COMP3004 Midterm Notes

William Findlay

February 21, 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

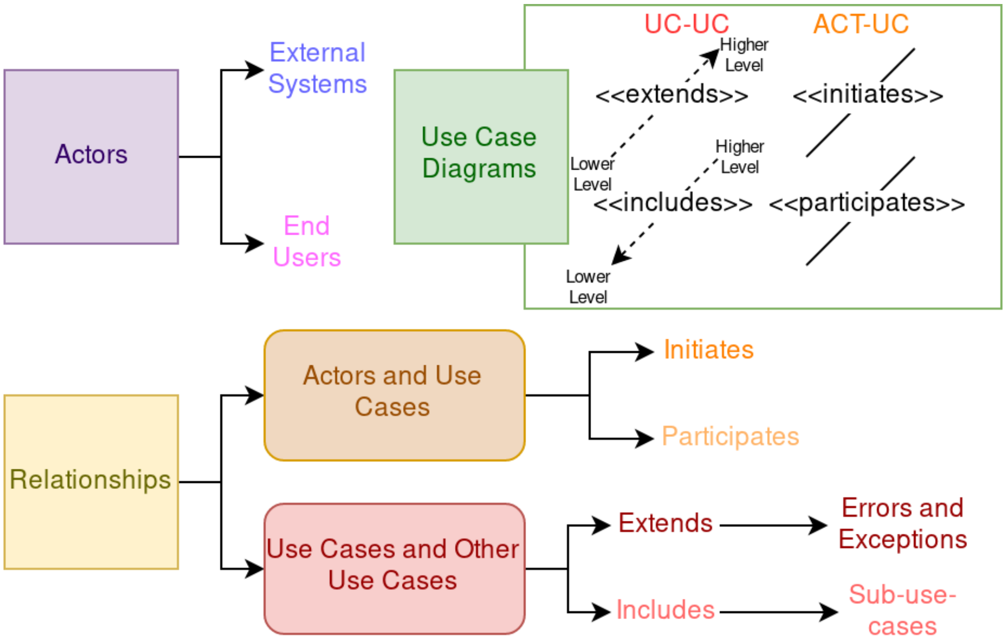


Figure 2.1: Components of use case diagrams and tables.

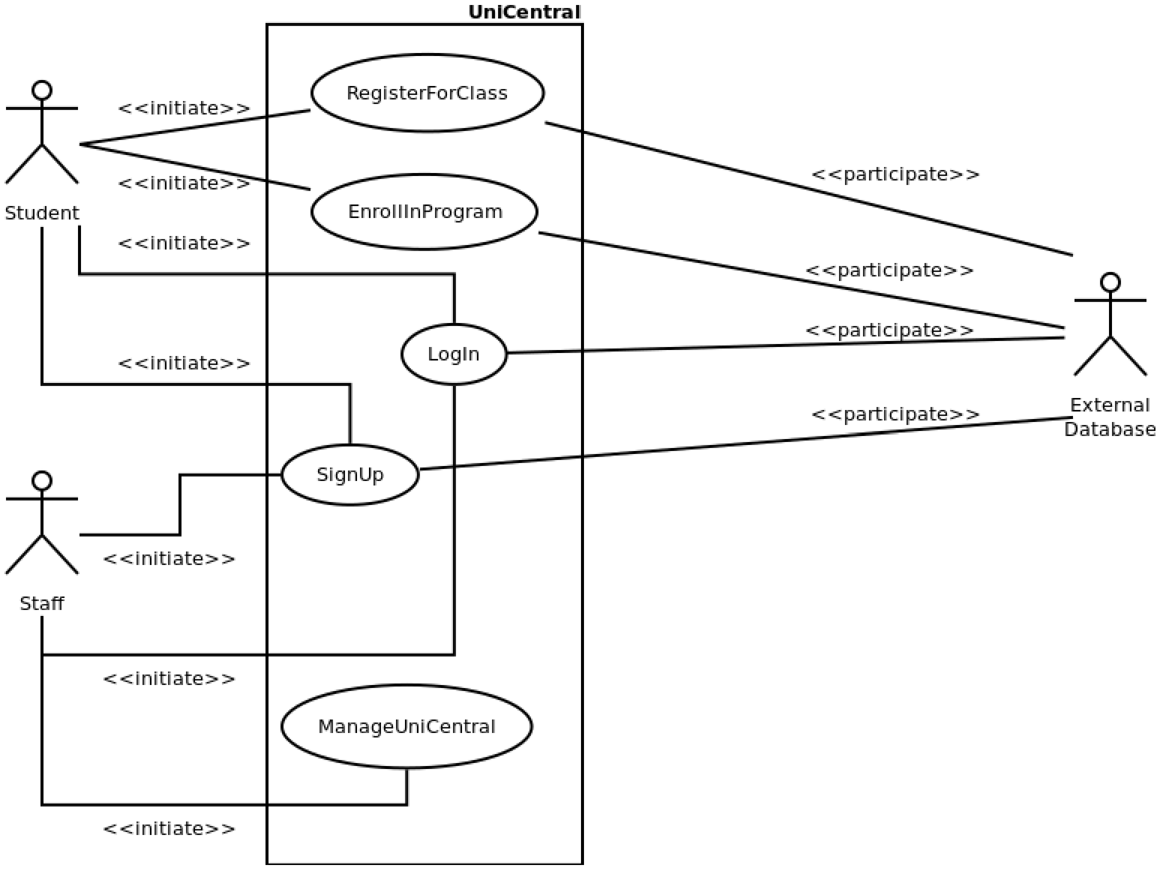


Figure 2.2: Example high level use case diagram.

