30. *The robustness diagram from Exercise 5-3, with errors highlighted*

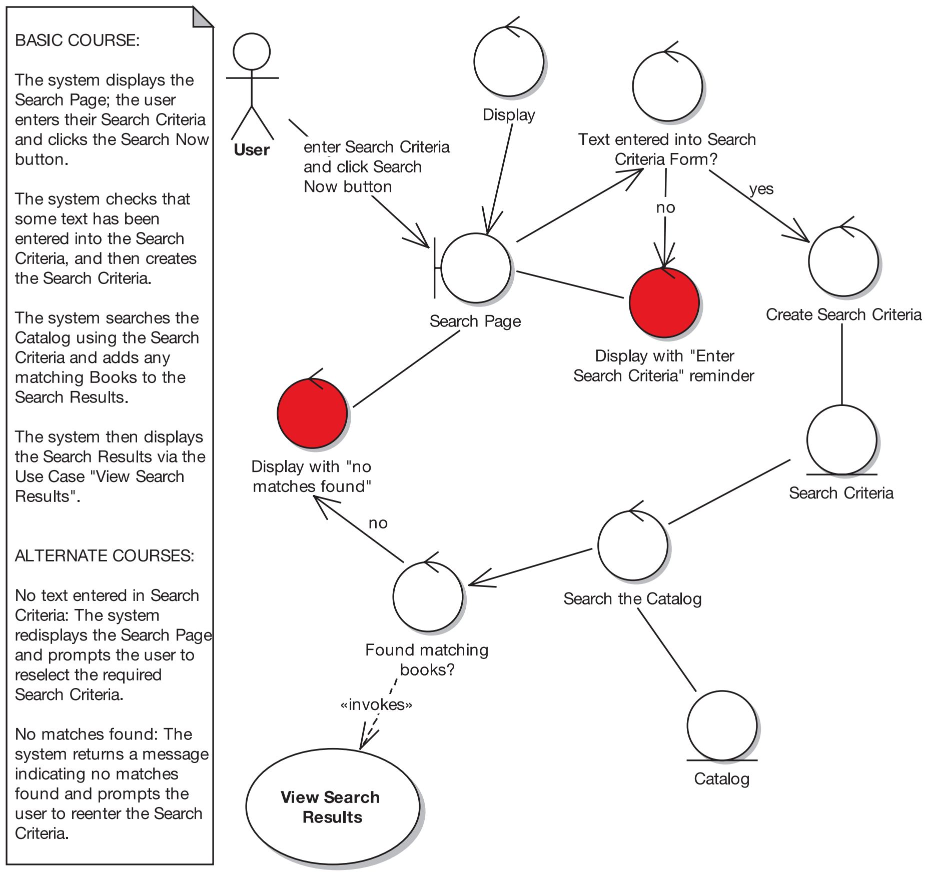
There's another naming mismatch on the "Verify search details" controller: it should be called "Verify Search Criteria." And while we're at it, this is really *validating* the search form, but the diagram doesn't walk through the validation checks, which it should.

■**Note** The robustness diagram doesn't need to specify the validation checks in detail, as that would be quite cumbersome, but it should at least name the function, with a controller per function (or group of closely related functions).

These validation checks have been left off because of the next (and biggest) error, which is that the diagram doesn't show *any* of the alternate courses! (This happens surprisingly often, but the alternate courses represent a huge proportion of the system functionality, so they must be given the same amount of attention as the basic course.)

Finally, the diagram is missing entity objects. Two should be on there: Search Criteria and Catalog.

Figure 5-31 shows the corrected robustness diagram. We've also done a small amount of additional tidying-up in the corrected version: see if you can spot what else we've fixed.



BASIC COURSE:

The system displays the Search Page; the user enters their Search Criteria and clicks the Search Now button.

The system checks that some text has been entered into the Search Criteria, and then creates the Search Criteria.

The system searches the Catalog using the Search Criteria and adds any matching Books to the Search Results.

The system then displays the Search Results via the Use Case "View Search Results".

ALTERNATE COURSES:

No text entered in Search Criteria: The system redisplays the Search Page and prompts the user to reselect the required Search Criteria.

No matches found: The system returns a message indicating no matches found and prompts the user to reenter the Search Criteria.

User
enter Search Criteria and click Search Now button
Display
Text entered into Search Criteria Form?
yes
no
Search Page
Display with "Enter Search Criteria" reminder
Create Search Criteria
Display with "no matches found"
Search Criteria
no
Search the Catalog
Found matching books?
«invokes»
View Search Results
Catalog

Figure 5-31. *The corrected robustness diagram from Exercise 5-4*

More Practice

This section provides a list of modeling questions that you can use to test your knowledge of robustness analysis.

1. Which of the following is *not* accomplished during robustness analysis?
   - a) Object discovery
   - b) Disambiguation of use case text
   - c) Modeling the problem domain
   - d) Validation of requirements by doing conceptual design
2. Which of the following is probably *not* a boundary class?
   - a) Login screen
   - b) Account table
   - c) Mailbox
   - d) Error dialog
3. Arrows on a robustness diagram can represent
   - a) Data flow
   - b) Control flow
   - c) Communication associations
   - d) All of the above
4. Which of the following is probably *not* a controller?
   - a) Validate password
   - b) Transaction manager
   - c) Line item
   - d) Display error message
5. Performing robustness analysis as an intermediate step between writing use cases and drawing sequence diagrams adds an additional modeling step and an additional required diagram to a software process, as opposed to drawing sequence diagrams immediately after writing use cases. Does adding this additional step make the process more or less efficient? Give at least three reasons to support your answer.
6. Robustness diagrams and collaboration diagrams (also called *communication diagrams* in UML2) both show objects collaborating. Does this mean that a robustness diagram and a collaboration diagram are the same, or are they different? Explain.

