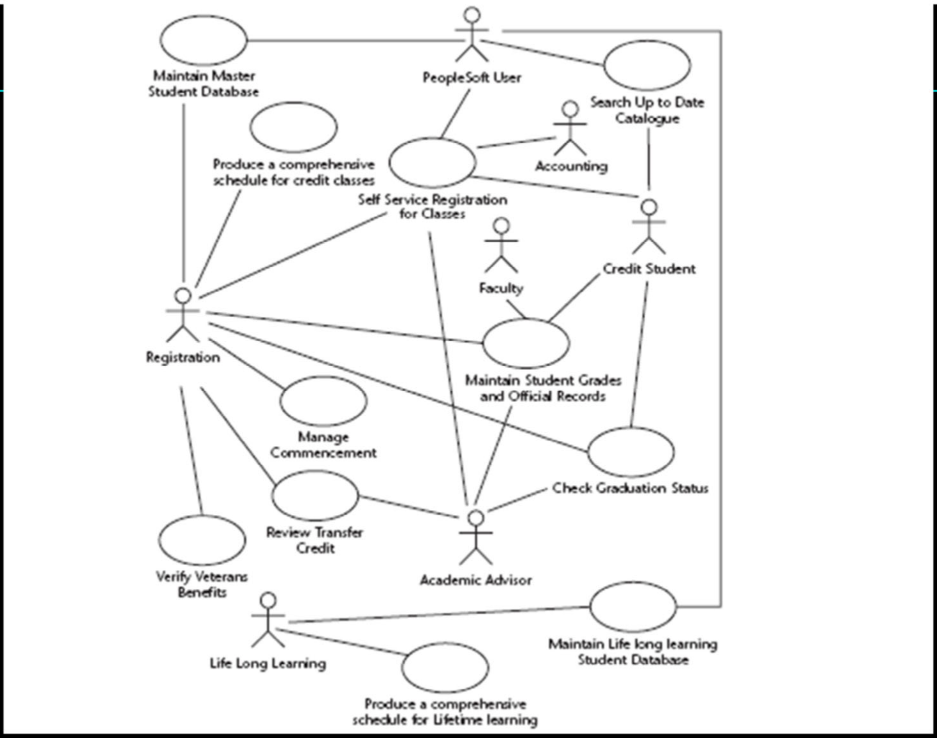
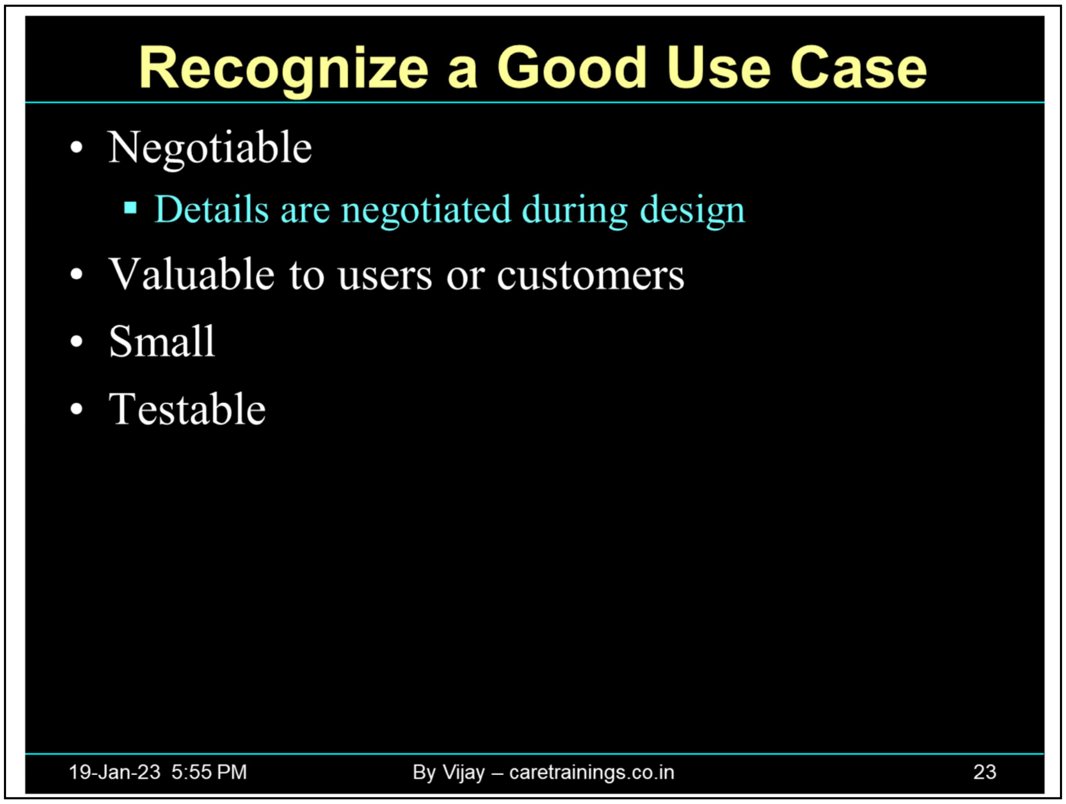
