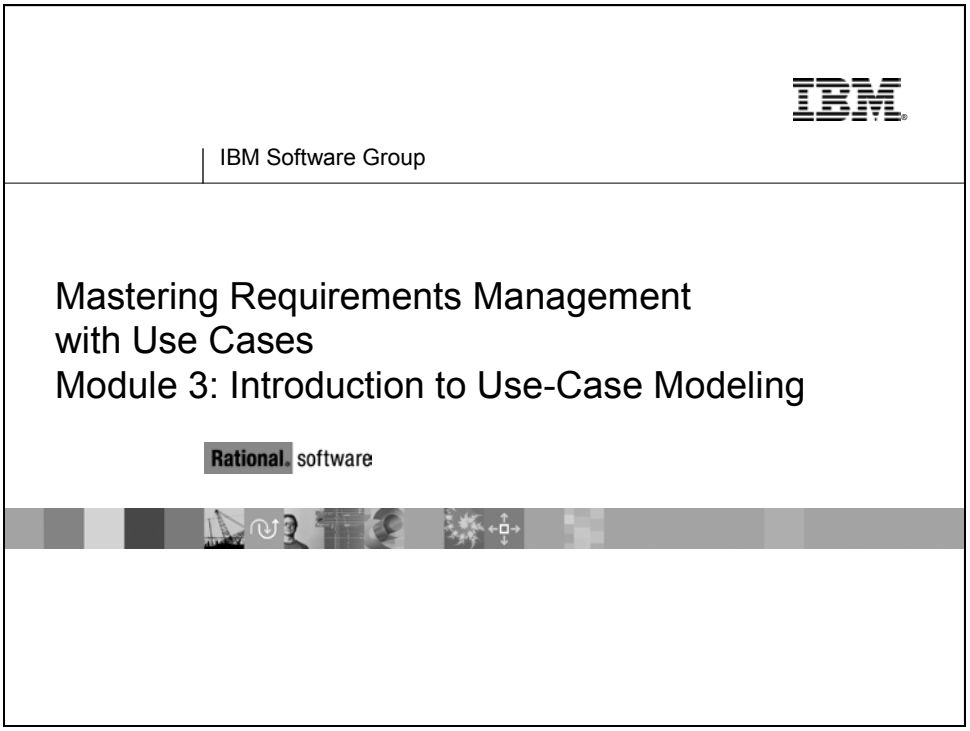
► ► ► Module 3
Introduction to Use-Case Modeling



Topics

Objectives

Objectives

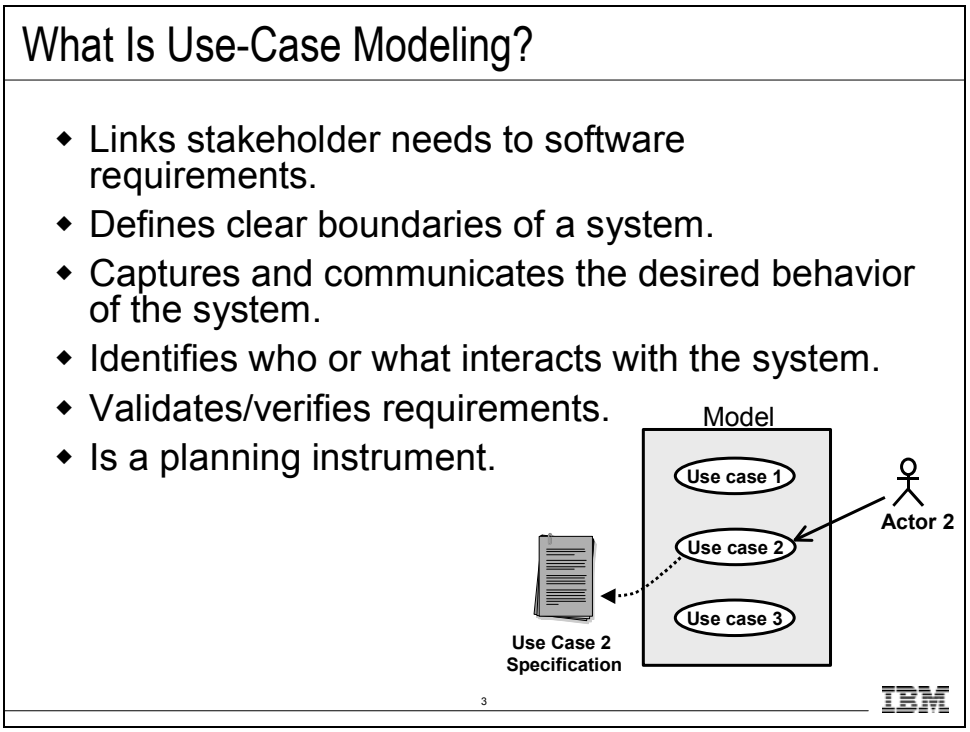
- Define key concepts of use-case modeling.
  - List the benefits of use-case modeling.
- Find actors and use cases.
  - Describe their relationships to each other.
- Define functional decomposition.
- Read and draw a use-case diagram.

2

IBM

This module provides a brief introduction to use-case modeling concepts, its benefits, and how to create and read a use-case model.

What Is Use-Case Modeling?



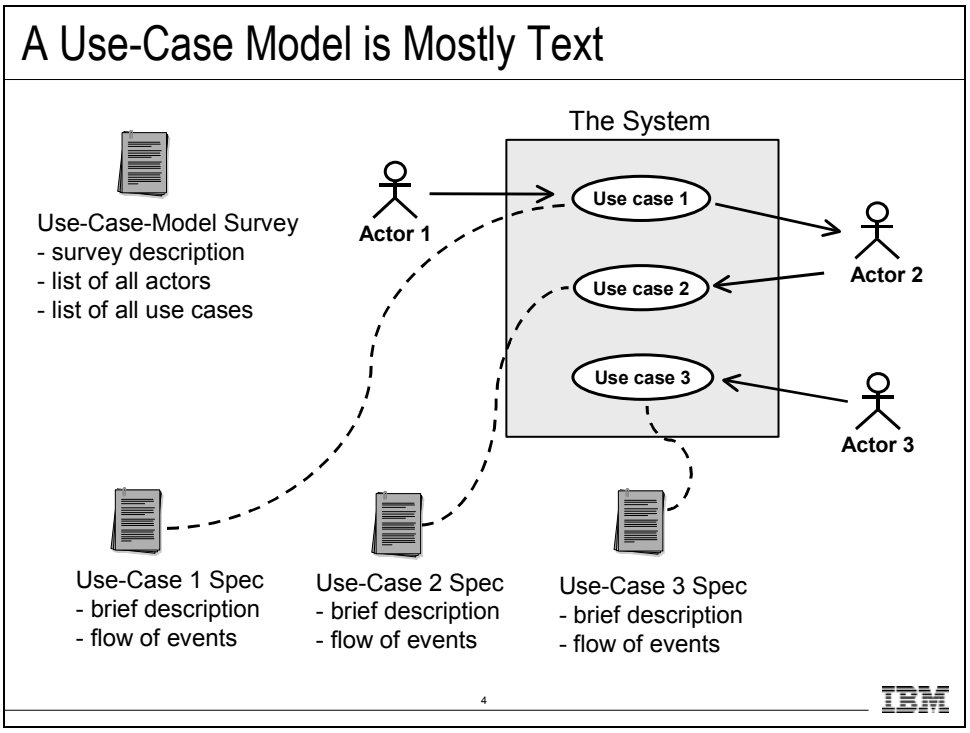
A use-case model describes a system's functional requirements in terms of use cases.

It is a model of the system's intended functionality (use cases) and its environment (actors). Use cases enable you to relate what you need from a system to how the system delivers on those needs.

Think of a use-case model as a menu, much like the menu you'd find in a restaurant. By looking at the menu, you know what's available to you, the individual dishes as well as their prices. You also know what kind of cuisine the restaurant serves: Italian, Mexican, Chinese, and so on. By looking at the menu, you get an overall impression of the dining experience that awaits you in that restaurant. The menu, in effect, "models" the restaurant's behavior.

Because it is a very powerful planning instrument, the use-case model is generally used in all phases of the development cycle by all team members.

A Use-Case Model is Mostly Text

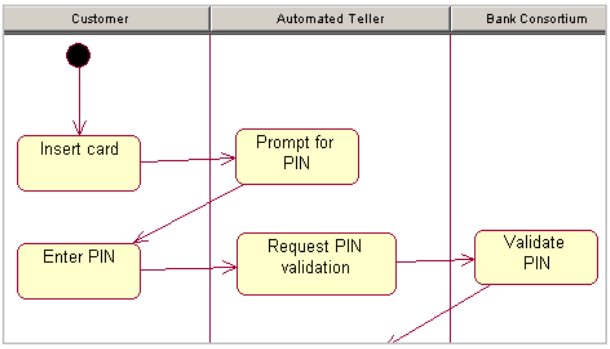


The use-case model consists of both diagrams and text. The diagrams give a visual overview of the system. The text gives descriptions of the actors and the use cases.

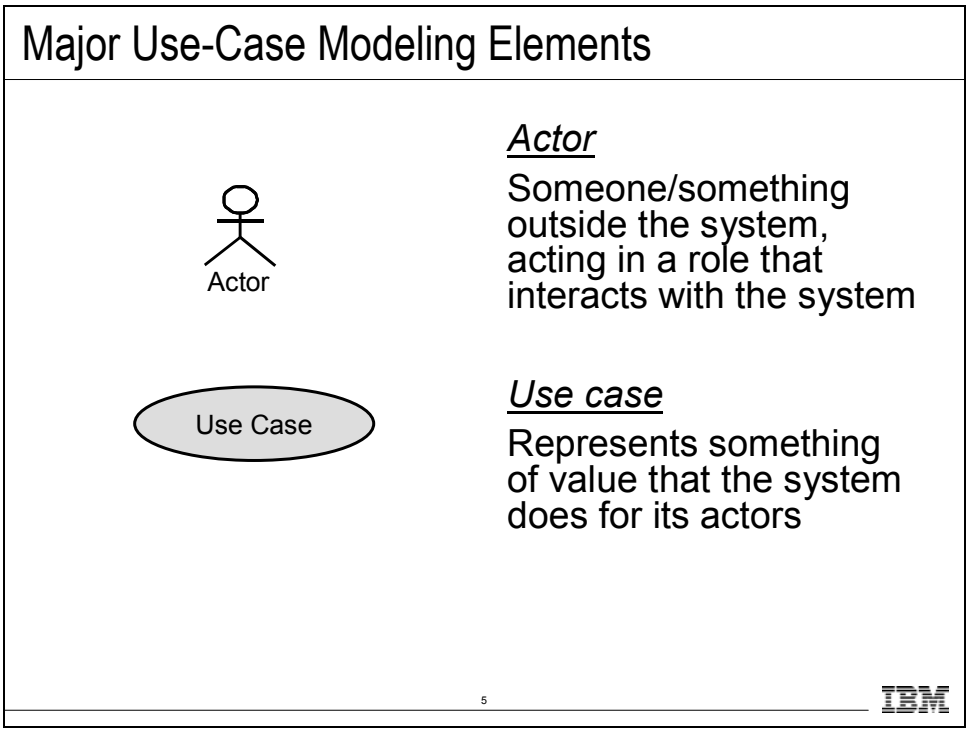
Use cases involve writing text. Drawing the pictures is only a small part of the effort. Typically, more than 80 percent of all effort during requirements capture is to write the textual description of what happens in each use case, the non-functional requirements, and rules. The description of what happens is called the flow of events.

Activity diagrams (previously known as flow charts) are another useful tool you can use to describe a use case. It is quite common to use an activity diagram to describe complex flows of events. When using activity diagrams, it is advisable to use swimlanes to represent each actor and the system. Without swimlanes, the activities float in a "semantic emptiness" and quickly become meaningless.

The following diagram is an example from a Withdraw Cash use case for an ATM.



Major Use-Case Modeling Elements



An **actor** represents a role that interacts with the system.

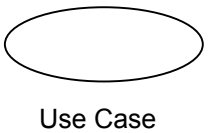
A **use case** describes a sequence of interactions between actors and the system that occur when an actor uses the system to achieve a certain business goal.

A use case describes:

- The system, its environment, and the relationship between them.
- How things outside the system interact with the system.
- The desired behavior for the system.