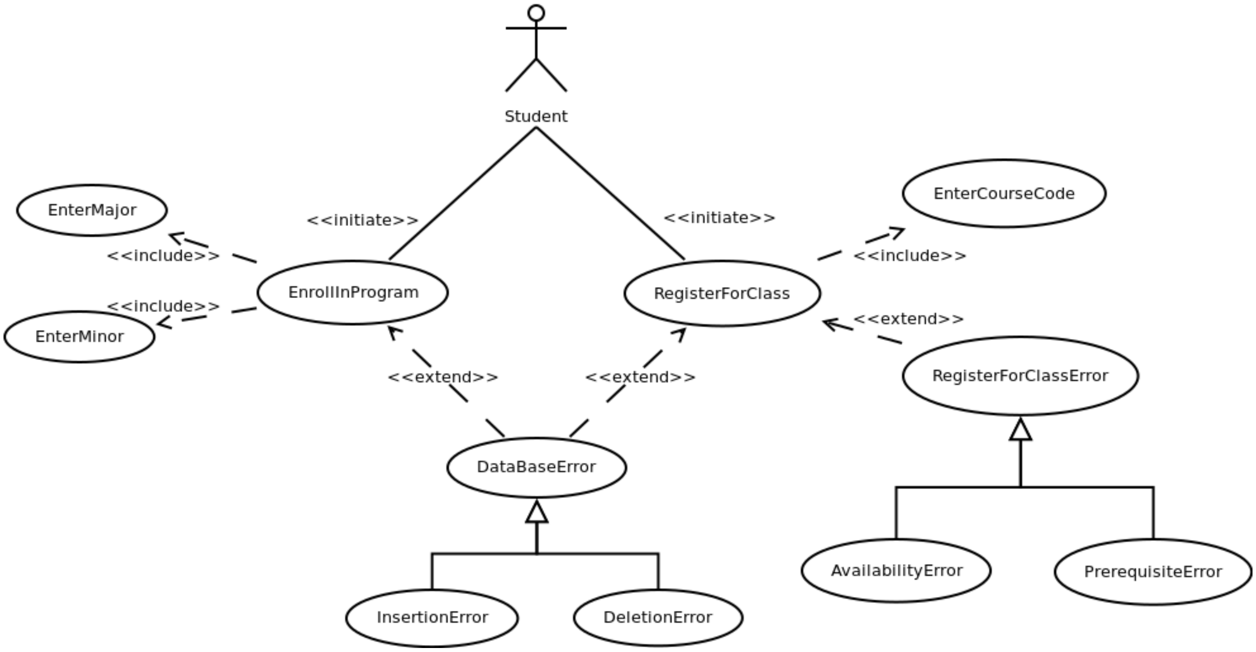
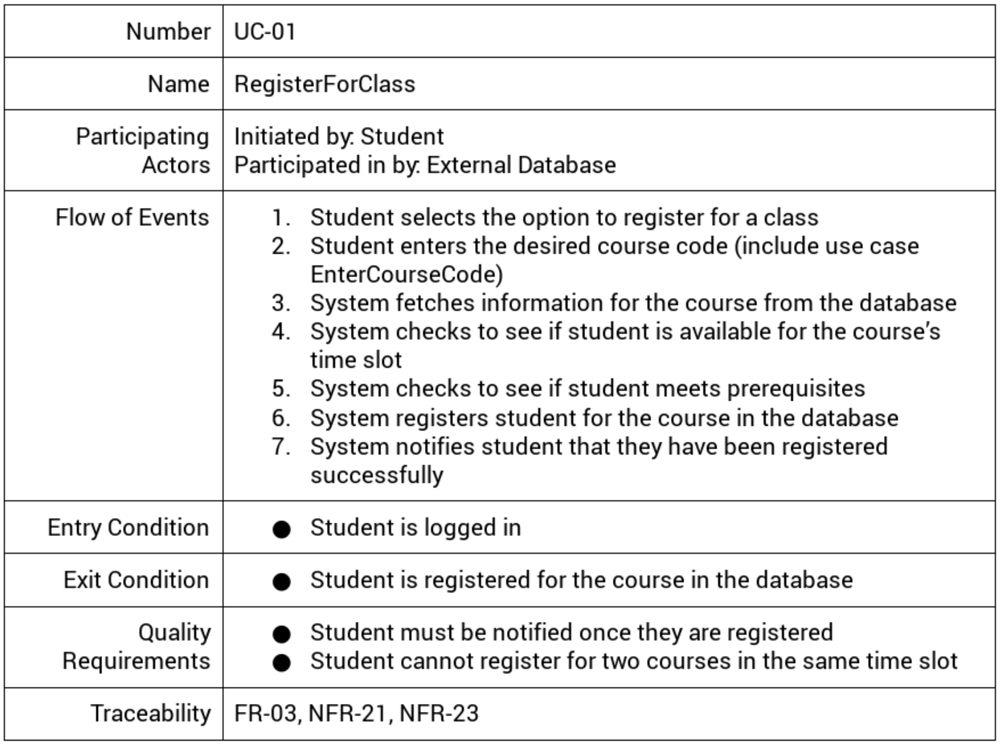


