Unit 5: Object-Oriented Software Engineering: Requirements Model

Objectives:

This Unit will introduce the Object-Oriented Software Engineering *(OOSE)* method from Jacobson et al. It will describe the basics of *'a use case driven approach'*. The focus of the Unit is the development of its *Requirements Model.* It will discuss actors, use cases, interface descriptions and problem domain objects. Relevant notations are drawn from the UML (Unified Modelling Language)

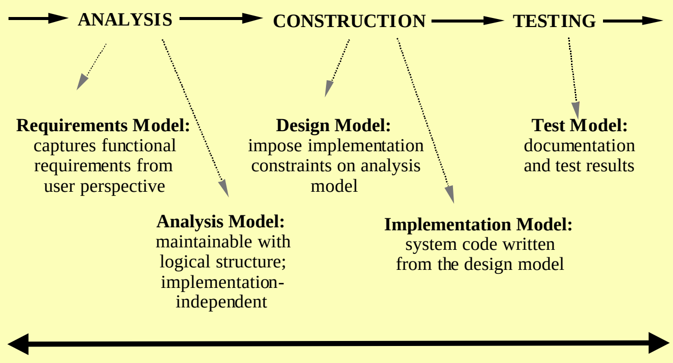
OOSE Background

- Originated in Sweden
- " Object-Oriented Software Engineering *A Use Case Driven Approach* "by Ivar Jacobson, Magnus Christerson, Patrik Jonsson & Gunnar Overgaard, Addison-Wesley , 1992
  - Pragmatic method based on experience
  - Popular and successful
  - Complete method

What Comprises a Method?

- Method described via
  - syntax (how it looks)
  - semantics (what it means)
  - pragmatics (heuristics, rules of thumb for use)

System Development as “Building Models”



