

L7 - Introducing Use Cases

CS3028 - Principles of Software Engineering

Ernesto Compatangelo

Department of Computing Science



7.1 Reminding past issues and mapping them to current topics

Introducing use cases

Where are we now?

Software development paradigms

- ⇒ The Unified Process (UP) paradigm
 - ⇒ UP phases and UP disciplines (activities) within each phase
 - ⇒ Inception (first UP phase)
 - ⇒ Activities and deliverables during inception
 - ⇒ Elaboration (second UP phase)
 - ⇒ Moving from inception to elaboration
 - ⇒ Requirements expressed as use cases

⇒

7.2 A few things about requirements

A few words of wisdom about requirements

- Requirements elicitation, modelling, and analysis mainly focuses on **functional requirements** because they are easier to specify
- ... but non-functional requirements **are equally important**
- It is common practice to specify all non-functional requirements together with the functional one they refer/are linked to
- Requirements elicitation is a recursive activity — you neither get ALL the requirements nor you get ALL OF THEM RIGHT the first time
- Elicitation is a typical pen-and-paper activity where the most valuable CASE tool is your brain. Automated support only helps you to **document** requirements, never to extract, organise, and model them
- Requirements should be (i) **ranked** and (ii) **prioritised** as soon as elicited to separate relevant issues, (to be addressed immediately) from marginal ones (to be addressed late, time permitting)

Why requirements elicitation?

- The primary goal of requirements elicitation is to understand and state accurately users' requirements for a software product
- Secondary goals include:
 - To communicate the current understanding of the proposed system
 - To prevent expensive mistakes
 - To prepare the basis for stating a design problem
 - To state the conditions for system acceptance

7.3 The UP approach to requirements

How are requirements organised in UP artifacts?

- **Use case model:** a set of typical scenarios of using a system. There are primarily for functional (behavioural) requirements
- **Supplementary specification:** everything not in the use cases. This is primarily for all non-functional requirements. It is also the place to record functional features not expressed (or expressible) as use cases
- **Glossary:** defines any noteworthy terms and elements (e.g., attributes, parameters of operation calls, report layouts) as well as data-related requirements (e.g., validation rules, acceptable values)
- **Vision:** summarises high-level requirements in the use case model and supplementary specification, and the business case for the project as a short executive overview document (white paper)
- **Business rules (a.k.a. domain rules):** requirements or policies required in the domain or business, which all applications need to conform to (e.g. government tax laws).

A step-to-step approach to UP requirements elicitation

Eliciting requirements means performing the following activities:

- **Identify actors**, *i.e.*, the different 'users' of the system
- **Identify scenarios**, *i.e.*, concrete examples of the future system in use in a specific situation
- **Identify use cases**, *i.e.*, abstractions describing all possible scenarios and thus the scope of a functionality
- **Refine use cases**, detailing each use case and describing the behaviour of the system in case of errors and exceptional conditions
- **Identify relationships among use cases**, finding dependencies among use cases and factoring out common functionalities to ensure specification consistency
- **Identify nonfunctional requirements**, *i.e.*, aspects that are visible to the user, but not directly related functionalities

Functional requirements: use case narrative

Expanded essential use case narrative for Register for Classes

Use case:	Register for Classes
Actors:	Student
Purpose:	Register a student for classes and record the student's schedule.
Overview:	A Student requests the sections of class desired for a term. The system adds the Student to each section if there is space available. On completion, the system provides the Student with a list of the classes in which he or she is enrolled.
Type:	Essential
Preconditions:	Class schedule must exist. Student is known by the system.
Postconditions:	Student was enrolled in the section.
Special Requirements:	Student must get a system response within 10 seconds.

Flow of Events

ACTOR ACTION

1. This use case begins when a Student desires to register for classes.
2. The Student provides the Student's identifier and a list of the department code, course number, and section number for each section desired.
4. On completion of entry of the section requests, the Student indicates that the request is complete.
6. The Student receives the student class list.