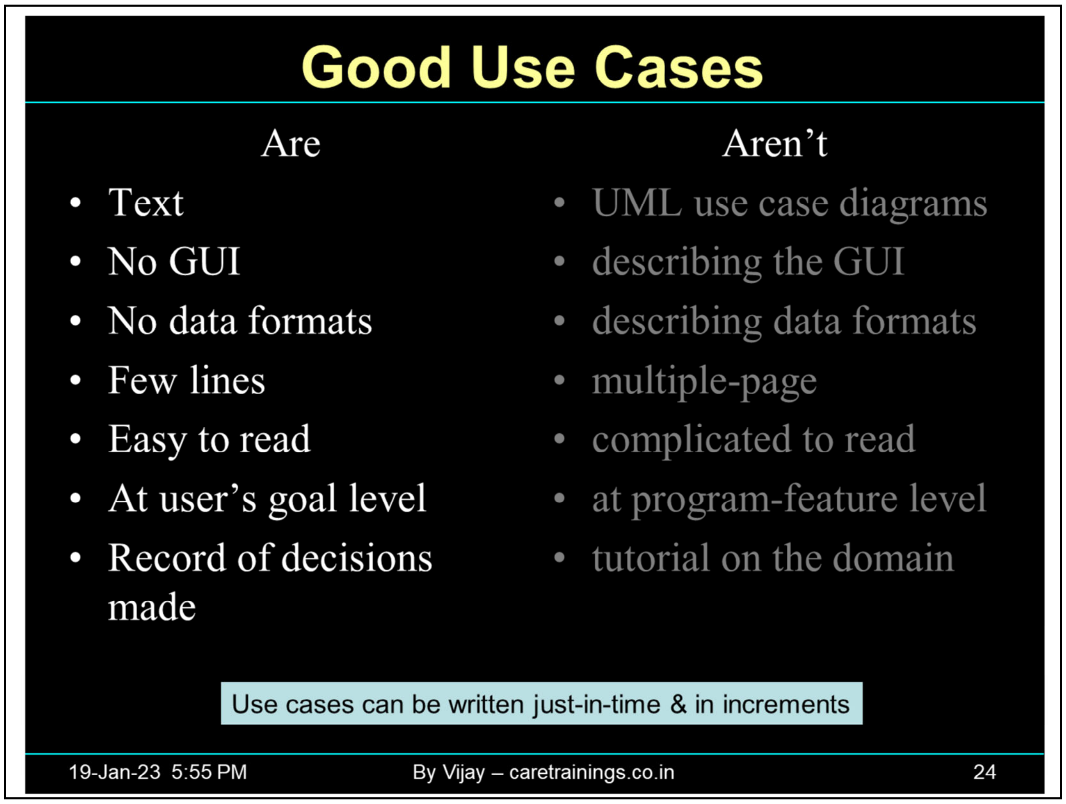