ANALYSIS → **CONSTRUCTION** → **TESTING**

Requirements Model: captures functional requirements from user perspective

Analysis Model: maintainable with logical structure; implementation-independent

Design Model: impose implementation constraints on analysis model

Implementation Model: system code written from the design model

Test Model: documentation and test results

Seamless, incremental transition between stages and models, iterations possible

3 stages, 5 models

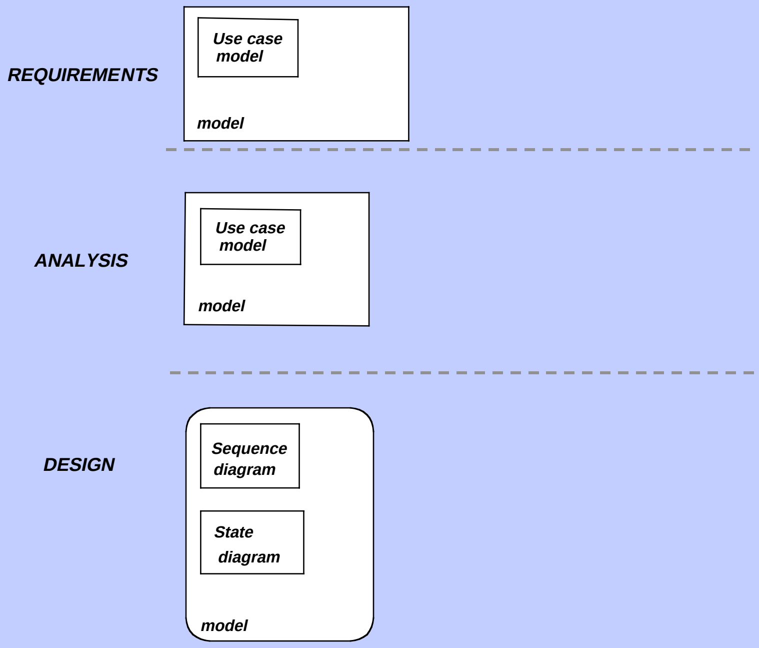
OOSE Models **UML Representations**

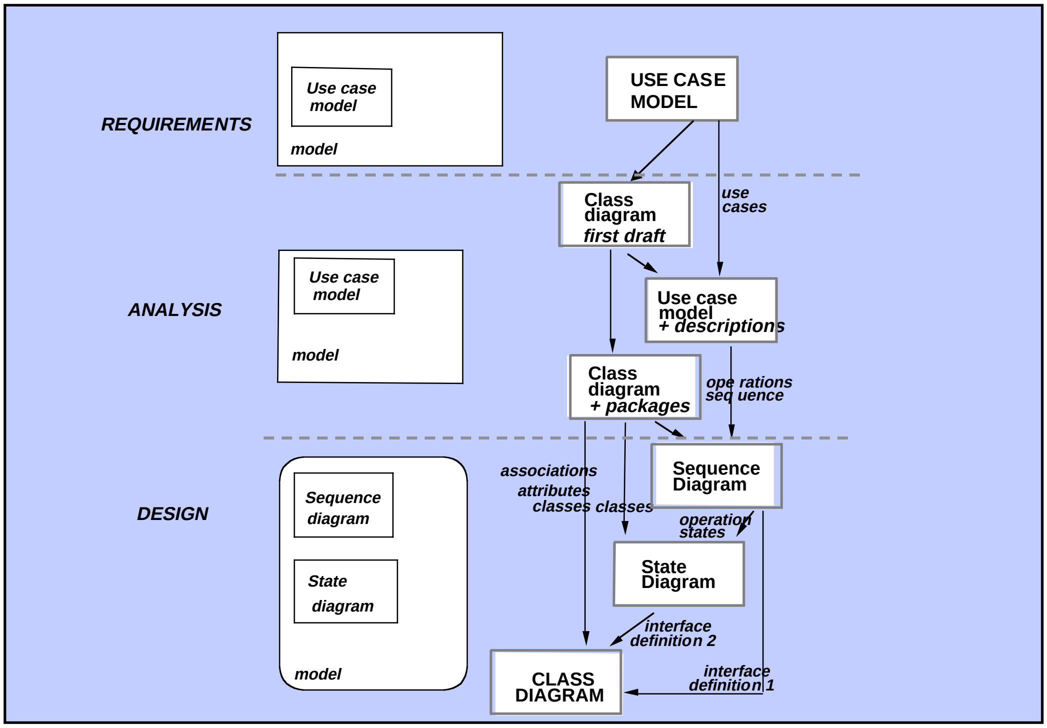
REQUIREMENTS

ANALYSIS

DESIGN

OOSE Models **UML Representations**





REQUIREMENTS

Use case model

model

USE CASE MODEL

Class diagram *first draft*

use cases

ANALYSIS

Use case model

model

Use case model *+ descriptions*

Class diagram *+ packages*

ope rations seq uence

DESIGN

Sequence diagram

State diagram

model

associations attributes classes classes

Sequence Diagram

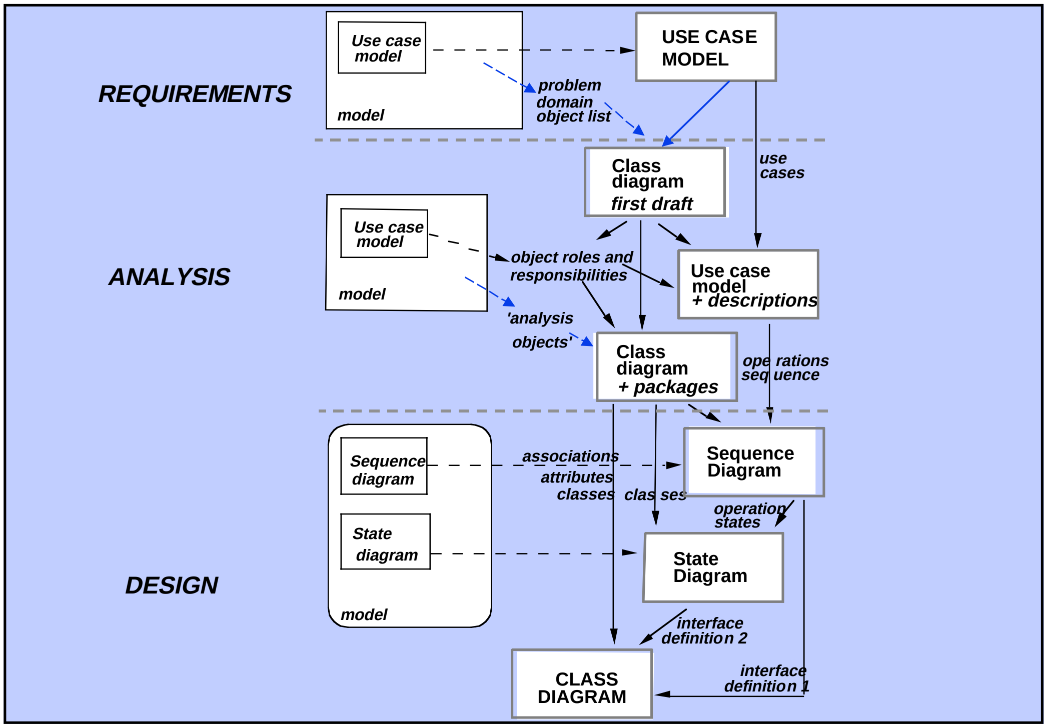
operation states

State Diagram

interface definition 2

CLASS DIAGRAM