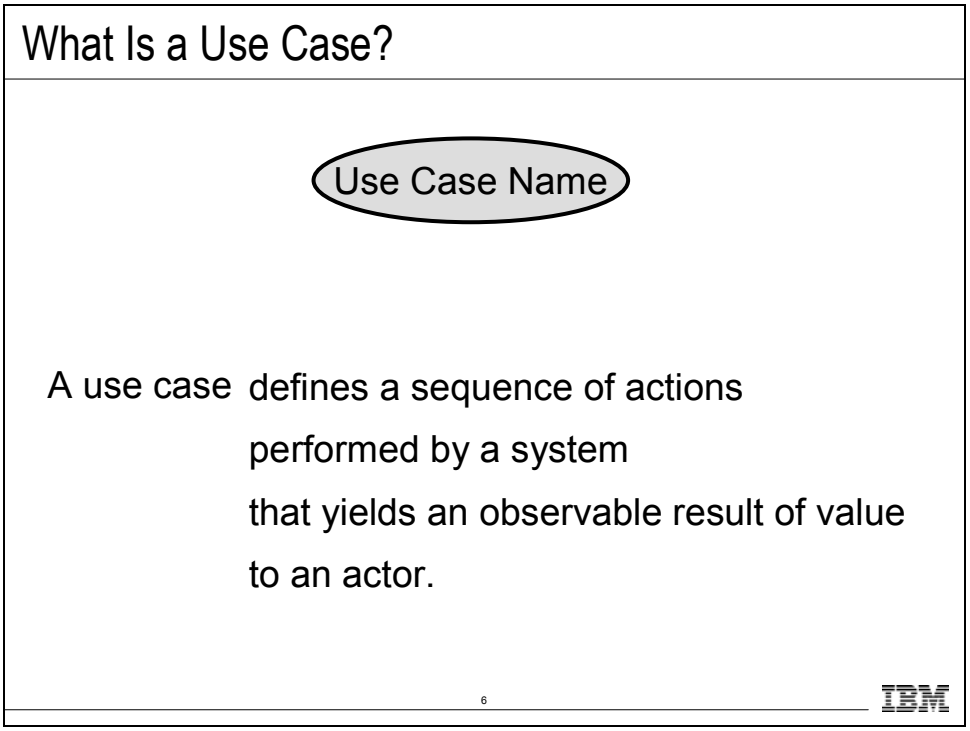
Use cases are containers for contextually related requirements of the system under development. They are containers because they group all requirements related to achieving a particular goal into a single story of how that is achieved.

A use case is shown as an ellipse containing the name of the use case. An optional presentation is to place the use case name under the ellipse.



Use Case

What Is a Use Case?



A use case describes a set of possible executions of the system. It describes a complete flow through the system until the resulting value is achieved for some actor.

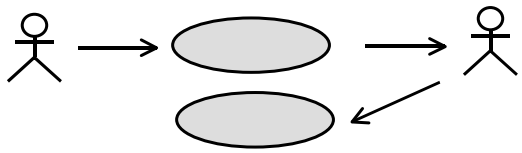
- "**Sequence of actions**": Atomic activities, decisions, and requests. Each action is performed fully or not at all.
- "**Performed by a system**": The actions performed by the system are the functional requirements.
- "**Observable result of value**": Make sure that the use case results in an observable value. Why would anybody use the system if it does not achieve a result of value? If nobody receives value from the use case, then the use case is probably too small. It needs to be combined with other use cases to provide a complete set of steps for achieving some goal for an actor.
- "**To an actor**": Decide which particular actor receives the value and helps avoid use cases that are too large. If more than one actor receives value from a use case, it might indicate that the use case is too big; it tries to do too much for too many actors. For example, in the Course Registration System, there could be just one use case, called "Operate the Registration System." It would achieve a variety of results for a variety of actors. It would be so big that it is hard to understand. It needs to be broken into smaller use cases. Use cases that do not deliver something of value are too small and usually describe individual functions of the system. These use cases do not make sense on their own and must be avoided.

Often the sequence of interactions in a use case involves several actors. But one actor receives the result of value. The actor receiving the value is sometimes called the primary actor. The actor receiving the value is usually, but not necessarily, the one who initiated the use case.

Use Cases Contain Software Requirements

Use Cases Contain Software Requirements

- Each use case
  - Describes actions the system takes to deliver something of value to an actor.
  - Shows the system functionality an actor uses.
  - Models a dialog between the system and actors.
  - Is a complete and meaningful flow of events from the perspective of a particular actor.



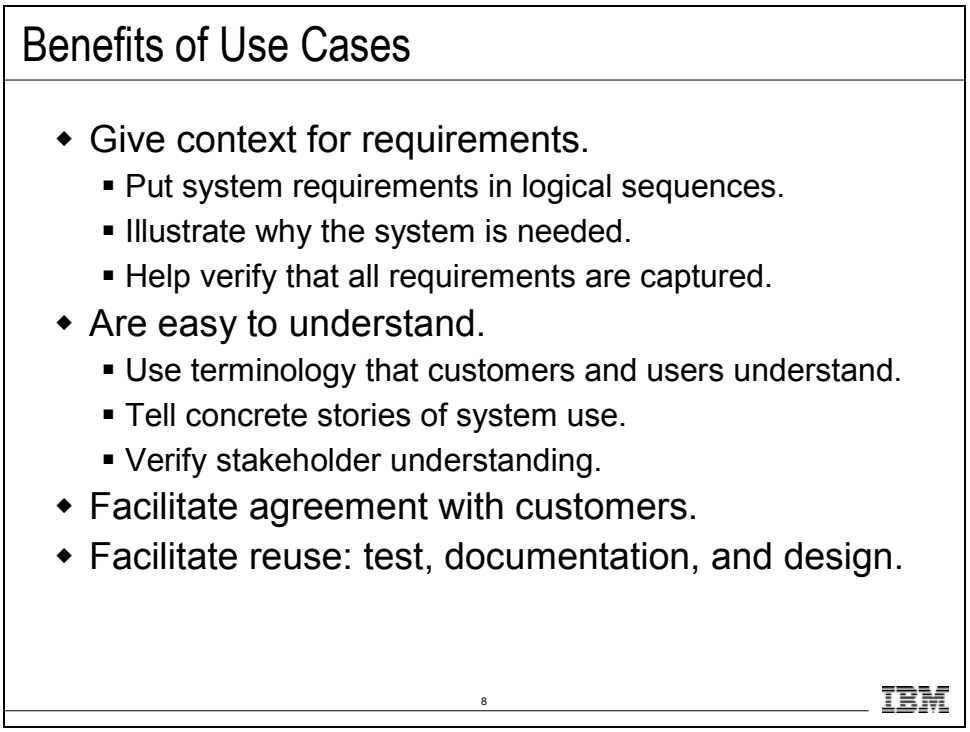
7

IBM

Use cases contain the detailed functional software requirements. Every time a use case says, "The system …," is a detailed requirement of what the system must do.

Also, remember the definition: "A sequence of actions *performed by a system* that yields a measurable result of value for a particular actor."

Benefits of Use Cases



Do you have any problems in these areas with your current requirement process?

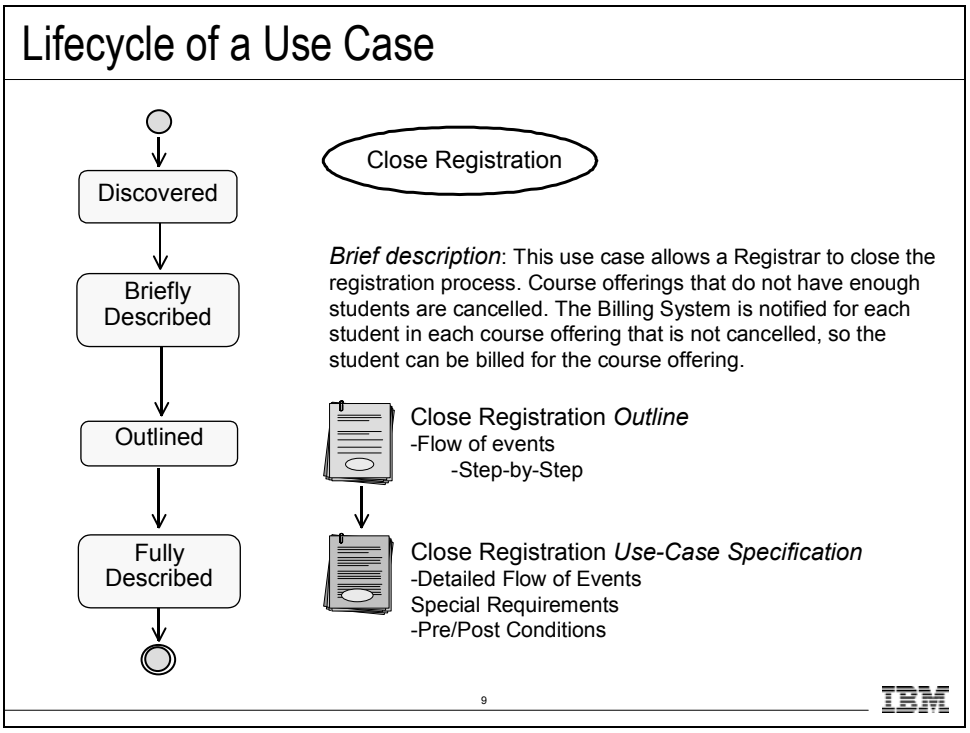
Use cases are a way to organize requirements **from a user perspective**. All the requirements for a user to accomplish a particular task are gathered together in a single use case. The use-case model is the collection of all the individual use cases.

Advantages of use-case modeling include:

- Use cases show why the system is needed. Use cases show what goals the users can accomplish by using the system.
- System requirements are placed in context. Each requirement is part of a logical sequence of actions to accomplish a goal of a user.
- Use cases are easy to understand. The model expresses requirements from a user perspective and in the user's own vocabulary. The model shows how users think of the system and what the system should do. Traditional forms of requirements capture need some translation to make them useable to different stakeholder types. When you translate something, information is often lost or misinterpreted. Use cases require no such translation and therefore provide a more consistent and reliable form of requirement capture.
- The model is a means to communicate requirements between customers and developers to make sure that the system we build is what the customer wants.

Use-case modeling is the best tool (so far) to capture requirements and put them in context.

Lifecycle of a Use Case



In all but the most trivial of systems, use cases are not written in one sitting. As with any iterative process, a use case evolves from the spark of an idea to a fully described description of how your system behaves.

Initially you "discover" a use case. This is done when you identify and name the goal that the actor is trying to achieve. A use case is discovered once you have named it.

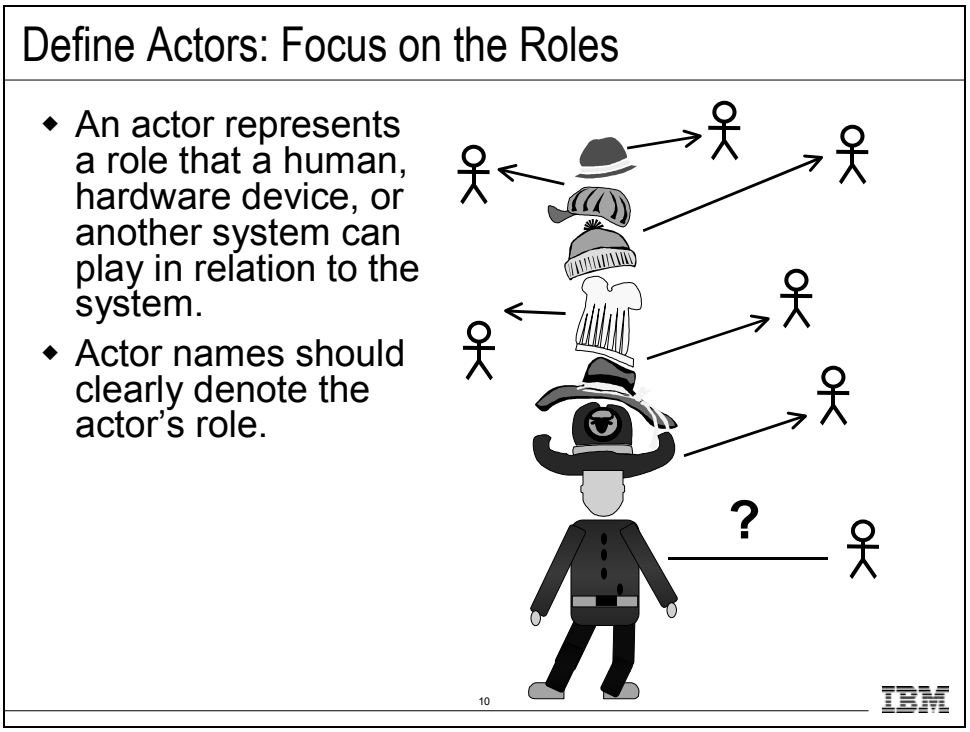
Immediately after discovering a use case (usually while you are discussing the name), you create a brief description of the use case. A brief description describes the goal in two or three sentences. For example, the Close Registration Use Case's brief description is: *This use case allows a Registrar to close the registration process. Course offerings that do not have enough students are cancelled. The Billing System is notified for each student in each course offering that is not cancelled, so the student can be billed for the course offering.*

The next stage of evolution is to outline the use case. This can be a bulleted list of just the basic flow. You may also choose to identify identify (not outline) possible alternate flows. This enables you to identify scenarios and also gain an understanding of the possible size and complexity of the use case.

