COMP 3111
# SOFTWARE ENGINEERING

SYSTEM REQUIREMENTS CAPTURE
PART 2

USE-CASE MODELING

● Captures the _____ *from the users' point of view.*

● Developed incrementally *in cooperation with the domain model*.

● Helps in: – capturing data and functional requirements.

  – planning iterations of development.

  – validating the system.

☞ **USE CASES DRIVE THE DEVELOPMENT EFFORT.**

> All **required functionality** is described in the **use cases**.

USE-CASE MODELING: ACTORS
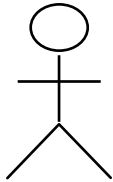
actor name

An *actor* represents something **outside** the system that interacts *directly* with it.

Can be a person or another system.

Provides input to or receives output from the system.

A role a user can play → multiple roles per user;
multiple users per role.

☞ **Actors are usually the source for discovering use cases.**

An actor is a stereotype of a UML class:

an actor is a classifier; a specific user or system is an instance.

USE-CASE MODELING: IDENTIFYING ACTORS

- Who or what uses the system?
- What roles do they play in the interaction?
- Who gets and provides information to the system?
- What other systems interact with this system?
- Who installs, starts and shuts down, or maintains the system?

☞ It is possible to have both a **domain model class** and an **actor** that **represent the same thing.**

☞ **Input/output devices** are **NEVER** actors!

For each actor, briefly describe the role it plays when interacting with the system.

ASU Course Registration System

System Requirements Capture

Use-case Model

Actors

The analysis and solution of this example will be posted in the Lecture Examples section of the course's web page *after* the lecture.

ASU USE-CASE MODEL: ACTORS

At the beginning of each term, students may request a course catalogue containing a list of course offerings needed for the term. Information about each course, such as instructor, department, and prerequisites are included to help students make informed decisions.

The new system will allow students to select four course offerings for the coming term. In addition, each student will indicate two alternative choices in case a course offering becomes filled or is canceled. No course offering will have more than forty students or fewer than ten students. A course offering with fewer than ten students will be canceled. Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the term.

Instructors must be able to access the online system to indicate which courses they will be teaching, and to see which students signed up for their course offerings.

For each term, there is a period of time that students can change their schedule. Students must be able to access the system during this time to add or drop courses.

USE-CASE MODELING: USE CASE
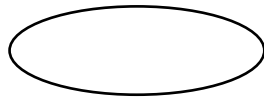
use-case name

A *use case* is a specific way of using the system by performing some part of its functionality.

Describes the _____ that takes place between an actor and the system and what the system must do.

☞ Considered from the actor's viewpoint.

Constitutes a *complete sequence* of events/actions.

Always initiated by an actor.

● Initially, only consider normal sequence of events/actions.

☞ Ignore alternative events/actions.

USE-CASE MODELING: SCENARIO

> **A *scenario* is a concrete, focused, informal description of a *single use* of the system from the viewpoint of a *single actor*.**

☞ **It is an actual attempt to carry out a use case.**

Note: There are two viewpoints of use-case modeling:

1. top-down: start with use cases, refine with scenarios.

2. bottom-up: start with scenarios, abstract to use cases.

☞ **In reality, the use-case specifier uses both viewpoints.**

> A use case is a stereotype of a UML class:
>
> use case is a classifier; scenario is an instance

USE-CASE MODELING:
IDENTIFYING USE CASES AND SCENARIOS

- What are the tasks that an actor wants the system to perform?
- What information does an actor access (create, store, change, remove or read) in the system?
- Which external changes does an actor need to inform the system about?
- Which events does an actor need to be informed about by the system?
- How will the system be supported and maintained?