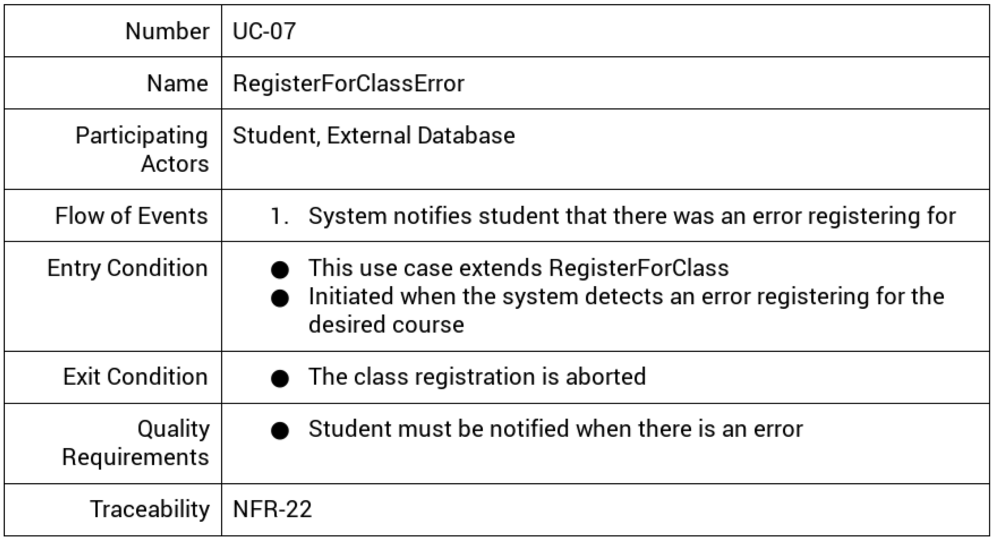
Figure 2.3: Example detailed use case diagram.

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.



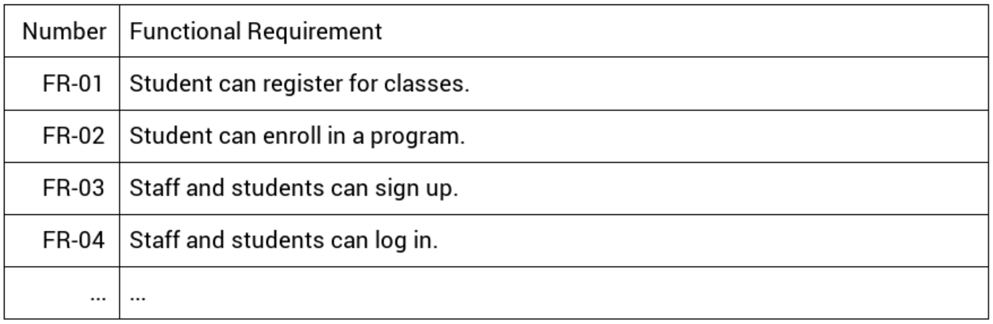
Number	UC-07
Name	RegisterForClassError
Participating Actors	Student, External Database
Flow of Events	1. System notifies student that there was an error registering for
Entry Condition	● This use case extends RegisterForClass ● Initiated when the system detects an error registering for the desired course
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)

Functional
Usability
Reliability
Performance
Supportability
+ Operation, Interface, Implementation, Packaging, Legal

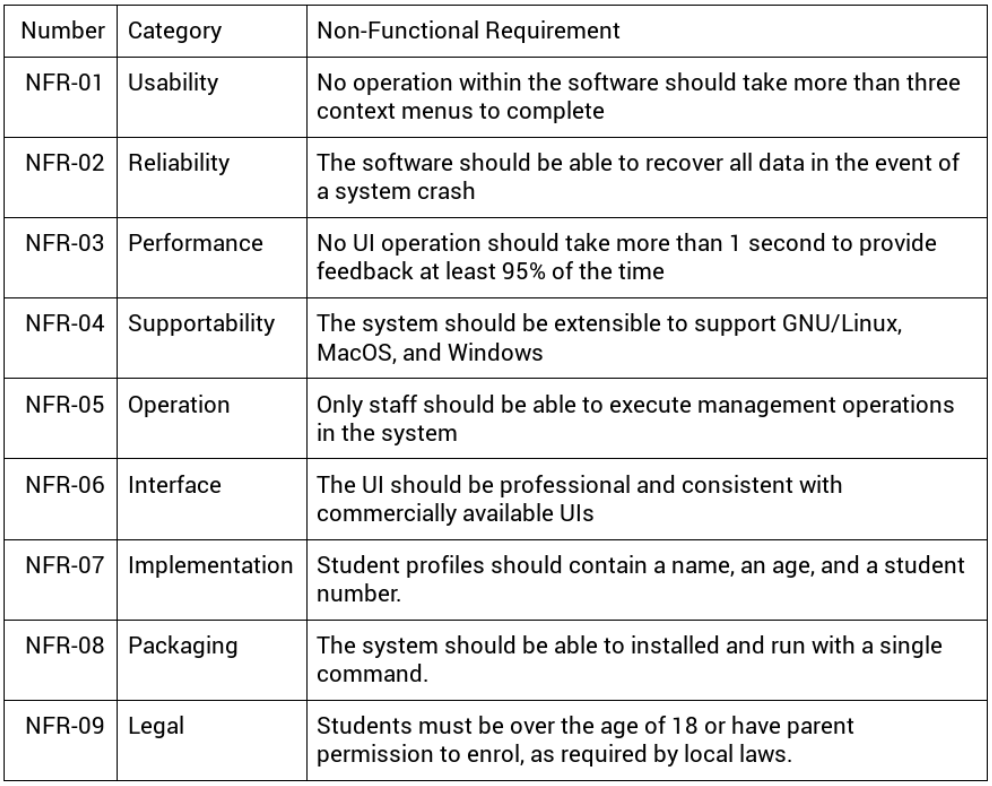
- types of requirements
    - functional
        - what can the actors do?
    - usability
        - ease of use requirements
        - measurable, specific
    - reliability
        - recovery from error
        - stability
        - security
    - performance
        - how the system performs under certain conditions
        - specific, quantifiable
        - realistic
    - supportability
        - what kinds of platforms/hardware can the system run on
        - ability for future maintenance
    - implementation
        - implementation-specific requirements
    - interface
        - how the system interacts with the actors
        - UI stuff that doesn't fall under usability
        - how it interfaces with external systems
    - operation
        - which users are allowed to do what
        - constraints on operation
    - packaging
        - how the system should be delivered to the customer
    - legal
        - any legal restrictions on the software
- see Table 2.3 for a functional requirements table
- see Table 2.4 for a non-functional requirements table