Finally, you fully describe the use case. This is done iteratively. You identify particular flows for development in an iteration and then fully describe and implement just those flows. You then identify flows for subsequent iterations and fully describe and implement those during that iteration. At the end of the project, you have all of your use cases fully described.

Prioritization of flows is done during the Manage the Scope of the System workflow detail.

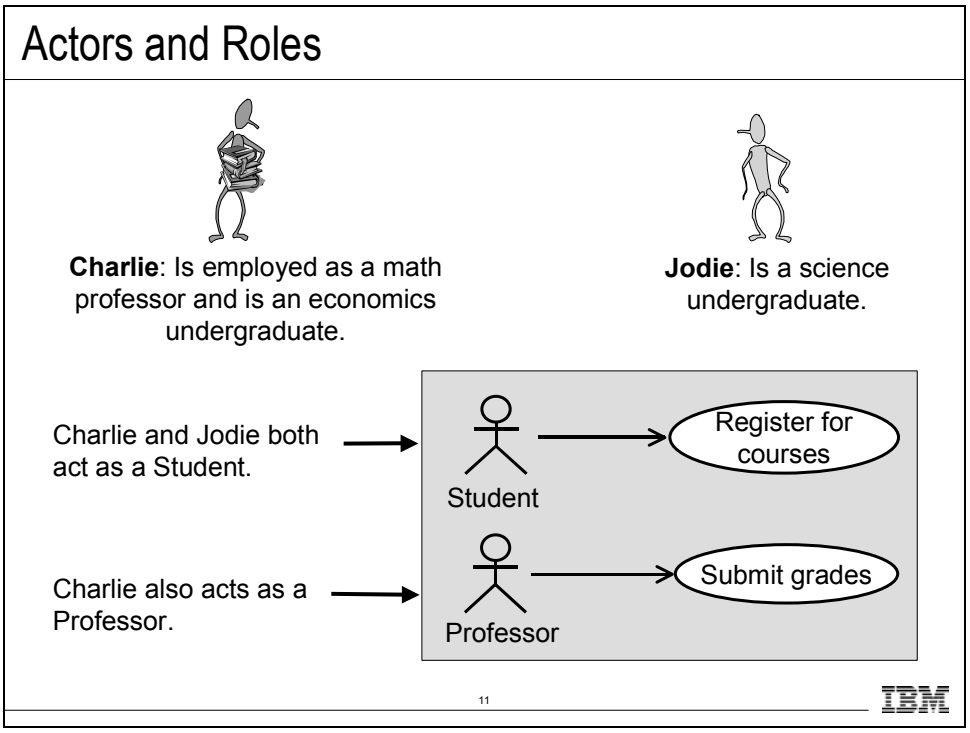
Define Actors: Focus on the Roles



The difference between an actor and an individual system user is that an actor represents a particular class of user rather than an actual user. Several users can play the same role, meaning they can be the same actor. In that case, each user constitutes an instance of the actor.

However, in some situations, only one person plays the role modeled by an actor. For example, there may be only one individual playing the role of system administrator for a rather small system.

Actors and Roles



Actors are roles. It is important to get away from thinking of actors as persons. In most situations, many different people can play the role of a particular actor. In the Course Registration example, there are many people who are students.

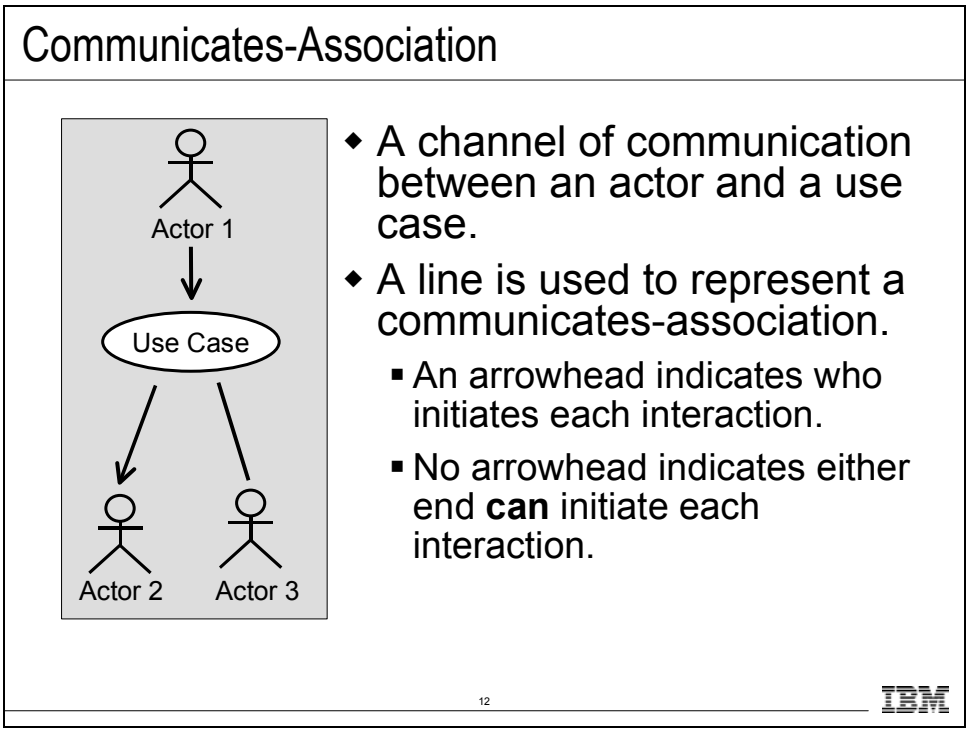
At First Community College of Centerville, Charlie is a Professor. Charlie also takes courses at First Community College of Centerville. When he is enrolling in courses on the Course Registration system, he is taking on the role of the Student actor.

Jodie is studying as a science undergraduate. She also takes on the role of a Student actor when using the system.

The difference between an actor and a system user is that the actor represents a particular type of user rather than an actual user. Several users can play the same role, which means they are all instances of the same actor. Each user playing that role is an instance of the actor.

In this example, Charlie and Jodie are students using a course registration system. When they are using the system to register for a course, each person is an instance of the Student actor.

Communicates-Association



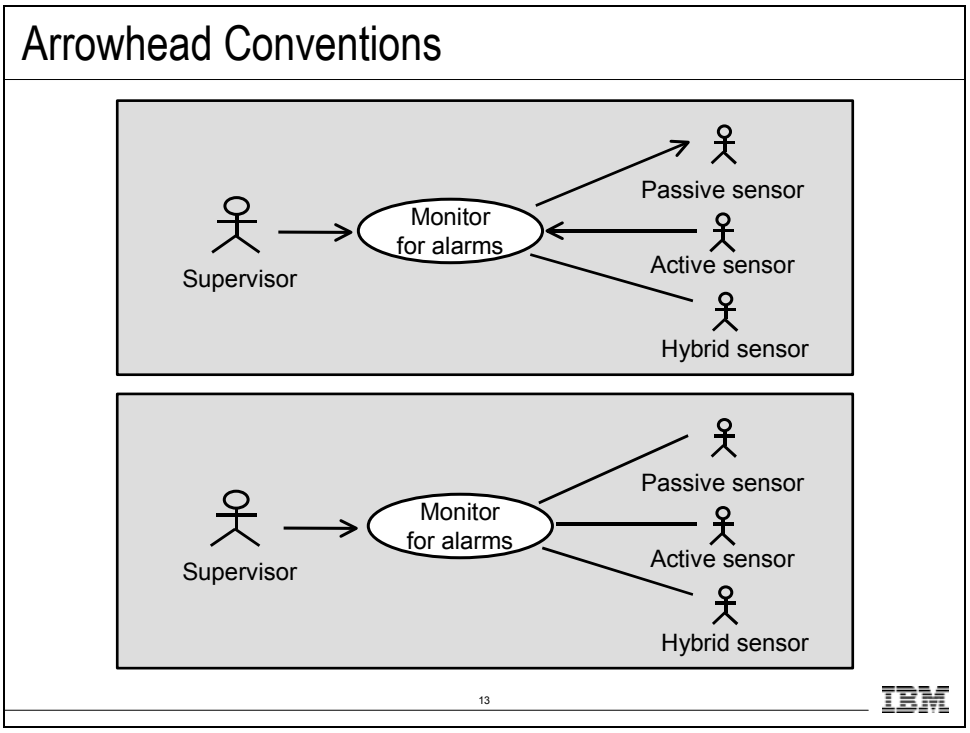
Relationships between actors and use cases are called communicates-associations. A communicates-association between an actor and a use case indicates that they interact: the actor participates in and communicates with the system containing the use case.

A communicates-association is represented on UML diagrams as a solid line. The line can be adorned with an arrow head. The line and arrow represent a two-way dialog between the actor and the system. If an arrow head is present, then only the "thing" at the tail of the association can initiate communication. No arrowhead means that either end **can** initiate communication (not that both ends **do** initiate communication).

In the example above, Actor 1 always initiates communication with the system. The system may respond to messages from Actor 1, but can never send an unsolicited message to Actor 1. For the second association, the system always initiates communication with Actor 2. Actor 2 may respond to messages from the system, but can never send an unsolicited message to the system. For the third association, Actor 3 or the system may initiate the dialog.

For more details, refer to the UML 1.3 Spec. Part 2 – Foundation, 2.5 – Core.

Arrowhead Conventions



The above is an example use case in a fire alarm monitoring system. The Supervisor is the person who administers the system and starts the monitoring.

A passive sensor is a smoke detector that requires the system to ask for its status. An active sensor is a smoke detector that automatically sends its status to the system every 60 seconds. A hybrid sensor is a smoke detector that sends its status every 60 seconds, but it can also receive status requests from the system. In this system, if the system does not hear from it in 75 seconds then the system asks the sensor for its status.

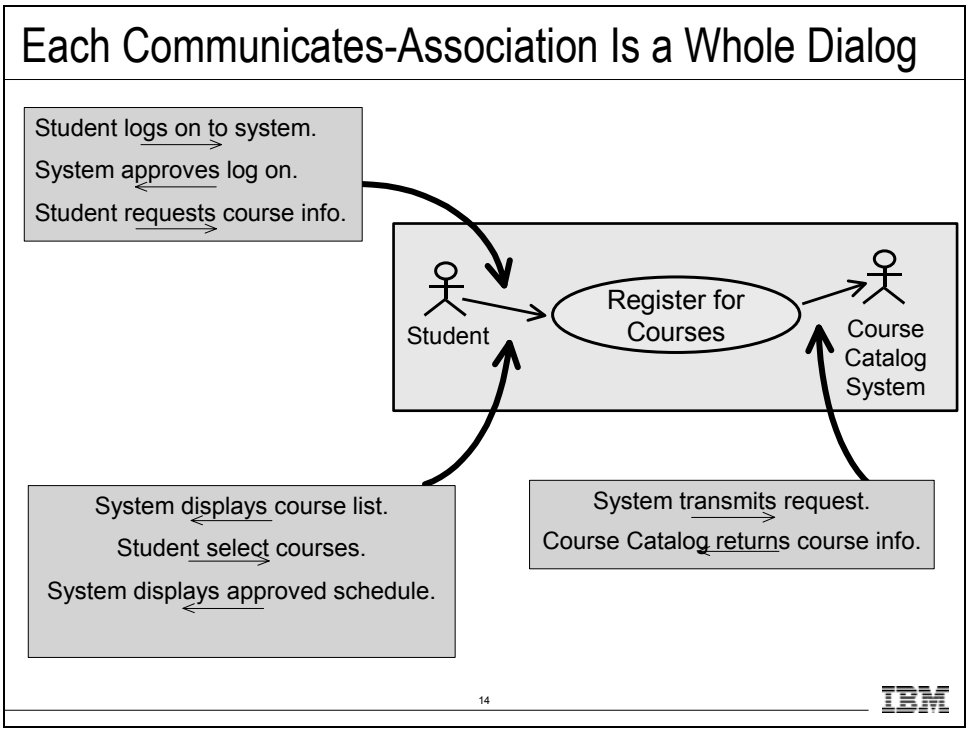
Observe how the top example conveys important characteristics of the actors surrounding the system. The information describing who starts the use case would be contained in the use-case description. For example: "This use case starts when the *Supervisor* activates the alarm monitoring."