☞ **State a use case name from the perspective of the actor as a present-tense, verb phrase in active voice.**

☞ **Provide a description of the purpose of the use case and an outline of its functionality.**

☞ **Use application domain terms in descriptions (i.e., from the glossary/data dictionary).**

ASU USE-CASE MODEL: USE CASES

At the beginning of each term, students may request a course catalogue containing a list of course offerings needed for the term.

functionality:

Information about each course, such as instructor, department, and prerequisites are included to help students make informed decisions.

functionality:

The new system will allow students to select four course offerings for the coming term.

functionality:

ASU USE-CASE MODEL: USE CASES

In addition, each student will indicate two alternative choices in case a course offering becomes filled or is canceled.

functionality:

No course offering will have more than forty students or fewer than ten students.

functionality:

A course offering with fewer than ten students will be canceled

functionality:

Once the registration process is completed for a student, the registration system sends information to the billing system so the student can be billed for the term.

functionality:

ASU USE-CASE MODEL: USE CASES

Instructors must be able to access the online system to indicate which courses they will be teaching, and to see which students signed up for their course offerings.

functionality:

For each term, there is a period of time that students can change their schedule.

functionality:

Students must be able to access the system during this time to add or drop courses.

functionality:

WHAT IS A GOOD USE CASE?

A use case typically represents a major piece of functionality that is complete from beginning to end.

A use case must deliver something of value to an actor.

Generally, it is better to have longer and more extensive use cases than smaller ones.

We want real and complete use cases, not several sub-cases of a use case.

USE-CASE MODELING EXAMPLE

The following are the requirements for a web-based system to computerize the management of the sale and rental of videos for a video shop.

- The system must be able to handle both physical and digital videos.

- It must be able to record which videos are sold and rented and by whom.

- For sold videos, the quantity sold should be recorded; for physical video rental, which copy is rented and when it is due back should be recorded.

- The system should keep track of overdue rentals of physical videos and send email notices to customers who have videos overdue.

- There will be a customer membership option for an annual fee, which will entitle a member to discounts (10%) on the sale and rental of videos.

- Members should be able to make reservations for physical video rentals either in person at the shop, by telephone or via the Web.

- A member can reserve at most five physical videos at any one time, but there is no limit on how many physical videos a member or nonmember can rent at any one time.

- As an added feature, the shop would like to allow customers (either members or nonmembers) to input, via the Web, mini-reviews (up to 100 words) and a rating (from 1, lowest, to 10, highest) of videos they have purchased or rented.

USE-CASE MODELING EXAMPLE (cont'd)

- These reviews should be anonymous if the customer so wishes (i.e., customers can specify whether they want their name to be made known when other customers browse the reviews).

- A sales clerk should be able to enter and update the following information about all customers (members or nonmembers): name, address, phone number, age, sex, and email address.

- Members are assigned a membership number by the shop when they become members and a password, which allows them to change their personal information and to buy and rent digital videos via the Web.

- The shop manager should be able to generate various reports on the sale and rental of videos.

- A sales clerk should be able to sell and rent physical videos and process the return of rented physical videos.

- When selling or renting physical videos, a sales clerk must be able to look up customer information and determine whether the customer is a member.

- A sales clerk must be able to enter basic information about a video (i.e., video id, title, leading actor(s), director, producer, genre, synopsis, release year, running time, selling price, and rental price).

USE-CASE MODELING EXAMPLE (cont'd)

From the video sale and rental shop requirements statement:

a)   **identify all actors and their required functionality.**

b)   **group the functionality into use cases and show the uses cases and their related actors in a use-case context diagram.**

USE-CASE MODELING: GENERALIZATION

Since actors and use cases are classifiers, they can be generalized.

