

# Use cases

---



**SoftEng**  
<http://softeng.polito.it>

Version 2.1 - April 2012

## Contents

---

- Use cases
  - ◆ History
  - ◆ Types
  - ◆ Writing
- Use case diagrams

# History

---

- Introduced by Ivar Jacobson in 1992
  - ♦ codified a visual modeling technique
- Found widespread adoption in the 1990's
  - ♦ initially in the Object-Oriented community

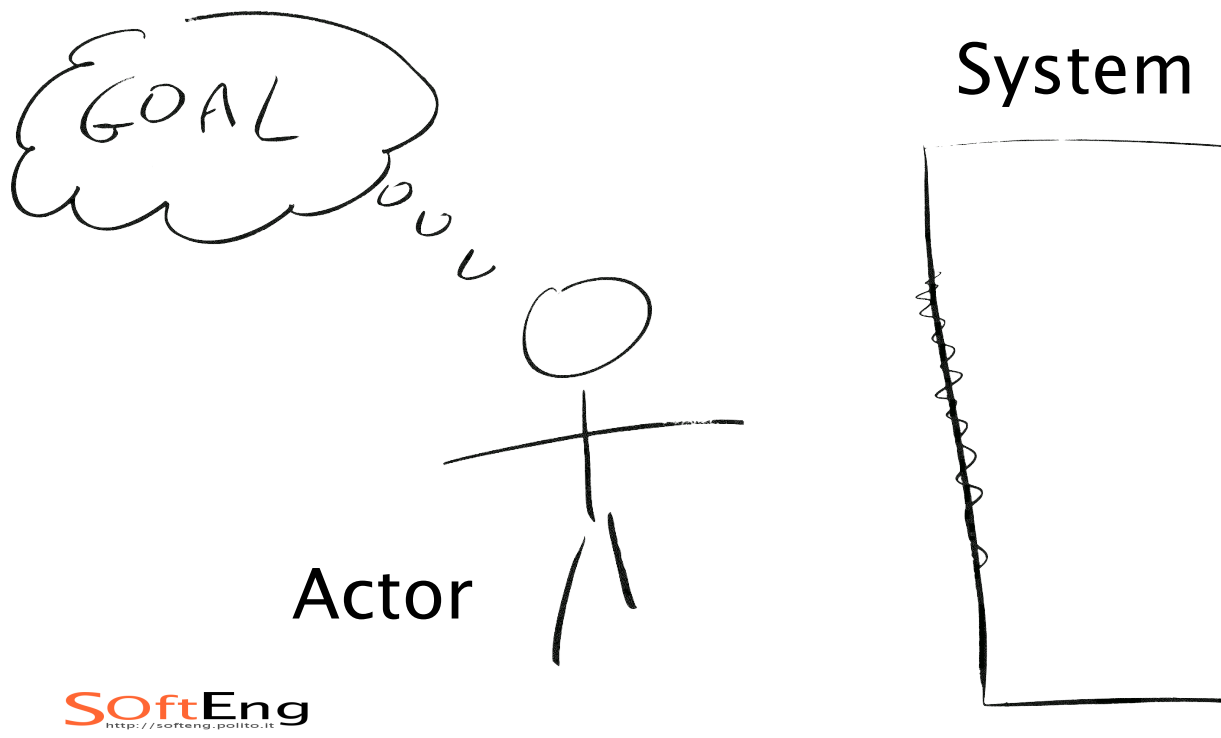
# Definition

---

A use case defines a **goal-oriented** set of **interactions** between external actors and the system under consideration.

# Key elements

---



# Key elements

---

- The **actor** involved
  - ♦ type of user that interacts with the system
- The **system** being used
  - ♦ treated as a black-box
- The functional **goal** that the actor achieves using the system
  - ♦ the reason for using the system

# Description

---

- Use cases describe the sequence of interactions between actors and the system
  - ♦ to deliver the service that satisfies the goal
- They also include possible variants of this sequence
  - ♦ either successful or failing

# Actor I

---

- Actors are parties **outside the system** that interact with the system.
- An actor may be a class of users, roles users can play, or other systems.
  - ♦ A **primary actor** is one having a goal requiring the assistance of the system.
  - ♦ A **secondary actor** is one from which the system needs assistance.