interface definition 1



REQUIREMENTS

Use case model

model

USE CASE MODEL

problem domain object list

Class diagram *first draft*

use cases

ANALYSIS

Use case model

model

object roles and responsibilities

'analysis objects'

Use case model *+ descriptions*

Class diagram *+ packages*

ope rations seq uence

DESIGN

Sequence diagram

State diagram

model

associations attributes classes clas ses

Sequence Diagram

operation states

State Diagram

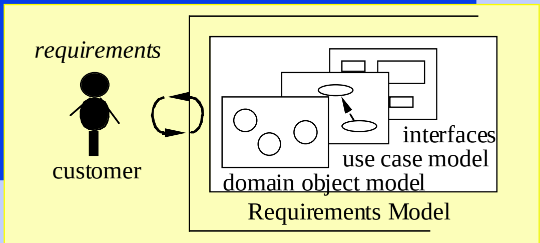
interface definition 2

CLASS DIAGRAM

interface definition 1

Analysis Stage

- *Primary objectives*
  - to determine what the system must do
  - to embed the software system in its environment
- *Two concerns*
  - to get the right thing
  - to get the thing right (now and for future)
- *Products*
  - Requirements Model
  - Analysis Model



Producing a Requirements Model

1 Derive possible use cases

2 Discriminate between possible use cases

3 Generate use case descriptions

4 Identify associations between use cases

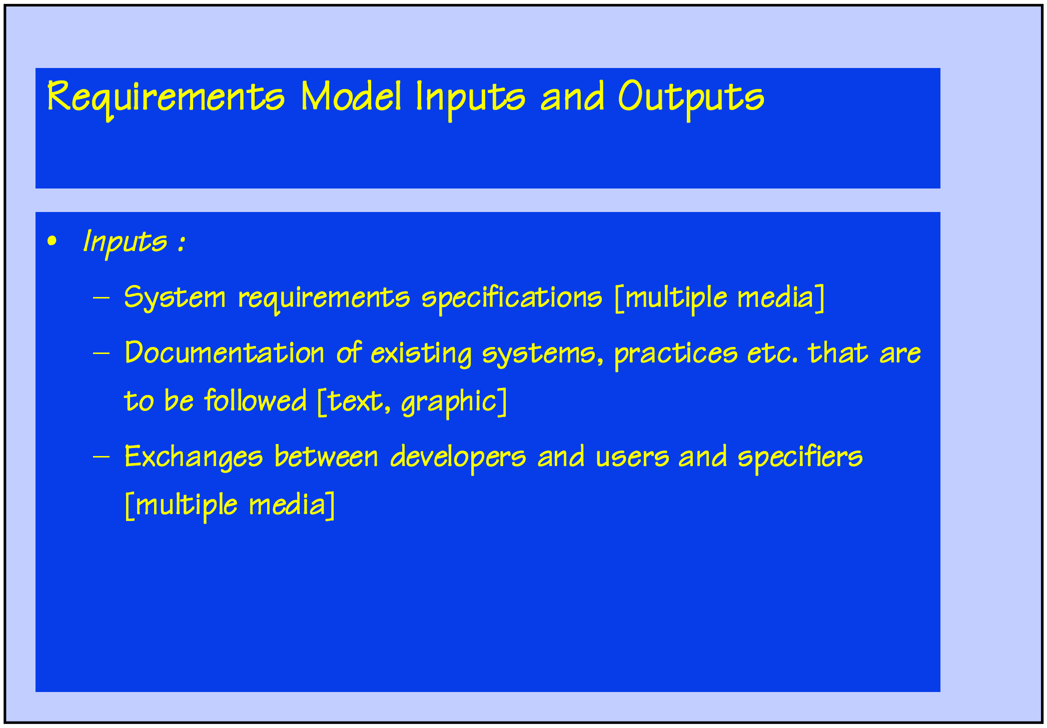
5 Refine and complete use cases and use case model

