Principles and Practices of Software Production

01/04/2011

David Russell | James Nightingale | Amanda Patterson | Scott Dennison

TABLE OF CONTENTS

SECTION 1 – PLAN, STRATEGY & DOCUMENTATION

1.1 - 2.3.3_01: PLAN

- Build quality assurance document
- Establish final set of requirements
    - These are based on the partner groups requirements and feedback from Kelvin
- Discuss and document on methods to implement the system
    - Language
    - IDE
- Establish who will do what
    - Developer
    - Analysts
    - Test Case Engineers
    - Testers
- Establish configuration management
    - Keep track of version control and changes
- Begin development of system
    - Unit tests
    - Quality Audits
    - Implementation reviews
    - Testing
        - White Box:
            - Basis path testing
        - Black Box:
            - Stress testing
- Build/Engineer test cases
    - Build tests from requirements
    - Visual/Flow charts
        - Testing logical flow through the system
    - Black box
    - White box
- Finish development
    - Alpha/Beta tests
    - Acceptance testing

1.2 - 2.3.3_02: STRATEGY

STAGE ONE

REVIEW SPECIFICATION

- Ensure entire team has a clear understanding of what is required.
- Establish a clear idea of how development is to proceed.
- Produce a final, clear document of all system requirements.
- Break down into assignable tasks.
    - To be delegated to correct team members.
- Ensure team members with tasks understand what is required of them.

STAGE TWO

IDENTIFY RESOURCES AND RESOURCE REQUIREMENTS

Developers	Begin development: • Unit test all methods as they are written. • Ensure work is kept to a standard defined in Quality Assurance. • Quality audits to be carried out periodically.
Test Engineers	Engineer test cases: • Basis Path testing. • Test cases to match requirements. • Usability tests. • Acceptance tests.
Testers	Carry out tests developed by test engineers: • Develop reports on test results. • Compare to expected results.
Analysts	Analyse test reports: • Establish areas of improvement, • Ensure system is sticking to requirements.
Project Manager	Team management: • Carry out quality audits. • Ensure team is adhering to quality strategy.

AVAILABLE RESOURCES

Resource	Roles
David Russell	• Project Manager • Developer • Tester
Amanda Patterson	• Tester • Analyst
James Nightingale	• Developer • Tester
Scott Dennison	• Developer • Tester

1.3 - 2.3.3_03: DESIGN REVIEW

OVERVIEW

Planned to meet at 2pm, Tuesday 8th of April in Octagon concourse. Aim is to exchange documents and establish which requirements in each specification are unneeded and if there are any immediate changes either group will make to the specification.

MINUTES

14.00 – Everyone arrived but Scott – Text message says he's running late and that he is on his way

14.05 - Decided to go ahead with exchange whilst waiting for Scott. Exchanged our documents with BHH. BHH does not have access to their assessed requirement specification. Began to go through our requirements with BHH, eliminating unneeded requirements.

14.30 – Scott arrives. Finished requirement exchange with partner group. As BHH do not have their assessed document, they go to find Kelvin to get access to their document.

14.40 – Waiting on Kelvin to finish tutorial for BHH document feedback. Discussed possible implementation with BHH.

14.50 – Amanda has to leave for an exam. Members of BHH group go to find Kelvin again.

14.55 – Richard (BHH) has to leave for lecture.

15.00 - BHH unable to locate Kelvin, agreed to email their assessed requirements ASAP.

19.07 – BHH email their scanned, assessed requirements. Evidence located: 2.5.5_02CommLog02_DesignReview

1.4 - 2.3.3_04: REQUIREMENTS

3.1.1 Log operator onto the system.

- All operators must login to the system with a username and password to access the system.

3.1.2 Record the operators' bookings

- Record a log of the bookings by each operator.

3.1.3 Book Seats up to 6 hours before film

- Book up to 10 seats in one booking (1 row)
- Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11

3.1.4 Cancel Bookings

- Customer's bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for refund

3.1.5 View which seats are available

- Operators can view which seats are available.
- It is possible for customers to request which seats are available for booking.

3.1.6 Different prices for premium seats

- Rows H and I are premium seats

3.1.7 Able to set prices of premium seats as a percentage increase of the costs for standard seating

3.1.8 Store film information

- File name and age rating for each film as text information

3.1.9 Check customer is correct age for viewing the film

- System function, enter date of birth of customer when booking film, check against age rating of film and if age of customer is equal to or greater than the age rating of film then booking can be completed

3.2.1 Personal information not to be kept for more than one year on database

- No credit card information is to be kept of customer

3.5.1 Software colour scheme to be red, white and green

3.5.2 Minimise number of steps to complete booking to keep it as simple as possible

1.5 - 2.3.3_05: RESOURCE ALLOCATION

DEVELOPMENT

Language:	• Java
Resources:	• Scott Dennison. • James Nightingale. • David Russell.
Strategy:	• Initially develop functionality with Command Line. • Implement to web interface once fully functional.
Version Control and Repository:	• github.com/staffs-ppsp
Allocation:	• Scott: Booking and Film classes. • James: Staff and Customer classes (adding Person interface). • Dave: FilmShowing and Seats classes. • Scott: Web interface • James & Dave: Java back end functionality
Quality Assurance:	• Unit testing on methods as code is written. • Adhering to quality assurance document. • Testing carried out by Amanda.

TEST CASE ENGINEERING

Resources:	• Amanda Patterson.
Strategy:	• Writing test cases to ensure program meets the specification. • Writing test cases to ensure program is fit for purpose. • Audit tests to ensure coding meets agreed standards.

TESTING

Resources:	• Amanda Patterson. • David Russell. • James Nightingale. • Scott Dennison.
Strategy:	• Adhere to written test cases and testing standards to ensure usability. • Allocate specific testing to team members.

WHITE BOX

Resources:	• David Russell. • James Nightingale.
Strategy:	• Construct basis path test diagrams from code. • Follow logical paths from code and ensure flow is correct. • Construct unit tests. • Ensure objects are constructed and referenced correctly.

BLACK BOX

Resources:	- Scott Dennison - Amanda Patterson
Strategy:	- Prepare test data. - Subject system to test data. - Record and analyse results. - Determine results are acceptable or within established boundaries.

ACCEPTANCE TESTING

Resources:	- Amanda Patterson. - David Russell. - James Nightingale. - Scott Dennison.
Strategy:	- Prepare test data which ensures the implementation meets requirements and functions as expected.

3.1.3 Book Seats up to 6 hours before film

- Book up to 10 seats in one booking (1 row)
  The interface that is shown does not allow this

- Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11
  ? Don't understand?

3.1.4 Cancel Bookings

- Customer's bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for refund
  There is no interface designed for this

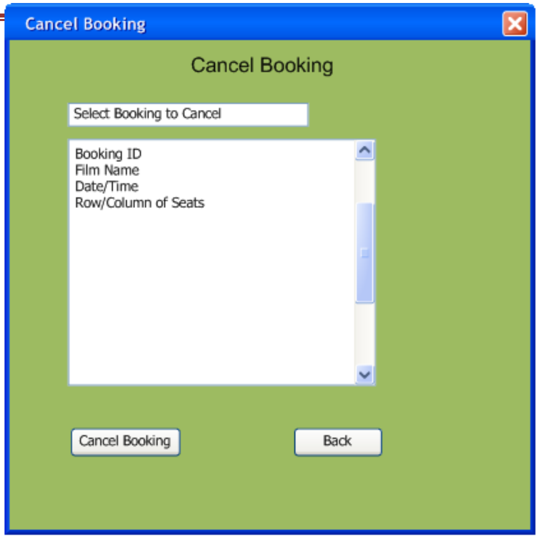
3.1.8 Store film information

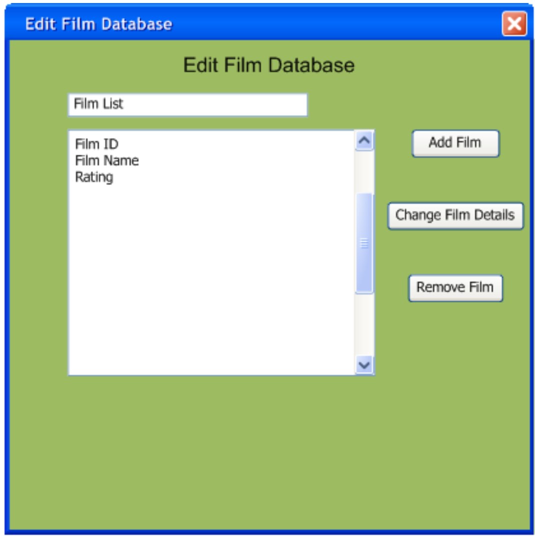
- File name and age rating for each film as text information
  there is no mention of being able to ADD films! Nor an interface

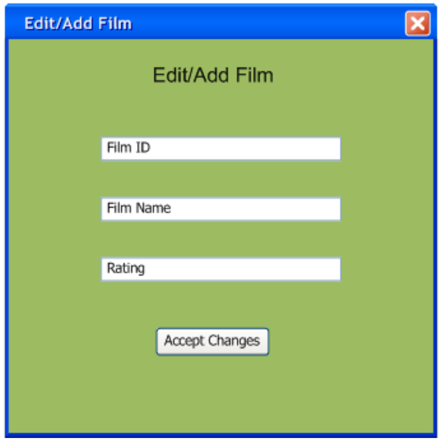
3.5.1 Software colour scheme to be red, white and green
Please can we discuss this with both the other team and the customer.

3.5.2 Minimise number of steps to complete booking to keep it as simple as possible.

1.7 - 2.3.3_07: ADDITIONAL DESIGNS







1.8 – 2.3.3_08: QUALITY ASSURANCE

The following points must be adhered to throughout the project.

Code

- Variables named correctly.
- Placeholders for code.
- Layout is industry standard/indentation.
- Comments to briefly describe functionality.
- Easy to understand flow of code.
- Separate into blocks/ classes.
- Correctly aligned vertically.
- Correct versioning.
- Variables declared properly.
- Easy to maintain, test and debug.
- Easy to fix, modify.
- Easy to read and understand by others - commented/ documented.