Table 2.3: An example functional requirements table.



Number	Functional Requirement
FR-01	Student can register for classes.
FR-02	Student can enroll in a program.
FR-03	Staff and students can sign up.
FR-04	Staff and students can log in.
...	...

Table 2.4: An example non-functional requirements table.



Number	Category	Non-Functional Requirement
NFR-01	Usability	No operation within the software should take more than three context menus to complete
NFR-02	Reliability	The software should be able to recover all data in the event of a system crash
NFR-03	Performance	No UI operation should take more than 1 second to provide feedback at least 95% of the time
NFR-04	Supportability	The system should be extensible to support GNU/Linux, MacOS, and Windows
NFR-05	Operation	Only staff should be able to execute management operations in the system
NFR-06	Interface	The UI should be professional and consistent with commercially available UIs
NFR-07	Implementation	Student profiles should contain a name, an age, and a student number.
NFR-08	Packaging	The system should be able to installed and run with a single command.
NFR-09	Legal	Students must be over the age of 18 or have parent permission to enrol, as required by local laws.

2.2 Dynamic Model (Analysis)

- state machines
- sequence diagrams
- activity diagrams

2.2.1 State Machines

- diagram for each use case
- models system state
- initial state
  - ➢ dark circle
- final state
  - ➢ dark circle surrounded by light circle
  - ➢ looks like a target
- other states
  - ➢ bubbles with verb phrases
- transitions with labels
  - ➢ “from initial” or “to final” optionally has no label
- Figure 2.4 for an example