6 Describe and test user interfaces

7 Describe system interfaces

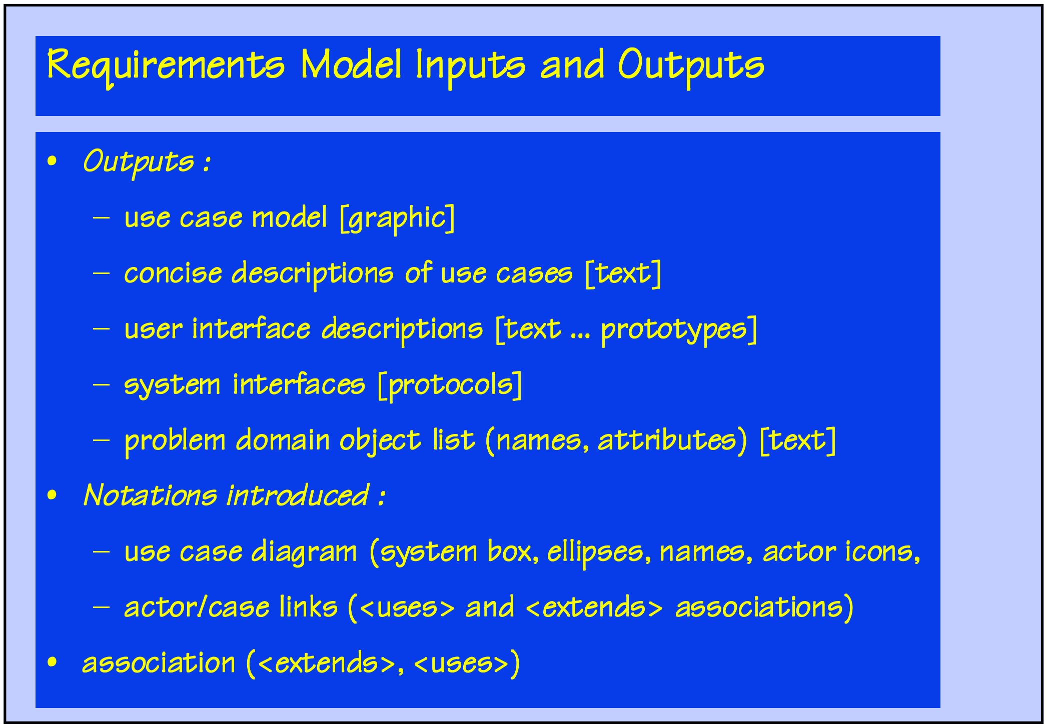
8 Identification of problem domain objects

9 Check incorporation of requirements



Requirements Model Inputs and Outputs

- *Inputs :*
  - System requirements specifications [multiple media]
  - Documentation of existing systems, practices etc. that are to be followed [text, graphic]
  - Exchanges between developers and users and specifiers [multiple media]

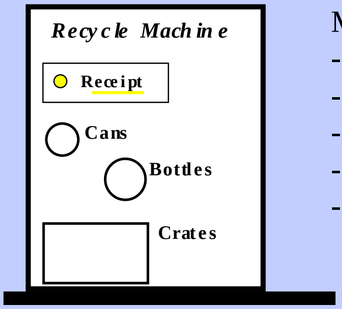


Requirements Model Inputs and Outputs

- *Outputs :*
  - use case model [graphic]
  - concise descriptions of use cases [text]
  - user interface descriptions [text ... prototypes]
  - system interfaces [protocols]
  - problem domain object list (names, attributes) [text]
- *Notations introduced :*
  - use case diagram (system box, ellipses, names, actor icons,
  - actor/case links (<uses> and <extends> associations)
- association (<extends>, <uses>)