Maintain Master Student Database
PeopleSoft User
Search Up to Date Catalogue
Produce a comprehensive schedule for credit classes
Accounting
Self Service Registration for Classes
Credit Student
Faculty
Registration
Maintain Student Grades and Official Records
Manage Commencement
Check Graduation Status
Review Transfer Credit
Academic Advisor
Verify Veterans Benefits
Life Long Learning
Maintain Life long learning Student Database
Produce a comprehensive schedule for Lifetime learning



Recognize a Good Use Case

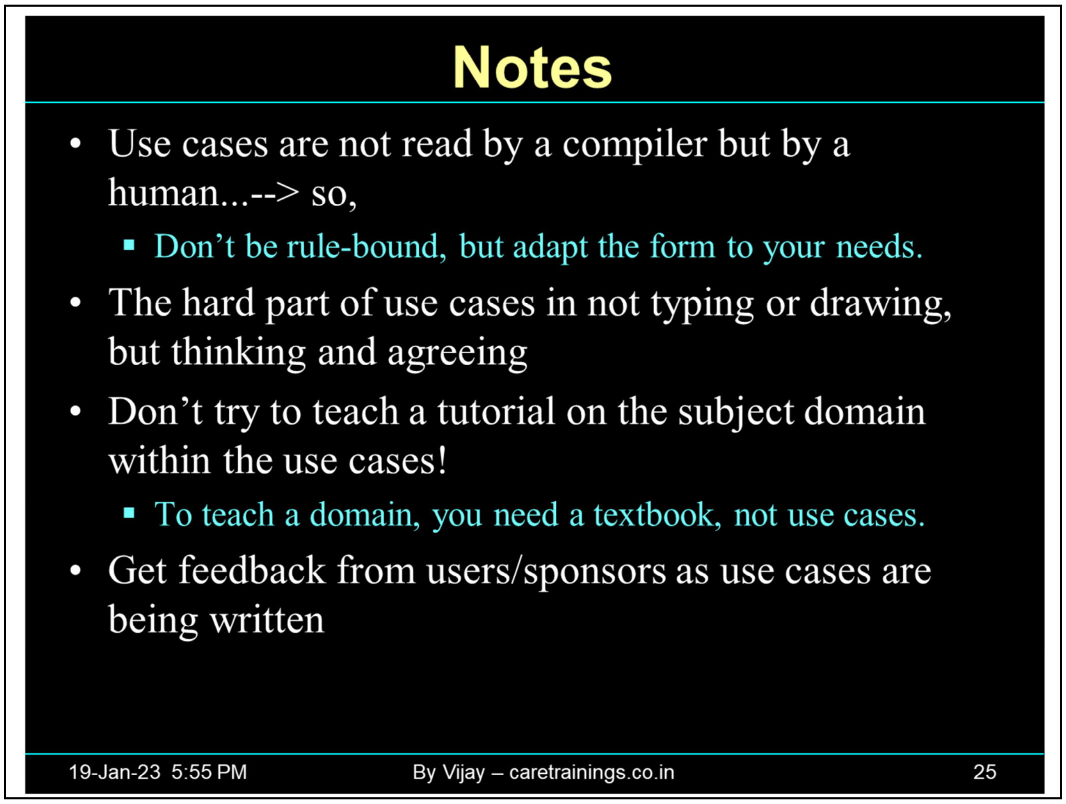
- Negotiable
  - Details are negotiated during design
- Valuable to users or customers
- Small
- Testable



Good Use Cases

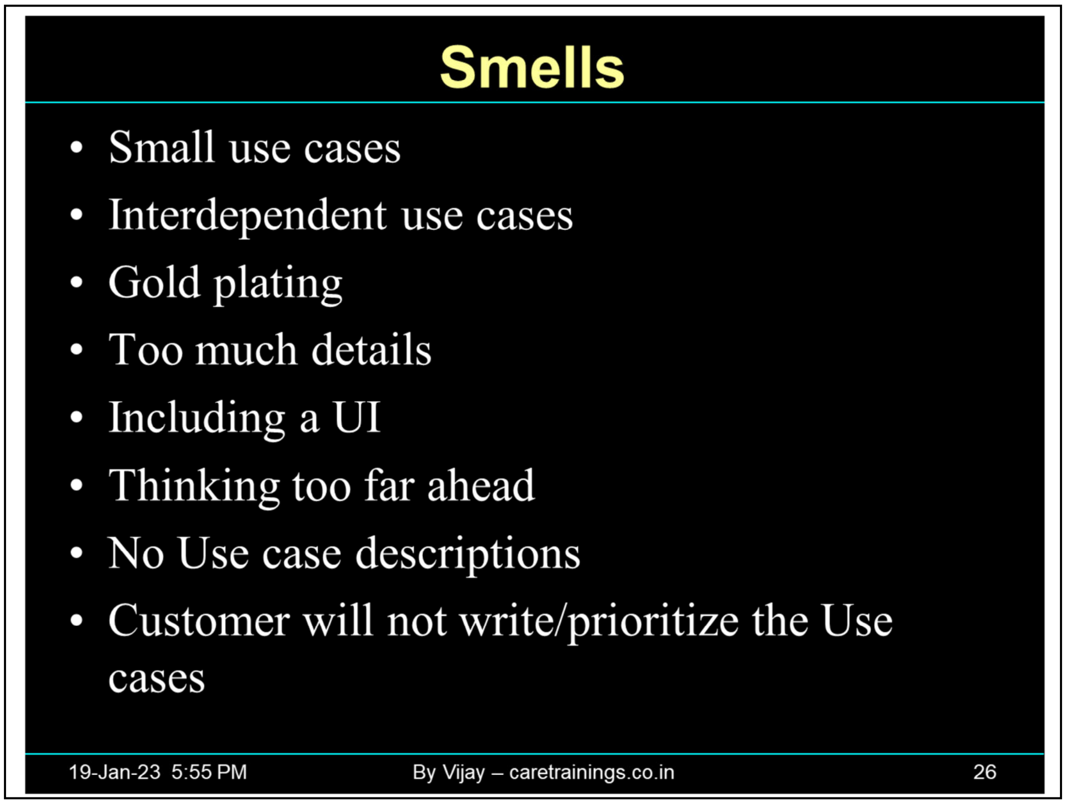
Are	Aren't
• Text	• UML use case diagrams
• No GUI	• describing the GUI
• No data formats	• describing data formats
• Few lines	• multiple-page
• Easy to read	• complicated to read
• At user's goal level	• at program-feature level
• Record of decisions made	• tutorial on the domain

