COMP3004 Midterm Notes

William Findlay

February 17, 2019

Contents

List of Figures

List of Tables

List of Listings

1 Software Engineering

- what is it?
  - requirements analysis
  - building a *software system*
- why is it necessary?
  - systems get huge and difficult to manage
  - we need a plan
  - *reliability*
  - *modifiability*

2 Build Models

- what is a model?
  - representation of how to build system
  - get a better idea of how to do it
  - clarify requirements

2.1 Functional Model (Elicitation)

- use case diagrams
- use case tables
- FR, NFR tables

2.1.1 Use Cases (Tables and Diagrams)

- see Figure 2.1 for components of use case diagrams and tables
- see Figure 2.2 for an example high level use case diagram
- see Figure 2.3 for an example detailed use case diagram
- see Table 2.1 and Table 2.2 for example use case tables

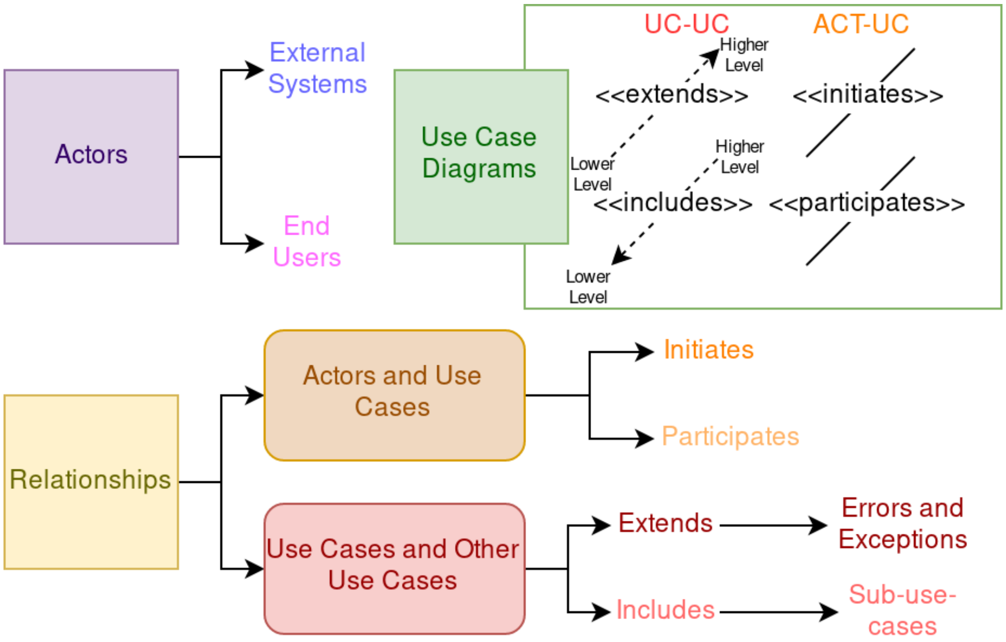


Figure 2.1: Components of use case diagrams and tables.