The bottom example only shows the actor that starts the use case. This means the way the actors communicate with the system is relegated to the description of the actor. But this form is usually more palatable for non-UML savvy stakeholders.

Arrowheads are optional in UML, only use them if they help to clarify the diagrams. Whatever standard you adopt, it should be clearly specified in Use-Case-Modeling Guidelines for the project. The Golden Rule for use-case-model guidelines: **to effectively communicate the requirements to ALL stakeholders**.

Note: It is the exception to see two navigation arrows pointing towards a use case. Most of the time you have one arrow pointing in and the rest pointing out. The top example is present to reinforce that the navigation arrow is about indicating who initiates each interaction, not who obtains the goal or who initiates the use case.

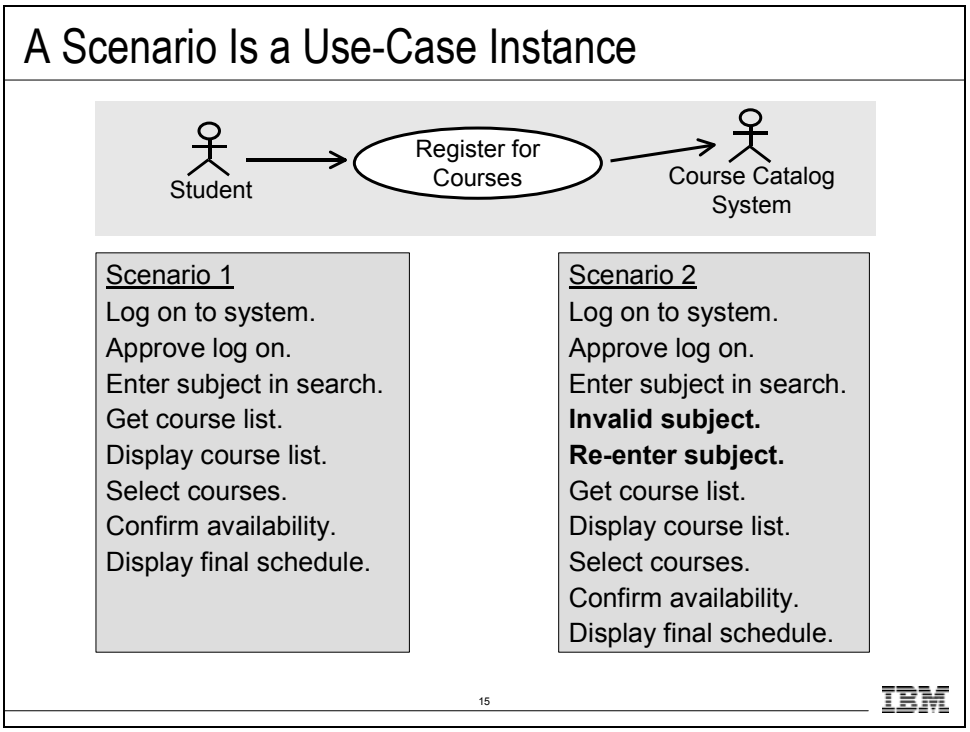
Each Communicates-Association Is a Whole Dialog



A communicates-association is a two-way dialog. The relationship between an actor and a use case is represented on the diagram by a single line or arrow, but the single line or arrow represents a whole dialog. For example, the relationship between the Student actor and the Register for Courses use case might include the whole dialog shown on the left side of this slide. The relationship between the Register for Courses use case and the Course Catalog System actor might include the whole dialog shown on the right side of this slide.

The dialogs are recorded in the Flows of Events in the use-case specification.

A Scenario Is a Use-Case Instance

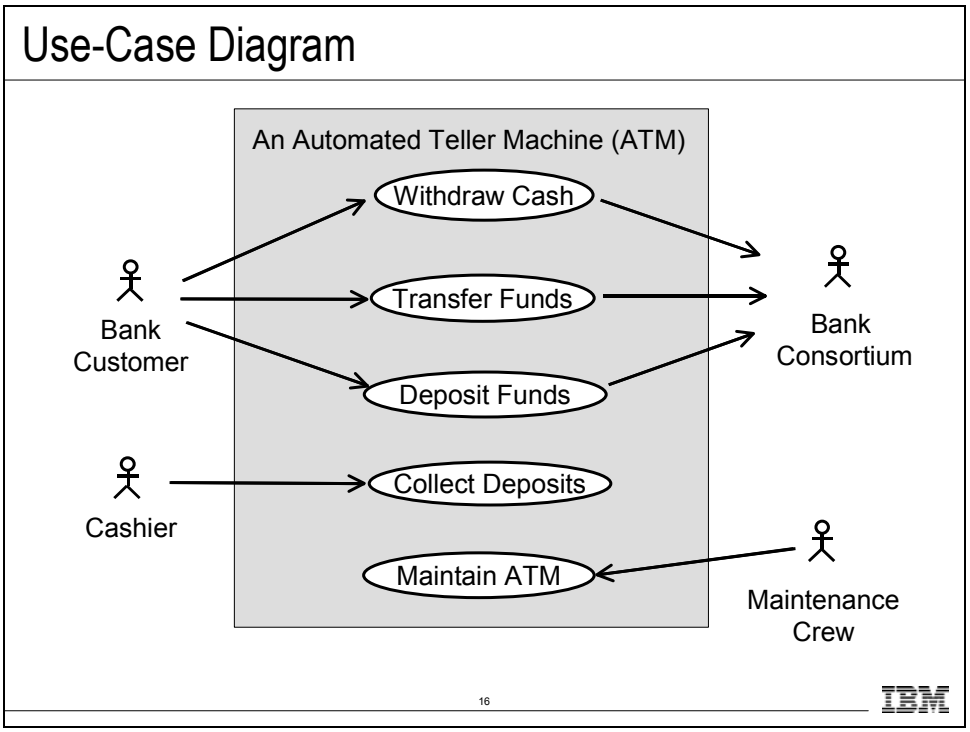


A use-case instance describes one particular path through the flows of events described in a use case. It is a specific sequence of actions that illustrates the behavior of the system. This is also called a scenario. In the example here, the sketch of Scenario 1 for the Register for Courses use case shows a Student interacting with the Course Registration System and successfully enrolling in a course the first time. Scenario 2 shows a Student interacting with the Course Registration System and entering an invalid subject; that student must re-enter the subject before successfully getting the course list.

A use case defines a set of related scenarios. A use case represents all the possible sequences that might happen until the resulting value is achieved or until the system terminates the attempt. The Register for Courses use case represents both of these sequences, and all the other possible sequences of interactions that may occur when a Student tries to enroll in a course.

Users and stakeholders can often identify the sequence of actions they want to perform to obtain a result. Asking stakeholders for the sequence of actions they perform is a good way to start identifying the steps in a use case.

Use-Case Diagram



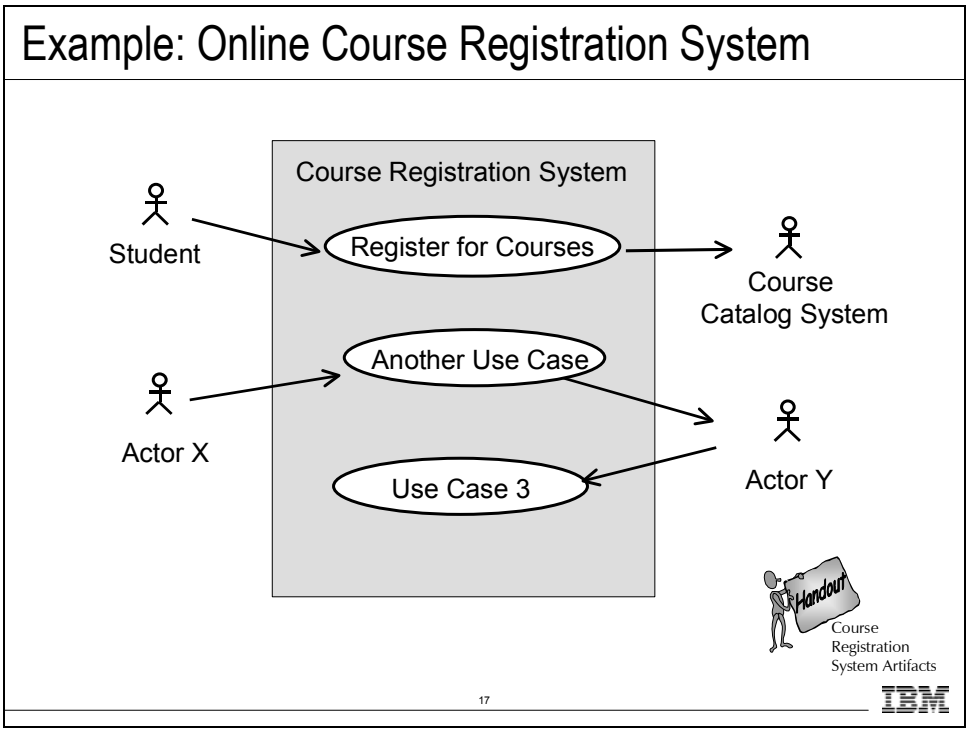
A use-case model shows what the system is supposed to do (use cases), the system's surroundings (actors), and the relationship between actors and use cases.

The use-case diagram is a graphical description of the use-case model. Can you tell at a glance what users can do with an ATM?

What do you think the priorities for the use cases above would be? The diagram already helps to determine these priorities; without the first one we do not have an ATM, so we better get it right, (e.g. during elaboration); other use cases like deposit funds are nice to have, but would also have quite an architectural impact.

It is also useful to distinguish between primary use case and secondary use cases: Primary use cases support the customers/actors and business. Secondary use cases we get, because we decided on a particular technical solution and thus have to specify "extra" functionality to run this technical solution; thus, Maintain ATM, Print logs, Start, Stop, Backup the System, etc are required. These could be also be organized in a separate diagram.

Example: Online Course Registration System



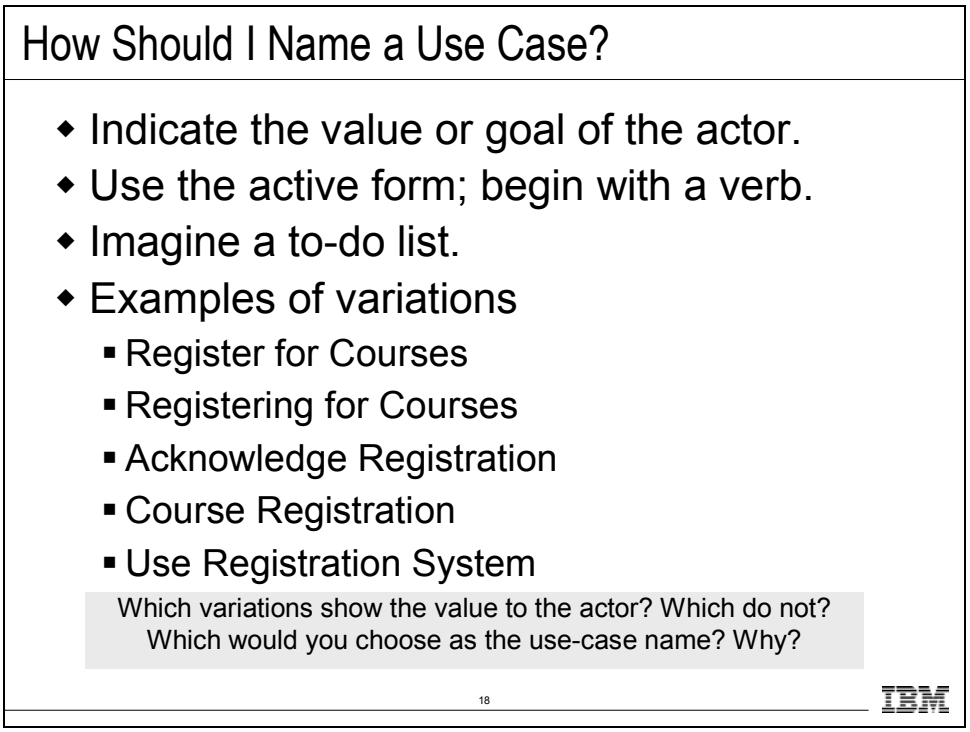
The use-case diagram shown here is a graphic description of the use-case model for an online Course Registration System. It shows two of the actors (Student and Course Catalog System) and one use case (Register for Courses) that they participate in.

