1. Mapping done.
2. Each use case somewhat maps to a set of subsystems.
   1. Each use case somewhat maps to a set of subsystems.

Use Case:

* + 1. Manage Course (Add, Remove, Update)
       1. CourseManager, Course, CourseScheduler, CourseType, CourseView, UserInterface
    2. Manage Course Offering (Add, Remove, Update)
       1. CourseManager, Course, CourseScheduler, CourseOffering, Operator, CourseView, UserInterface
    3. Assign Operator to Course Offering
       1. CourseScheduler, CourseOffering, Course, Operator, UnavailabilityRecord, CourseView, UserInterface
    4. View Operator
       1. Operator, UnavailabilityRecord, RosterScheduler, Roster, Shift, SingletonController
    5. Give Shift to Operator
       1. Operator, Roster, UnavailablilityRecord, RosterScheduler, Shift, CourseOffering, Course, School, CourseScheduler
    6. Manage School (Add, Remove, Update)
       1. School, SchoolManager, SchoolView, UserInterface, SingletonController
  1. Some use cases do have a control object, while others do not.
     1. Manage Course (CourseManager control) [Mapped Above]
     2. Manage Course Offering (CourseOffering control) [Mapped Above]
     3. Assign Operator (CourseScheduler control) [Mapped Above]
     4. View Operator (No Control) (Mapped Above)
     5. Give Shift (CourseScheduler control) [Mapped Above]
     6. Manage School (SchoolManager control) [Mapped Above]
  2. The component-based system architecture seems to be used in the design of the subsystem, as it uses common libraries and standardised components throughout the codebase. Some design goals for this subsystem that should have been considered would be user-friendliness, good documentation, readability, reliability among some other helpful goals.
  3. The System here has relatively high coupling and very low cohesion, so it likely wasn’t a big consideration when originally designing the system. It could be improved by isolating the subsystems further than they currently are which would increase cohesion. Many classes provide information that another class does, somewhat breaking encapsulation. If encapsulation was improved, the coupling of the system would also decrease.
  4. The design model is readable, the rough map of subsystems can be within the system have names that are relatively unique, but not too a major degree. Many classes within the system also contains attributes with near identical names to many other classes in the system, which whilst does lead to them being related to similar concepts it also creates some conflict with the classes if necessary to use them together.
  5. The subsystem does not handle all boundary conditions, such as checking if a minimum amount of operators, course, course offerings, etc., exist before using them in operations. Furthermore there are nearly no additional checks throughout the code that confirm whether the data that is available is valid.
  6. The design does partially interfere with some non-functional requirements. The response time of the system is questionable with how it is coded. It also doesn’t take into account any wrong inputs from the user or system. These could be fixed or at least alleviated through ensuring that there are safety checks on any user input, and on critical system processes. Response time could be improved through better coding practices.
  7. The system uses the strategy pattern in regards to the type of course, which is either fire or medical. This lets the course type be more accurately defined for more situations then if it were a static variable. It also uses the observer pattern in regards to notifying an operator about shift/roster changes with the NotficationService class.

1. Done – Changed only the NotificationService class, adding the adaptor of FaxService to its usage.
2. Testing Strategies:

Create a testing control class which allows the user to directly test a subsystem.

A – Unit Test

Within Course class

Do setCourseName(“TestCourse”), which when using getCourseName() should return “TestCourse”.

Do setCourseName(15), which would crash the system due to having no testing for the input.

This is seen in this code, where there is no check for or way to catch any error.

public void setCourseName(String courseName) {

this.courseName = courseName;

}

Using a similar testing method with other set/get methods in other class in different classes should yield similar results.

B – Integration Test

Using CourseManager, Course and School

CourseManager - addCourse("testCourse", "This is a test course", "Test Content", "Learning outcome is test")

Course- addOffering(FakeSchool, 31/02/2024, 9:00 am, 3:00 pm)

If the system crashes it means there is no safety check to see if the school exists. This code ideally should have something to catch any errors.

public CourseOffering addOffering(School requestingSchool, LocalDate deliveryDate, LocalTime startTime, LocalTime endTime)

{

CourseOffering newOffering = new CourseOffering(this, requestingSchool, deliveryDate, startTime, endTime);

offerings.add(newOffering);

return newOffering;

}

C – System Test

Using CourseManager, Course and School, and SchoolManager

CourseManager - addCourse("testCourse", "This is a test course", "Test Content", "Learning outcome is test")

SchoolManager – addSchool(“FakeSchool”, “[fakeschool@gmail.com](mailto:fakeschool@gmail.com)”, “0412 345 678”, 500, “123 Fakeroad Rd, Faketown”)

Course- addOffering(FakeSchool, 31/02/2024, 9:00 am, 3:00 pm)

School- FakeSchool.getCourses()

Expected output should be testCourse, if not then it fails. If the system takes longer than X amount of time at any step, then it also fails.

D – User Acceptance Test

Ensuring all customer requirements are met.

Test boundaries, using negatives as such below.

SchoolManager – addSchool(“FakeSchool”, “[fakeschool@gmail.com](mailto:fakeschool@gmail.com)”, “0412 345 678”, -500, “123 Fakeroad Rd, Faketown”)

This will likely cause a large error, or crash due to not properly checking for a negative input.

public void addSchool(String[] data) throws DuplicateObjectException{

new School(data[0], data[1], data[2], data[3], Integer.parseInt(data[4]), data[5]);

}

Using decimals instead of integers also can work for testing.

Use previously working tests and run them again, as a form of regression testing to ensure that no newer changes have affected the system negatively.