# Actor II

---

- The use case cares only what is the relationship of the actor to the system
- The goal must be of value to the (primary) actor:
  - ♦ “Enter PIN code” is not
  - ♦ “Withdraw cash” is

# Actor III

---

- External entities:
  - ♦ Which persons interact with the system (directly or indirectly)?
    - Don't forget maintenance staff!
  - ♦ Will the system need to interact with other systems or existing legacy systems?
  - ♦ Are there any other hardware or software devices that interact with the system?
  - ♦ Are there any reporting interfaces or system administrative interfaces?

# Essential use cases

---

- Intended to capture the essence of problems through technology-free, idealized, and abstract description
- Independent of the technical implementation of the system.
  - ◆ Not “the user presses the enter button”
  - ◆ Better “the user confirms his choice”

# Conventional use case

---

User action	System response
insert card	read magnetic strip request PIN
enter PIN	verify PIN display transaction option menu
press key	display account menu
press key	prompt for amount
enter amount	display amount
press key	return card
take card	dispense cash
take cash	

---

# Essential use case

---

User action	System response
identify self	verify identity offer choices
choose	
take cash	dispense cash

## Degree of detail

---

- Brief
  - ◆ few sentences summarizing the use case
- Casual
  - ◆ few paragraphs of text elaborating the use case in the form of a summary or story
- Detailed
  - ◆ formal document based on a long template with fields for various sections

# Granularity

---

- Cockburn (2001) identified three different goal levels:
  - ♦ **summary level** is the 50,000 feet perspective,
  - ♦ **user-goal level** is the sea-level perspective,
  - ♦ **subfunction** is the underwater perspective.

## Summary level

---

- Summary level use cases:
  - ♦ are large grain use cases that encompass multiple lower-level use cases; they provide the context (lifecycle) for those lower-level use cases.
  - ♦ they can act as a table of contents for user goal level use cases.

# Summary level

---

- Use Case:
  - ♦ Manage Funds By Bank Account
- Scope:
  - ♦ Bank Accounts and Transactions System
- Intention in Context:
  - ♦ The intention of the Client is to manage his/her funds by way of a bank account. Clients do not interact with the System directly; instead all interactions go through either: a Teller, a Web Client, or an ATM, which one depends also on the service.
- Primary Actor: Client
- Main Success Scenario:
  - ♦ 1. Client opens an account.
  - ♦ Step 2 can be repeated according to the intent of the Client
  - ♦ 2. Client performs task on account.
  - ♦ 3. Client closes his/her account

# User Goal Level

---

- User-goal level use cases:
  - ♦ describe the goal that a primary actor or user has in trying to get work done or in using the system.
  - ♦ *are usually done by one person, in one place, at one time; the (primary) actor can normally go away happy as soon as this goal is completed.*

# User Goal Level

---

- Use Case: Deposit Money
- Scope: Bank Accounts and Transactions System
- Intention in Context:
  - ♦ The intention of the Client is to deposit money on an account. Clients do not interact with the System directly; instead, for this use case, a client interacts via a Teller. Many Clients may be performing transactions and queries at any one time.
- Primary Actor: Client
- Main Success Scenario:
  - ♦ 1. Client requests Teller to deposit money on an account, providing sum of money.
  - ♦ 2. Teller requests System to perform a deposit, providing deposit transaction details\*.
  - ♦ 3. System validates the deposit, credits account for the amount, records details of the transaction, and informs Teller.

# User Goal Level

---

- Extensions:
  - ♦ 2a. Client requests Teller to cancel deposit: use case ends in failure.
  - ♦ 3a. System ascertains that it was given incorrect information:
    - 3a.1. System informs Teller; use case continues at step 2.
  - ♦ 3b. System ascertains that it was given insufficient information to perform deposit:
    - 3b.1. System informs Teller; use case continues at step 2.
  - ♦ 3c. System is not capable of depositing (e.g. transaction monitor of System is down)\*\*:
    - 3c.1. System informs Teller; use case ends in failure.
- Notes:
  - \* a hyperlink to a document that contains data details and formats.
  - \*\* this is an example of an IT infrastructure failure, we only write it in a use case if there is a corresponding project constraint that states a physical separation, e.g., transaction supported by a legacy system.

# Subfunction level

---

- Subfunction level use cases
  - ♦ provide “execution support” for user–goal level use cases; they are low–level and need to be justified, either for reasons of reuse or necessary detail

# Subfunction level

---

- Use Case: Identify Client
- Scope: Automatic Teller Machine (ATM for short)
- Intention in Context:
  - ♦ The intention of the Client is to identify him/herself to the System. A project (operational) constraint states that identification is made with a card and a personal identification number (PIN).
- Primary Actor: Client
- Main Success Scenario:
  - ♦ 1. Client provides Card Reader with card; Card Reader informs System of card details\*.
  - ♦ 2. System validates card type.
  - ♦ 3. Client provides PIN to System.
  - ♦ 4. System requests BAT System to verify identification information\*.
  - ♦ 5. BAT System informs System that identification information is valid, and System informs Client.