The diagram is incomplete. You use the online course registration system as a case study in this module. You develop the use-case model for this example as we go through the module.

Take some time to look at the use-case model for an online Course Registration System in the Student Workbook. This example gives you an idea of what a use-case model looks like before you begin to develop one.

The use-case model for an online Course Registration System in the Student Workbook is incomplete. It contains only enough artifacts for the purpose of this module: Introduction to Use-Case Modeling. A larger, much more fully developed case study is presented later in the course.

How Should I Name a Use Case?



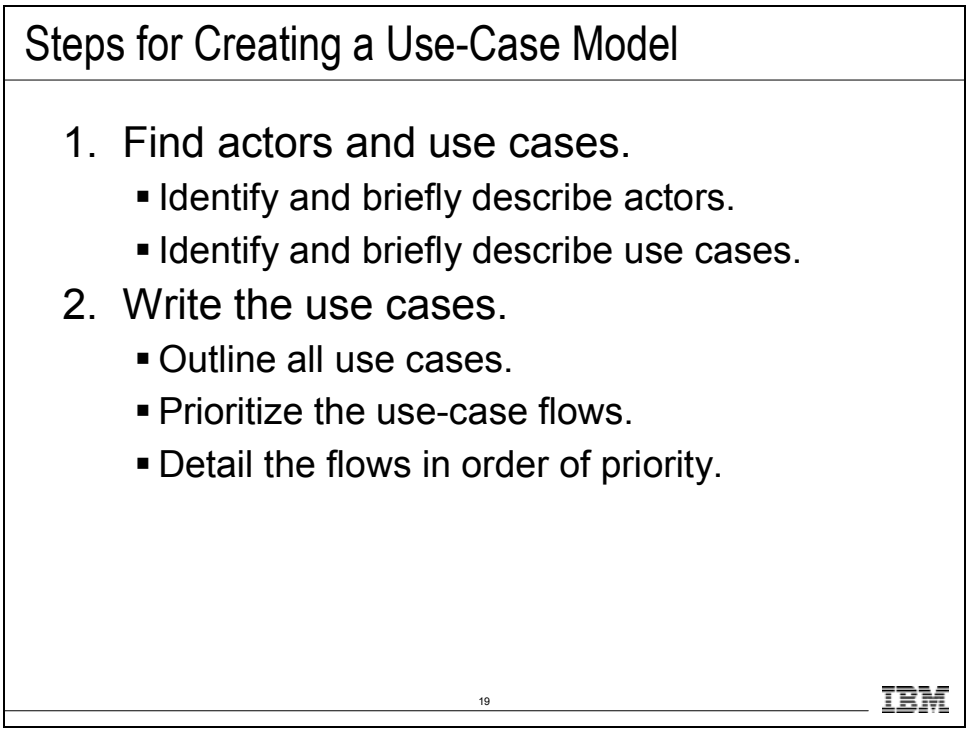
Each use case should have a name that indicates what is achieved by its interactions with the actor(s).

A good rule of thumb (but not dictated by any standard) is to attach the actor's name (of the primary actor) at the beginning of the use-case name to see if it makes a meaningful sentence. For example, does it make sense to say "The student registers for a course?" Does it make sense to say, "The student takes a course?"

Another approach is to ask, "Why does the actor want to use the system?"

Be focused on identifying the goal that is trying to be attained by using the system.

Steps for Creating a Use-Case Model



Creating a use-case model involves putting the pieces you have learned together. First, the actors and the use cases are found by using the requirements of customers and potential users as vital information. As they are discovered, the use cases and the actors are identified and illustrated in a use-case diagram. Next, the steps in a use case are outlined to get a sketch of the flow.

The actor's name must clearly denote the actor's role. Make sure there is little risk at a future stage of confusing one actor's name with another.

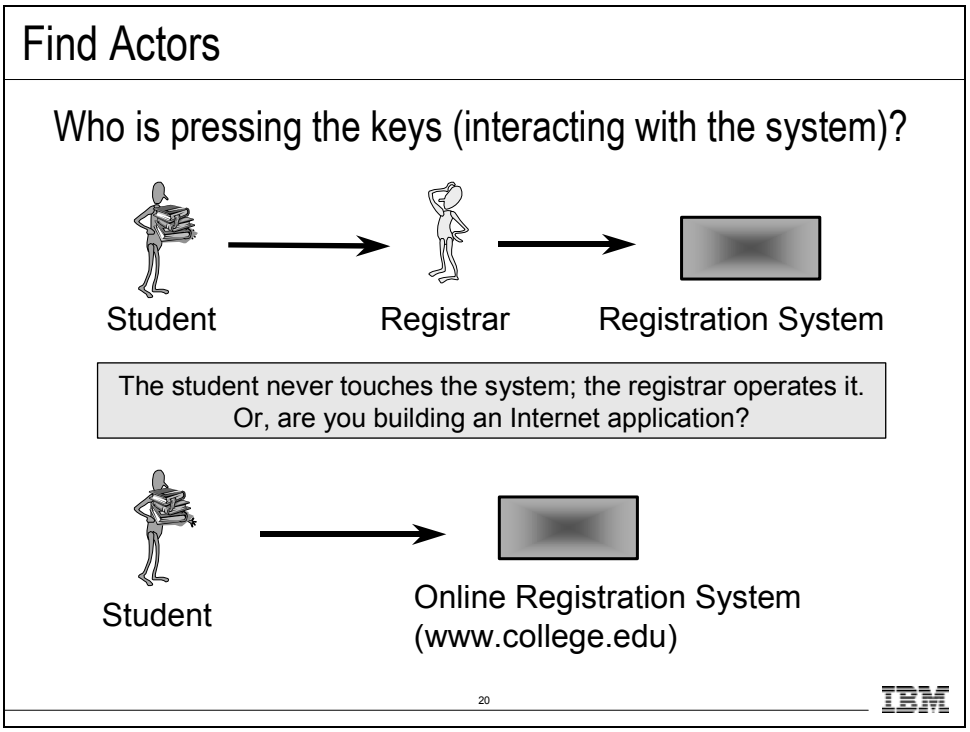
Define each actor by writing a brief description that includes the actor's area of responsibility and what the actor needs the system for. Because actors represent things outside the system, you need not describe them in detail.

Each use case should have a name that indicates what is achieved by its interactions with the actor(s). The name may have to be several words to be understood. No two use cases can have the same name.

Define each use case by writing a brief description of it. As you write the description, refer to the glossary and, if you need to, define new terms.

When the actors and use cases have been found, each use case is described in detail. These descriptions show how the system interacts with the actors and what the system does in each individual case. In an iterative development environment, you select a set of use case flows to be detailed in each iteration. These are prioritized in such a way that technical risk is mitigated early and the customer gets important functionality before less important functionality.

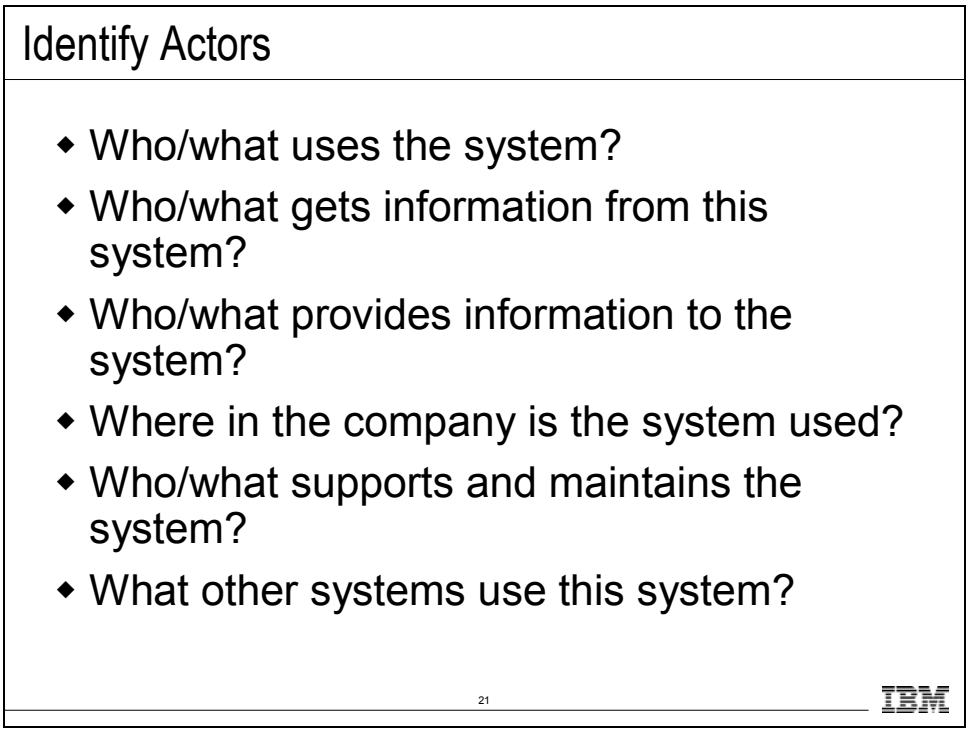
Find Actors



To identify the actors for a system, the simplest question to ask is: "Who is doing the actual interaction?" The actor is the one who is interacting with the system.

What if a person is using a speech recognition system? Then the actor is the one talking with the system.

Identify Actors



Here are some questions that may be used to help identify the actors in the system.

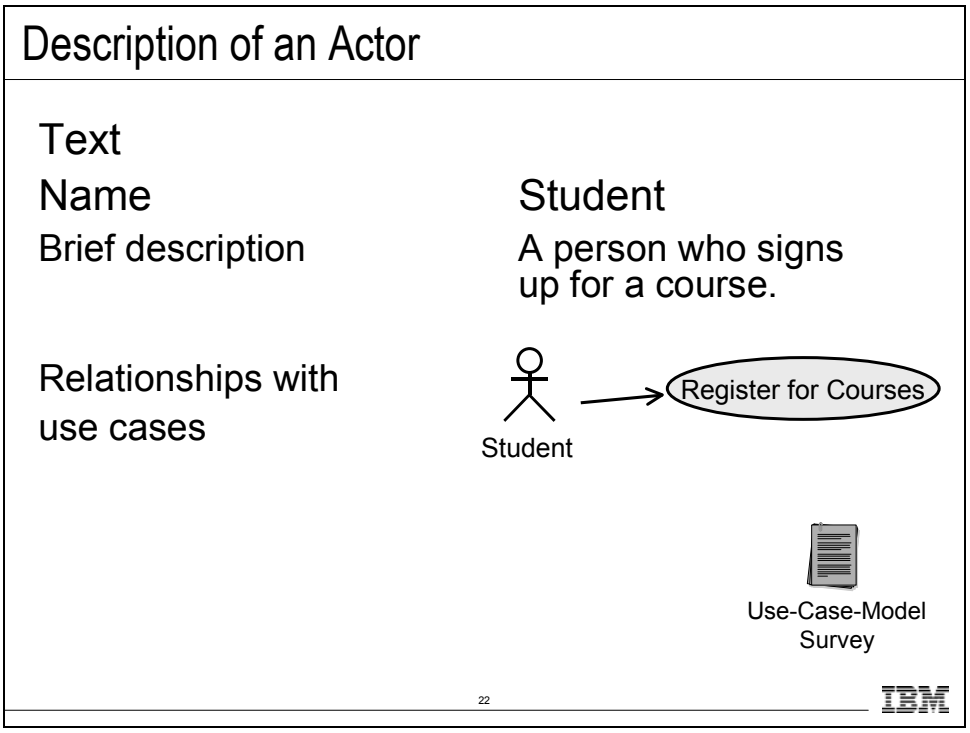
To fully understand the system's purpose, you must know who or what the system is for, that is, who or what use it. Different user types are represented as actors.

An actor is any person or any thing that is outside the system and that exchanges data with the system. An actor can either give information, receive information, or give and receive information.

Actor is a role, not a particular person or thing. The name of the actor should represent, as clearly as possible, the actor's role.
Make sure that there is little risk of confusing one actor's name with another at a future stage of the work. Also, try to avoid an actor called "User," rather try to figure out the role of that particular user.

In most instances, some person or some other system does something to trigger the start of the use case. If a use case in your system is initiated at a certain time, for example, at the end of the day or the end of the month, this can be modeled with a special actor, such as the "scheduler" or "time." "Scheduler," as opposed to "time," is a useful name for such an actor because scheduler may be human or non-human. "Time" leaves an element of design in your use case model.