Validate User

Employee

Sales clerk   Manager

Check Password   Scan Finger
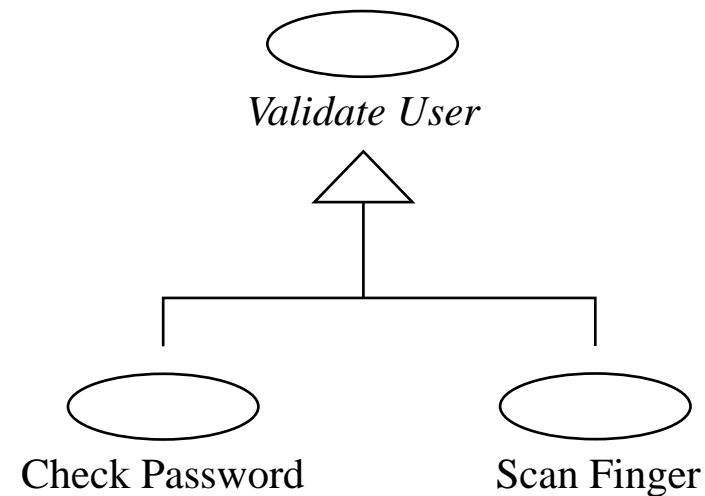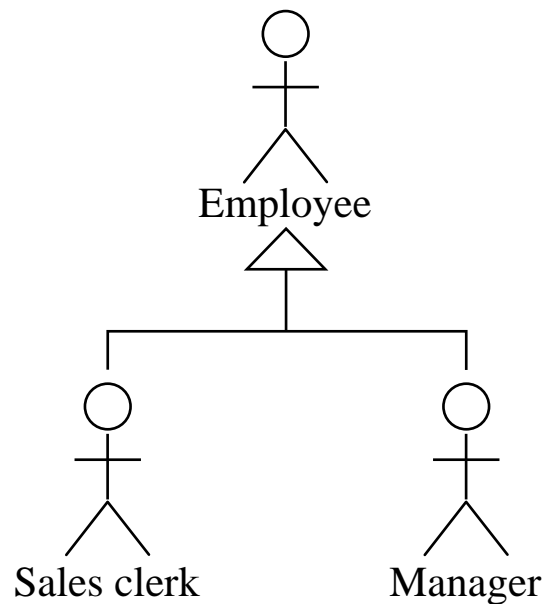
Do not use use-case generalization in your project. There is no need to do so and it is often used incorrectly.

The use of generalization represents a design decision!

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 preformed.

☞ 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 might do.)

USE CASE DETAIL:
FLOW OF EVENTS SPECIFICATION

This is a suggested format only!

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

  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, …, A*n*
  - 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, …, S*n*;

– 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: WRITING GUIDELINES

- Start with an outline of the flow of events.
- The boundary of the system should be clear.
- Name use cases with verb phrases that indicate what the user is trying to accomplish.
- Name actors with noun phrases.
- When first referring to an actor, precede the actor name with the identifier "actor".
- A use case should describe a *complete user transaction* (i.e., a complete sequence of events from beginning to end).
- Describe how the flow starts and ends.
- Describe what data is exchanged between the actor and the use case.
- Describe the flow of events, not only the functionality. To enforce this, start every action with "The actor …" or "The system …".
- *Do not describe details of the user interface,* unless they are necessary to understand the behaviour of the system.
- Describe what the system does, *but be careful not to describe how* the system is designed.
- Detail the flow of events—all "what" questions should be answered.
- Describe things clearly enough that an outsider could easily understand them.

USE-CASE DETAIL: WRITING GUIDELINES

- Use straightforward vocabulary (simple terms) where possible.
- Write short, concise sentences.
- Avoid adverbs, such as *very*, *more*, *rather*, and the like.
- Avoid vague terminology, such as *information*, *etc.*, *appropriate*, *required*, *relevant*, and *sufficient*.
- Use correct punctuation.
- Avoid compound sentences.
- Make sure that the sequence of events is clear. If the order of events is not important, make sure this is clearly stated and do not describe it as though it has to be fixed.
- Use terminology consistently throughout the use-case model (e.g., use a glossary).
- A use-case description should not exceed two or three pages in length.

Extracted from:

Use Case Modeling, Kurt Bittner and Ian Spence, Addison-Wesley, 2003.