SYSTEM RESPONSE

3. Adds the student to the section if there are seats available.
5. Produces a student class list for the Student.

Alternative Flow of Events

Line 3: Invalid department code and course number entered. Indicate error. Return to Step 2.
 Invalid section number entered. Indicate error. Return to Step 2.
 No seats remaining. Inform the Student. Return to Step 2.

Navigation icons: back, forward, search, etc.

The very notion of use case

- A use case is an abstraction describing all possible scenarios of system usage in a particular situation, thus modelling a functionality (behavioural) provided by the (software) system
- A scenario is a specific sequence of actions and interactions between actors and the system; it is also called a *use case instance*
- A use case is a collection of ALL the related success and failure scenarios that describe an actor using a system to support a goal
- Use cases are not diagrams but text documents. Use-case modelling is the act of writing text, not drawing diagrams

Navigation icons: back, forward, search, etc.

A closer look at use cases

- An **actor** is something with behaviour, such as a person (identified by role), external computer system, or organisation
- A **scenario** is a specific sequence of actions and interactions between actors and the system; it is also called a *use case instance*
- The scenario records the steps, of which there are four kinds:
 - Step 1, which indicates the trigger event that starts the scenario
 - An interaction between actors
 - A validation (usually by the system)
 - A state change by the system (e.g., recording or modifying something)
- A **use case** is a collection of ALL the related success and failure scenarios that describe an actor using a system to support a goal
- **Use cases are not diagrams but text documents.** Use-case modelling is the act of writing text, not drawing diagrams

7.4 Detailed focus on use cases

How to create a use case

- Identify the **system boundaries**, the **primary actors** (humans, computers) and all the **possible different scenarios** (e.g. the expected course of action and all the alternatives)
- List the **goals** of each primary actor
- Write each step in the form of **subject-verb-direct object** (and sometimes preposition-indirect object)
- Clarify **initiator and receivers** of action
- Write the steps from the viewpoint of an independent observer
- Write each step at **same level of abstraction**
- Ensure that the use case has a **sensible set of steps**
- Write repeating instructions **after the steps** to be repeated

How many different kinds of actors are there?

- **Primary actor** - has user goals fulfilled through using services of the system under discussion.
Why identify? To find user goals, which drive the use cases.
- **Supporting actor** - provides a service (for example, information) to the system under discussion. The automated payment authorisation service is an example. Often a computer system, but could be an organisation or person.
Why identify? To clarify external interfaces and protocols.
- **Offstage actor** - has an interest in the behaviour of the use case, but is not primary or supporting; for example, a government tax agency.
Why identify? To ensure that all necessary interests are identified and satisfied. Offstage actor interests are sometimes subtle or easy to miss unless these actors are explicitly named.

Heuristics for actor identification Once the name of the most important system functions has been found, actors are identified by answering the following questions:

1. Which users are supported by the system through the functions?
2. Which users execute the system functions?
3. Which users perform secondary functions associated to the system functions, *e.g.*, maintenance and administration?
4. What external HW or SW system will interact with the system functions being currently specified?

Heuristics for scenario identification Once actors and function names have been found, scenarios are identified by answering the following questions:

1. What are the tasks that the actor wants the system to perform?
2. What information does the actor access? Who creates that data? Can it be modified or removed? By whom?
3. Which external changes does the actor need to inform the system about? How often? When?
4. Which events does the system need to inform the actor about? With what latency?