Use cases can be written just-in-time & in increments



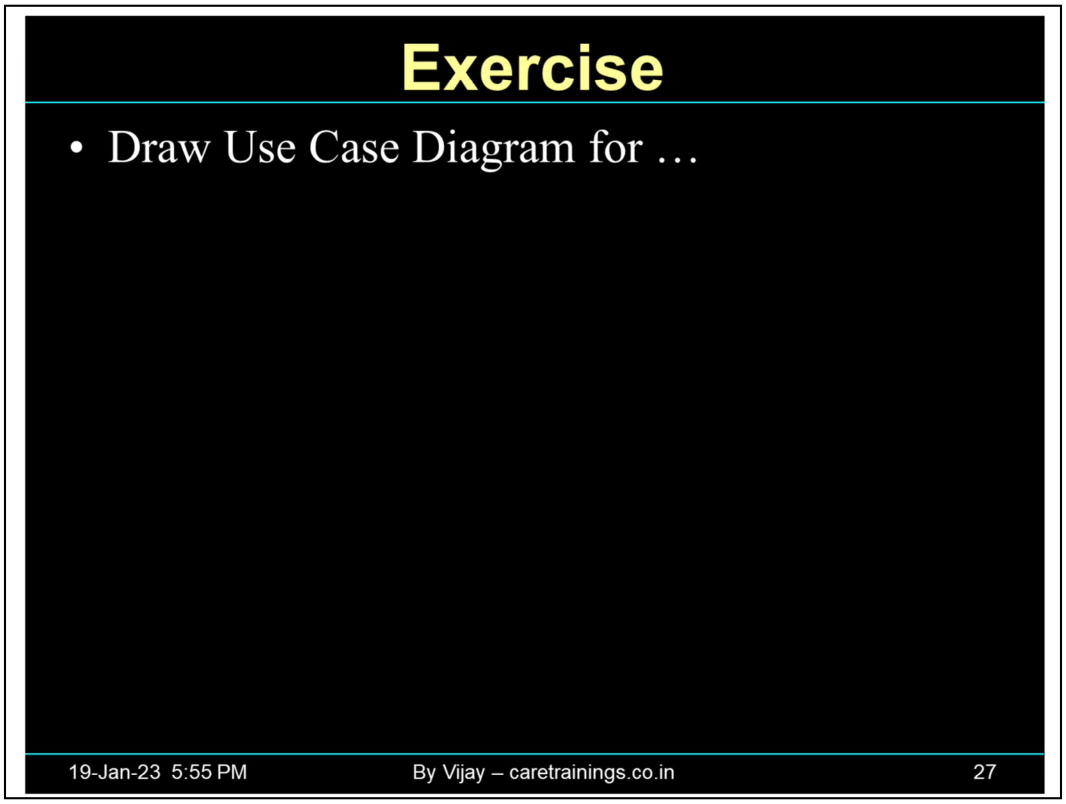
Notes

- Use cases are not read by a compiler but by a human...--> so,
  - Don't be rule-bound, but adapt the form to your needs.
- The hard part of use cases in not typing or drawing, but thinking and agreeing
- Don't try to teach a tutorial on the subject domain within the use cases!
  - To teach a domain, you need a textbook, not use cases.
- Get feedback from users/sponsors as use cases are being written



