Criteria 7 Critical analysis of your work and the techniques you have used.

Design Patterns are an integral part of coding methodology when developing software applications. This is because they are modular and can be used to help solve particular problems. They can be defined as a template or description to solve a problem that occurs in multiple instances while designing a software application or software framework. The language used is generic so they can be adapted to a range of different problems. The idea behind them is that you don't reinvent the wheel, you use previous working examples, it's much quicker and better tested. By practicing reusability of good design, you can keep design methods to a better standard. Design patterns are a vital part of Software Engineering because they are efficient and easy to maintain, reusable and provide transparency to the design of an application.

In my project, I used a variety of software engineering techniques, to make my code efficient, easy to understand and reproduceable.

MVC design pattern

A major design pattern I used in the assignment was the MVC pattern. Model, View, Controller pattern. The model relates to logic and is used to interact with the database using CRUD operations (create, retrieve, update and delete). The controller receives input from the model and passes data to the view, in my project the view is mainly concerned with the web content files, such as the index.jsp page. The view is concerned with representation, displaying the data to the user.

In my project the model acts as a starting position, with film.java and the DAO, data accessor object, the DAO holds a variety operations such as getAllFilms, getFilmByTitle, deleteFilm and insertFilm, these methods contain SQL query language which is used to interact with and manipulate the film database. The controller acts as a middle man, implementing the model, specifically the DAO class, and then does something with the data retrieved by the model. For example, the PopulateTable.java servlet class invokes getAllFilms() from the DAO, this gets a list of films from the database and then it converts the list of films into a JSON array, and serves requests to a web browser. The view, index.jsp retrieves the JSONArray and displays it in a table format on the browser so the end user can view the list of films in a readable form, rather than as a JSONArray.

There is a wealth of advantages to using the MVC design pattern. Firstly, it makes an application easy to understand because it is split into 3 sections, it separates the user interface from the logic. This is hugely advantageous when it comes to the development process as it keeps each separate part organised and means modification of one area doesn't affect the entire project, as each part has low dependency on one another, this in turn means an application can be developed quickly or worked on by multiple developers. It could be argued that a disadvantage of the MVC design pattern is that the controller is dependent upon on model, however making changes to the model doesn't affect the architecture, meaning it's relatively simple to make modifications. Ultimately, the separation of responsibility makes it easier to apply modifications for current or future use.

DAO design pattern

DAO or data accessor object is a design pattern used to access data from persistence storage, for example a database. It is used to perform the CRUD operations and it acts as an interface to the database.

It features three main parts, the value object, the data access object interface and the concrete class that implements the interface. For example in my project, the value object is Film.java, this is a POJO, plain old java object containing the getter and setter methods to store data that is retrieved by the DAO class. The data access object interface and concrete class implementation are in the FilmDAO.java class, this defines the operations used on the value objects for example, getFilmByTitle retrieves the corresponding film from the database.

An advantage of the DAO design pattern is that all storage details remain hidden from the rest of the application this means, the persistent storage can be changed relatively easily. For example in my project, when creating a Google app engine, I wanted to use a Google cloud database as the persistent storage instead of the MMU database, though I couldn't actually implement this, if I could get the DAO to link to the cloud database, this wouldn't affect the rest of the app. The rest of the application doesn't need to know where the data originates from, and all the changes only need to be implemented in the DAO class, making it easy to switch persistent storage. This is especially advantageous because it means a DAO can be reproduced for multiple projects with only some modification.

To evaluate, even though I could not get the amendFilms() part of the DAO properly working, I am happy with the DAO because it is easy to amend and easily maintainable, I found this especially useful when trying creating different SQL statements and implementing the getFilmsByID attribute. Although I couldn't get the DAO to connect with the Google Cloud Database, I am confident the interface and CRUD operations wouldn't need to be changed to interact with the cloud database.

Singleton design pattern

A singleton design pattern is used when you need to manage a shared resource, this stops conflict. For example when connecting to a database. An advantage of the Singleton design pattern is that it saves memory because an object is not created at each request. Only a single instance is reused again and again. The singleton pattern is implemented by creating a class with a method that create a new instance of the class if one no longer exists. Given the time I would of liked to include more Singleton design patterns in my code.

SE Techniques

Throughout the project I have used SE engineering techniques such as version control and commenting to help make a better project. These have been invaluable tools, as version control through github has allowed me to test different versions of the project, and revise it when errors have occurred. I've also tried to use correct name conventions