Requrements Example

Multi-purpose recycling machine



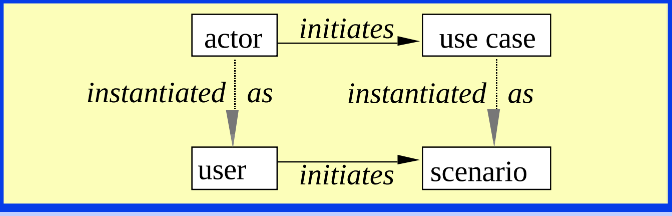
Machine must:
- receive & check items for customers,
- print out receipt for items received,
- print total received items for operator,
- change system information,
- signal alarm when problems arise.

ACTORS

- An *actor* is:
  - anything *external* to the system, human or otherwise
  - a *user type* or *category*
- A *user* doing something is an *occurrence* of such a type
- A *single user* can instantiate several *different actor types*
- Actors come in two kinds:
  - *primary* actors, using system in daily activities
  - *secondary* actors, enabling primary actors to use system

USE CASES

- A *use case*
  - constitutes *complete course* of events initiated by actor
  - defines *interaction* between actor and system
  - is a member of the set of all use cases which together define *all existing ways* of using the system



actor *initiates* use case

instantiated as · *instantiated as*

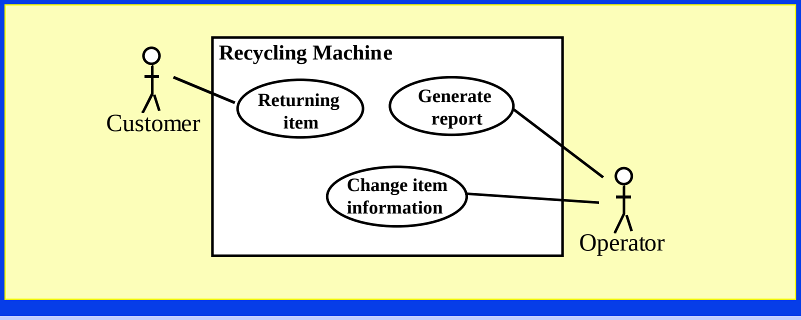
user *initiates* scenario

Examples of Use Cases

- *Returning items* is started by *Customer* when she wants to return cans, bottles or crates. With each item that the *Customer* places in the recycling machine, the system will increase the received number of items from *Customer* as well as the daily total of this particular type. When *Customer* has deposited all her items, she will press a receipt button to get a receipt on which returned items have been printed, as well as the total return sum.
- NB Particular *instances of use* would be different " *The morning after the party Sarah goes to the recycling centre with three crates containing ....* "

Use Case Model

- A *use case* model
  - *presents a collection of use cases*
  - characterise *behaviour of whole system*, plus external actors



Identifying Use Cases

- Consider *situation*,
- Identify *actors*,
- Read *specification*,
- Identify *main tasks*,
- Identify *system information*,
- Identify *outside changes*,
- Check *information* for actors,
- Draft initial *use cases*, [text]
- Identify system *boundary*,
- Draft initial *use case model* [graphic]

When is a Use Case ... ?

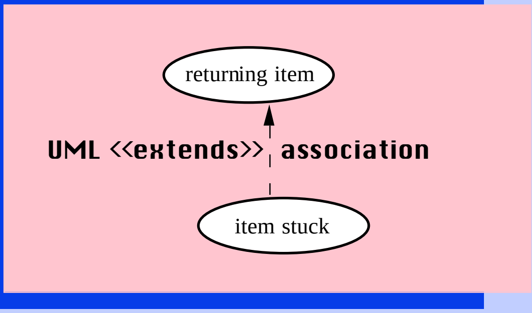
- *Discrimination between possible use cases*
  - Estimate *frequency* of use,
  - Examine degree of *difference* between cases
  - Distinguish betweeen *'basic'* and *'alternative'* courses of events
  - Create new use cases where necessary

Elaborated Example

- BASIC
  - When the *Customer* returns a deposit item, it is measured by the system. The measurements are used to determine what kind of can, bottle or crate has be deposited. If accepted, the *Customer* total is incremented, as is the daily total for that specific item type.
- ALTERNATIVE
  - If the item is not accepted, 'NOT VALID' is highlighted on the panel.
- BASIC
  - When *Customer* presses the receipt button, the printer prints the date. The customer total is calculated and the following information printed on the receipt for each item type: name, number returned, deposit value, total for this type. Finally the sum that the *Customer* should receive is printed on the receipt.