Smells

- Small use cases
- Interdependent use cases
- Gold plating
- Too much details
- Including a UI
- Thinking too far ahead
- No Use case descriptions
- Customer will not write/prioritize the Use cases



Exercise

- Draw Use Case Diagram for …

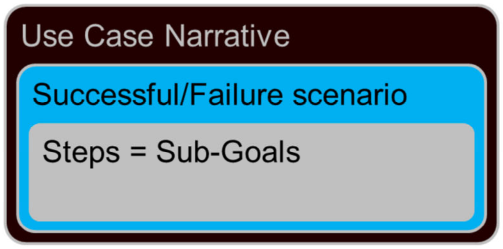
19-Jan-23 5:55 PM By Vijay – caretrainings.co.in 27

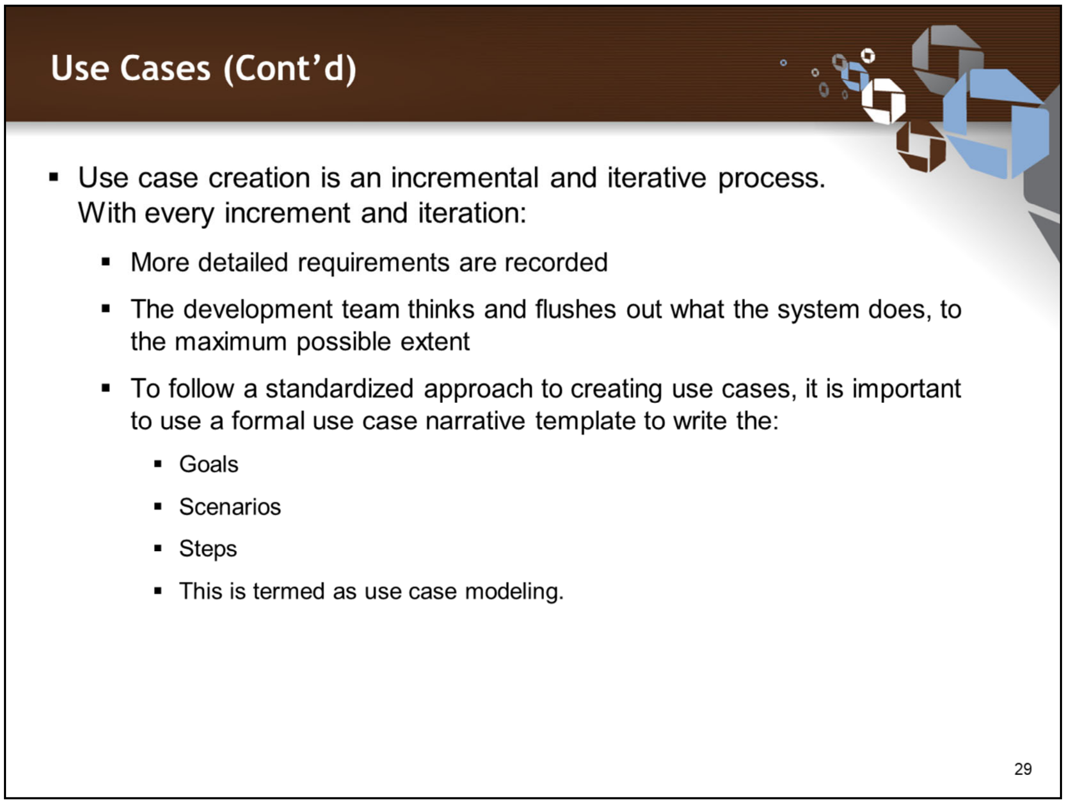
Banking?

Amazon?

Use Cases

- A use case is a description of sequence of actions between a system and a user that allows the user to achieve a goal.
- Each sequence of actions is called a scenario.
- Use cases can be represented:
  - Textually through use case narratives
  - Diagrammatically through use case diagrams





Use Cases (Cont'd)

- Use case creation is an incremental and iterative process. With every increment and iteration:
  - More detailed requirements are recorded
  - The development team thinks and flushes out what the system does, to the maximum possible extent
  - To follow a standardized approach to creating use cases, it is important to use a formal use case narrative template to write the:
    - Goals
    - Scenarios
    - Steps
    - This is termed as use case modeling.