A comprehensive list of coding standards can be found in index 2.3.1_QualityDocuments.

The produced software must also meet the following standards.

Product

- Use of requirement's or program specification.
- No bugs.
- Completed product.
- Documentation.

SECTION 2 - STANDARDS

2.1 – 2.3.1.1_01: CODING STANDARDS

COMMENTING STYLE

As the coding is to be a collaborative effort, developers must adhere to the commenting style standards defined here. This is so other developers who may need to read, update or remove code will be able to understand with ease the functionality of methods, algorithms or data structures in the code. Comments must not impact the formatting conventions, for example, comments must not make altering blocks of code unnecessarily task heavy.

CONVENTIONS:

- Javadoc must include Author, version and java version.
- Javadoc all functions and methods.
- Variable declarations are to be grouped with a comment above.
- Comments at the side of variables to describe their use. However this must not conflict with formatting conventions described above.
- Comments must be descriptive without being too large and avoiding 'walls of text'.
- Comments covering few lines (1-2 lines) may use the standard // feature.
- Larger comments which cover more than a few lines must be grouped with the /* */ feature for code clarity and neatness.

VARIABLE NAMING STYLE

Variables must be named in accordance to the standards defined below. Again, the collaborative nature of the project, there must be a standard to variable declaration used by all developers for the ease of altering and reading others' code. Variables must avoid being ambiguous for ease of referencing, passing and casting.

CONVENTIONS:

- Declaring variables 'on the fly' must be avoided, with the exception of for loops, and declared at the top of classes and functions.
- All variables declared must use title case, as seen in the examples below
- Underscores must be avoided in variable declaration.
- Underscores must be used to denote parameter variables.
- The exception to these standards is for loops, which may use single variable names, for example; i, j, k etc.
- Arrays and data structures must be declared as such: arrValues. Followed by a comment which describes what data types are in the array.

- Variables must follow the 3 (Boolean types break this exception) Hungarian notation.

Data Type	Correct Notation	Parameter
Integer	intIntegerVar	_intIntegerVar
Double	dblDoubleVar	_dblDoubleVar
Boolean	boolBooleanVar	_boolBooleanVar
String	strStringVar	_strStringVar
Char	chrCharVar	_chrCharVar
Long	lngLongVar	_lngLongVar
Float	fltFloatVar	_fltFloatVar
Short	shtShortVar	_shtShortVar
Byte	bytByteVar	_bytByteVar
Object	objObjectVar	_objObjectVar

FUNCTION AND METHOD NAMING STYLE

Again, correctly named functions and methods are essential for code clarity and ease of modification for other developers. The names must give a hint as to what the method or function does, this is for ease of calls to the method or function in code.

CONVENTIONS:

- As with variables, names must use title case with the first word in lowercase regardless of the circumstance.
- Underscores must be avoided, unless in the case of parameter variables.

BRACE AND INDENTATION FORMATTING

For code clarity and neatness, all developers must adhere to the following conventions. Code clarity is essential when working on a collaborative project and poor formatting is inexcusable, as all IDE's have customisable formatting options.

INDENTATION CONVENTIONS:

- Declarations within class bodies must be indented.
- Statements within methods/constructors, blocks, switch and case statements must be indented.
- Vertical lines must be adhered to in the case of line breaks and comments occurring after variables.

BRACE CONVENTIONS:

- Braces must follow the declaration on the same line.
- Catch/finally blocks must begin on a new line after the brace.
- Else if blocks must also being on a new line after the brace.

EXAMPLE:

```
/**
  * @author  Jeff Jones
  * @version 1.01
  * @since         1.6
  */
    class MyClass {

        // Variable Declaration
        int intHouseNumber; // Number of the house
        double dblGrams;          // Weight in grams
        /*
        Constructor for MyClass
        I will make
        this comment span
        several lines.
        */
        public MyClass() {
            super.MySuperClass;
        }
        // switch statement for something
        switch (intHouseNumber) {

            case 1:
                return true;
                break;
            case 2:
                return false;
                break;
            default:
        }
        /**
         * @param _intHouseNum passed house number
         */
        public void setHouseNum(int _intHouseNum) {
            intHouseNumber = _intHouseNum;
        }
    }
```

2.2 – 2.3.1.1_02: TESTING STANDARDS

TESTING

The testing process is arguably one of the most important steps of a collaborative project, especially when releasing a product to a client – as they are the ones that will be paying for it. Testing must demonstrate that all the functionality that was specified in the requirements document is adhered to and is functional in a manner the client expects.

It is because of this that a clear and concise level of testing standards must be defined that developers and test engineers will use when testing.

DEVELOPING – WHITE BOX TESTING

During development, developers must unit test all methods as they are written in a separate test case class. All the test cases must pass and be approved of by the project manager before they can be included in the final implementation.

- Tests must have an ID that can be mapped to a requirement and vice versa.
- Tests must be time stamped to display when they were carried out.
- Test cases must be relevant and rigorous to demonstrate absolute functionality in the methods that they are testing.
- Test data must be recorded and documented with the above information.

TEST CASE ENGINEERING

Test case engineers have the task of writing test cases for the code. Test case engineers must have a good idea of how the system works in order for them to write exhaustive tests for the system.

- Test cases must be mapped to the requirement, in order to demonstrate the requirements have been addressed and tested.
- Test cases must be rigorous.
- They must cover boundaries in data types.

TESTERS –BLACK BOX TESTING

Black box testing is an important process, for the black box tests, users who have no knowledge of the system will be given the user manual and asked to use the system in a way which they'd expect the system to work. Once the user get familiar with the system, they may start to attempt to do other things which will test the functionality and rigor of the system which may not be highlighted in the initial white box tests.

- Test data must be recorded and documented with the test.
- Any faults, bugs or errors discovered must be reported and a developer tasked to investigate the problem

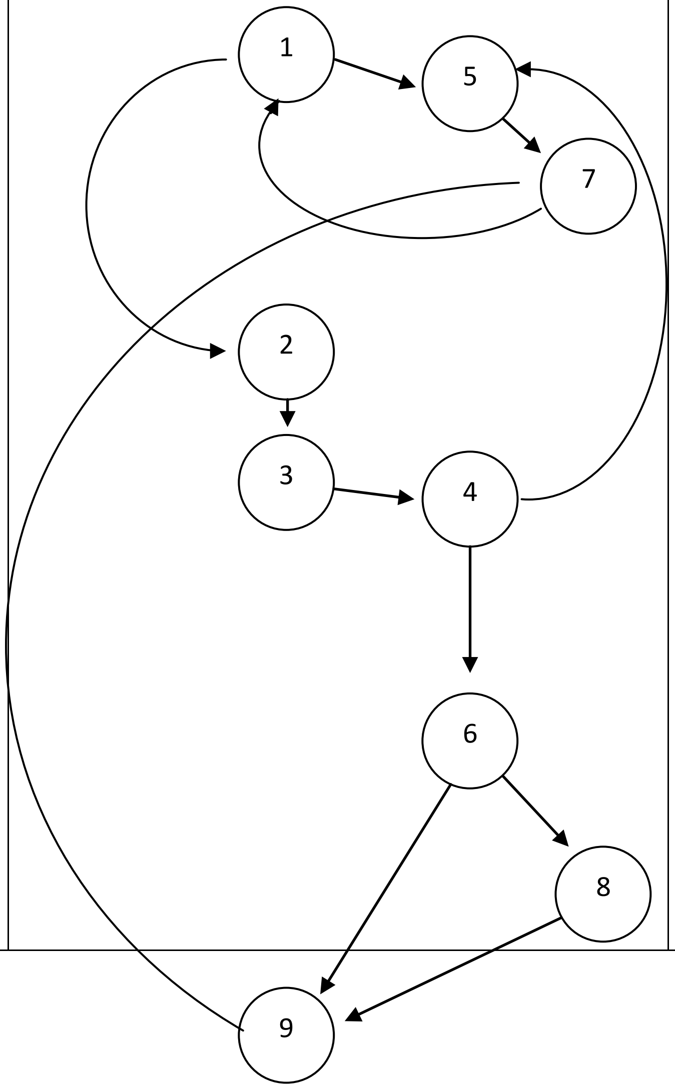
SECTION3 - TESTING

3.1 – 2.3.4_01 TEST CASES

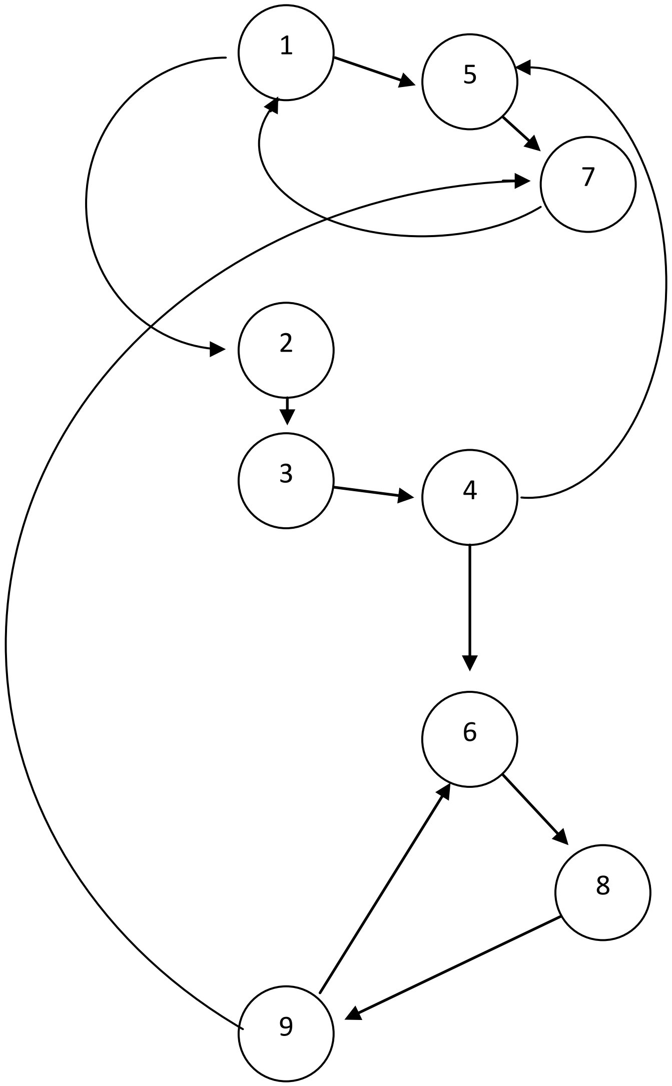
Case	Testing Requirements	Test Case Completed
01.001	To access the system the user must enter their username and password	Yes (30/03/2011)
01.002	To access the admin privileges the user must enter the admin username and password	Yes (30/03/2011)
02.001	Bookings recorded	Yes (30/03/2011)
02.002	Log kept of bookings made by the operator	Yes (30/03/2011)
03.001	Bookings not allowed less than 6 hours before film	No
03.002	A user must be able to select up to 10 seats to be booked at a time	Yes (30/03/2011)
03.003	Peak times are to be set between specified hours	No
04.001	User must be able to cancel film bookings at request of customer	Yes (31/03/2011)
04.002	Details required for cancellation: Customer, film times, payment details	No
05.001	User is able to view the seats currently available for booking in a showing	Yes (30/03/2011)
06.001	User able to see which seats are premium seats, Rows H and I	No
06.002	Price difference in place for booking premium seats	Yes (30/03/2011)
07.001	User able to add: new film, with details: Film name and age rating	Yes (30/03/2011)
07.002	User able to edit film information currently available	Yes (30/03/2011)
08.001	System function, user must enter age of customer booking film, this is checked against age rating of film, if age of customer is equal to or greater than the film rating then booking can be completed.	Yes (30/03/2011)