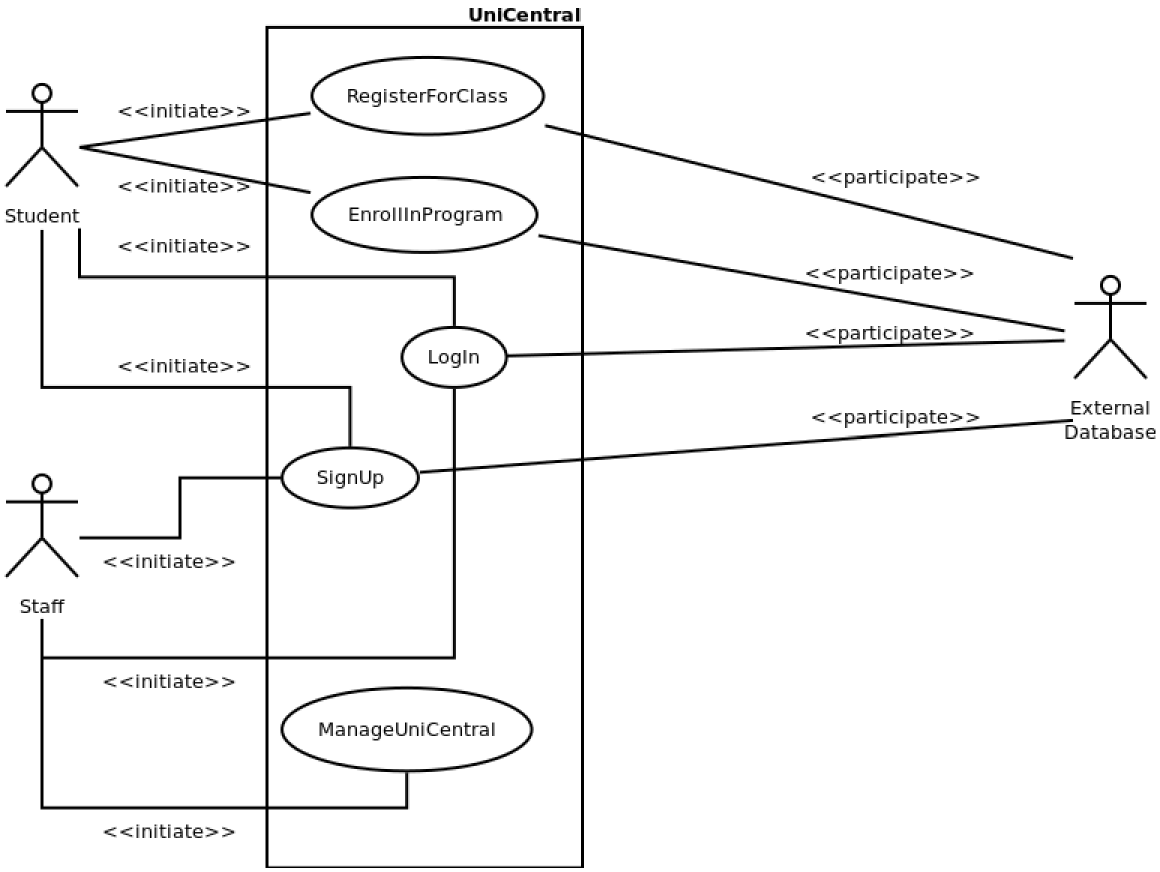


Figure 2.2: Example high level use case diagram.

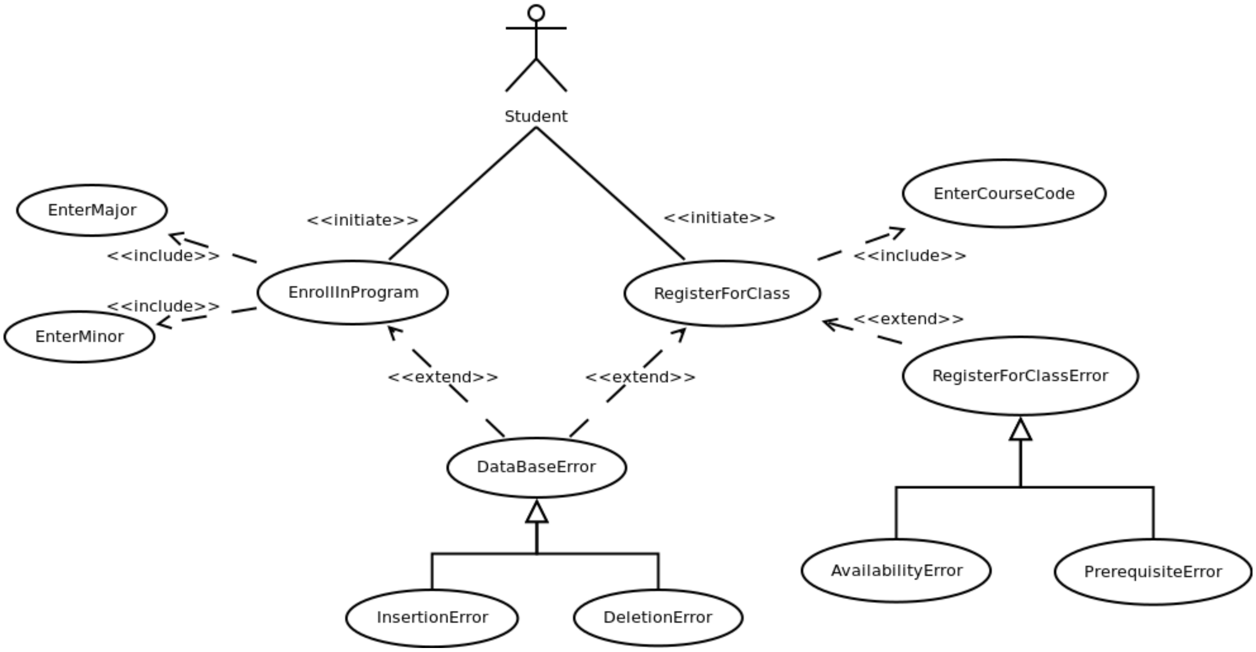
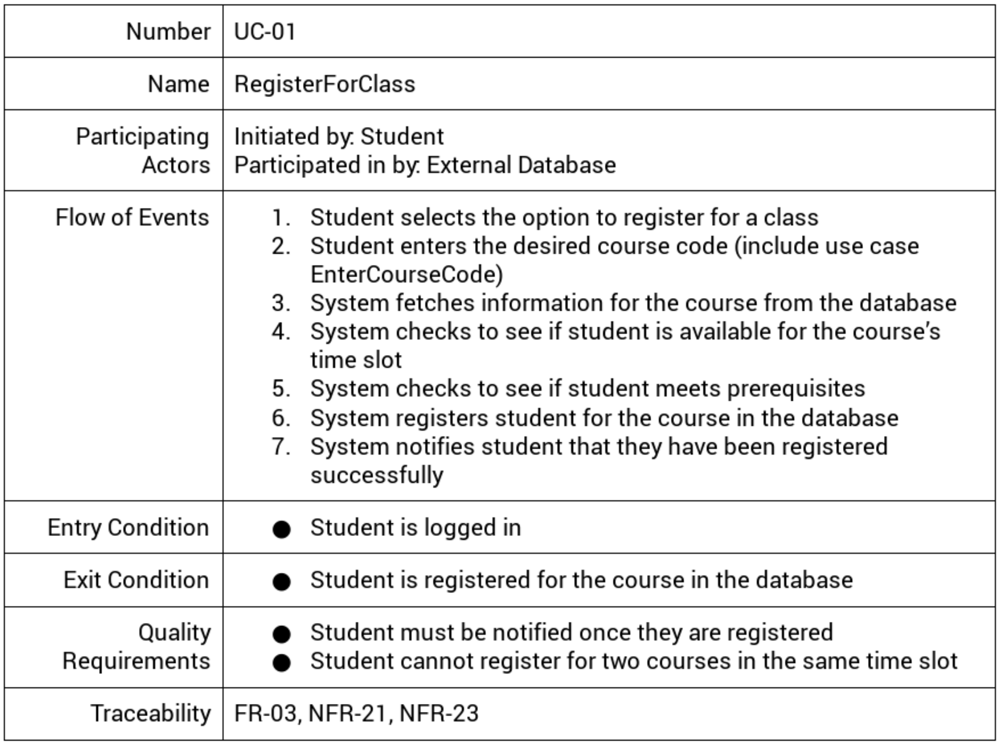