Use case specification

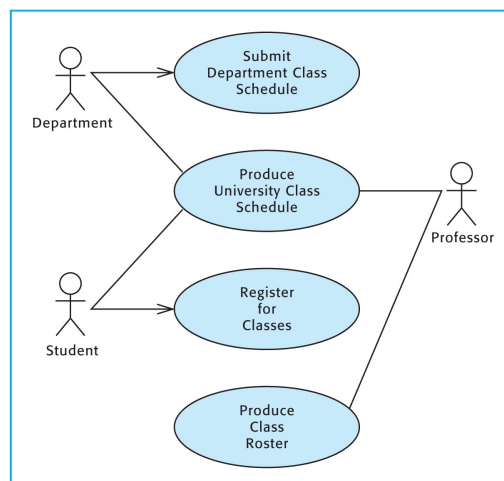
Once actors, function names, and scenarios have been found, use cases are specified using a template as follows

- ① The **use case name** must reflect the function name
- ② **Initiating** (active) and **participating** (passive) **actors** must be stated
- ③ **Preconditions** (entry conditions) and **postconditions** (exit conditions) must be stated
- ④ **Expected flow of events** must be specified, showing actors / actions / triggering events (on the left) and system reactions (on the right)
- ⑤ All **unexpected flows of events** must be specified one by one by stating where in the above list one expected condition goes wrong, and what happens then (termination, return to previous action, etc.)
- ⑥ The non-functional reqs. associated to the use case must be stated

7.5 Use case diagrams

Use case modelling: UML use case diagram example



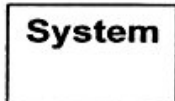

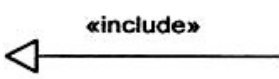


Example of a use case diagram



How to create UML use case diagrams

- Start with **system boundary**
- Place elements **in order** to be easy to read
- Place **actors** on the diagram
- Conclude by **connecting actors to use cases** by lines (links)

Notations for UML use case diagrams

<p>AN ACTOR:</p> <ul style="list-style-type: none"> • Is a person or system that derives benefit from and is external to the system • Is labeled with its role • Can be associated with other actors using a specialization/superclass association, denoted by an arrow with a hollow arrowhead • Is placed outside the system boundary 	 <p>Actor/Role</p>
<p>A USE CASE:</p> <ul style="list-style-type: none"> • Represents a major piece of system functionality • Can extend another use case • Can include another use case • Is placed inside the system boundary • Is labeled with a descriptive verb-noun phrase 	 <p>Use Case</p>
<p>A SYSTEM BOUNDARY:</p> <ul style="list-style-type: none"> • Includes the name of the system inside or on top • Represents the scope of the system 	 <p>System</p>
<p>AN ASSOCIATION RELATIONSHIP:</p> <ul style="list-style-type: none"> • Links an actor with the use case(s) with which it interacts 	
<p>AN INCLUDE RELATIONSHIP:</p> <ul style="list-style-type: none"> • Represents the inclusion of the functionality of one use case within another • The arrow is drawn from the base use case to the used use case 	 <p>«include»</p>
<p>AN EXTEND RELATIONSHIP:</p> <ul style="list-style-type: none"> • Represents the extension of the use case to include optional behavior • The arrow is drawn from the extension use case to the base use case 	 <p>«extend»</p>
<p>A GENERALIZATION RELATIONSHIP:</p> <ul style="list-style-type: none"> • Represents a specialized use case to a more generalized one • The arrow is drawn from the specialized use case to the base use case 	

7.6 Associated non-functional requirements

Non-functional requirements — glossary

- Non-functional requirements capture behavioural properties of a system. Use a **glossary** to specify terms and definitions as well as data-related requirements (e.g., validation rules, acceptable values)
- In the UP, the Glossary also plays the role of a data dictionary, a document that records data about the data (metadata)
- Term attributes include:
 - aliases
 - description
 - format (type, length, unit)

Supplementary non-functional specification

- Use a **supplementary specification** to capture and report everything not in the use cases or glossary. It is also the place to record functional features not expressed or expressible as use cases.
- Elements of the Supplementary Specification also include:
 - reports
 - development constraints (for example, process or development tools)
 - other design and implementation constraints
 - standards (technical, safety, quality)

7.7 Preparing for the topics ahead

Introducing use cases

Next lecture...

Elaboration requirements models

More specifically, we will focus on:

- Functional (activity) model
- Structural (class & object) model