3.2 – 2.3.4_02: BASIS PATH TESTS

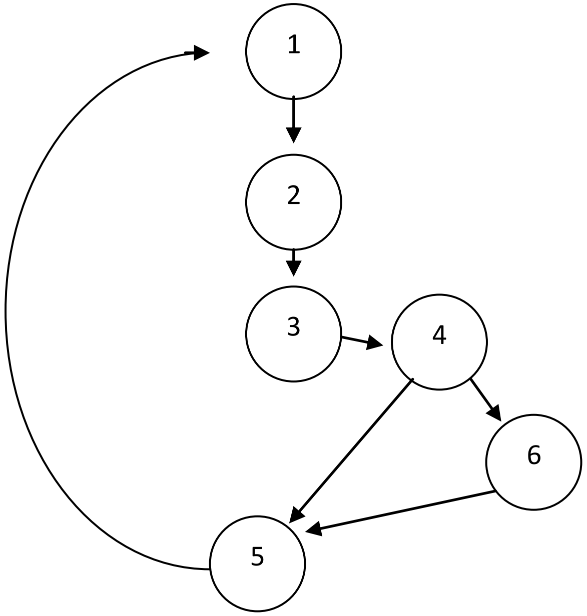
Test Case #	Flow Graph	Code

01.001		if (loginAdminFunction() == 1) {
		//CASE BODY
		}
		else {
		break;
		}
		public static int loginUserFunction() {
		System.out.println("User Name:");
		strInputUserName = kybd.next();
		kybd.reset();
		System.out.println("Password:");
		strInputPassword = kybd.next();
		
		if
		(strInputUserName.equalsIgnoreCase(strUsername)&&strInputPassword.equalsIgnoreCase(strPassword)) {
		System.out.println("Success");
		intAttemptCtr = 0; // Reset the attempt counter
		return 1;
		} else {
		intAttmptCtr++;
		System.out.println("\nUnsuccessful Attempt #" + intAttemptCtr);
		if (intAttemptCtr == 3) {
		System.out.println("3 Consecutive Unsuccessful Attempts.. Exiting Program");
		return escape = 2;
		}
		return 3;
		}
		}

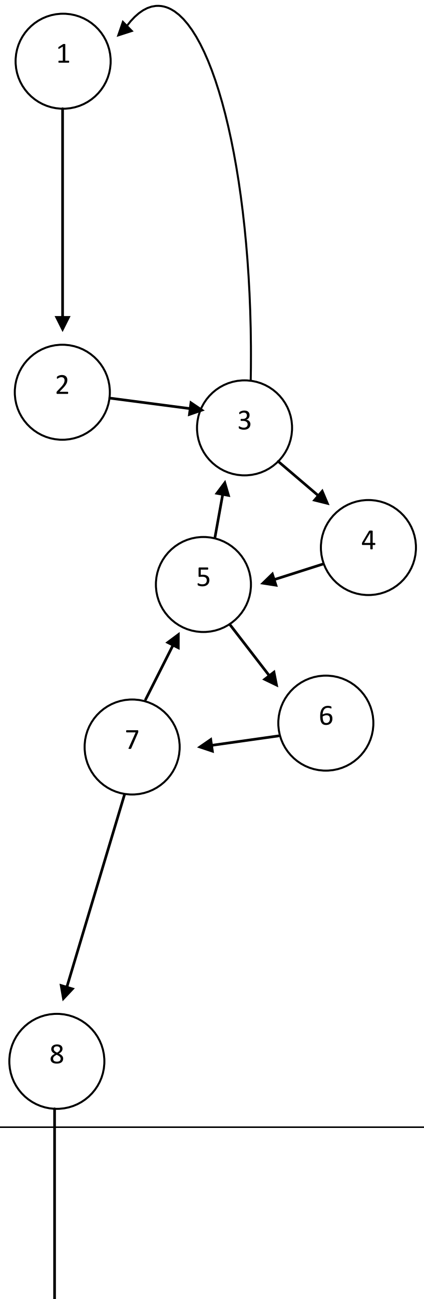
01.002



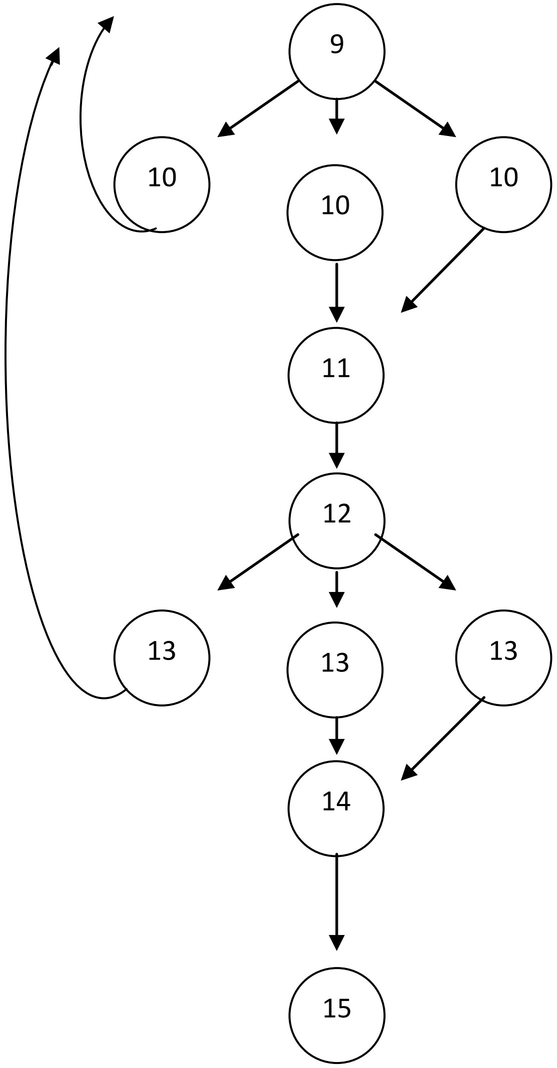
```
if (loginAdminFunction() == 1) {
       //CASE BODY
}
else {
       break;
}

public static int loginAdminFunction() {

       System.out.println("User Name:");
       strInputUserName= kybd.next();
       kybd.reset();
       System.out.println("Password:");
       strInputPassword = kybd.next();

       if (strInputUserName.equalsIgnoreCase(strAdminUsername)
                     && strInputPassword.equalsIgnoreCase(strAdminPassword)){

                     System.out.println("Success");
                     intAttemptCtr = 0; // Reset the attempt counter
                     return 1;

       } else {

              intAttemptCtr++;
              System.out.println("\nUnsuccessful Attempt #" + intAttemptCtr);

       if (intAttemptCtr == 3) {

              System.out.println("3 Consecutive Unsuccessful Attempts.. Exiting
Program");
              return escape = 2;
              }
       }
       return 3;
}
```

08.001		// Call to method
		
		public static boolean advancedBooking() throws Exception {
		
		// Add a customer
		// Make a booking
		// Add a showing to the booking
		
		if (customer.getIntAge() > newBooking.getFilmShowing().getIntRating()) {
		System.out.println("Customer Under Age");
		return false;
		} else {
		
		// Add booking and choose seats
		return true;
		}

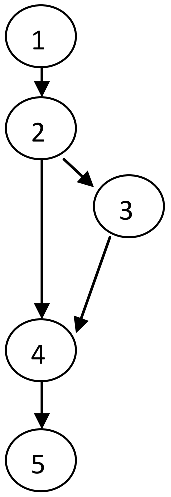
03.002



```
// CREATE CUSTOMER OBJECT

customers.addCustomer(customer);
Booking newBooking = new Booking(customer, stf1);
System.out.println("Available Showings");
showings.showShowings();
System.out.println("ID Of Showing");
kybd.reset();
intCase = kybd.nextInt();
newBooking.setIntShowingID(showings.getByID(intCase));
newBooking.setIntFilmRating(showings.getByRating(intCase));
newBooking.setDblTotalPrice(showings.getByPrice(intCase));

if (customer.getIntAge() < newBooking.getIntFilmRating()) {
        System.out.println("Customer Under Age");
        return false;
} else {
        Seats seatBooking = new Seats(newBooking);
        System.out.println("Number of Seats: ");
        intCase = kybd.nextInt();
        newBooking.setDblTotalPrice(newBooking.getDblTotalPrice() * intCase);

        if (intCase > 10 || intCase <= 0) {
                System.out.println("Exceeds maximum number of seats per booking");
                return false;
        }
        if (intCase <= 0) {
                System.out.println("Number of seats must be greater than 0");
                return false;
        }


        seatBooking.setIntNumSeats(intCase);
```