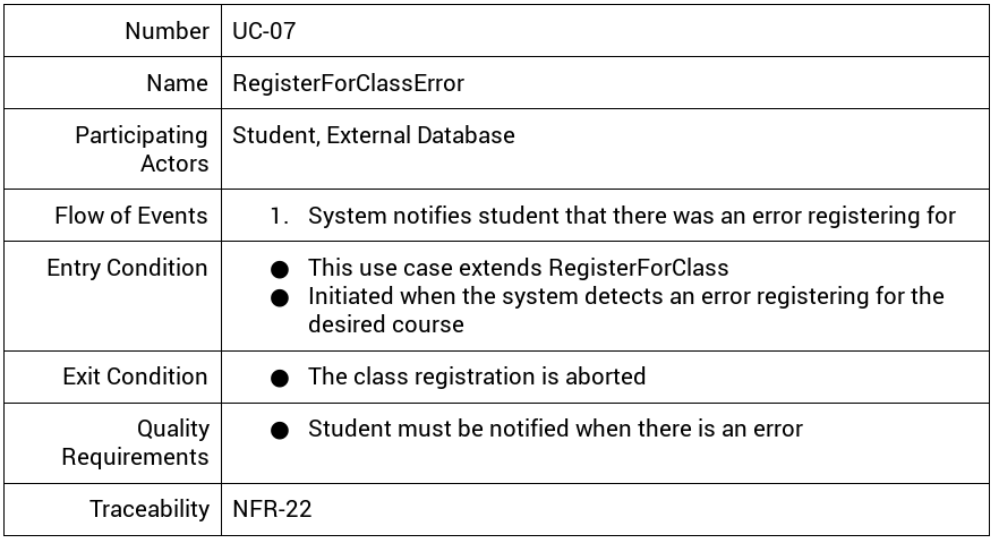
Figure 2.3: Example detailed use case diagram.

Table 2.1: An example use case table for a high level use case.



Number	UC-01
Name	RegisterForClass
Participating Actors	Initiated by: Student Participated in by: External Database
Flow of Events	1. Student selects the option to register for a class 2. Student enters the desired course code (include use case EnterCourseCode) 3. System fetches information for the course from the database 4. System checks to see if student is available for the course's time slot 5. System checks to see if student meets prerequisites 6. System registers student for the course in the database 7. System notifies student that they have been registered successfully
Entry Condition	● Student is logged in
Exit Condition	● Student is registered for the course in the database
Quality Requirements	● Student must be notified once they are registered ● Student cannot register for two courses in the same time slot
Traceability	FR-03, NFR-21, NFR-23

Table 2.2: An example use case table for an extend use case.



