

1 Lecture 3: Use Case Modeling

Course: Software Engineering and Systems Analysis

Topic: Use Cases and User Goals

Recommended duration: 90 minutes

1.1 Learning objectives

By the end of this lecture, students should be able to define an actor and a use case, establish a system boundary, write a structured use case description, distinguish include from extend, and identify quality problems in use case models.

1.2 1. Purpose of use case modeling

A **use case** describes a goal that an external actor achieves by interacting with a system. Use case modeling focuses on externally visible behavior. It does not begin with database tables or programming classes. It begins with what a person, organization, device, or external system needs from the system under study. Use cases are especially useful for discovering functional requirements. They provide a shared language for customers, analysts, testers, designers, and developers because the central question is simple: “What does the actor need to accomplish?”

1.3 2. Actors and the system boundary

An **actor** is a role played by an external entity. A person may play several roles, and the same role may be played by many people. For example, “Bank Customer” is an actor role; it is not the name of a specific person.

The **system boundary** separates behavior supplied by the system from behavior supplied by external actors. An actor is outside the boundary. A use case is inside it. External payment gateways, identity providers, and hardware devices may also be actors when the system communicates with them.

1.4 3. Naming use cases

Use case names should express an actor goal with a verb and an object. Examples include “Submit Assignment,” “Reserve Room,” and “Issue Refund.” Avoid names that describe internal mechanisms, such as “Call Database” or “Run Validation,” unless those mechanisms are themselves an externally meaningful goal.

A good use case has a clear result. “Manage Account” may be too broad because it could include registration, password reset, address update, and deletion. Splitting it into focused goals often creates requirements that are easier to estimate, test, and prioritize.

1.5 4. Structured use case description

A diagram provides an overview, but the detailed requirements normally appear in a textual description. A practical template includes the following sections:

Use case: Submit Assignment

Primary actor: Student

Goal: Store an assignment before the deadline

Preconditions: The student is authenticated and enrolled

Trigger: The student selects Submit

Main success scenario:

1. The system displays the submission form.
2. The student uploads a file and confirms submission.
3. The system validates the file and deadline.
4. The system stores the submission and shows a receipt.

Alternative flows:

- A1. The file type is not allowed; the system explains the rule.
- A2. The deadline has passed; the system rejects the submission.

Postconditions: A timestamped submission record exists.

The main success scenario should be written as a sequence of actor–system interactions. It should be understandable without knowledge of the implementation.

1.6 5. Relationships between use cases

An **association** connects an actor to a use case that the actor participates in. An **include** relationship indicates behavior that is required and reused by several use cases. For example, “Place Order” may include “Authenticate User” when authentication is always required in that context.

An **extend** relationship represents optional or conditional behavior inserted into a base use case at a defined extension point. For example, “Apply Discount” may extend “Place Order” only when a valid coupon is supplied.

Generalization can be used when one actor or use case is a specialized form of another. These relationships should be used sparingly. If the meaning can be explained more clearly in ordinary text, do not force it into a complex diagram.

1.7 6. Example: online food delivery

The system boundary contains Browse Restaurants, Place Order, Track Delivery, and Request Refund. A Customer actor interacts with the first three use cases. A Restaurant Staff actor confirms orders. A Delivery Partner actor updates delivery status. A Payment Gateway actor authorizes payment.

“Place Order” may include “Authorize Payment.” “Request Refund” may extend “Place Order” when a customer reports a qualifying problem, although the exact choice depends on the business rule. The model must reflect the actual requirement rather than a memorized pattern.

1.8 7. Common quality problems

Use cases should not be written as user-interface screen lists. “Display Page 1” and “Click Button” are usually design details, not user goals. They should also not be written as vague system capabilities such as “Process Data.” The actor, goal, and expected result should be identifiable.

Another problem is mixing unrelated goals into one enormous use case. Large use cases become difficult to review and lead to unclear acceptance tests. Conversely, excessively small use cases can describe every keystroke and create unnecessary maintenance work.

1.9 8. From use cases to tests

Each main scenario and alternative flow can inform acceptance tests. The preconditions define the test setup. The steps define the interaction. The postconditions define the expected result. This creates traceability from a user goal to a verifiable system behavior.

For “Submit Assignment,” tests should cover a valid file, an unsupported file type, a missing file, a deadline boundary, and a network or storage failure. A use case is not a complete test specification, but it is a strong source of test scenarios.

1.10 9. Practice activity

Write a structured use case for “Withdraw Cash” in an ATM system. Include the primary actor, supporting actors, preconditions, main success scenario, at least three alternative flows, and postconditions. Then draw the system boundary and identify which external entities are actors.

1.11 Summary

Use case modeling captures system behavior from the perspective of external goals. High-quality use cases define a clear boundary, name meaningful goals, describe interactions in user language, and include alternatives and outcomes. The resulting model supports requirements validation and acceptance testing.

1.12 References