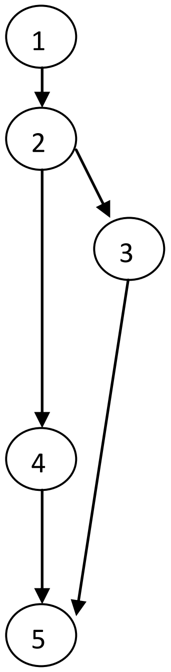


```
        System.out.println("Row: (A - I)");
        strCase = lineReader.readLine();

        switch (strCase.toUpperCase().charAt(0)) {
                case 'A':
                        seatBooking.setObjRow(Row.A);
                        break;
                // ETC
                case 'I':
                        seatBooking.setObjRow(Row.I);
                        newBooking.setDblTotalPrice(newBooking.getDblTotalPrice() *
        1.5);
                        break;
                default:
                        System.out.println("Invalid Seat");
                        return false;
        }
        System.out.println("Column: (0 - 15)");
        intCase = kybd.nextInt();
        switch (intCase) {
                case 0:
                        seatBooking.setObjColumn(Column.J);
                        break;
                // ETC
                default:
                        System.out.println("Invalid Seat");
                        return false;
        }
        bookings.addBooking(newBooking);
        screen1.addSeat(seatBooking);
        System.out.println(newBooking);
        System.out.println(screen1);
        return true;
        }

}
```

<table>
<tr>
<td>


</td>
<td>

```
public static boolean addFilmFunction() {
        Film newFilm = new Film();
        System.out.println("Film Title:");
        try {
                newFilm.setStrFilmName(lineReader.readLine());
        } catch (IOException e) {
                System.out.println("Invalid Entry");
        }
        System.out.println("Film Rating:");
        newFilm.setIntRating(kybd.nextInt());
        films.addFilm(newFilm);
        FilmShowing showing = new FilmShowing();
        showing.addFilm(newFilm);
        return true;
}
```

</td>
</tr>
<tr>
<td>


</td>
<td>

```
public void addSeat(Seats _objSeats) throws Exception {
try {
        for (int i = 0; i < _objSeats.intNumSeats; i++) {
                if
(arrSeats[_objSeats.objRow.getRowNum()][_objSeats.objColumn.getColNum() + i] ==
0) {
arrSeats[_objSeats.objRow.getRowNum()][_objSeats.objColumn.getColNum() + i] =
_objSeats.objBooking.getIntBookingID();
                        bookings.add("Booking ID: "+
_objSeats.objBooking.getIntBookingID() + "\n"+ "Row: " +
_objSeats.objRow.getRowNum() + 1 + "\n"+ "Column: " + _objSeats.objColumn);
} else
        throw new Exception();
}
} catch (Exception e) {
        System.out.println("Seat Already Booked: " + _objSeats.objRow+
_objSeats.objColumn);
}
```

</td>
</tr>
</table>

3.3 – 2.3.4_03: UNIT TESTS

UNIT TEST – SEATS
DATE: 30.03.11 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

1.0	static Seats screen1 = new Seats("Screen 1");
	static Staff stf1 = new Staff("James", "Nightingale", 20, "07123283823", 456, "Manager", "jn1", "lalalala");
	static Film flm1 = new Film("Paul", 15);
	static FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99);
	static Customer cst1 = new Customer("Dave", "Russell", 21, "07533475113");
	static Booking booking = new Booking(cst1, stf1, fs1);
	Seats seats = new Seats(booking);				

1.1	assertNotNull("Booking ID is null", seats.getIntBookingID());	Pass	Pass	N/A	None taken
1.2	assertNotNull("Showing ID is null", seats.getIntShowingID());	Pass	Pass	N/A	None taken
1.3	assertNotNull("Customer ID is null", seats.getIntCustomerID());	Pass	Pass	N/A	None taken
1.4	assertNotNull("Screen is null", screen1.getStrScreen());	Pass	Pass	N/A	None taken

1.5	assertNotNull("Price is null", seats.getDoubleTotalPrice());	Pass	Pass	N/A	None taken
1.6	assertSame("Screens do not match", screen1.getStrScreen(), fs1.getStrScreen());	Pass	Pass	N/A	None taken

1.7	int test;
	assertSame("Return error", seats.setIntNumSeats(2), test = seats.setIntNumSeats(2));
	assertSame("Number of seats do not match", seats.setIntNumSeats(2),seats.getIntNumSeats());
		Pass	Pass	N/A	None taken

1.8	assertNotNull("Number of seats is null", seats.getIntNumSeats());	Pass	Pass	N/A	None taken
1.9	assertTrue("Row is unrecognised", seats.setObjRow(Row.A));	Pass	Pass	N/A	None taken

1.10	seats.setObjRow(Row.A);
	assertNotNull(seats.getObjRow());
		Pass	Pass	N/A	None taken

1.11	assertTrue("Column is unrecognised", seats.setObjColumn(Column.J));	Pass	Pass	N/A	None taken
1.12	seats.setObjColumn(Column.L); assertNotNull("Column has not been set", seats.getObjColumn());	Pass	Pass	N/A	None taken

1.13	try {
	    assertTrue("Adding has failed - Returned false", screen1.addSeat(seats));
	} catch (Exception e) {
	    fail("Adding has failed - Exception caught");
	}
		Pass	Pass	N/A	None taken

1.14	screen1.addSeat(newSeats);
	assertNotNull(screen1.toString());
		Pass	Pass	N/A	None taken

1.15	assertNotNull("Get date/time failed",seats.getCurrentDateTime());				

UNIT TEST – BOOKING

DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

2.0	`Seats screen1 = new Seats("Screen 1");`
	`Booking booking = new Booking("Bookings");`
	`Staff stf1 = new Staff("James", "Nightingale", 20, "07123283823", 456, "Manager", "jn1", "lalalala");`
	`Film flm1 = new Film("Paul", 15);`
	`FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99);`
	`Customer cst1 = new Customer("Dave", "Russell", 21, "07533475113");`
	`Booking staffBooking = new Booking(stf1);`
	`Booking sncBooking = new Booking(cst1, stf1);`
	`Booking fullBooking = new Booking(cst1, stf1, fs1)`
	`Booking testBooking;`				

2.1	`assertNotNull(testBooking = new Booking());`	Pass	Pass	N/A	None taken
2.2	`assertNotNull(staffBooking.toString());`	Pass	Pass	N/A	None taken
2.3	`assertNotNull(testBooking = new Booking(stf1));`	Pass	Pass	N/A	None taken
2.4	`assertNotNull(testBooking = new Booking(cst1,stf1));`	Pass	Pass	N/A	None taken

2.5	assertNotNull(testBooking = new Booking(cst1, stf1, fs1));	Pass	Pass	N/A	None taken

2.6	assertNotNull(staffBooking.getIntBookingID());
	assertNotSame(staffBooking.getIntBookingID(), sncBooking.getIntBookingID());
		Pass	Pass	N/A	None taken

2.7	assertNotNull(fullBooking.getDblTotalPrice());	Pass	Pass	N/A	None taken
2.8	assertNotNull(staffBooking.getIntStaffID());	Pass	Pass	N/A	None taken
2.9	assertNotNull(fullBooking.getIntShowingID());	Pass	Pass	N/A	None taken
2.10	assertNotNull(fullBooking.getIntCustomerID());	Pass	Pass	N/A	None taken
2.11	assertNotNull(fullBooking.getIntFilmRating());	Pass	Pass	N/A	None taken
2.12	assertNotNull(fullBooking.getShowingTime());	Pass	Pass	N/A	None taken

2.13	booking.addBooking(fullBooking);
	assertNotNull(booking);
		Pass	Pass	N/A	None taken

2.14	assertNotNull(fullBooking.generateBookingID());	Pass	Pass	N/A	None taken
2.15	assertNotNull(fullBooking.getDateTime());	Pass	Pass	N/A	None taken
2.16	assertNotNull(fullBooking.toString());	Pass	Pass	N/A	None taken

2.17	booking.addBooking(fullBooking);
	assertNotNull(booking);
	booking.showBookings();
	booking.cancelBooking(57);
	booking.showBookings();
		Pass	Pass but unexpected entry being ‘made’	Unsure	Tasked developer to investigate problem

UNIT TEST – FILMSHOWING
DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

3.0	Seats screen1 = new Seats("Screen 1");
	FilmShowing showings = new FilmShowing("Showings");
	Film flm1 = new Film("Paul", 15);
	FilmShowing testFS;
	FilmShowing fs1 = new FilmShowing(flm1, "27/03/2011", screen1, 6.99);				

3.1	showings.addFilmShowing(fs1);
	System.out.println(fs1);
		Pass	Pass	N/A	None taken

3.2	assertNotNull(testFS = new FilmShowing(flm1));	Pass	Pass	N/A	None taken
3.3	assertNotNull(testFS = new FilmShowing("Showings"));	Pass	Pass	N/A	None taken
3.4	assertNotNull(testFS = new FilmShowing(flm1, "21/03/2009", screen1, 4.99));	Pass	Pass	N/A	None taken

3.5	showings.addFilmShowing(fs1);
	showings.showShowings();
		Pass	Pass	N/A	None taken

3.6	assertNotNull(fs1.getIntShowingID());	Pass	Pass	N/A	None taken
3.7	fs1.setStrDate("21/03/2009");	Pass	Fail	java.lang.IllegalArgumentException: Cannot format given Object as a Date	Tasked developer to fix method.
3.8	assertNotNull(fs1.getStrDate());	Pass	Pass	N/A	None taken
3.9	assertNotNull(fs1.getStrScreen());	Pass	Pass	N/A	None taken

3.10	fs1.setStrScreen("Screen 2");
	System.out.println(fs1.toString());
	assertNotNull(fs1.getStrScreen());
		Pass	Pass	N/A	None taken