Use Case Extensions

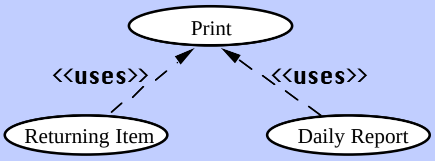
- *Extensions* provide opportunities for :
  - *optional* parts
  - *alternative* complex cases
  - *separate* sub-cases
  - *insertion* of use cases



Refinements

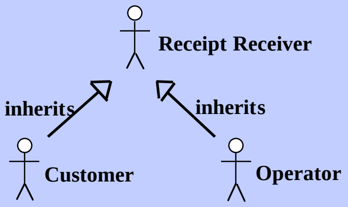
Abstract use case

Concrete use case



Abstract actors

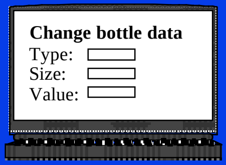
Concrete actors



User Interface Descriptions

- *Describe* user interfaces
- *Test* on potential users,
- *if necessary using*
- *simulations* or *prototypes*

Operator's interface



Change bottle data
Type:
Size:
Value:

- Describe system *interfaces* for *non-human actors*

Problem Domain Objects

- Object in specification
- Direct counterpart in the application environment
- Refinement in stages :

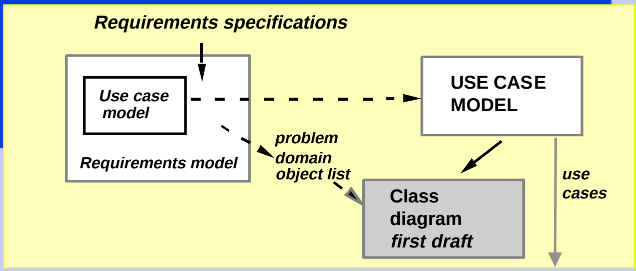
Object noun ->
Logical attributes ->
Static associations
Inheritance ->
Dynamic associations ->
Operations

Object Examples

OBJECT	*ATTRIBUTES*
name	*characteristic / information : type*
Deposit item	name: string, total: integer, value: ECU
Can	width: cm, height: cm
Bottle	width: cm, height: cm, bottom: cm
Crate	width: cm, height: cm, lenght: cm
Receipt	total cans: int, total bottles: int, ...
Customer panel	receipt button: button
Operator panel	bottle data: cm, ...

Requirements Model

- Outputs :
  - *use case model* [graphic]
  - *concise descriptions of use cases* [text]
  - *user interface descriptions* [text ... prototypes]
  - *system interfaces* [protocols]
  - *problem domain object list* (name, attribute: type) [text]


Requirements specifications
Use case model
Requirements model
USE CASE MODEL
problem domain object list
use cases
Class diagram first draft

Key Points

- System development can be viewed as model building.
- Special attention should be devoted to the requirements model in order “to get the right thing”. The first step is to get system use in context via the use case model . This is complemented by user interface descriptions.
- Problem domain objects are identified as a prelude to class diagram.