# Structure – inclusion

---

- An include relationship between two use cases means that the sequence of behavior described in the included (or sub) use case is included in the sequence of the base (including) use case.
  - ♦ Including a use case is thus analogous to the notion of calling a subroutine

# Structure – extension

---

- The extends relationship provides a way of capturing a variant to a use case.
  - ♦ Extensions are not true use cases but changes to steps in an existing use case.
- The extends relationship includes
  - ♦ the condition that must be satisfied if the extension is to take place, and
  - ♦ references to the extension points defined in the extended use case

# Scope

---

- System = Enterprise
  - ♦ Business use cases
- System = Software System
  - ♦ System use cases
- System = Component of the sw

# Template

---

- SWEED template
  - ♦ One column (no table)
  - ♦ Sequenced: Numbered steps (Dewey decimal numbering system) and extensions to main scenario use alphabetic letters to differentiate from main steps

# Template

---

- **Use Case:**
  - ♦ Name of the use case. This is the goal stated by a short active verb phrase.
- **Scope:**
  - ♦ The scope of the “system” being considered (black-/white- box and enterprise/system/component).
- **Level:**
  - ♦ Summary, User-goal, or Subfunction

# Template

---

- **Intention in Context:**
  - ♦ A statement of the primary actors intention and the context within which it is performed.
- **Primary Actor:**
  - ♦ The primary actor of the use case.
- **\*Stakeholders' Interests:**
  - ♦ The list of stakeholders and their key interests in the use case.
- **\*Precondition:**
  - ♦ What we can assume about the current state of the system and environment.

# Template

---

- **\*Minimum Guarantees:**
  - ◆ How the interests of the stakeholders are protected in all circumstances.
- **\*Success Guarantees:**
  - ◆ The state of the system and environment if the goal of the primary actor succeeds.
- **\*Trigger:**
  - ◆ What event starts the use case.

# Template

---

- **Main Success Scenario:**
  - ◆ `<step_number> "." <action_description>`  
The numbered steps of the scenario, from trigger to goal delivery, followed by any clean-up.
  - ◆ Conditions and alternatives are shown in the extension part.
- **\*Extensions:**
  - ◆ `<step_altered> <condition> ":" <action_description> or <sub-use_case>`

# Template

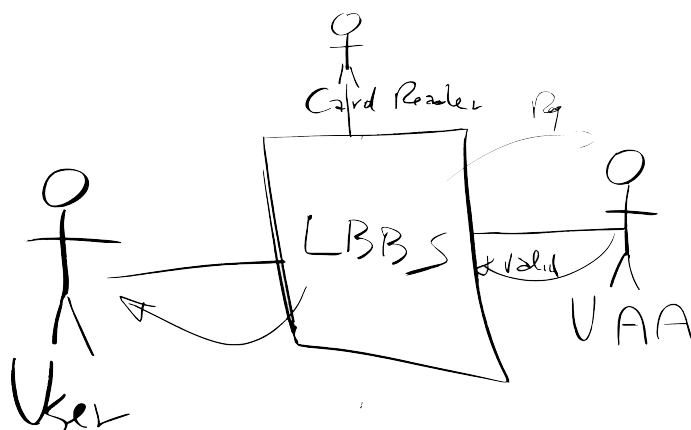
---

- Steps can be nested. Dewey numbers are then used, e.g. 3a.1
- An extension always refers to a step of the Main Success Scenario.
- An extension step takes place in addition to the respective main step, notation: 2 ||,
- or as an alternative, notation: 2a.
- An extension might correspond to regular behavior, exceptional behavior that is recoverable, or unrecoverable erroneous behavior.

## Example: LBBS

---

- An automated library book borrowing system (LBB system). The goal is to relieve the librarians from processing book loans.



# Example: Check out books

---

- **Scope:** Library Book Borrowing System
- **Level:** User Goal
- **Intention in Context:**
  - ◆ The intention of the User is to check-out books from a library. Only one User can check-out books at any one time.
- **Primary Actor:**
  - ◆ User (becomes Member once s/he has identified him/herself with the System)

# Example: Check out books

---

- **Main Success Scenario:**
  1. User requests System to check-out books.
  2. User identifies him/herself to System.
  - Step 3 is repeated for each book that is checked-out by Member.*
  3. Member registers book with System.
  4. Member indicates to System that s/he has finished checking out books.
  5. System records all books registered by Member as on loan, requests Printer to print out a receipt\* for the session, and puts itself in a state to receive the next User.

