Specifying Requirements
with Use Case Diagrams
SOURCE OF REQUIREMENTS

Initial requirements come from the customer, by:
- Documents, Meetings, reports

Advanced requirements come from the analysts, after studying:
- Scope and price
- Feasibility (technological, organizational etc)
- Prototypes

Final requirements are stabilized in an iterative process.
REQUIREMENTS VS. DESIGN

Requirements:
- **What** the system should do
- More abstract

Design:
- **How** the system should do it
- More detailed
TYPES OF REQUIREMENTS

Visible Functional Requirements

• “The system will deliver cash to the customer”

• “Cash will be delivered after card was taken out”

Qualitative Requirements

• “The authorization process will take no more than 1 sec”

Hidden Requirements

• “Database maintenance processes will occur every night”
A use case is a contract of an interaction between the system and an actor.

A full use-case model comprise of:

- A diagram, describing relations between use-cases and actors.
- A document describing the use case in details
USE CASE DIAGRAM OBJECTIVE

1. Create a semi-formal model of the functional requirements
2. Analyze and define:
   - Scope
   - External interfaces
   - Scenarios and reactions
WHAT MAKES GOOD USE-CASE SPECIFICATION?

Lack of ambiguity

- Each requirement must be interpreted in a single manner.

Completeness

- The collection of all use cases is everything that can be done to/with the system.

Consistency

- Requirements should not conflict with each other. If there are, tradeoffs must be detected and discussed.

Avoid design

- Requirements should raise a need, not answer it.
USE CASES AS MEANS OF COMMUNICATION

The use case should stimulate a discussion about what the system should do, mainly with people who are outside of the development team.
OUTLINE

Introduction

Use Case Diagrams

Writing Use Cases

Guidelines for Effective Use Cases
A SIMPLE EXAMPLE

- **Handle Message**
- **Handle Call**
- **Bill Management**

**Actors**
- **Cellular Phone**
- **Customer**
- **External Phone Company**

**System boundary**

**Association**

**Use Case**

**Introduction | Diagrams | Writing | Guidelines**
FINDING ACTORS

External objects that produce/consume data:

- Must serve as sources and destinations for data
- Must be external to the system

Humans

Machines

External systems

Organizational Units

Sensors
ACTORS CAN BE GENERALIZED

The child actor inherits all use-cases associations

Should be used if (and only if), the specific actor has more responsibility than the generalized one (i.e., associated with more use-cases)
Linking enables flexibility in requirements specification

- Isolating functionality
- Enabling functionality sharing
- Breaking functionality into manageable chunks

Three mechanism are used:

- Include
- Extend
- Inheritance
USE-CASE LEVELS

Base Use Case: Used directly by the user

User goals

Sub-functionality

Perform Sale

Choose Products

Fill-in billing info

Alistair Cockburn “Writing Effective Use Cases”
THE “INCLUDE” CONSTRUCT

Include is used when:

- Decomposing complicated behavior
- Centralizing common behavior

The base use case explicitly incorporates the behavior of another use case at a location specified in the base.

Example:

```
Perform Sale <<include>> Fill-in billing info
```

Example
The base use case can incorporate another use case at certain points, called extension points.

Note the direction of the arrow

- The base use-case does not know which use-case extends it
EXAMPLE: AMAZON

Product Page

Shopping Cart

Review Writing
GENERALIZATION BETWEEN USE-CASES

The child use case inherits the behavior parent use case:

- The interaction (described in the textual description)
- Use case links (associations, include, extend, generalization)

Child use-case can substitute parent Use case

Overriding occurs through the textual description

Example

1. Transfer call to available representative
2. Mark representative as busy
3. Start record call
4. Stop record call
5. Log call details
6. Mark representative as free
GENERALIZATION HAZARDS

Combining generalizations of actors and use-cases can be dangerous

**Bad:** Undergrad can submit thesis

**Good:** Only graduate student can submit thesis
EXAMPLE: EASTERN STATE UNIVERSITY (ESU) REGISTRATION SYSTEM.

1. Professors indicate which courses they will teach on-line.
2. A course catalog can be printed
3. Allow students to select on-line four courses for upcoming semester.
4. No course may have more than 10 students or less than 3 students.
5. When the registration is completed, the system sends information to the billing system.
6. Professors can obtain course rosters on-line.
7. Students can add or drop classes on-line.
EXAMPLE: EASTERN STATE UNIVERSITY (ESU) REGISTRATION SYSTEM.
OUTLINE

Introduction

Use Case Diagrams

Writing Use Cases

Guidelines for Effective Use Cases
**Bottom-up Process**

Starting with throwing all scenarios on the page, and then combining them:

- save
- print
- load
- Save as preview
- Bullets format
- Paragraph format
- Font format

**Top-down Process**

Starting with an overview of the system, and then splitting Use-cases:

- File actions
- Formattin g actions
- Viewing Actions
Most of the analysis process are actually Combined
COMBINING PROCESSES

Number Limit:

- The diagram should have between 3 to 10 base use-case. No more than 15 use cases (base + included + extending).

Abstraction:

- All use-cases should be in similar abstraction levels.

Size:

- Use cases should be described in half a page or more.

Interaction:

- Use-cases which are carried out as part of the same interaction.
DIVIDING PROCESSES

Size:

- If a use-cases takes more than a page, consider include/extend

Weak dependency:

- If the dependency between two parts of a use-case is weak, they should be divided.

UC
MORE GUIDELINES

Factor out common usages that are required by multiple use cases

• If the usage is required use <<include>>
• If the base use case is complete and the usage may be optional, consider use <<extend>>

A use case diagram should:

• contain only use cases at the same level of abstraction
• include only actors who are required
WHEN ARE WE DONE?

When every actor is specified.

When every functional requirement has a use-case which satisfies it.

A tractability matrix can help us determine it:

Use Cases

Requirements
SUMMARY

✓ Introduction
  ➤ to the Unified Modeling Language (UML)
  ➤ To Use Case Diagram

✓ Use Case Diagrams
  ➤ Dual presentation of use-cases
  ➤ Include, Extend, Inheritance

✓ Writing Use Cases
  ➤ Preconditions & Post-conditions
  ➤ Main scenario vs. Alternative Flow

✓ Guidelines for Effective Use Cases