REQUIREMENTS MODEL

Stages of production

Inputs:

- System requirements specifications [multiple media]

- Documentation of existing systems, practices etc. that are to be followed [text, graphic]

- Exchanges between developers and users and specifiers [m m]

1) Derive possible use cases from requirements specification

- consider possible scenes or situations

- identify actors

- read spec from each possible actor's perspective,

- identify main tasks associated with each individual actor,

- identify system information read, written or changed by actor,

- identify outside changes which actor informs system about,

- check if actor needs to be informed of unexpected changes,

- draft initial use cases (? using templates) [text]

- identify system boundary and draft initial use case model [graphic]

2) Discriminate between possible use cases

- estimate frequency of use,

- examine degree of difference between cases

- distinguish betweeen 'basic' and 'alternative' courses of events

- create new use cases where necessary

3) Generate for each use case a desciption in natural language text and create a full use case model [text, graphic]

4) Identify <extends> associations between use cases by modelling:

- optional parts

- complex and alternative cases that are rare

- separate sub-cases executed in some cases or circumstances

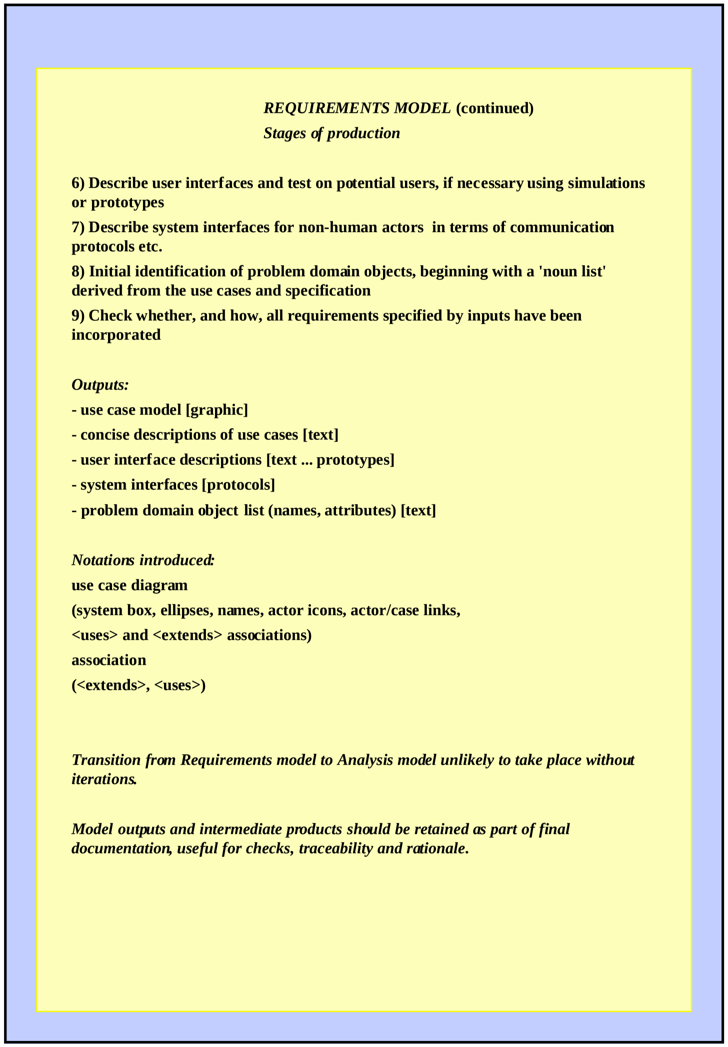
- situations where different use cases can be inserted into or interrupt a use case

5) Refine and complete use cases and use case model

- identification of ‘abstract’ and ‘concrete’ use cases (<uses>)

- identification of ‘abstract’ and ‘concrete’ actors (<generalizes>)

continued ...



***REQUIREMENTS MODEL* (continued)**

Stages of production

6) Describe user interfaces and test on potential users, if necessary using simulations or prototypes

7) Describe system interfaces for non-human actors in terms of communication protocols etc.

8) Initial identification of problem domain objects, beginning with a 'noun list' derived from the use cases and specification

9) Check whether, and how, all requirements specified by inputs have been incorporated

Outputs:

- use case model [graphic]

- concise descriptions of use cases [text]

- user interface descriptions [text ... prototypes]

- system interfaces [protocols]

- problem domain object list (names, attributes) [text]

Notations introduced:

use case diagram

(system box, ellipses, names, actor icons, actor/case links,

<uses> and <extends> associations)

association

(<extends>, <uses>)

Transition from Requirements model to Analysis model unlikely to take place without iterations.

Model outputs and intermediate products should be retained as part of final documentation, useful for checks, traceability and rationale.