Object-Oriented Software Engineering: Using UML, Patterns, and Java, Bernd Bruegge and Allen H. Dutoit, Prentice Hall, 2004.

USE-CASE DETAIL:
HOW MUCH DETAIL IS ENOUGH?

- A use case is a technique for _____—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:
  - Login
  - 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?

NONFUNCTIONAL REQUIREMENTS

A *nonfunctional requirement* places a constraint on a use case or on the system.

It is identified by asking questions about the system's:

Design Qualities – reliability, supportability, maintainability, etc.

Performance – speed, throughput, response time, accuracy, etc.

Interface –   user interface (learnability, usability);
external system interface (formats, timing).

Hardware – implementation platform, memory size, storage capacity.

Implementation – standards, languages, error handling.

Security – system access; data access; physical access.

Physical environment – abnormal conditions; distributed operation

Documentation – what is required and for who?

Management – system back up, installation, maintenance.

IDENTIFYING NONFUNCTIONAL REQUIREMENTS

To help identify nonfunctional requirements some categories to consider and "trigger questions" to ask for each category are given below.

1. **Design Quality Issues**
   - What are the requirements for reliability?
   - Must the system trap faults?
   - Is there a maximum acceptable time for system restart after a failure?
   - What is the acceptable system downtime per 24-hour period?
   - Is it important that the system be portable (able to move to different hardware or operating system environments)?

2. **Performance Characteristics**
   - Are there any speed, throughput or response time constraints on the system?
   - Are there size or capacity constraints on data processed by the system?

3. **User Interface and External System Interface**
   - What type of user will be using the system?
   - Will more than one type of user be using the system?
   - What sort of training will be required for each type of user?
   - Is it particularly important that the system be easy to learn?
   - Is it particularly important that users be protected from making errors?

IDENTIFYING NONFUNCTIONAL REQUIREMENTS
(cont'd)

- What sort of input/output devices for the human interface are available and what are their characteristics?
- Is input coming from systems outside the proposed system?
- Is output going to systems outside the proposed system?
- Are there restrictions on the format or medium that must be used for input or output?

4. Hardware Considerations
- What hardware is the proposed system to be used on?
- What are the characteristics of the target hardware, including memory size and auxiliary storage space?

5. Implementation Considerations
- What standards need to be followed?
- What programming language(s) should be used?
- How should the system respond to input errors?
- How should the system respond to extreme conditions?

6. Security Issues
- Must access to any data or the system itself be controlled?
- Is physical security an issue?

IDENTIFYING NONFUNCTIONAL REQUIREMENTS
(cont'd)

7. Physical Environment
- Where will the target equipment operate?
- Will the target equipment be in one or several locations?
- Will the environmental conditions in any way be out of the ordinary (for example, unusual temperatures, vibration, magnetic fields)?

8. Documentation
- What kind of documentation is required?
- What audience is to be addressed by each document?

9. Management Issues
- How often will the system be backed up?
- Who will be responsible for the back up?
- Who is responsible for system installation?
- Who will be responsible for system maintenance?

Source: http://www.csee.umbc.edu/courses/undergraduate/345/spring04/mitchell/nfr.html

NONFUNCTIONAL REQUIREMENTS EXAMPLE: SATWATCH

- SatWatch is a wrist watch that:
  - uses GPS satellites to determine its location and displays the time based on its current location.
  - uses internal data structures to convert this location into a time zone.
  - never requires the owner to reset the time due to the information it stores and its accuracy.
  - has no buttons or controls available to the user since it adjusts the time and the date displayed as the watch owner crosses time zones.
  - assumes that it does not cross a time zone boundary during a GPS blackout period, but adjusts its time zone as soon as possible after the blackout period.
  - has a two-line display showing, on the top line, the time (hour, minute, second, time zone) and on the bottom line, the date (weekday, day, month, year).
  - has a readable display even under poor light conditions.
  - can have its software upgraded using the WebifyWatch device (provided with the watch) and a personal computer connected to the Internet.