Description of an Actor

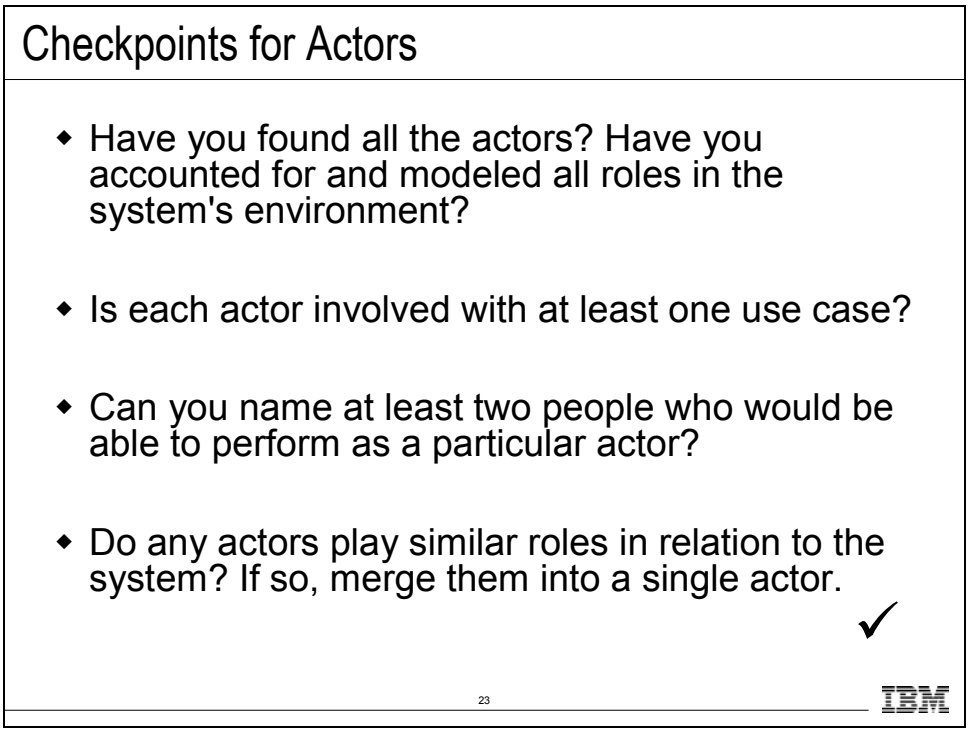


This is the information used to describe each actor. Some of the information contained in the brief description is:

- What or whom the actor represents.
- Why the actor is needed.
- What interests the actor has in the system.

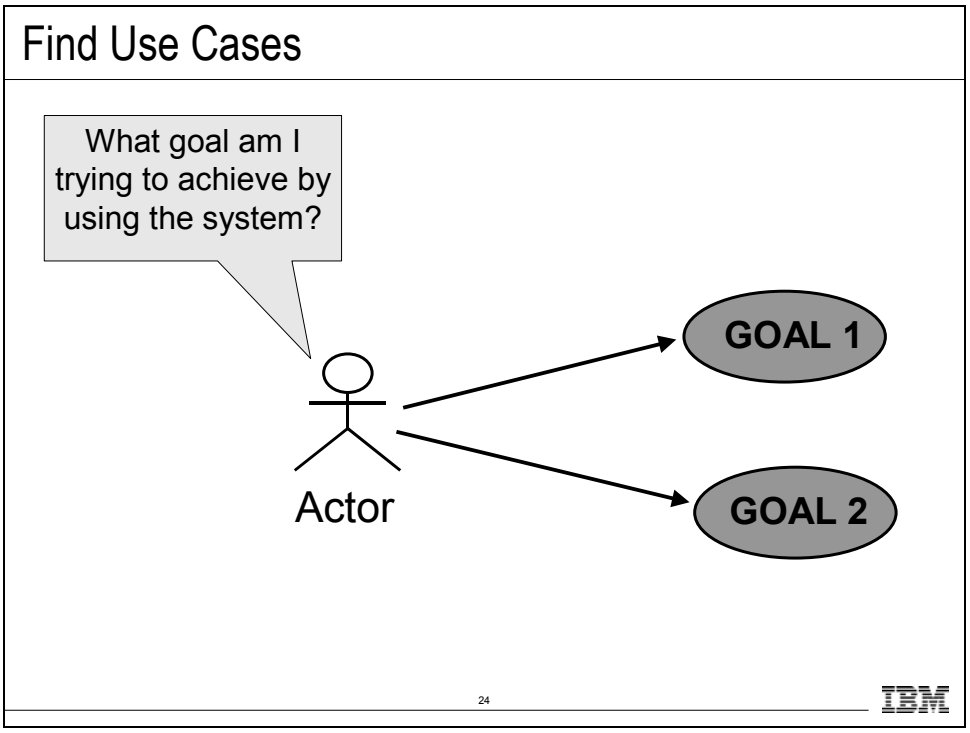
Some information about the actors is also shown on use-case diagrams. The use-case diagrams show the name of the actor and the actor's relationships with use cases.

Checkpoints for Actors



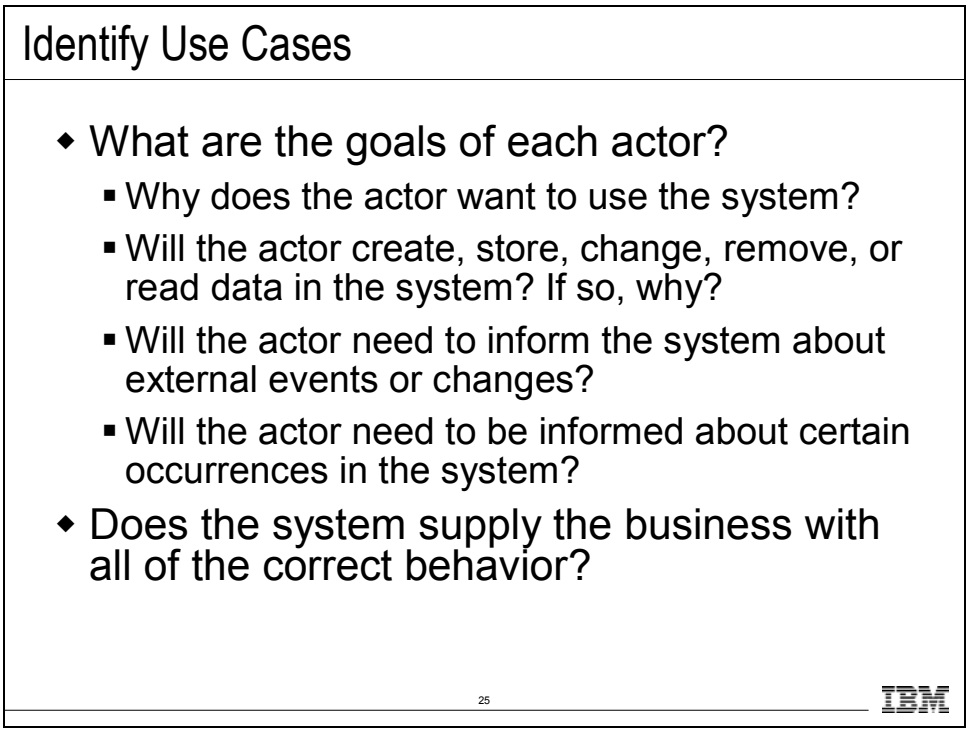
The above list of actor checkpoints is a summary of the list in the Rational Unified Process®.

Find Use Cases



Identifying the use cases is the next step in developing your use-case model, once the actors have been identified. Use cases describe what an actor wants a system to do that provides some value to the actor. Use this process to identify the use cases for each actor.

Identify Use Cases



Here are some questions that are useful to help you find the use cases in your system. The best way to find use cases is to consider what each actor requires of the system. Go through all the actors and identify the particular needs of each actor.

When you have made your first list of use cases, verify that all required functionality has been treated. Do not forget special use cases for system startup, termination, or maintenance. Also, do not forget to include use cases for automatically scheduled events. For example, a time-initiated job may run the payroll at midnight on the last day of each month. Use cases that concern automatically scheduled events are usually initiated by a special actor: the scheduler.

Try to keep all use cases at approximately the same level of importance. The notion of use-case levels as popularized by some authors is not recommended. It can lead to a functionally decomposed system.

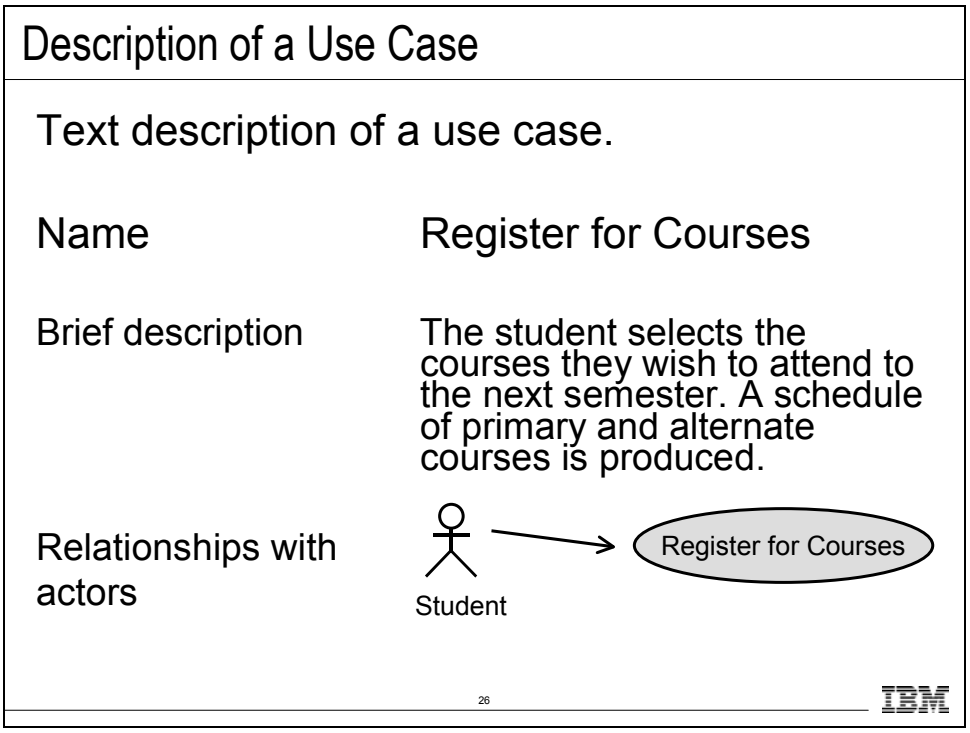
The above questions can help get the right granularity (refer also to the definition of a use case). Ask yourself:

- Would you ever want to perform a particular activity as an end result in itself?
- Are there any activities that should be kept together so you can review, modify, test, and document them together?
- Is the use case too complex? If it is, you may want to split it (a use-case report significantly longer than 10 pages may be a candidate).

Give each use case a name that indicates what an instance of the use case does. The name may have to consist of several words to be clearly understood.

Does it deliver something of value to multiple actors? If so, it may be too large.

Description of a Use Case



This is the information used to describe the use case in the use-case model:

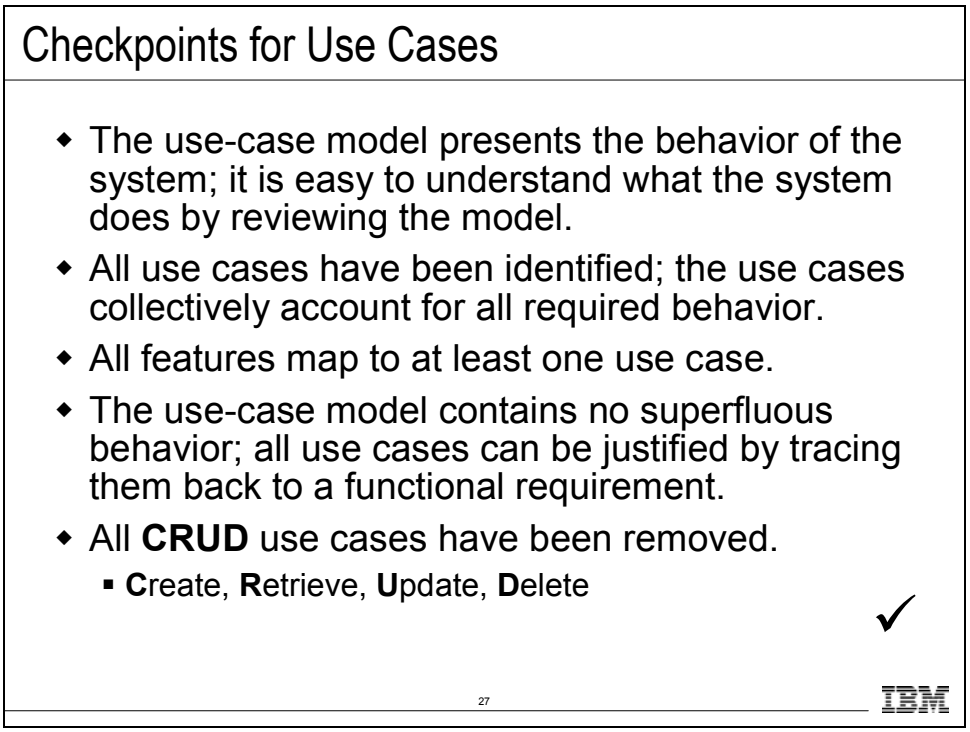
Name: The name of the use case.