3.11	assertNotNull(fs1.getDblPrice());	Pass	Pass	N/A	None taken

3.12	fs1.setIntPrice(7.99);
	System.out.println(fs1);
		Pass	Pass	N/A	None taken

3.13	showings.addFilmShowing(fs1);	Pass	Pass	N/A	None taken
3.14	showings.showShowings();	Pass	Pass	N/A	None taken

3.15	showings.addFilmShowing(fs1);
	assertNotNull(showings.getByID(2));
	System.out.println(showings.getByID(2));
		Pass	Pass	N/A	None taken

3.16	`showings.addFilmShowing(fs1);`
	`assertNotNull(showings.getByRating(2));`
	`System.out.println(showings.getByRating(2));`
		Pass	Pass	N/A	None taken

3.17	`showings.addFilmShowing(fs1);`
	`assertNotNull(showings.getByPrice(2));`
	`System.out.println(showings.getByPrice(2));`
		Pass	Pass	N/A	None taken

3.18	`testFS = new FilmShowing();`
	`assertNotNull(testFS.generateFSID());`
		Pass	Pass	N/A	None taken

3.19	`assertNotNull(fs1.getDateTime());`	Pass	Pass	N/A	None taken

UNIT TEST – CUSTOMER

DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

4.0	Customer customers = new Customer("Customers");
	Customer testArray;
	Person testCust;
	Customer cust1 = new Customer("Dave", "Russell", 21, "01232 123123");
	Customer cust2 = new Customer();				

4.1	assertNotNull(testCust = new Customer());	Pass	Pass	N/A	None taken
4.2	assertNotNull(testCust = new Customer("James", "Nightingale", 20, "01232 123123"));	Pass	Pass	N/A	None taken
4.3	assertNotNull(testArray = new Customer("Customers"));	Pass	Pass	N/A	None taken
4.4	assertNotNull(cust1.getCustomerID());	Pass	Pass	N/A	None taken
4.5	customers.addCustomer(cust1);	Pass	Pass	N/A	None taken
4.6	assertNotNull(cust2.generateCustomerID());	Pass	Pass	N/A	None taken
4.7	System.out.println(cust1);	Pass	Pass	N/A	None taken

UNIT TEST – STAFF

DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

5.0	`Staff staff = new Staff("Staff");`
	`Staff testStaff;`
	`Staff stf1 = new Staff("Amanda", "Patterson", 21, "01010 929292","Head Honcho", "ap1", "ap1");`				

5.1	`assertNotNull(stf1.getStrFirstName());`	Pass	Pass	N/A	None taken

5.2	`stf1.setStrFirstName("Manda");`
	`assertNotNull(stf1.getStrFirstName());`
		Pass	Pass	N/A	None taken

5.3	`assertNotNull(stf1.getStrLastName());`	Pass	Pass	N/A	None taken

5.4	`stf1.setStrLastName("patterson");`
	`assertNotNull(stf1.getStrLastName());`
		Pass	Pass	N/A	None taken

5.5	`assertNotNull(stf1.getIntAge());`	Pass	Pass	N/A	None taken

5.6	`stf1.setIntAge(22);`
	`assertNotNull(stf1.getIntAge());`
		Pass	Pass	N/A	None taken

5.7	assertNotNull(stf1.getStrTelephone());	Pass	Pass	N/A	None taken

5.8	stf1.setStrTelephone("01010 124123");
	assertNotNull(stf1.getStrUsername());
		Pass	Pass	N/A	None taken

5.9	testStaff = new Staff();
	assertNotNull(testStaff);
		Pass	Pass	N/A	None taken

5.10	testStaff = new Staff("Staff");
	assertNotNull(testStaff);
		Pass	Pass	N/A	None taken

5.11	testStaff = new Staff("Scott", "Dennison", 20, "01010 231231", "Crew Member", "sd1", "sd1");	Pass	Pass	N/A	None taken
5.12	staff.addStaff(stf1);	Pass	Pass	N/A	None taken
5.13	staff.viewStaff();	Pass	Pass	N/A	None taken
5.14	assertNotNull(stf1.getStaffID());	Pass	Pass	N/A	None taken
5.15	assertNotNull(stf1.generateStaffID());	Pass	Pass	N/A	None taken
5.16	assertNotNull(stf1.getStrRole());	Pass	Pass	N/A	None taken

5.17	stf1.setStrRole("THE BOSS");
	assertNotNull(stf1.getStrRole());
	System.out.println(stf1.getStrRole());
		Pass	Pass	N/A	None taken

5.18	assertNotNull(stf1.getStrUsername());	Pass	Pass	N/A	None taken

5.19	stf1.setStrUsername("ap_m1");
	assertNotNull(stf1.getStrUsername());
	System.out.println(stf1.getStrUsername());
		Pass	Pass	N/A	None taken

5.20	assertNotNul(stf1.getStrPassword());	Pass	Pass	N/A	None taken

5.21	stf1.setStrPassword("password");
	assertNotNull(stf1.getStrPassword());
	System.out.println(stf1.getStrPassword());
		Pass	Pass	N/A	None taken

5.22	System.out.println(stf1);	Pass	Pass	N/A	None taken

UNIT TEST – FILM

DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

6.0	`Film films = new Film("Films");`
	`Film testFilm;`
	`Film newFilm2 = new Film("Harry Potter and the Mysterious Case of Missing Work", 15);`				

6.1	`assertNotNull(testFilm = new Film("Films"));`	Pass	Pass	N/A	None taken
6.2	`assertNotNull(testFilm = new Film());`	Pass	Pass	N/A	None taken
6.3	`assertNotNull(testFilm = new Film("The Exorcist", 18));`	Pass	Pass	N/A	None taken

6.4	`films.addFilm(newFilm2);`
	`assertNotNull(films);`
		Pass	Pass	N/A	None taken

6.5	`assertNotNull(newFilm2.getIntFilmID());`	Pass	Pass	N/A	None taken

6.6	`newFilm2.setStrFilmName("The Exorcist 2");`
	`assertNotNull(newFilm2.getStrFilmName());`
	`System.out.println(newFilm2.getStrFilmName());`
		Pass	Pass	N/A	None taken

6.7	`assertNotNull(newFilm2.getStrFilmName());`	Pass	Pass	N/A	None taken

6.8	newFilm2.setIntRating(15);
	assertNotNull(newFilm2.getIntRating());
	System.out.println(newFilm2.getIntRating());
		Pass	Pass	N/A	None taken

6.9	assertNotNull(newFilm2.getIntRating());	Pass	Pass	N/A	None taken
6.10	System.out.println(newFilm2);	Pass	Pass	N/A	None taken
6.11	assertNotNull(newFilm2.generateFilmID());	Pass	Pass	N/A	None taken

UNIT TEST – MAIN

DATE: 30.03.2011 – JUNIT – WHITEBOX

ID	Test Data	Expected	Actual	Cause	Action Taken

7.0	`1 – admin, admin`
	`2 – y, 1.8`
	`3 – user, user`
	`4 – Harry Potter, 15`
	`5 – Dave, Russell, 21, 01231 123132, 2`
	`6 - 2`				

7.1	`assertNotNull(main_package.Main.loginAdminFunction());`	Pass	Pass	N/A	None taken
7.2	`assertNotNull(main_package.Main.premiumSeatPrice());`	Pass	Pass	N/A	None taken
7.3	`assertNotNull(main_package.Main.loginUserFunction());`	Pass	Pass	N/A	None taken
7.4	`assertNotNull(main_package.Main.addFilmFunction());`	Pass	Pass	N/A	None taken

7.5	`try {`
	`      assertTrue(main_package.Main.advancedBooking());`
	`} catch (Exception e) {`
	`      fail("Will fail regardless. Has no showings to reference");`
	`}`
		Fail	Fail	N/A	None taken

7.6	try {
	    assertTrue(main_package.Main.makeBooking());
	} catch (Exception e) {
	    fail("Will fail regardless. Has no showings to reference");
	}
		Fail	Fail	N/A	Test method separately

3.4 – 2.3.4_04: BLACK BOX TESTING

ID	Date	Description	Test Data	Expected	Actual	Cause	Action Taken

8.1	25.03.11	User enters an option not supported at menu	`***   Cinema Login System Ver. 1.6     ***`
			
			`1:    Admin`
			`2:    User`
			`s`
			`Invalid Entry`
				Invalid Entry	Invalid Entry	Print statement displaying "Invalid Entry"	None taken

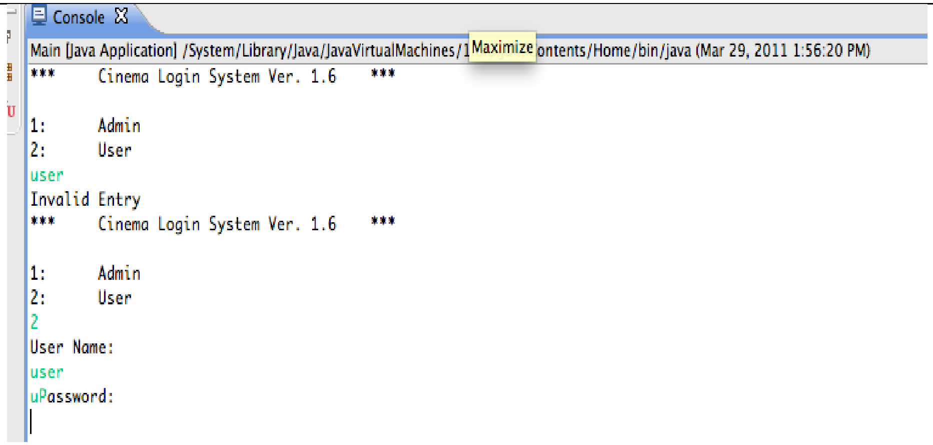
8.2	25.03.11	User enters number outside the range	`***   Cinema Login System Ver. 1.6     ***`
			
			`1:    Admin`
			`2:    User`
			`12`
			`Unrecognised Command: 12`
				"Unrecognised command"	"Unrecognised Command"	Print statement displaying "Unrecognised Command"	None taken

8.3	30.03.11	User enters string into an integer value	`Film Title:`
			`winnie the pooh`
			`Film Rating:`
			`u`
			`Invalid Entry`
				As "U" is a rating of film expected it to be valid input	"invalid entry"	Value needs to be integer in order to do age comparison with rating of film	Code altered so that if string input no comparison is needed

