C3 : Requirement Capture - Use Case

Non-Functional Requirements

- describe characteristics of the system to be achieved that are not related to its functionality
- How well it provides the functional requirements - performance criteria

Criterias of non-functional requirements :

↳ Operational
- ↳ physical and technical environments in which the system will operate
- ↳ e.g. The system should be able to fit in a pocket or purse.
  The system should be able to integrate with the existing inventory system

↳ Performance
- ↳ The speed, capacity and reliability of the system
- ↳ e.g. Any interaction between the user and the system should not exceed 2 seconds.
  The system should receive updated inventory information every 15 minutes.

↳ Security
- ↳ Who has authorized access to the system under what circumstances
- ↳ e.g. Only direct managers can see personnel records of staff.
  Customers can see their order history only during business hours.

↳ Cultural and political
- ↳ Cultural, political factors and legal requirements that affect the system
- ↳ e.g. The system should be able to distinguish between United States and European currency.
  Company policy says that we buy computers only from Dell.

Reference :

NON-FUNCTIONAL REQUIREMENTS

Nonfunctional Requirement	Description	Examples
Operational	The physical and technical environments in which the system will operate	■ The system should be able to fit in a pocket or purse. ■ The system should be able to integrate with the existing inventory system. ■ The system should be able to work on any Web browser.
Performance	The speed, capacity, and reliability of the system	■ Any interaction between the user and the system should not exceed 2 seconds. ■ The system should receive updated inventory information every 15 minutes. ■ The system should be available for use 24 hours per day, 365 days per year.
Security	Who has authorized access to the system under what circumstances	■ Only direct managers can see personnel records of staff. ■ Customers can see their order history only during business hours.
Cultural and political	Cultural, political factors and legal requirements that affect the system	■ The system should be able to distinguish between United States and European currency. ■ Company policy says that we buy computers only from Dell. ■ Country managers are permitted to authorize customer user interfaces within their units. ■ The system shall comply with insurance industry standards.

C8 : Specifying Operations

Object Constraint Language (OCL)

↳ a formal language used to express constraints on a model, that remains easy to read and write

↳ to specify many constraints that cannot be expressed directly in class diagram

↳ Constraints specify invariant (不变条件) conditions that the system being modelled

Consist of :

- Context
  - ↳ defines domain within which expression is valid
  - ↳ instance of a type
  - ↳ link (association interface)
  - ↳ e.g. context Company inv : self.numberOfEmployees > 50 (invariant; self 指向自己的 class)
- A property of that instance
  - ↳ often an attribute, association-end (role name) or query operation
  - ↳ e.g. self.manager.isUnemployed = false
- OCL operation applied to the property
  - ↳ arithmetical operators * / + / - / `/'
  - ↳ set operators (e.g. size(), isEmpty(), forAll(), select(), sum())
  - ↳ e.g. self.stockPrice() > 0

* extra notes:

```
context Person
    self.husband → notEmpty() implies
    self.husband.sex = male
```

Interpretation: if the set "husband" associated with a Person is not empty, then the value of the property "sex" of the husband must be male.

Arrowhead "→" is used for "SET". If the instance is a "SET", then "→" can be used with default operation (e.g. size(), isEmpty(), sum(), count())

Application of OCL

- specify invariants (不变) on classes and types in the class model
- specify type invariant for Stereotypes. (定型)
- describe pre- and post conditions on Operations and Methods
- specify constraints on operations

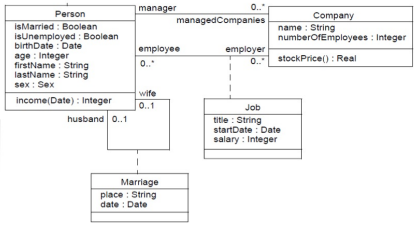
Reason of using OCL

- Allows suitably enabled modeling tools to generate code based on OCL expressions.