Brief Description: A brief description of the purpose of the use case.

Relationships: The relationships (communicates-associations, include, extend and generalization) in which the use case participates.

Some information about the use cases is also shown on use-case diagrams in the Use-Case Model. The use-case diagrams show the name of the use case and the relationships with actors.

Checkpoints for Use Cases



Checkpoints (continued from slide)

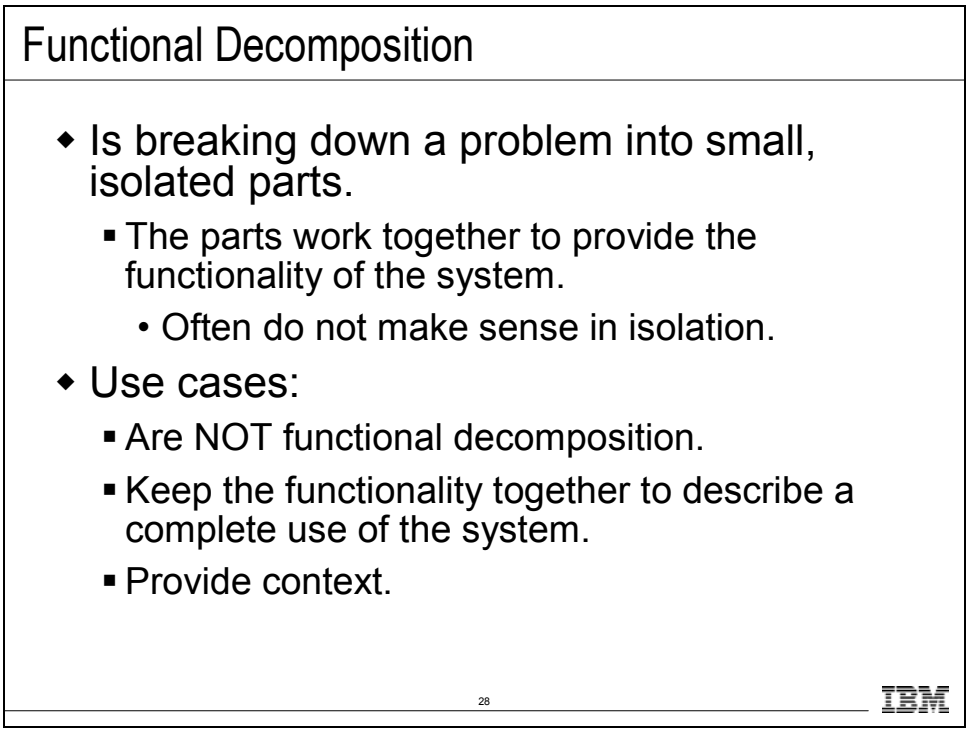
- Do the use cases have unique, intuitive, and explanatory names so that they cannot be mixed up at a later stage? If not, change their names.
- Do customers and users alike understand the names and descriptions of the use cases?
- Does the brief description give a true picture of the use case?
- Is each use case involved with at least one actor?
- Do any use cases have very similar behaviors or flows of events?

The list of use-case checkpoints is a summary of the list in the Rational Unified Process.

A CRUD use case is a Create, Read, Update or Delete use case. Data management use cases like these are another form of function decomposition. Managing the data like this is usually part of some higher goal (typically maintenance of information), and they should be combined into a single use case.

For example, instead of having four use cases: “Add Customer”, “Delete Customer”, Update Customer Information”, and “View Customer Details”. You could have a single use case named: “Maintain Customer Details”.

Functional Decomposition

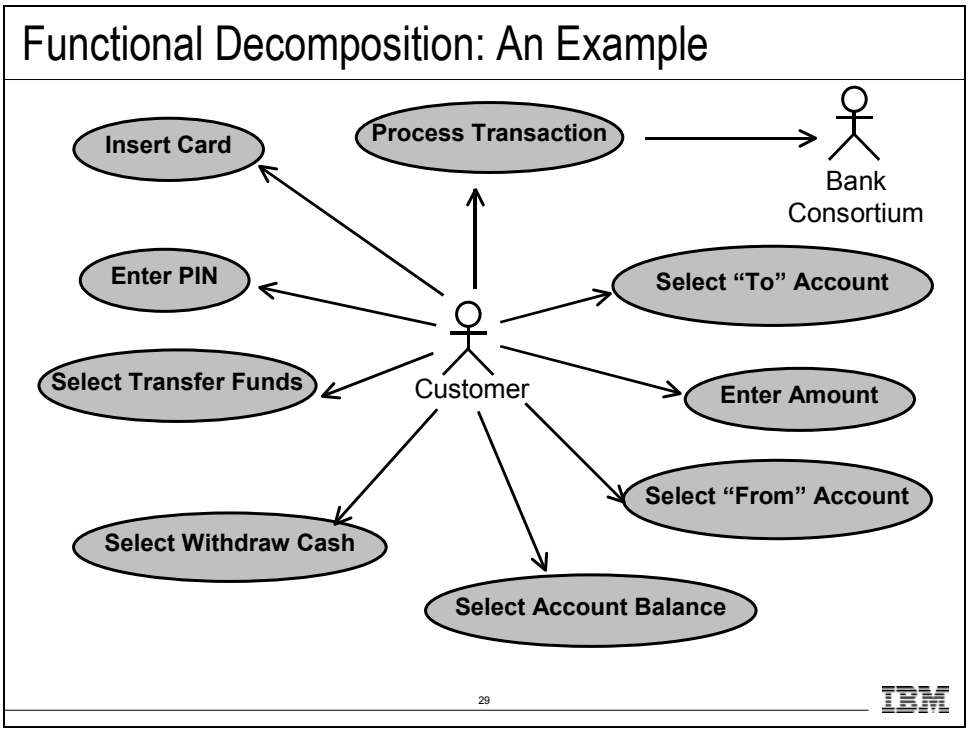


Humans deal with complex problems by breaking them up into small manageable pieces. The solutions to these pieces are then assembled to provide a complete solution to the original problem. When applied to computer science, this is called functional decomposition.

This microscopic approach means that the requirements for these pieces lose their context. The greater the number of smaller pieces you have, the greater the number of interfaces you need between these pieces and the greater the potential for misfit.

Use cases allow you to capture your requirements in a macroscopic form. They still contain the same level of detail as functional decomposition, but the requirements are put into context with each other and thus provide a technique that enables you to develop requirements that are more likely to be understood by your stakeholders as well as enabling you to deliver end-to-end functionality of the system.

Functional Decomposition: An Example



The diagram is an example of functional decomposition. The use cases look like a laundry list of steps that the actor performs instead of goals the actor is trying to achieve. In order to withdraw cash from the system, the actor needs to perform these use cases in a particular order. There is no way to show this required sequence.

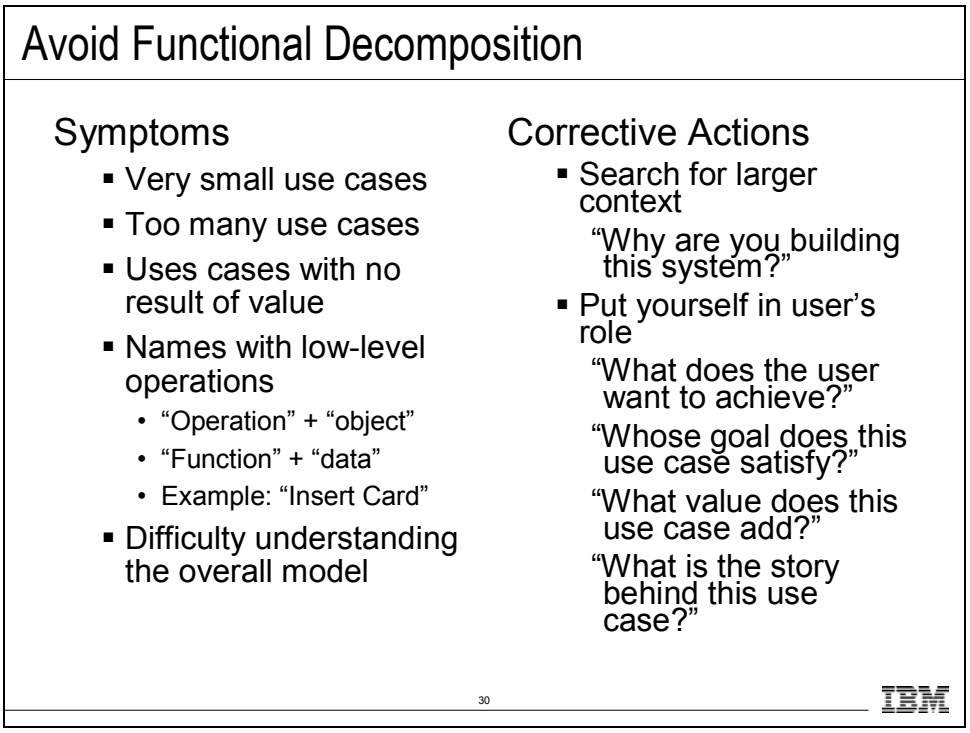
One of the biggest advantages of use cases over functional decomposition is that the requirements in the use cases have context. In the example, the requirements lose their context and their implementation becomes more difficult.

What is often not realized about functional decomposition is that the pain is not felt during the requirements gathering activities. Instead it is felt by the implementers and testers of the system. This is because the implementers do not see the relationship between the requirements and testers that are testing use cases end up writing more test cases.

The problems caused by this approach are:

- The requirements have no context, so they are harder to understand.
- Stakeholders have to read and cross-reference more isolated pieces of information, resulting in a more fragmented development and implementation experience.
- The goals of the actor are ambiguous.
- Implementers cannot see the relationship between the requirements.
- Testers have to write more test scripts (one per use case). These test scripts are "partial tests" because there is no end-to-end description of what the system must do.

Avoid Functional Decomposition



Use cases are not functional decomposition. You want to avoid functional decomposition. Use cases describe the complete use of the system to achieve a goal.

"Small" names are usually a symptom of too small use cases, tiny things that are operations. Nobody understands what the system can do for them by reading a lot of small, tiny bits that are not in context or in order.

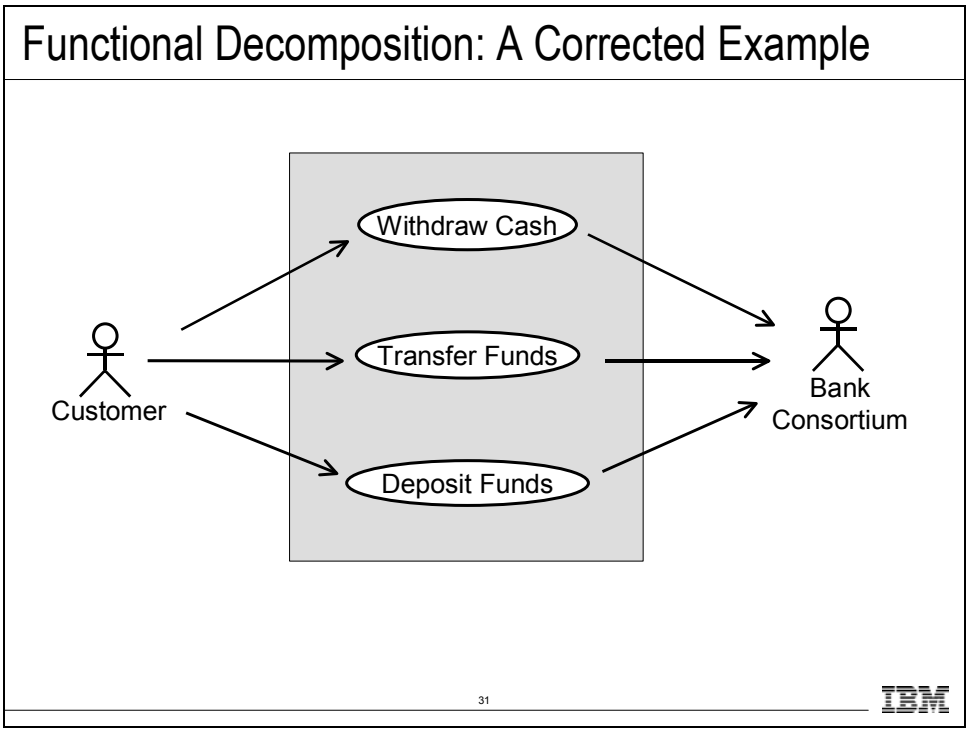
From a requirements analyst's point of view, functional decomposition may not appear to be a serious problem. Problems with functional decomposition manifest themselves later in the lifecycle, particularly in the project schedule.