Number	UC-07
Name	RegisterForClassError
Participating Actors	Student, External Database
Flow of Events	1. System notifies student that there was an error registering for
Entry Condition	● This use case extends RegisterForClass ● Initiated when the system detects an error registering for the desired course
Exit Condition	● The class registration is aborted
Quality Requirements	● Student must be notified when there is an error
Traceability	NFR-22

2.1.2 FURPS+ Requirements (Tables)

2.2 Dynamic Model (Analysis)

2.2.1 State Machines

2.2.2 Sequence Diagrams

2.2.3 Activity Diagrams

2.3 Object Model (Analysis)

2.3.1 Class Diagrams

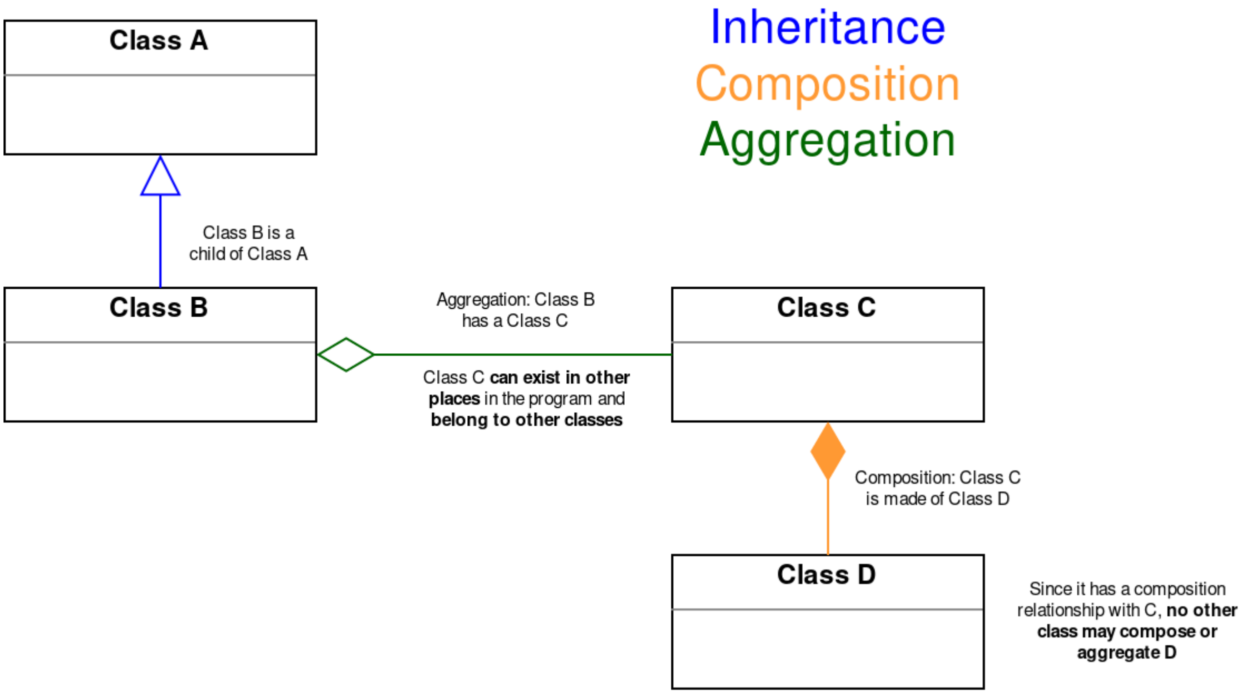

Inheritance
Composition
Aggregation
Class A
Class B is a child of Class A
Class B
Aggregation: Class B has a Class C
Class C **can exist in other places** in the program and **belong to other classes**
Class C
Composition: Class C is made of Class D
Class D
Since it has a composition relationship with C, **no other class may compose or aggregate D**

Figure 2.4: Inheritance, composition, and aggregation in UML class diagrams.

2.3.2 Data Dictionaries

2.4 Traceability

- required changes?
  - ➢ traceability lets us figure out *what parts are affected*
- numbers on all table rows
  - ➢ FR-01, . . .
  - ➢ NFR-01, . . .
  - ➢ UC-01, . . .

3 Software Development Life Cycle

1. Requirements Elicitation
2. Analysis

Client Knowledge Disappears

3. High Level System Design
4. Detailed Object Design
5. Implementation

Client Knowledge Reappears

6. Testing
7. Deployment and Maintenance

4 Requirements Elicitation

- what does the client want?
- requirements (FURPS+)
  - functional
    - what do the actors do?
  - non-functional
    - constraints
    - quality requirements
- scenarios, use cases
- work products
  - functional model
    - FR, NFR
    - use case diagrams

5 Analysis

- work products
  - object model
    - class diagrams
  - dynamic model
    - sequence diagrams
    - state machine diagrams
    - activity diagrams

6 High Level System Design