- Allows suitably enabled modeling tools to the consistency of UML models

EXERCISE

i. One company has at least one employee
ii. A person who is married must be more than 18 years old
iii. The salary of an employee cannot be greater than the salary of his/her supervisor
iv. If an employer's age is above 60 years old, the minimum salary must be 5000.



i. Context Company
inv: self.employee → size()>0
Or
Context Company
inv: self.employee → notEmpty()

ii. Context Person
inv: self.isMarried → notEmpty() implies
self.age>18

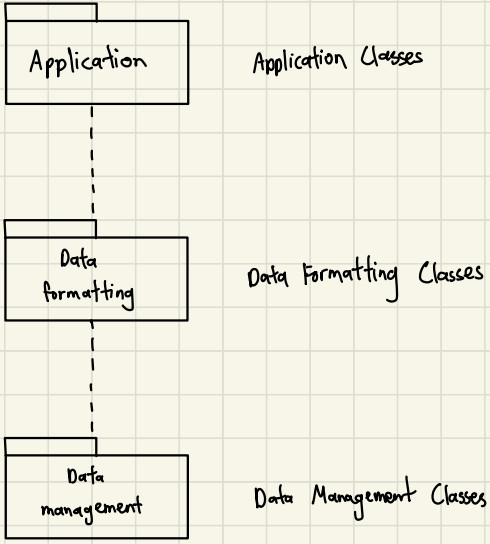
iii. Context Company
inv: self.employer → notEmpty() implies
self.employee.salary < self.employer.salary

iv. Context Company
inv: self.employer → select(age > 60) implies
self.employer.salary >= 5000

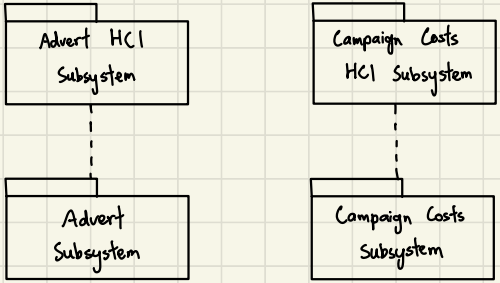
C10 : Software Architecture

Layering and Partitioning

- Layering
  - ↳ different subsystems usually represent different levels of abstraction
  - ↳ each subsystem is decomposed into many layers.
  - ↳ each layer is used to group the classes that will provide a particular service.



- Partitioning
  - ↳ each subsystem focuses on a different aspect of the functionality of the system as a whole.
  - ↳ decompose a software system into various subsystems.
  - ↳ each subsystem typically groups together elements of the system that share some common properties



Conclusion

↳ Both layering & partitioning helps system developers to improve maintainability and to maximize the reusability of a system.

C12 : System Design

Logical Design

↳ independent of the implementation language and platform
↳ high-level abstraction
↳ independent of specific technologies
↳ goal : define structure and behaviour
↳ components : class diagrams, interaction diagrams, use cases

Physical Design

↳ based on the actual implementation platform and the language that will be used
↳ low-level abstraction
↳ dependent on specific technologies (databases, programming languages)
↳ goal : define implementation and deployment
↳ components : deployment diagrams, file structure, database schemas

Qualities of Design

- **Reusable**

  Elements of the system can be reused in other systems (in the context of object-oriented system)

- **Usable**

  Provides users with a satisfying experience (not a source of dissatisfaction)

- **Maintainable**

  Design makes it possible for the maintenance programmer to understand the designer's intention.

- **Manageable**

  Easy to estimate work involved and to check progress

- **Buildable**

  Design is not too complex for the developer to be able to implement it

- **General**

  General-purpose and portable (mainly applies to utility programs)

- **Flexible**

  Capable of being adapted to new uses, to run in different countries, or to be moved to a different platform

- Secure

  Protect against errors, attacks and loss of valuable data

- Reliable

  Not prone to hardware or software failure, will deliver the functionality when the users want it

- Economical

  Running costs of the system will not be unnecessarily high

- Efficient

  The system performs those functions efficiently in terms of to time and resources

- Functional

  The system will perform the functions that are required

Criteria for Good Design

- Cohesion
  - ↳ a measure of the degree to which an element contributes to a single purpose
  - ↳ maximise cohesion : produces modules that perform a clearly defined process / a group of processes that are functionally related to 1 another.
  - ↳ how single-minded a module (class, object or method) is within a system.
  - ↳ a class or object should only represent one thing, and a method should only solve a single task. Thus, high cohesion is desirable in a system.
  - ↳ focus on how much the functionality are related to each other within the module

- Coupling
  - ↳ the degree of interconnectedness between design components.
  - ↳ determined by the number of links an object has and by the degree of interaction the object has with other objects
  - ↳ Minimise coupling : produce modules that are independent of 1 module
  - ↳ level of interdependency or interrelationship among the modules (classes, objects and methods) in a system.
  - ↳ the higher the interdependency is, the more likely changes in part of a design can cause changes to be required in other parts of the design. Thus, you want to minimize coupling in a system, if possible.
  - ↳ focus on how much one module is dependent on the other program modules within the whole application