# Example: Check out books

---

- Extensions:
  - 2a. User fails to identifies him/herself with System: use case ends in failures.
  - 3a. System informs Member that s/he has reached his/her maximum number of books allowed on loan; use case continues at step 4.
  - 3||a. Member requests System to remove book from the books that are checked out:
    - 3||a.1. Member identifies book to System.
    - 3||a.2a. System identifies book and removes it from the list of books registered by Member; use case continues from where it was interrupted.
    - 3||a.2b. System fails to identify book; use case continues from where it was interrupted.

# Example: Check out books

---

- (3-4)a. Member requests System to cancel check-out.
- (3-4)a.1. System removes all books that were registered by Member, and puts itself in a state to receive the next User; use case ends in failure.
- (3-4)b. System times-out waiting for input from Member:
  - (3-4)b.1. System removes all books that were registered by Member, and puts itself in a state to receive the next User; use case ends in failure.

# Applying use cases

---

- Identify your actors:
  - ♦ who will be using the system?
- Identify their goals:
  - ♦ what will they be using the system to do?
- Identify key scenarios:
  - ♦ in trying to achieve a specific goal, what distinct outcomes or workflows might we need to consider?
- Describe in business terms the interactions for a specific scenario
- Validate with user

# Warnings

---

- Use case alone are not the requirements
- Use cases should be based on some form of conceptual model or glossary
- Use cases should not include details about the user interface

# Typical errors

---

- Undefined or inconsistent system boundary
- Take System's viewpoint instead of Actor's
- Inconsistent Actor names
- Use cases refer to too many actors
  - ♦ Spider's web

# Typical errors

---

- Too long specifications
- Confused specifications
  - ♦ Use of conditional logic
  - ♦ Attempt to describe algorithms
- Actor non fully entitled
- Customer not understanding
- Never ending use cases

# Use case diagrams

---

- Use cases can be (collectively) represented using UML use-case diagrams
- Often the “brief” level of detail can be useful in this context

## Elements of use case diagrams

---



Actor

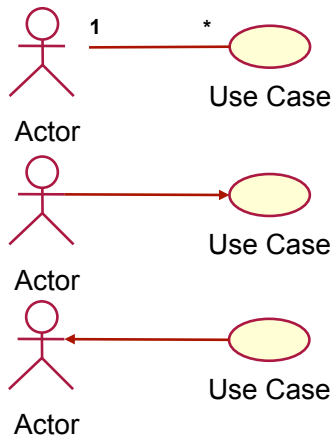
- Someone (user) or something (external system, hardware) that
  - ♦ Exchanges information with the system
  - ♦ Supplies input to the system, or receives output from the system



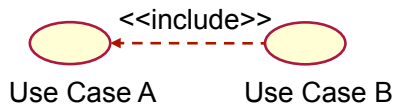
Use Case

- A functional unit (functionality) part of the system

# Relations



- Association models:
  - ♦ Which actors participate in a use case
  - ♦ Where execution starts
  - ♦ Adornments (e.g. multiplicity, direction) allowed

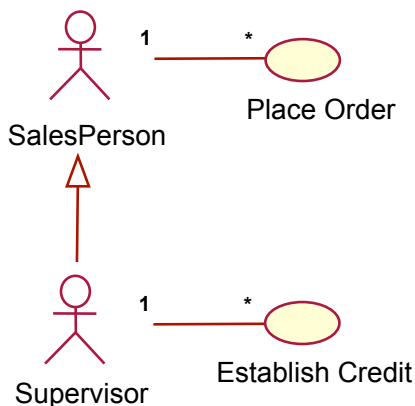


- Include
  - ♦ Models that functionality A is used in the context of functionality B (one is a phase of the other)