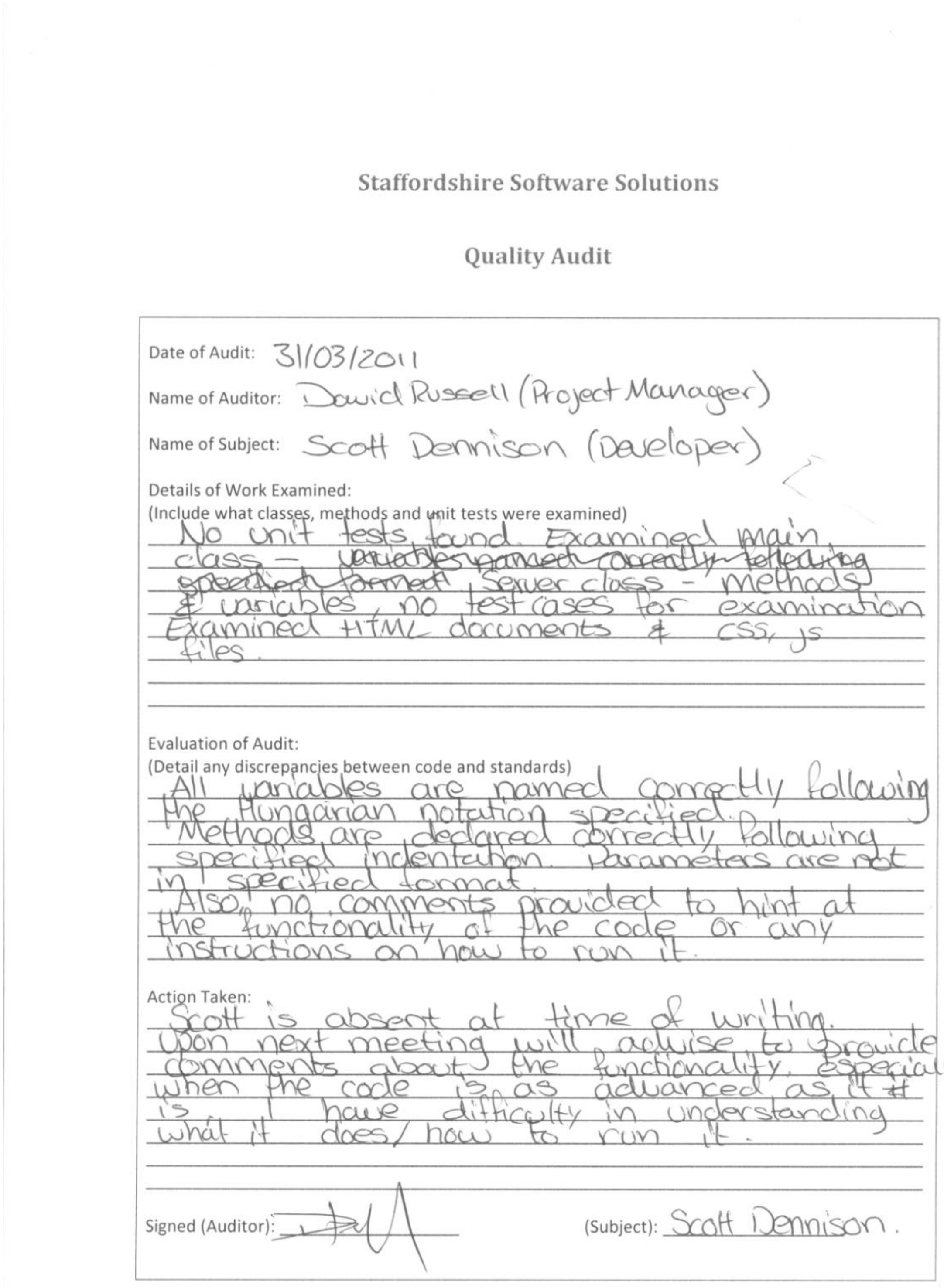
8.4	27.03.11	User enters password incorrectly	`1:    Admin`
			`2:    User`
			`1`
			`User Name:`
				Error message displayed and program	Error message displayed and program	Print statement displaying	None taken

		3 times	`user`
			`Password:`
			`;;k`
			
			`Unsuccessful Attempt #3`
			`3 Consecutive Unsuccessful Attempts.. Exiting Program`
				closes	closes	error message and showing the program is going to close	

8.5	28.03.11	User enters an option not available at user main menu	`***   Cinema System Ver. 1.6      ***`
			
			`1:   Make Booking`
			`2:   Display Showings`
			`0:   Logoff`
			
			`_`
			`java.util.InputMismatchException`
			`      at java.util.Scanner.throwFor(Scanner.java:840)`
			`      at java.util.Scanner.next(Scanner.java:1461)`
			`      at java.util.Scanner.nextInt(Scanner.java:2091)`
			`      at java.util.Scanner.nextInt(Scanner.java:2050)`
			`      at main_package.Main.main(Main.java:142)`
			`4`
				Should have displayed “invalid input”	Printed the description of error	Wrong output in catch exception	Code changed to a print statement displaying “invalid input”

8.6	29.03.11	User entered “/” before entering password causing issues for input and program	
			
			Main [Java Application] /System/Library/Java/JavaVirtualMachines/1Maximize ontents/Home/bin/java (Mar 29, 2011 1:56:20 PM)
			`***      Cinema Login System Ver. 1.6    ***`
			
			`1:       Admin`
			`2:       User`
			`user`
			`Invalid Entry`
			`***      Cinema Login System Ver. 1.6    ***`
			
			`1:       Admin`
			`2:       User`
			`2`
			`User Name:`
			`user`
			`uPassword:`
				Password input to be allowed	Program froze not allowing any user input and causing user to close program	Unknown cause	None taken as error did not occur a second time

8.7	29.03.11	String entered instead of integer when inputting customer age	Customer Age a Invalid Entry *** Cinema System Ver. 1.6 *** 1: Make Booking 2: Display Showings 0: Logoff _ java.util.InputMismatchException at java.util.Scanner.throwFor(Scanner.java:840) at java.util.Scanner.next(Scanner.java:1461) at java.util.Scanner.nextInt(Scanner.java:2091) at java.util.Scanner.nextInt(Scanner.java:2050) at main_package.Main.advancedBooking(Main.java:285) at main_package.Main.main(Main.java:153) 1	Should have displayed invalid input and allowed user to enter another input	Program printed the description of the error and didn't allow the user to put in another input	Wrong output in catch exception	Code changed to print out statement displaying "invalid input"



Staffordshire Software Solutions

Quality Audit

Date of Audit: 31/03/2011

Name of Auditor: Dawid Russell (Project Manager)

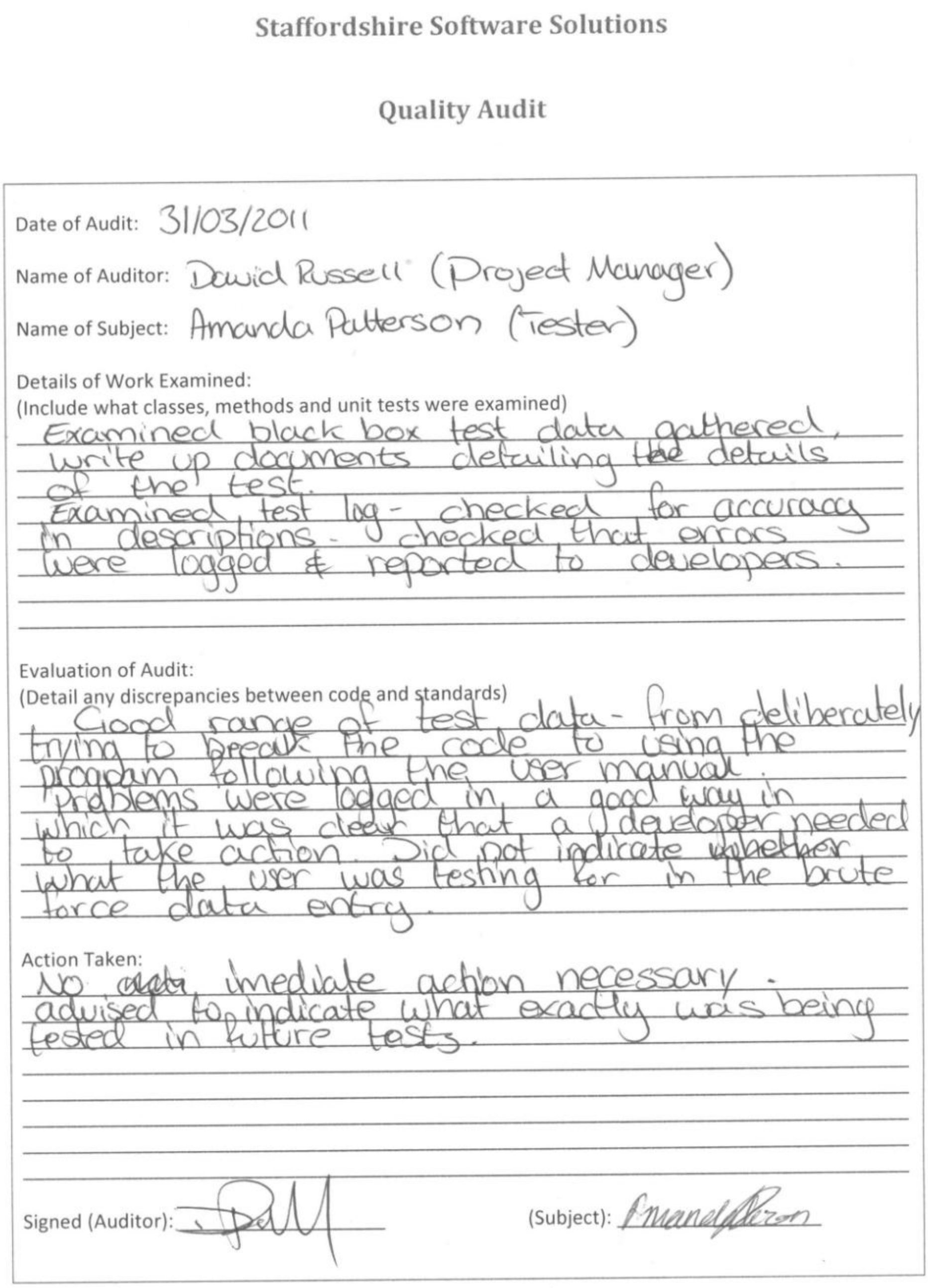
Name of Subject: Scott Dennison (Developer)

Details of Work Examined:
(Include what classes, methods and unit tests were examined)
No unit tests found. Examined main class – ~~variables named correctly following specified format~~. Server class – methods & variables, no test cases for examination. Examined HTML documents & CSS, js files.