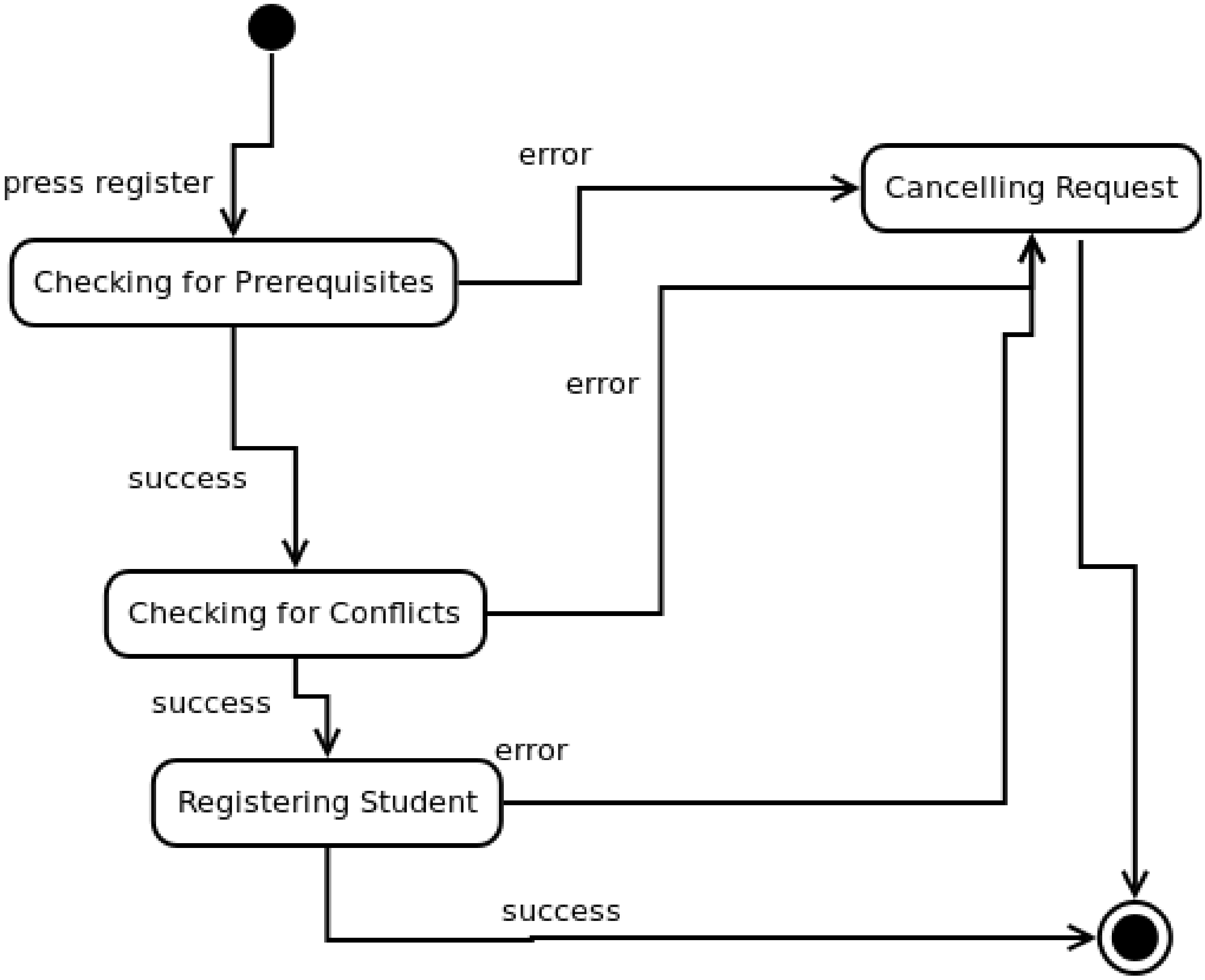


Figure 2.4: An example state machine diagram.

2.2.2 Sequence Diagrams

- diagram for each use case
- lifeline from each object
  - ➢ actors and boundary objects get infinite lifeline
  - ➢ other objects get destroyed with an X
- rectangle to indicate "focus of control"
- arrows with labels for actions
  - ➢ `select()`
  - ➢ `<<create>>`
  - ➢ `notify()`
  - ➢ `send()`
  - ➢ etc.

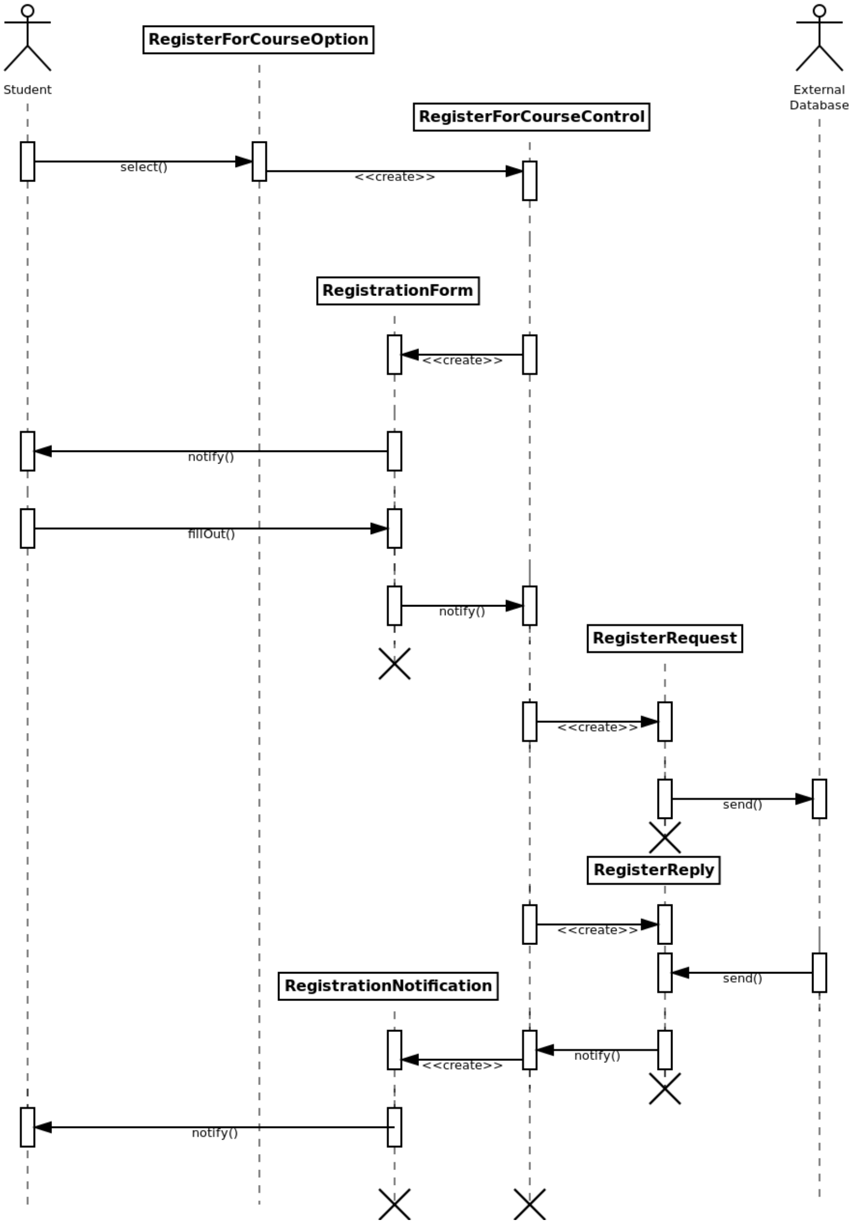


Figure 2.5: An example sequence diagram.