If you have a functional decomposition, there will be more use cases to analyze, resulting in more code. More code means more bugs.

A functional decomposition may also affect your testing team, resulting in more test cases.

The sum of these means that your project takes longer.

Functional Decomposition: A Corrected Example



The example above is a corrected example of the functionally decomposed use-case diagram seen previously. The benefits of this approach over the former are:

- The requirements are in context so they are easier to understand.
- The goals of the actor are clearly visible.
- All of the requirements related to the three goals are contained in the same use-case specification.
- Implementers see the relationship between the requirements.
- Testers can write fewer test cases.

Exercise 3.1: Identify Actors and Use Cases

Exercise 3.1: Identify Actors and Use Cases

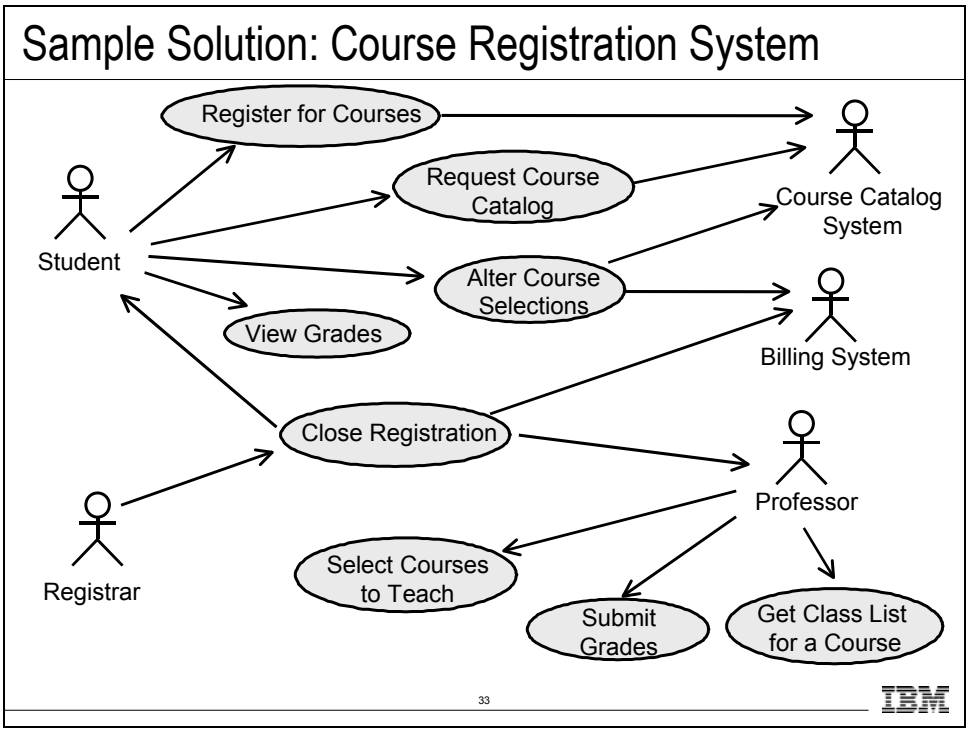
- Identify the actors who interact with the Course Registration System.
- Identify use cases for the system.
- Sketch a use-case diagram.
  - Refer to use-case and actor checkpoint slides.

32

IBM

See Student Workbook Exercise 3.1: Identify Actors and Use Cases.

Sample Solution: Course Registration System

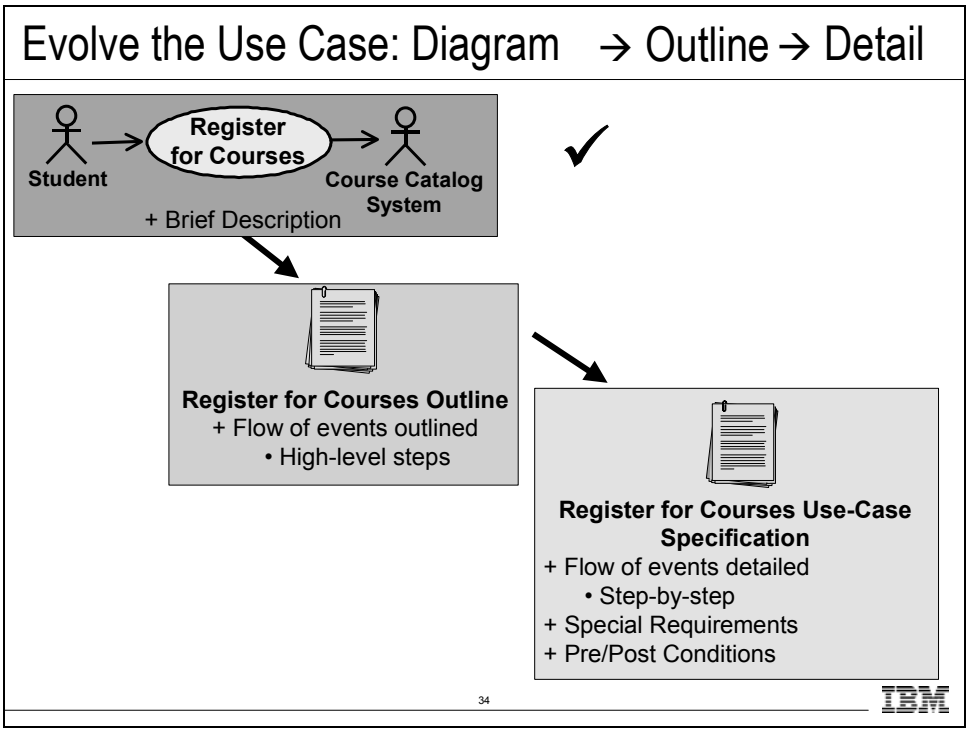


Here is a sample (not necessarily optimal) solution, which brings up a lot of questions that need to be answered by the customer or user:

- Are the use cases too small? For example, should we combine Register for Courses Use Case and Alter Course Selections Use Case and have an alternative flow for registering after the registration period?
- Does the diagram cover all activities? For example, in which use case do you think canceling too-small classes is done?
- How do you know what is done in each use case?

You might notice that the student and the professor interact with the system during Close Registration. Does this mean they have to be present when registration is closed? No, rather they are informed of the outcome via e-mail or post, etc. Another technique would have been to put the actor as an e-mail server or the postal service. There is no right or wrong answer; both are correct, and you should use whichever more clearly conveys your requirements.

Evolve the Use Case: Diagram

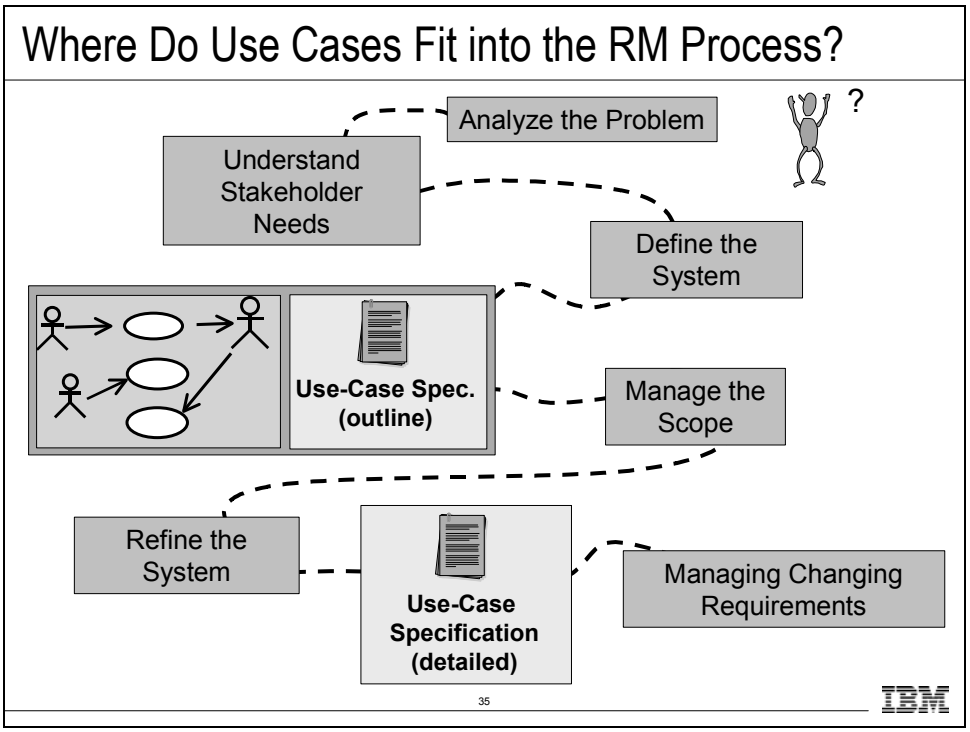


This module exposed you to use cases and use-case terminology. There are two more steps in creating your use-case model. After identifying the use cases, you should outline them to get a rough idea of what the system will do. Once the use cases have been outlined and everyone agrees that the general idea is correct, you progressively detail each use case until you develop a complete description of what the system must do.

You'll work more with use cases in the next few days when creating artifacts for the class project.

The Register for Courses diagram, outline, and specification are located in the case study section of the Student workbook.

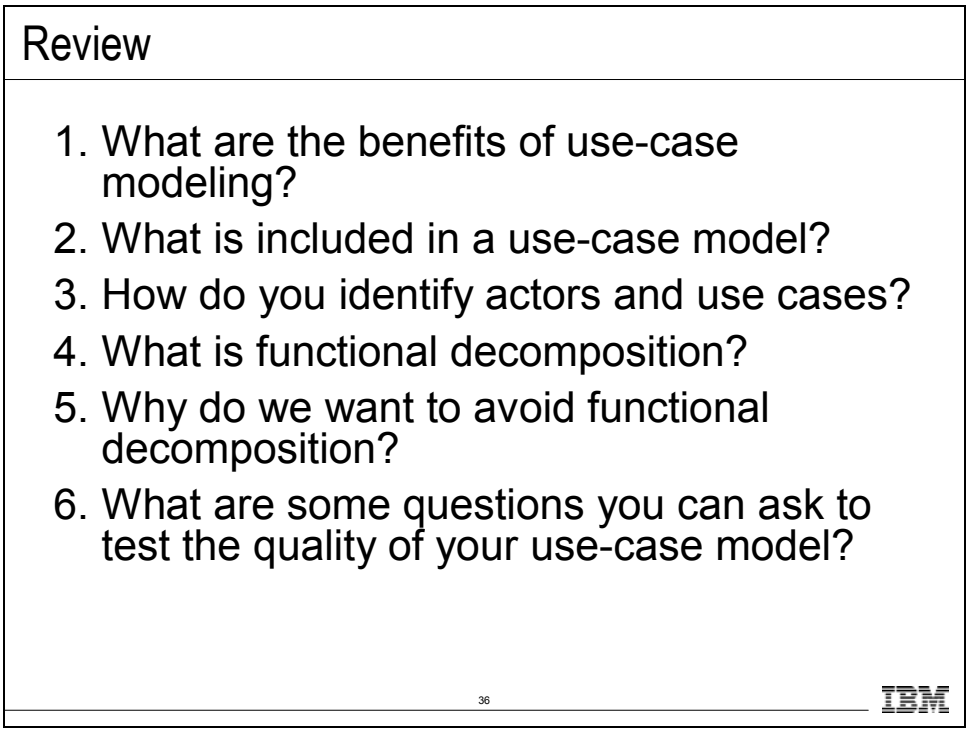
Where Do Use Cases Fit into the RM Process?



It is important to understand that use cases are only one tool in your armory for effective requirements management. Use cases are developed progressively over the course of a project, and there are many activities in the requirements management discipline that help you find the information that you want to capture in the use case model.

This module has given you a brief introduction to give you a flavor for what use cases are. The remaining modules show you how to effectively manage your requirements and where in the process you work with use cases.

Review



1.

2.

3.

4.

5.

6.