Evaluation of Audit:
(Detail any discrepancies between code and standards)
All variables are named correctly following the Hungarian notation specified.
Methods are declared correctly following specified indentation. Parameters are not in specified format.
Also, no comments provided to hint at the functionality of the code or any instructions on how to run it.

Action Taken:
Scott is absent at time of writing. Upon next meeting will advise to provide comments about the functionality, especially when the code is as advanced as it is, I have difficulty in understanding what it does/ how to run it.

Signed (Auditor): ______ (Subject): Scott Dennison.



Staffordshire Software Solutions

Quality Audit

Date of Audit: 31/03/2011

Name of Auditor: David Russell (Project Manager)

Name of Subject: Amanda Patterson (Tester)

Details of Work Examined:
(Include what classes, methods and unit tests were examined)

Examined black box test data gathered, write up documents detailing the details of the test.
Examined test log - checked for accuracy in descriptions - checked that errors were logged & reported to developers.

Evaluation of Audit:
(Detail any discrepancies between code and standards)

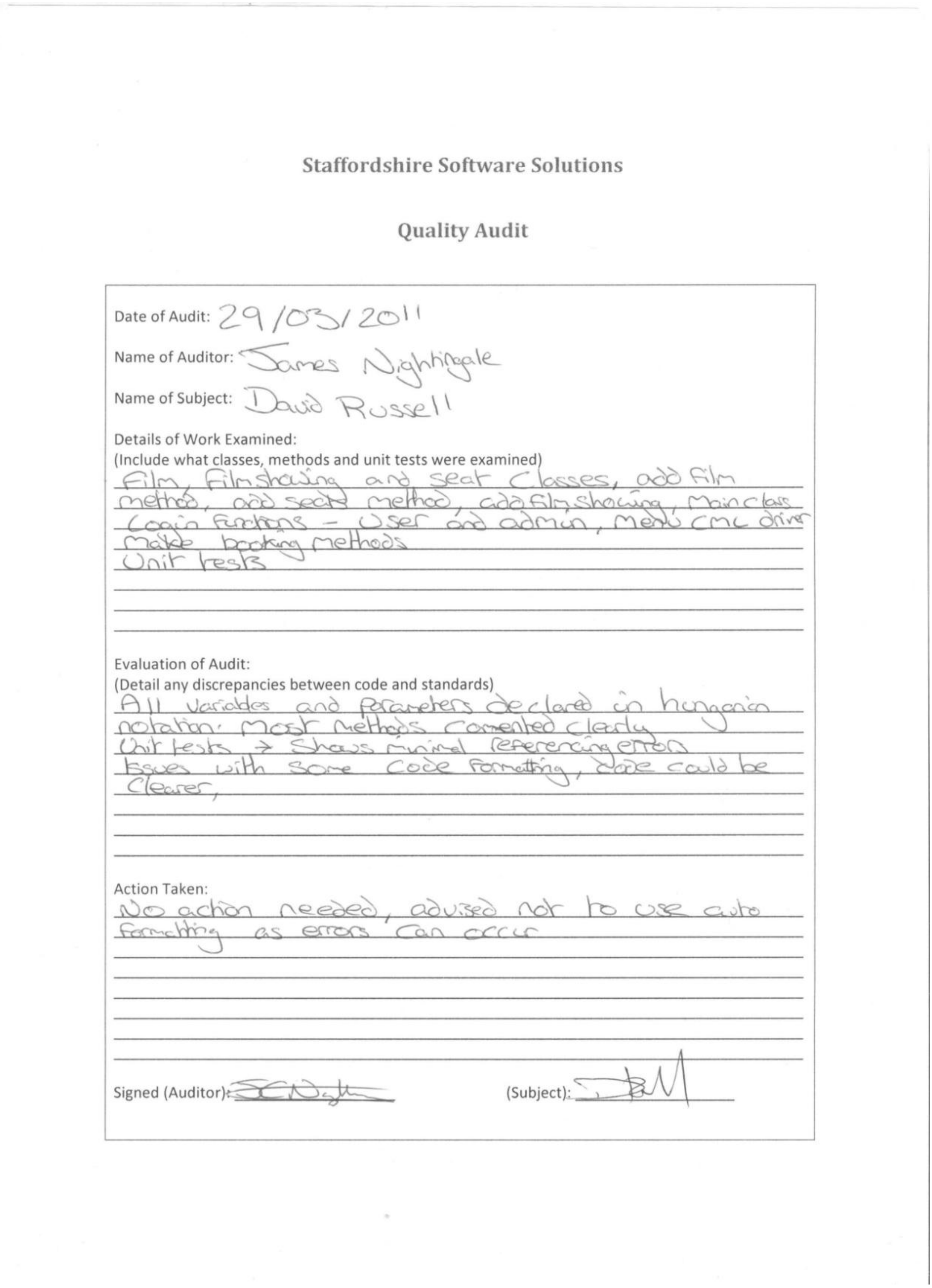
Good range of test data - from deliberately trying to break the code to using the program following the user manual.
Problems were logged in a good way in which it was clear that a developer needed to take action. Did not indicate whether what the user was testing for in the brute force data entry.

Action Taken:

No ~~data~~ imediate action necessary. advised to indicate what exactly was being tested in future tests.

Signed (Auditor):

(Subject):



Staffordshire Software Solutions

Quality Audit

Date of Audit: 29/03/2011

Name of Auditor: James Nightingale

Name of Subject: David Russell

Details of Work Examined:
(Include what classes, methods and unit tests were examined)

Film, Filmshowing and seat Classes, add film method, add seats method, add filmshowing, Main class Login functions - User and admin, Menu CMC driver Make booking methods
Unit tests

Evaluation of Audit:
(Detail any discrepancies between code and standards)

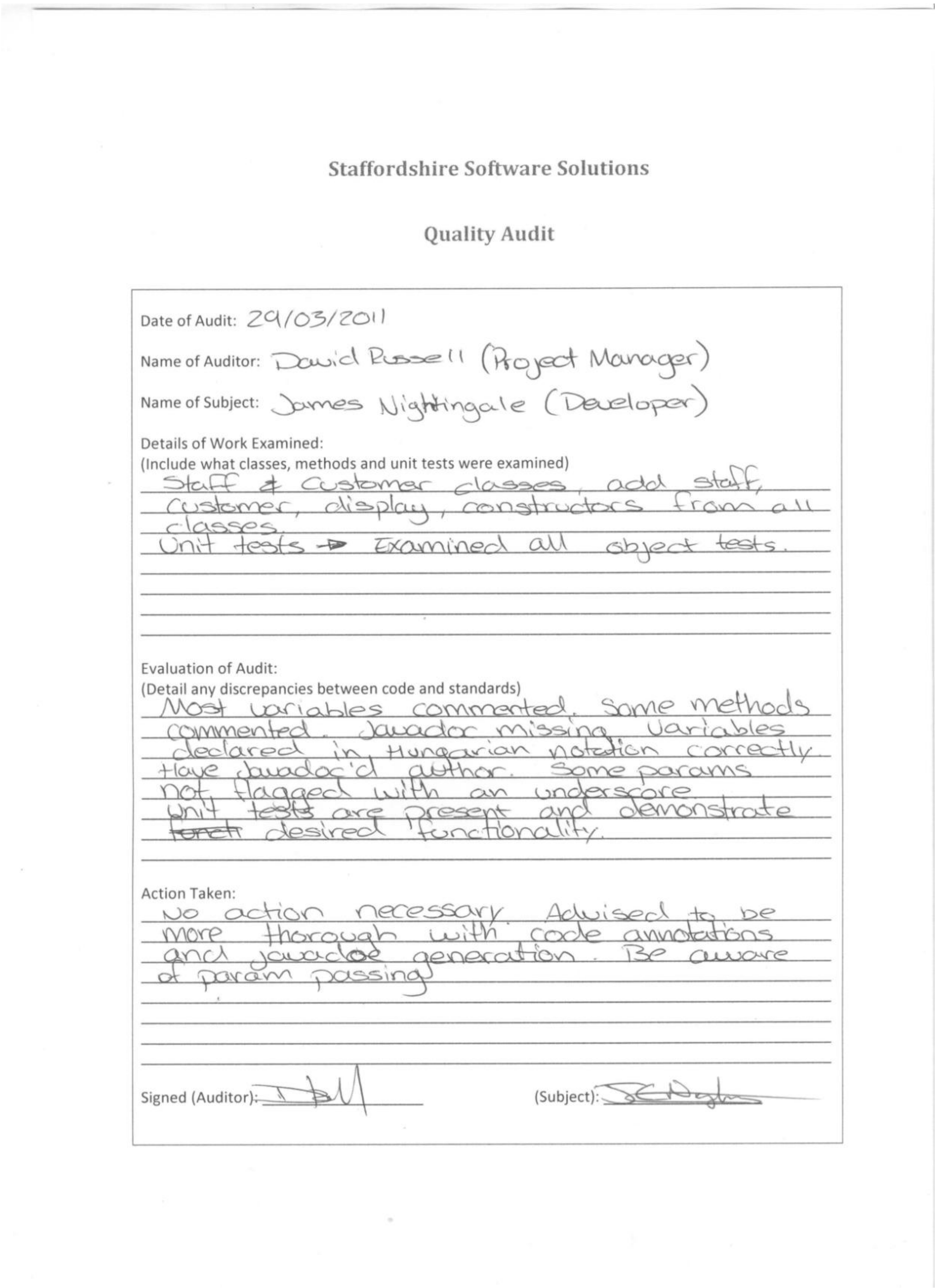
All variables and parameters declared in hungarian notation. Most methods comented clearly
Unit tests → Shows minimal referencing errors
Issues with some code formatting, code could be clearer,

Action Taken:

No action needed, advised not to use auto formatting as errors can occur

Signed (Auditor):

(Subject):



Staffordshire Software Solutions

Quality Audit

Date of Audit: 29/03/2011

Name of Auditor: David Russell (Project Manager)

Name of Subject: James Nightingale (Developer)

Details of Work Examined:
(Include what classes, methods and unit tests were examined)
Staff & Customer classes, add staff, customer, display, constructors from all classes.
Unit tests → Examined all object tests.

