Development Proposal

DVD Selection System

Document Revision: **1.0**

Issue Date: 20 July 2012

Overview

Currently, all activities in DVD Selection – a DVD rental shop – are carried out manually, which makes it difficult to manage the processes of borrowing, renewing, returning and keeping track of the DVD resources. Therefore, the shop is considering developing a system to assist both its staff and customers in managing and borrowing the DVDs.

It is critical for the shop to have a high-quality system in place due to various reasons such as security, performance, availability and so on. Therefore, you are asked to use a *test-driven* approach to develop an e-shop system which needs to meet the following requirements.

Functional Requirements

1. Customer data (e.g. unique customer ID, password, last name, first name, and telephone), details of customers' loan (e.g. DVD borrowed, check-out date, due date), and details of all the DVDs in the shop (e.g. unique DVD barcode, title, director, starring, classification, publisher, genre, status[1]) are read from *two* files at program start-up.

   The first file should store customer and loan details. The second file should store details of the DVDs. The format of those files is your choice. **In your report you should include the test data that are stored in those two files.**

2. The system shall ask the customer to login with their ID and password. The system then checks this information against the customer data, and displays an appropriate error message if either the ID or the password is not correct.

3. If the system determines that the customer's password is invalid, the customer will be required to re-enter the password. If the customer is unable to successfully enter the password after **three** tries, his/her record is marked as *"locked"* and the customer is not allowed to login in the system.

[1] A DVD can be either Available, On Hold, or On Loan

4. If the system determines that the customer's record is "locked", the system will display a message to inform the customer that they will have to contact the shop to get it unlocked.

5. The system shall allow the customers to change their login password. A valid password must contain at least 8 characters and must include at least a number and a letter.

6. Upon successful login, the system shall display the customer's loan record, sorted by due date. If there is any overdue DVD, the system should display a warning message to the customer.

7. The system should allow the customer to renew a DVD (by entering its barcode) that they are borrowing. If the renew is successful[2], the due date is extended for 1 week. Otherwise, the system should display an error message.

8. The system should allow the customer to hold a DVD that is currently not available and is not hold by another customer.

9. If the customer has any overdue DVD, they are not allowed to renew or hold a DVD.

10. The system shall allow the customer to search for DVDs by *genre*.

11. Upon exit, the system shall update all details (customer data, loan details and DVD details) to the two files that are loaded earlier at program start-up.

Non-functional Requirements

1. The system is to be implemented in C++ and is to run on a Linux OS.

2. The system is to use a simple **textual** "menu-select" style of user interface.

Use of Integrated Development Environment – NetBeans

You are to create your solution as a set of C++ application projects within NetBeans.

- You are strongly advised to work in the laboratories for these have the NetBeans environment already set up and the necessary libraries (such as the cppUnit framework library) installed.

- You can install the development environment on your own machine and do some of the work at home; but you really should participate in scheduled laboratories as well.

The projects for classes and associated unit testing code will need to have compiled and linked properties both adjusted to include the cppUnit libraries (the header files for compile time, the compiled code at link time). The lecture notes on testing and cppUnit contain illustrations showing how this can be done.

Use of cppUnit

The file with the TestDriver code should be physically copied into each project that involves unit testing. This code is never changed.

[2] On-loan DVDs are allowed to renew if they are not on hold.

As noted above, you must edit the C++ “compile” and “link” properties of a project to reference the include directory containing the cppUnit header files and the libcppUnit.a file respectively. The lecture notes contain an illustration.

If you did the exercises in the laboratory, you will have learnt how to configure a project to use extra libraries.

The cppUnit classes will be subclasses of CppUnit::TestFixture.

Use CPPUNIT_TEST_SUITE and CPPUNIT_TEST macros to build the TestSuite instance for this class.

Work incrementally on each class. Define some operations in your class and add a test function to your tester class. Use CPPUNIT_ASSERT macros in your tester class to validate operations. Iterate through test/refactor/test until the new operations are working correctly (at the same time retesting all previously implemented operations).

Use of gcov code coverage tool

gcov is a part of the gnu suite of C/C++ tools; if you are using a different C++ environment you will have to identify the equivalent code coverage tool and resolve how it is used.

The lecture notes explain the usage of gcov.

If you did the exercises in the laboratory, you will have learnt how to configure and run an application with code coverage incorporated, and how to generate actual code coverage reports.

Generate reports that show graphically the number of times code corresponding to individual source statements has been executed.

gcov reports are long – it will suffice to include just a couple of screen shots showing parts of reports (or copy & paste some segment of text from a gcov report into your report and tidy formatting).

Suggested timetable

1. Week 2 (first week of laboratories): Complete exercises on basic use of NetBeans/C++ development. Start your analysis of the system requirements.

2. Week 3: Complete exercises involving partially implement Point and Bitmap classes and the use of cppUnit and gcov. Implement some basic functionalities.

3. Week 4: Refactor/refine existing code and implement the remaining functionalities. Develop unit test classes. Begin work on A1.pdf report with a draft of the sections presenting the classes, the use of cppUnit, and your incremental development approach.

4. Week 5: Test the whole system and fix bugs if found. Perform code coverage testing. Complete and submit the report.