NONFUNCTIONAL REQUIREMENTS EXAMPLE:
SATWATCH (cont'd)

- Any user who knows how to read a digital watch and understand international time zone abbreviations should be able to use SatWatch.

- As SatWatch has no buttons, no software faults requiring the resetting of the watch should occur.

- SatWatch should accept upgrades to its onboard processor via the USB interface.

NONFUNCTIONAL REQUIREMENTS EXAMPLE: SATWATCH (cont'd)

- SatWatch should display the correct time zone within 5 minutes of the end of a GPS blackout period.

- SatWatch should measure time within 1/100th second over 5 years.

- SatWatch should display time correctly in all 24 time zones.

NONFUNCTIONAL REQUIREMENTS EXAMPLE: SATWATCH (cont'd)

- All related software associated with SatWatch will be written using Java.

- SatWatch complies with the physical, electrical, and software interfaces defined by WebifyWatch API 2.0.

SPECIFYING NONFUNCTIONAL REQUIREMENTS

> **Nonfunctional requirements are specified as supplementary requirements on use cases or on the system as a whole.**

- Some nonfunctional requirements will be implemented as administration use cases.

    – Login

    – System start up

    – System shut down

    – System backup

> *Administration use cases* are use cases that deal with non-functional requirements such as security or system operation and maintenance.

VALIDATE SYSTEM REQUIREMENTS

The system requirements specification (SRS) should be validated continuously with the client/user to verify that they are:

complete – the requirements describe all possible features of interest, including the handling of exceptional behaviour.

➢ All aspects of the system are represented in the SRS.

consistent – the requirements do not contradict themselves.

clear – the requirements define exactly one system.

➢ It is not possible to interpret the SRS in two or more different ways.

correct – the requirements represent the features of interest to the client.

➢ Everything in the SRS accurately represents an aspect of the system.

realistic – the system can be implemented within the given constraints.

Acceptance tests are the primary means to validate that the system implementation satisfies the requirements.

REQUIREMENTS VALIDATION EXAMPLE: SATWATCH

Incompleteness

Problem: The SatWatch specification does not specify the boundary behavior when the user is standing within GPS accuracy limitations of a time zone's boundary.

Solution:

REQUIREMENTS VALIDATION EXAMPLE: SATWATCH (cont'd)

Inconsistent

Problem: SatWatch software should not have bugs nor need to be upgraded.

SatWatch software should be easily upgraded using the USB interface.

Solution:

REQUIREMENTS VALIDATION EXAMPLE: SATWATCH (cont'd)

Unclear

Problem: The SatWatch specification refers to time zones.

Does the SatWatch deal with daylight saving time or not?

Solution:

Incorrect

Problem: SatWatch supports supports only 24 time zones (24 hours).

There are more than 24 time zones. Several countries and territories are half an hour ahead of a neighboring time zone.

Solution:

SYSTEM REQUIREMENTS CAPTURE: RETROSPECTIVE

Domain Modeling

Captures the data requirements of an application.

class diagram – shows classes and the relationships among them.

Use-case Modeling

Captures the functional requirements of an application.

use-case model – shows use cases that provide system functionality and actors that use the functionality.

flow of events – describes the sequence of actions that comprise a use case's functionality.

Requirements Validation

Verifies that the system meets all stated requirements.

SYSTEM REQUIREMENTS CAPTURE: SUMMARY

- ● Requirements are captured over several iterations by specifying:
  - – a domain model
  - – a use-case model
  - – plus any nonfunctional requirements.

> These are all documented in the
> **System Requirements Specification (SRS).**

- ● In subsequent iterations/phases we refine and/or transform the requirements by specifying:
  - – more detail for each of the above artifacts.
  - – a set of test cases in the test model.
  - – matching use-case realizations in the analysis model.
  - – matching use-case realizations in the design model.

> **The use cases drive the subsequent iterations/phases.**