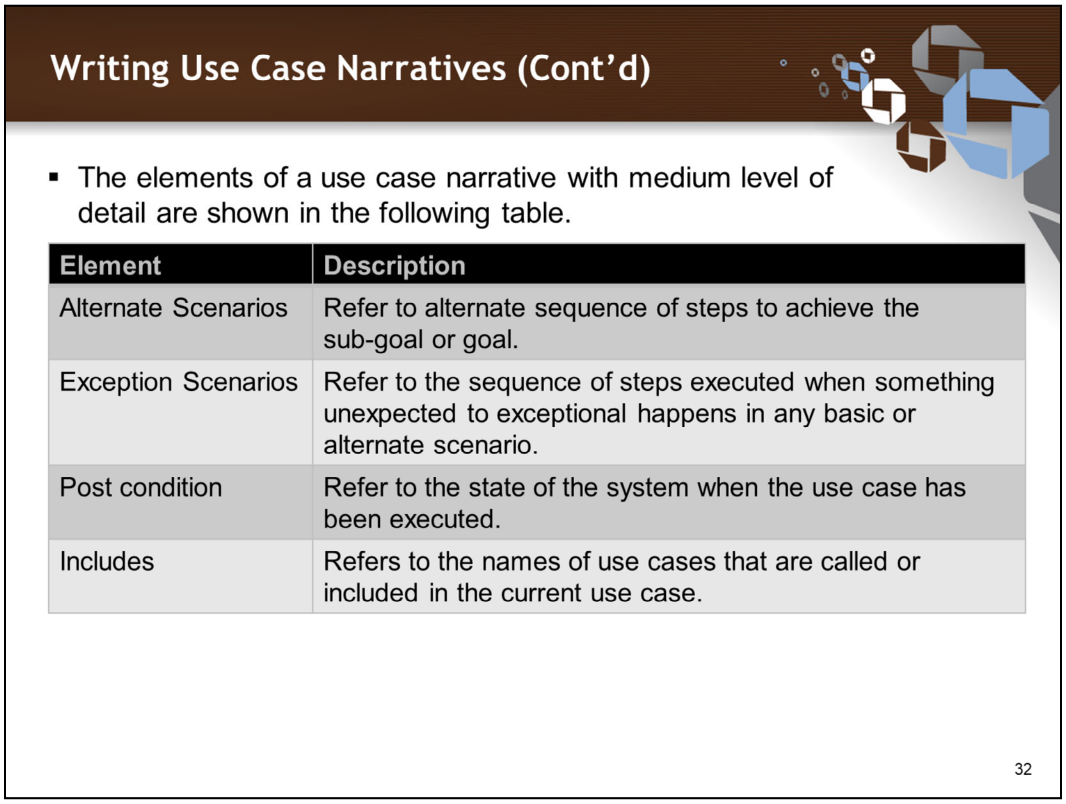
Writing Use Case Narratives

- To understand how to write use case narratives, you must identify the elements of a use case.
- A use case narrative can be written with:
  - Extensive details known as a fully-dressed use case
  - Medium level details
  - Basic details

Writing Use Case Narratives (Cont'd)

- The elements of a use case narrative with medium level of detail are shown in the following table.

Element	Description
Actor	Refers to a human or another system interacting with the system being modelled. Primary or active actors are those that the system uses to achieve its goals. Secondary or passive actors are those that provide some service to the system in order to help the system realize the goals of the primary actors.
Trigger	Refers to an event that causes the use case to start.
Preconditions	Refer to assumptions that can be made for a use case before it begins.
Basic Scenarios	Refers to the sequence of steps describing the normal case scenario.



Writing Use Case Narratives (Cont'd)