7. List four things accomplished during robustness analysis that help to close the gap between "what" (requirements analysis) and "how" (detailed design). Explain the benefits of each.

8. Attack or defend the following statement:

   > It's impossible to completely understand your requirements without doing some exploratory design.

   If you agree with the statement, list three benefits of doing this exploratory design as a "conceptual design" modeling step, as opposed to doing all exploratory design in code.

Summary

In this chapter, we moved from analysis to design, using one of the industry's most useful and yet best-kept secrets: robustness analysis.

The activity diagram in Figure 5-32 shows where we are (the tasks we discussed in this chapter are shown in red). This brings us to Milestone 2 (Preliminary Design Review), which we cover in the next chapter.

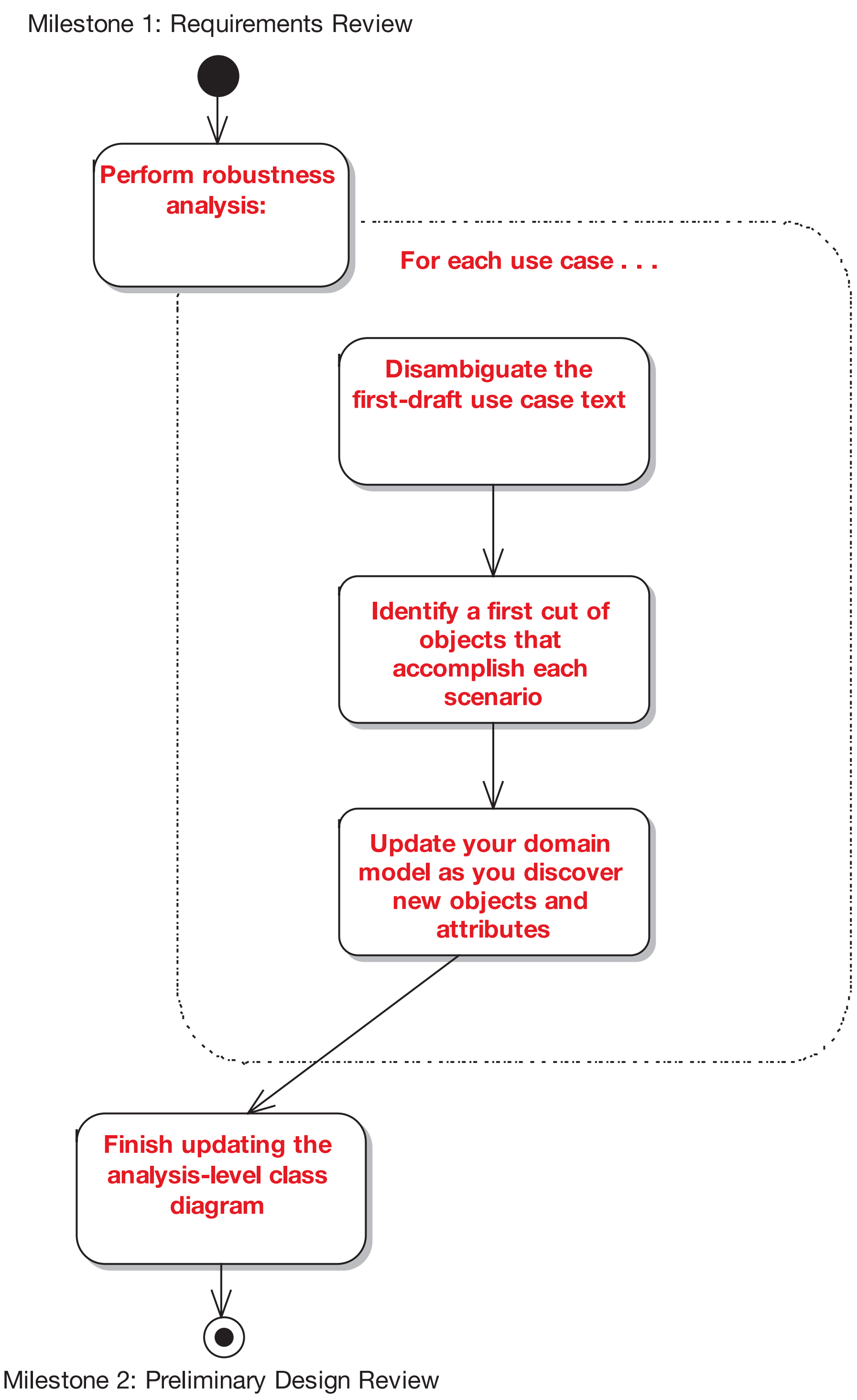

Milestone 1: Requirements Review
Perform robustness analysis:
For each use case . . .
Disambiguate the first-draft use case text
Identify a first cut of objects that accomplish each scenario
Update your domain model as you discover new objects and attributes
Finish updating the analysis-level class diagram
Milestone 2: Preliminary Design Review

Figure 5-32. *Analysis and Preliminary Design Checkpoint 2*