2.2.3 Activity Diagrams

- diagram for each use case
- bubbles represent use cases
  - ➢ labeled with verb phrases
  - ➢ connected with arrows
- black bars to split and join arrows

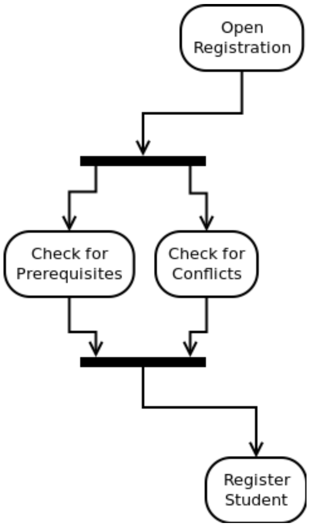


Figure 2.6: An example activity diagram.

2.3 Object Model (Analysis)

- class diagrams
- data dictionaries
  - ➢ define objects
  - ➢ list attributes and associations
  - ➢ explain when an attribute is set

2.3.1 Class Diagrams

- relationships
  - ➢ inheritance
  - ➢ composition
  - ➢ shared aggregation
- associations
  - ➢ directionality
  - ➢ cardinality
  - ➢ aggregation or composition
- classes
  - ➢ attributes
  - ➢ operations
- abstract classes
  - ➢ italic names
- instances
  - ➢ **instance_name:class_name**

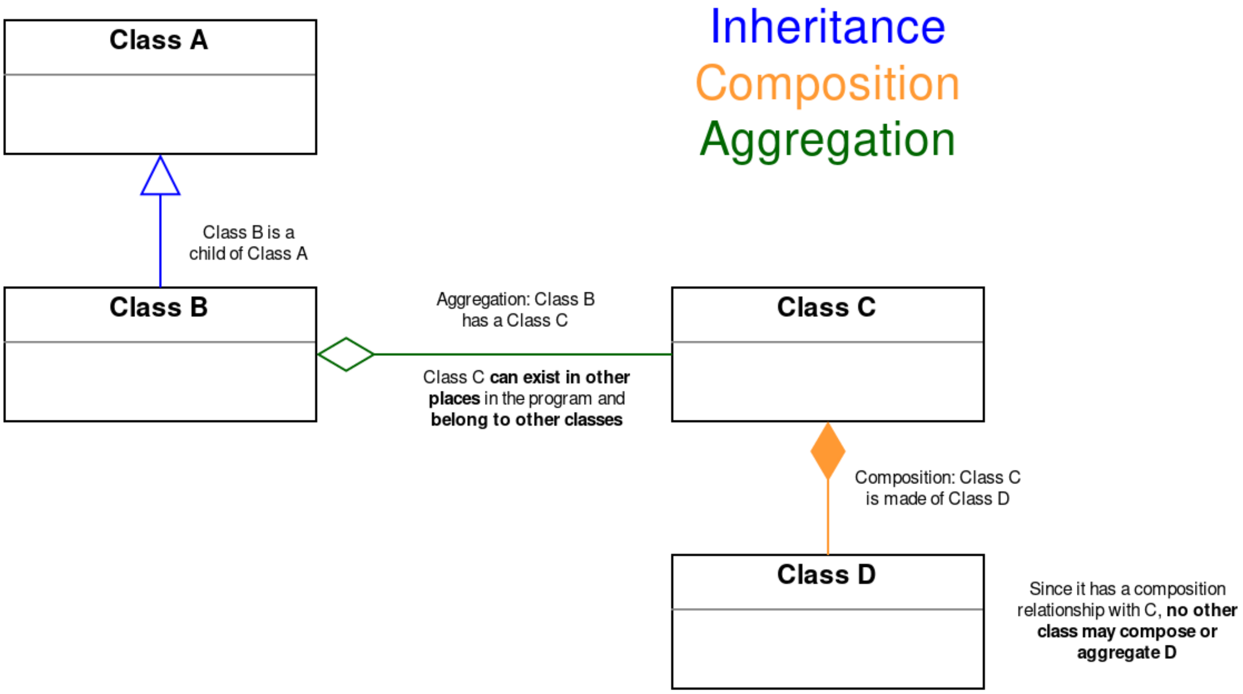

Class A
Inheritance
Composition
Aggregation
Class B is a child of Class A
Class B
Aggregation: Class B has a Class C
Class C
Class C **can exist in other places** in the program and **belong to other classes**
Composition: Class C is made of Class D
Class D
Since it has a composition relationship with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

Figure 2.7: Inheritance, composition, and aggregation in UML class diagrams.