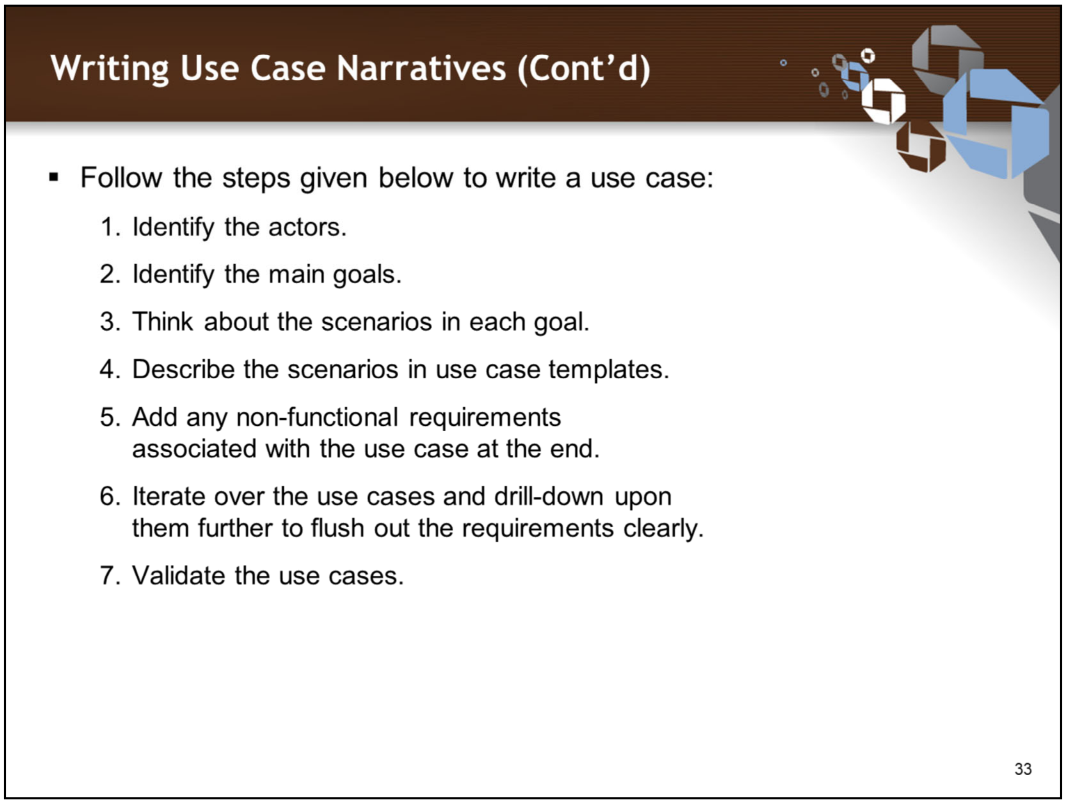
- The elements of a use case narrative with medium level of detail are shown in the following table.

Element	Description
Alternate Scenarios	Refer to alternate sequence of steps to achieve the sub-goal or goal.
Exception Scenarios	Refer to the sequence of steps executed when something unexpected to exceptional happens in any basic or alternate scenario.
Post condition	Refer to the state of the system when the use case has been executed.
Includes	Refers to the names of use cases that are called or included in the current use case.

32

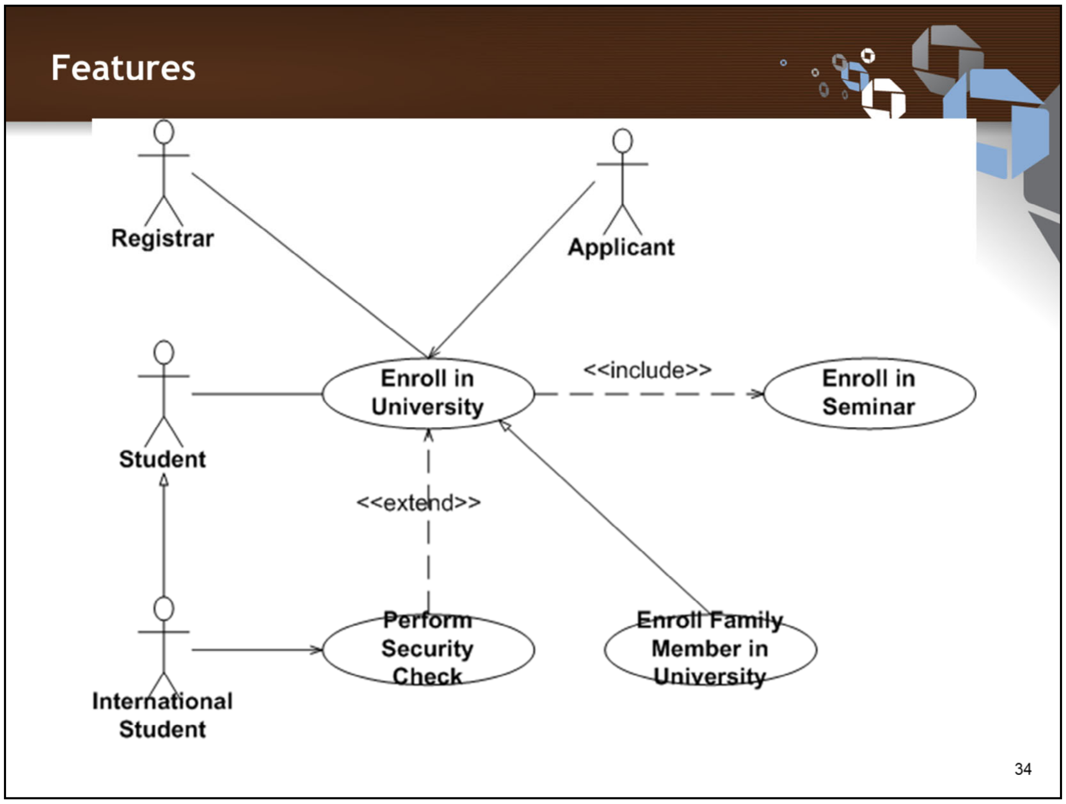
Alternate flows can be numbered as A.1, A.2 and so on and written in reference to the basic scenario.

