

COMP 3111

SOFTWARE ENGINEERING

LECTURE 9

SYSTEM REQUIREMENTS CAPTURE

SYSTEM REQUIREMENTS CAPTURE OUTLINE

✓ System Requirements Capture Overview

- Life-cycle Role
- Importance of Requirements Capture
- Why Requirements Capture is Difficult

➔ System Requirements Capture Activities

✓ Capture Data Requirements: **Domain Modeling**

✎ Capture Functional Requirements: **Use-Case Modeling**

- Capture Nonfunctional Requirements
- Validate System Requirements

USE-CASE DETAILED SPECIFICATION

- **use case name** (present-tense, verb phrase in active voice)
- **brief description**
- **participating actors** (can show using a use-case diagram fragment)
- **preconditions** (if any)
- **flow of events** (the required *normal* sequence of actions)
- **postconditions** (if any)
- **alternative flows** (if any), which describe:
 - **optional** actions performed *in addition to* the normal actions
 - **variant** actions performed *to replace* some normal actions
 - **exceptional** actions performed *to handle abnormal situations* (e.g., invalid input, system errors, etc.)
- **special (nonfunctional) requirements** (if any)

USE-CASE DETAIL: PRECONDITION

A precondition is a statement about the state the system and/ or the actor(s) must be in for the use case to be performed.

 A precondition is only needed if a use case cannot be performed unless certain conditions are true.

- Preconditions are written so that the use-case descriptions are as independent of each other as possible.

 A precondition states “what” should be true, not “how” it is made true.

- The preconditions are “necessary, but not sufficient” for the use case to be performed.

 Starting a use case **always** requires an actor to do something.

USE-CASE DETAIL: POSTCONDITION

A *postcondition* is a statement about the state the system is in at the conclusion of a use case.

 **A postcondition is only needed if the system state when the use case ends is important to an actor of the use case.**

- A postcondition is needed when:
 - The completion of the use case leaves the system in a particular state that *may need to be a precondition for another use case*.
 - The *possible outcomes of the use case are not obvious*, so that it will be difficult for developers, testers, or users to understand the result of performing the use case.

 **Postconditions help ensure that the reader understands what the result of executing the use case has been.**

USE-CASE DETAIL: FLOW OF EVENTS

A flow of events is a precise, but easy to read, description of the sequence of actions needed to accomplish a use case.

What the system and the actor should do to perform the use case (not how it is done; ignore use-case interactions).

Basic flow: Most common path (i.e., what *normally* happens).

Exactly one.

One complete sequence of actions from start to end.

Always required!

Alternative flows: Optional, variant or exceptional behaviour.

Zero or more.

Infrequently taken paths.

Start with the basic flow; add alternative flows as needed.
(So as to specify everything the user and system should do.)

USE CASE DETAIL: FLOW OF EVENTS SPECIFICATION

- A sequence of short steps that are numbered, declarative and time-ordered.

*This is a suggested
format only!*

n. The *something* *some-action*. (e.g., 3. The actor enters a name.)

Branching: If

n. If *boolean-expression*
 n.1. *declarative-statement*
 n.2. *declarative-statement*
n.3. ...
n+1.

Repetition: For

n. For *iteration-expression*
 n.1. *declarative-statement*
 n.2. *declarative-statement*
n.3. ...
n+1.

- ✓ A use-case specification should be kept as **simple as possible**.
- ✓ The narrative should be **event-response oriented**.
- ✓ Repetition should be **used sparingly!**

Repetition: While

n. While *boolean-expression*
 n.1. *declarative-statement*
 n.2. *declarative-statement*
n.3. ...
n+1.

USE-CASE DETAIL: EXTENSION POINT

An *extension point* is a named place in the flow of events where additional behaviour can be inserted.

- Kinds of extension points:
 - a single location: occurs at a single place.
 - a set of discrete locations: occurs at multiple places.
 - a region: a pair of extension points, suitably named to make clear that they are a matched pair, which delineates a set of locations between them.

Extension points are primarily used for defining alternative flows.

USE-CASE DETAIL: ALTERNATIVE FLOW

An alternative flow of events describes optional, variant or exceptional behaviour.

 **Allows infrequently used functionality to be added incrementally to the basic flow.**

- Alternative flows should be:
 - **numbered** A1, A2, ..., An
 - **uniquely named within a use case** with active names that indicate their purpose.
- Kinds of alternative flow:
 - **specific alternative flow** starts at a *specific named point* in another flow of events (i.e., basic, subflow or alternative) of the use case.
 - **bounded alternative flow** can only occur *between two extension points* of the use case.
 - **general alternative flow** can *start at any point* within a use case.

USE-CASE DETAIL: ALTERNATIVE FLOW (cont'd)

- The first line of an alternative flow has the form:

- For specific alternative flow

At {extension point} when <some event occurs> ...

At {extension point} if <some condition is true> ...

Depend on some event
or condition occurring.

- For bounded alternative flow

At any point between {extension point} and {extension point} ...

- For general alternative flow

At any time in the flow of events ...

USE-CASE DETAIL: ALTERNATIVE FLOW (cont'd)

- The last line of an alternative flow's flow of events, and any other exit point within it, must state explicitly where the actor resumes the flow of events.
 - For optional behaviour, this will usually be the original extension point where the alternative flow was triggered.
 - For variant behaviour, this is usually another extension point elsewhere in the flow of events.
 - For exceptional behaviour, this can be either the original or another extension point, or the use case may end, which should be explicitly stated in the alternative flow's flow of events.

USE-CASE DETAIL: SUBFLOW

A *subflow* is a segment of behaviour within a flow of events that has a clear purpose and is “atomic.”

✎ It is a technique to structure a use-case description to improve its readability.

✎ Excessive nesting of subflows should be avoided.

- Subflows should be:
 - numbered S1, S2, ..., Sn;
 - uniquely named within a use case with active names that indicate their purpose.
- A subflow is referenced using: Perform subflow <subflow name>


**A subflow *is not* a separate use case!
It is an integral part of the use case.**

USE-CASE DETAIL:

HOW MUCH DETAIL IS ENOUGH?

- A use case is a technique for *mitigating risk*—specifically the risk that someone might misunderstand what the system is required to do to produce value for its users.
- Therefore, the use cases must contain **enough detail so that all stakeholders are satisfied** that the system is defined in sufficient detail to allow the *right* system to be built.

 **The basic flow should unambiguously describe the required behaviour.**

 **Keep asking: “What does this mean?” until all ambiguities have been resolved.**

USE-CASE INTERACTION & DECOMPOSITION

- Use cases **do not directly communicate with each other!**

 **Use cases interact via the state of the system.**

- This is by design since

 **Each use case is intended to be independent of other use cases.**

- **Be careful** not to decompose the system behaviour into **too small use cases**, such as:

- Select Products
- Enter Order Information
- Enter Shipping Information
- Enter Payment Information
- Confirm Order

Is each of these things independently valuable?

Would you ever do just one without the others?

SYSTEM REQUIREMENTS CAPTURE USE-CASE MODELING EXERCISE