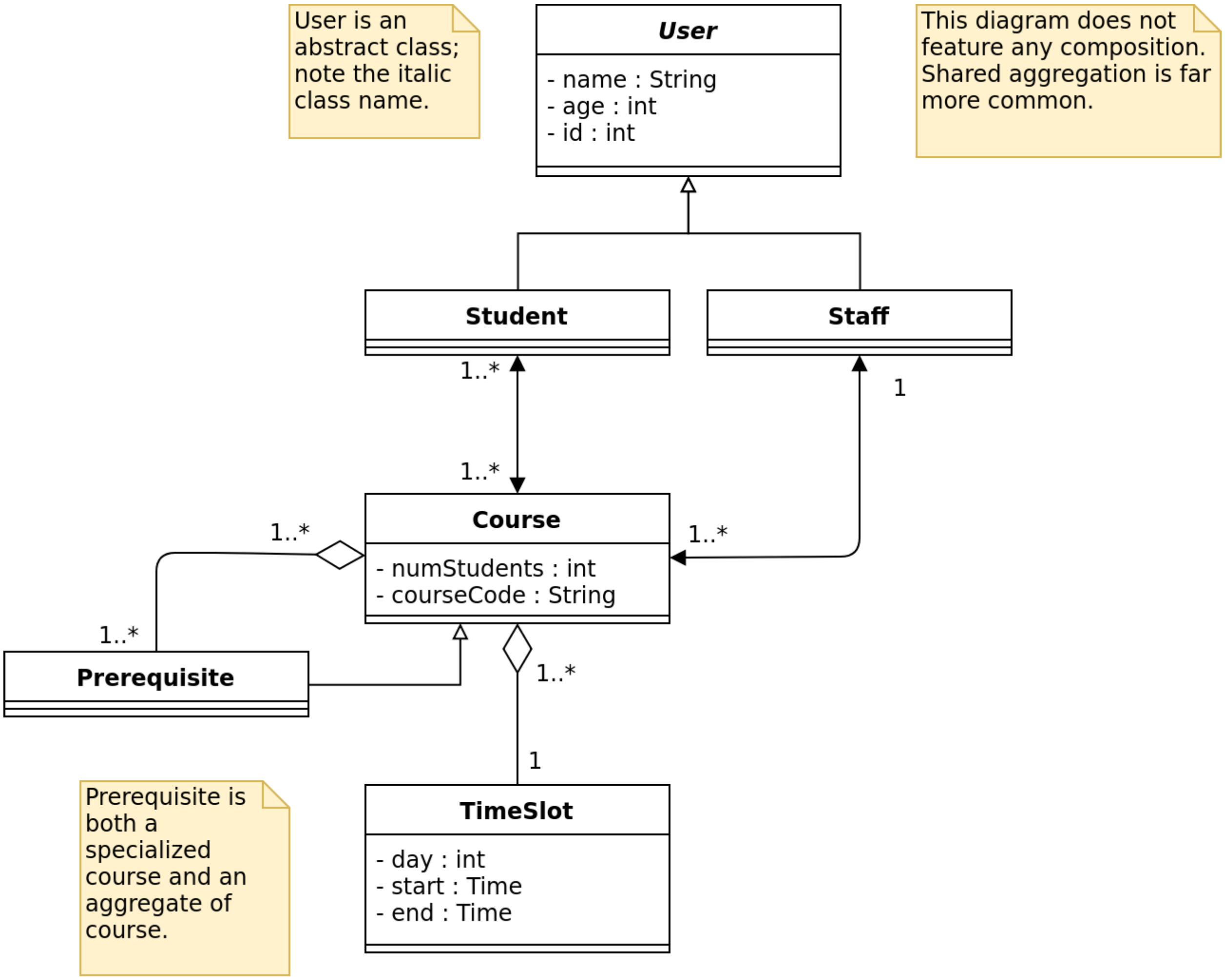
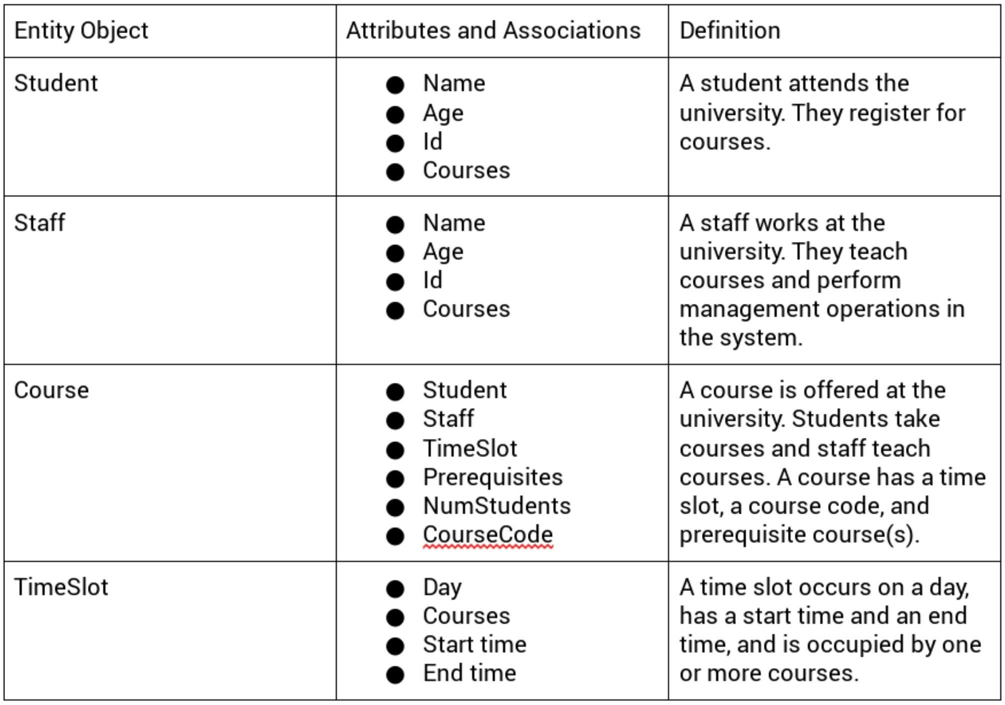

User is an abstract class; note the italic class name.
User
- name : String
- age : int
- id : int
This diagram does not feature any composition. Shared aggregation is far more common.
Student
Staff
1..*
1
1..*
Course
- numStudents : int
- courseCode : String
1..*
1..*
1..*
Prerequisite
1..*
1
TimeSlot
- day : int
- start : Time
- end : Time
Prerequisite is both a specialized course and an aggregate of course.