Exception flows can be numbered as Y.E.N, where Y is 0 if the exception occurs in basic scenario and a number > 0 if the exception occurs in an alternate scenario. E indicates that the sequence describes an exception scenario. N is the sequence number for the exception flow in the use case.



Writing Use Case Narratives (Cont’d)

- Follow the steps given below to write a use case:
  1. Identify the actors.
  2. Identify the main goals.
  3. Think about the scenarios in each goal.
  4. Describe the scenarios in use case templates.
  5. Add any non-functional requirements associated with the use case at the end.
  6. Iterate over the use cases and drill-down upon them further to flush out the requirements clearly.
  7. Validate the use cases.



Features

An extend dependency is a generalization relationship where an extending use case continues the behaviour of a base use case. The extending use case accomplishes this by conceptually inserting additional action sequences into the base use-case sequence. This allows an extending use case to continue the activity sequence of a base use case when the appropriate extension point is reached in the base use case and the extension condition is fulfilled. When the extending use case activity sequence is completed, the base use case continues. In figure , you see the use case “Perform” extends the use case “Enroll in University,” the notation for doing so is simply a normal use-case dependency with the stereotype of <<extend>>. In this case, “Enroll in University” is the base use case and “Perform Security Check” is the extending use case.

An extending use case is, effectively, an alternate course of the base use case. In fact, a good rule of thumb is you should introduce an extending use case whenever the logic for an alternate course of action is at a complexity level similar to that of your basic course of action. I also like to introduce an extending use case whenever I need an alternate course for an alternate course; in this case, the extending use case would encapsulate both alternate courses. Many modelers avoid the use of extend dependencies as this technique has a tendency to make use-case diagrams difficult to understand. My preference is to use extend dependencies sparingly.

You use include dependencies whenever one use case needs the behaviour of another. Introducing a new use case that encapsulates similar logic that occurs in several use cases is quite common. For example, you may discover several use cases need the behaviour to search for and then update information about students, indicating the potential need for an “Update Student Record” use case included by the other use cases.

Use cases can inherit from other use cases, offering a third opportunity to indicate potential reuse. Figure 1 depicts an example of this, showing that “Enroll Family Member in University” inherits from the “Enroll In University” use case. Inheritance between use cases is not as common as either the use of extend or include dependencies, but it is still possible. The inheriting use case would completely replace one or more of the courses of action of the inherited use case. In this case, the basic course of action is completely rewritten to reflect that new business rules are applied when the family member of a professor is enrolling at the university. Family members are allowed to enroll in the school, regardless of the marks they earned in high school, they don’t have to pay any enrollment fees, and they are given top priority for enrollment in the university.

Inheritance between use cases should be applied whenever a single condition, in this case, the student is a family member of a professor, would result in the definition of several alternate courses. Without the option to define an inheriting use case, you need to introduce an alternate course to rework the check of the student’s high-school marks, the charging of enrollment feeds, and for prioritization of who is allowed to enroll in the given semester.

The inheriting use case is much simpler than the use case from which it inherits. It should have a name, description, and identifier, and it should also indicate from which use case it inherits in the “Inherits From” section. This includes any section that is replaced, particularly the pre-conditions and post-conditions as well as any courses of action. If something is not replaced, then leave that section blank, assuming it is inherited from the parent use case (you might want to put text, such as “see parent use case,” in the section).