# Relations

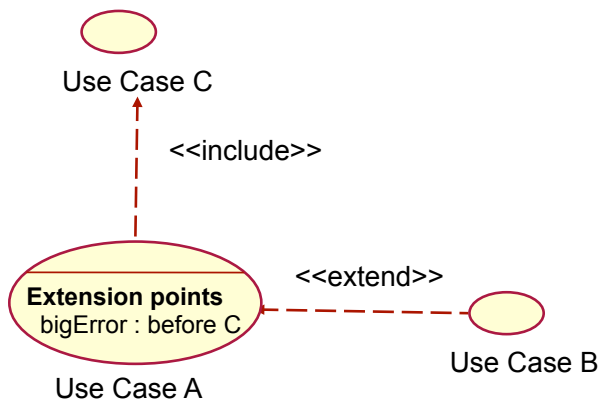


- Generalization
  - ♦ Defines functionality B as a specialization of functionality A (e.g. a special case)



- Generalization
  - ♦ A generalization from an actor B to an actor A indicates that an instance of B can communicate with the same kinds of use-case instances as an instance of A

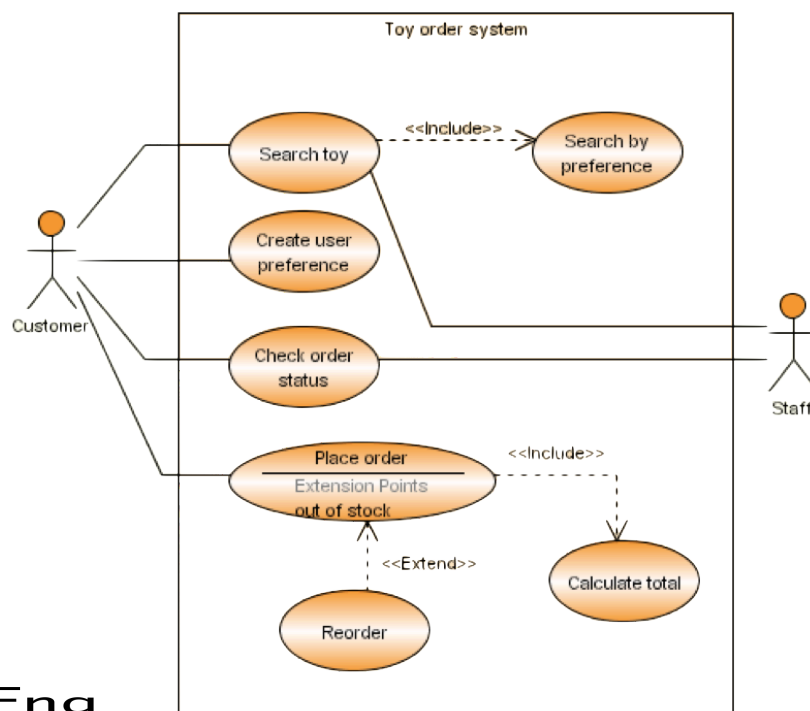
# Relations



## Extension

- ◆ An extend relationship from use case A to use case B indicates that an instance of use case B may be augmented by the behavior specified by A
- ◆ The behavior is inserted at the location defined by the extension point (name : where) in B, which is referenced by the extend relationship

# Use case – Example



# Pros

---

- Use case modeling is generally regarded as an excellent technique for capturing the functional requirements of a system.
- Use cases discourage premature design
- Use cases are traceable.
- Use cases can serve as the basis for the estimating, scheduling, and validating effort.

# Pros

---

- Use cases are reusable within a project.
  - ◆ The use case can evolve at each iteration from a method of capturing requirements, to development guidelines to programmers, to a test case and finally into user documentation.
- Use case alternative paths capture additional behaviour that can improve system robustness.

# Pros

---

- Use cases are useful for scoping.
  - ◆ Use cases make it easy to take a staged delivery approach to projects; they can be relatively easily added and removed from a software project as priorities change.
- Use cases have proved to be easily understandable by business users, and so have proven an excellent bridge between software developers and end users.

# Pros

---

- Use case specifications don't use a special language. They can be written in a variety of styles to suit the particular needs of the project.
- Use cases are concerned with the interactions between the user and the system.
  - ◆ UI designers can get involved in the development process either before or in parallel with software developers.

# Pros

---

- Use cases put requirements in context, they are clearly described in relationship to business tasks.
- Use case diagrams help stakeholders to understand the nature and scope of the business area or the system under development.

# Pros

---

- Test Cases (System, User Acceptance and Functional) can be directly derived from the use cases
- Use cases are critical for the effective execution of Performance Engineering

# Cons

---

- Use case flows are not well suited to easily capturing
  - ◆ non–interaction based requirements of a system (such as algorithm or mathematical requirements) or
  - ◆ non–functional requirements (such as platform, performance, timing, or safety–critical aspects)
- Use cases templates do not automatically ensure clarity. Clarity depends on the skill of the writer(s).