Figure 2.8: An example class diagram.

2.3.2 Data Dictionaries

Table 2.5: An example data dictionary table.



Entity Object	Attributes and Associations	Definition
Student	● Name ● Age ● Id ● Courses	A student attends the university. They register for courses.
Staff	● Name ● Age ● Id ● Courses	A staff works at the university. They teach courses and perform management operations in the system.
Course	● Student ● Staff ● TimeSlot ● Prerequisites ● NumStudents ● CourseCode	A course is offered at the university. Students take courses and staff teach courses. A course has a time slot, a course code, and prerequisite course(s).
TimeSlot	● Day ● Courses ● Start time ● End time	A time slot occurs on a day, has a start time and an end time, and is occupied by one or more courses.

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

6.1 Subsystem Decomposition

- subsystem = group of *related* classes
    - logical = no run-time equivalent
    - physical = run-time equivalent

6.1.1 Layers and Partitions

- layer
    - group of subsystems providing related services
    - depends on lower level layers
    - knows nothing about higher level layers
- closed architecture
    - layer **only** uses layer *immediately* below it
    - loose coupling
    - overhead
- open architecture
    - layer uses any layers below it
- partitioning
    - group of peer subsystems
    - very loosely coupled
    - can operate independently of each other

6.1.2 Services and Subsystem Interfaces

- class interface

    - ➢ public operations of a class
- subsystem interface
    - ➢ public operations of all classes in the subsystem
- service
    - ➢ subset of related operations in a subsystem
    - ➢ named with a noun phrase
    - ➢ one subsystem can provide multiple services

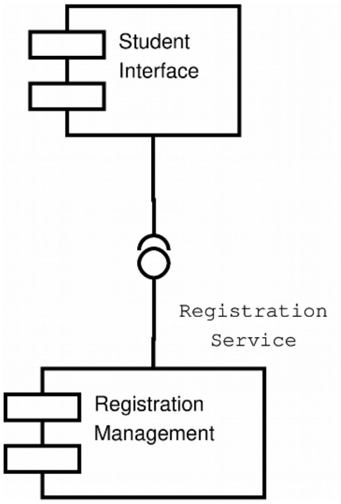


Figure 6.1: An example of a service provided by Registration Mangement to Student Interface.

6.1.3 Coupling and Cohesion

- coupling = how dependent is the subsystem on other subsystems?
- cohesion = how dependent is the subsystem on its components
- we want
    - ➢ **high cohesion**
    - ➢ **low coupling**

6.1.4 Architecture Styles

- grouping subsystems at the highest level

Repository.

- subsystems
    - ➢ access and modify a single data structure
    - ➢ independent
    - ➢ communicate through the repository
- control flow
    - ➢ repository has triggers on data
    - ➢ subsystems have repository locks
- examples
    - ➢ DBMS
    - ➢ compilers
- advantages
    - ➢ new services easily added

  - ➢ good for complex data processing
- disadvantages
  - ➢ bottleneck
  - ➢ high coupling between repository and subsystems

Model/view/controller (MVC).

- model
  - ➢ application domain knowledge
  - ➢ independent of view and control
- view
  - ➢ display to user
  - ➢ observer design pattern to propagate changes
- control
  - ➢ manage interaction sequences
- special case of repository
- example
  - ➢ multiplayer games
- advantages
  - ➢ loose coupling between model and view
  - ➢ maps well to boundary, entity, control

Client-server.

- server
  - ➢ provide services to client
  - ➢ handle requests with
    - ■ remote procedure calls
    - ■ sockets
- client
  - ➢ interact with users
- client and server are **independent**
  - ➢ independent control flow
  - ➢ synchronize only on requests and replies
- can be a special case of repository
- example
  - ➢ central IT database
- advantages
  - ➢ distributed systems
  - ➢ multiple clients and servers

Peer-to-peer.

- generalization of client-server
- a subsystem can be both client and server
- subsystems
  - ➢ independent control flows
  - ➢ synchronize only on requests and replies
- example
  - ➢ database that requests and notifies of changes
- advantages
  - ➢ distributed systems
  - ➢ multiple clients and servers
- disadvantages
  - ➢ deadlock if two send a request at the same time

Three-tier.

- three layers
- interface
  - boundary
- application
  - control and entity
  - processing and notification
- storage
  - persistent storage of data
  - can be shared by multiple applications
- example
  - cuACS

Four-tier.

- just like three-tier but:
  - interface layer is separated into:
    - presentation client layer
    - presentation server layer
- different clients $\implies$ different UIs
  - common data across all UIs is handled by the presentation server layer
- example
  - web browser

Pipe and filter.

- filters
  - subsystems
- pipes
  - association between subsystems
- filters are independent
  - synchronized by pipes
- examples
  - BASH
  - other UNIX-like shells
  - `ps aux | grep william | sort | less`
- advantages
  - good for streams of data
- disadvantages
  - not good for complex interactions between filters