Evaluation of Audit:
(Detail any discrepancies between code and standards)
Most variables commented. Some methods commented. Javadoc missing. Variables declared in Hungarian notation correctly. Have Javadoc'd author. Some params not flagged with an underscore.
Unit tests are present and demonstrate ~~foreth~~ desired functionality.

Action Taken:
No action necessary. Advised to be more thorough with code annotations and javadoc generation. Be aware of param passing

Signed (Auditor):

(Subject):

SECTION 4 – REQUIREMENTS TRACING

4.1 – 2.3.2_01: REQUIREMENTS TRACING

Req ID	Requirement	Description	Method which satisfies requirement	Class	Location	Method Description	Test	Version Added
3.1.1	Log operator onto the system	All operators must login to the system with a username and password to access the system	public static boolean loginAdmin() and public static boolean loginUser()	Main	Line 262–283 & 325 - 346	Authentication is done programatically using static strings, and equalsignorecase method of the string class. An unsecure method, however it fits the requirements	7.1 & 7.3	1.3
3.1.2	Record the operator's bookings	Record a log of the bookings by each operator.	public Booking(Customer, Staff, FilmShowing)& public void addBooking(Booking)	Booking	Line 71-79 & 134 - 136	Booking constructor includes the Staff object who made the booking, which contains all their details. The booking information is stored in a collection and can be extracted.	2.3 & 2.8	1.3

3.1.3	Book seats up to 6 hours before the film	Book up to 10 seats in one booking (1 row) Peak Times are fixed on Saturdays and Sundays between 12-4 and 6-11	public void addSeats(Seats)	Seats	154 - 176	Method checks if the current date/time is not equal to the date/time of the showing and proceeds. Method then enters a for loop which specifies how many times to loop based on the number of seats the customer wishes to book. The method checks that the specified seat is empty, if not it throws and error that is caught and an error specifying that the desired seat is already booked. The method then populates the position and loops until the loop is satisfied.	1.13	1.1
	Cancel Bookings	Customer's bookings can be cancelled via request of the operator, they require details of customer and film times and payment details for a refund	public boolean cancelBooking(int)	Booking	Line 172-184	Using an enhanced for loop, the method runs through the booking collection and if the passed booking id is found, the booking is removed	2.17	1.7

3.1.5	View which seats are available	Operators can view which seats are available. It is possible for customers to request which seats are available for booking.	public String toString() and public void listByBooking()	Seats& Booking	Line 251-274 (Seats) & 210 – 216 (Booking)	toString() method enters a loop which prints out the lseats row by row, an empty seat - denoted by [ ] is printed if the seat is empty. And an [X] is printed if the seat is taken. The listByBooking() method loops through the collection of bookings with an enhanced for loop and prints all the bookings from all screens by booking ID and seat reference	2.14	1.1
3.1.6	Different prices for premium seats	Rows H and I are premium seats	public static boolean makeBooking() and makeAdvancedBooking()	Main	Line 453 – 459 & 594 - 599	In the case statement, if the selected rows are H or I, the price of the film is multiplied by the premium percentage. If not then the price remains the same	N/A	1.6

3.1.7	Able to set prices of premium seats as a percentage increase of the costs for standard seating		public static boolean premiumSeatPrice()	Main	Line 291 – 315	The user is shown the current premium and prompted whether or not they want to adjust it. If the user wishes to adjust it the user is prompted to enter a new percentage and that is assigned to the variable dblPremium, which is then used in the calculations in the booking methods.	7.2	1.6.5
3.1.8	Store film information	File name and age rating for each film as text information	public Film(), Film(String, int) and public void addFilm(Film)	Film	Line 24 - 33	Constructor public Film() constructs a Film object with the attributes specified: Film title and rating. Method addFilm adds the film object to a collection.	6.3, 6.4, 6.6, 6.10	1.2
3.1.9	Check customer is correct age for viewing the film	System function, enter date of birth of customer when booking film, check against age rating of film and if age of customer is equal to or greater than the age rating of film then booking can be completed	public static boolean advancedBooking() throws Exception	Main	Line 243 - 285	When an advanced booking is made, the operator must enter the customers' details into the system. The booking is then made using the customers details. If the rating of the showing is greater than the customers'	7,5, 7.6	1.5

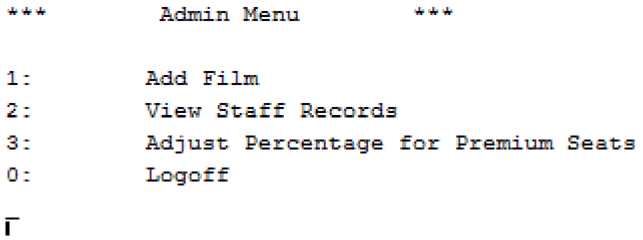
						age, the booking is rejected.		
3.2.1	Personal information not to be kept for more than one year on database	No credit card information is to be kept of customer	Not Implemented	N/A	N/A	Unsure of how to implement this. Possibly timestamping each customer when added for advanced booking and comparing the dates with that timestamp.	N/A	N/A
3.5.1	Software colour scheme to be red, white and green		Not Implemented	N/A	N/A	At time of writing, implementation is command line – possible future implementation will be with standard java colours.	N/A	N/A
3.5.2	Minimise number of steps to complete booking to keep it as simple as possible		Number of steps to make a booking: 12	Main	Line 380 - 665	Bookings are made through a series of menus as of current version. In total there are 12 steps in making a booking. This includes input of the users details. Future versions will streamline this process.	N/A	1.5

SECTION 5 - USER MANUAL

ADMINISTRATOR USER GUIDE

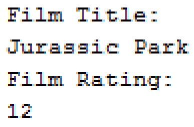
Default admin username: admin
Default admin password: admin

- To log in as admin, enter 1 from main menu, this will take you to username and password input screens.



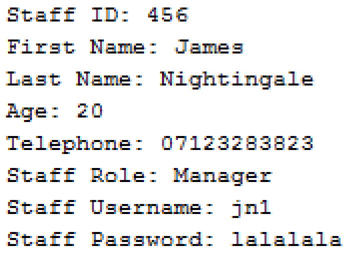
```
***         Admin Menu          ***

1:          Add Film
2:          View Staff Records
3:          Adjust Percentage for Premium Seats
0:          Logoff
```

- To add a film select option one to bring you to the next menu, this will ask you to enter a film name and age rating.



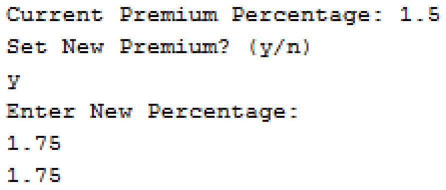
```
Film Title:
Jurassic Park
Film Rating:
12
```

- To look at staff records enter option 2 at the admin menu and will display the details of staff.



```
Staff ID: 456
First Name: James
Last Name: Nightingale
Age: 20
Telephone: 07123283823
Staff Role: Manager
Staff Username: jn1
Staff Password: lalalala
```

- As the admin it is also possible to change the percentage of the price increase from standard seats to premium seats. It will display the current percentage and then ask if you would like to change it, if you select yes it will ask you for the new percentage.

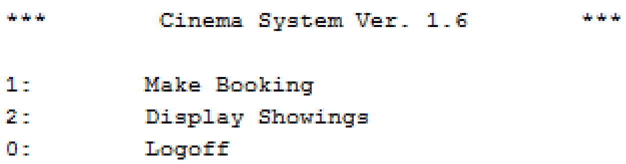


```
Current Premium Percentage: 1.5
Set New Premium? (y/n)
y
Enter New Percentage:
1.75
1.75
```

- To log out of the admin account press 0 at the main menu.

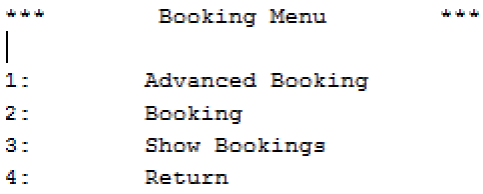
Default user username: user
Default user password: user

- To log in as user, enter 2 from main menu, this will take you to username and password input screens.



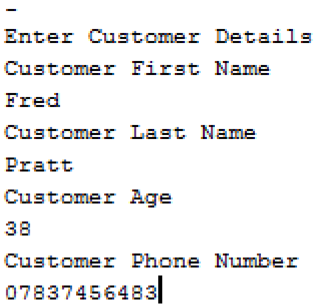
```
***         Cinema System Ver. 1.6         ***

1:         Make Booking
2:         Display Showings
0:         Logoff
```

- To make a new booking select option 1 to go to the booking menu.



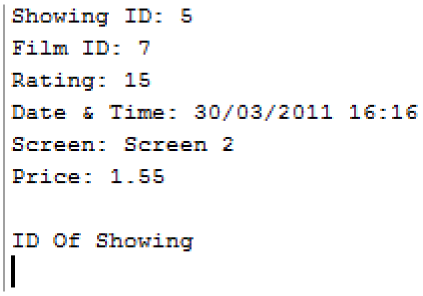
```
***         Booking Menu         ***

1:         Advanced Booking
2:         Booking
3:         Show Bookings
4:         Return
```

- To make an advanced booking select option 1, this will take you to another screen which will require you to input your name, age and telephone number.



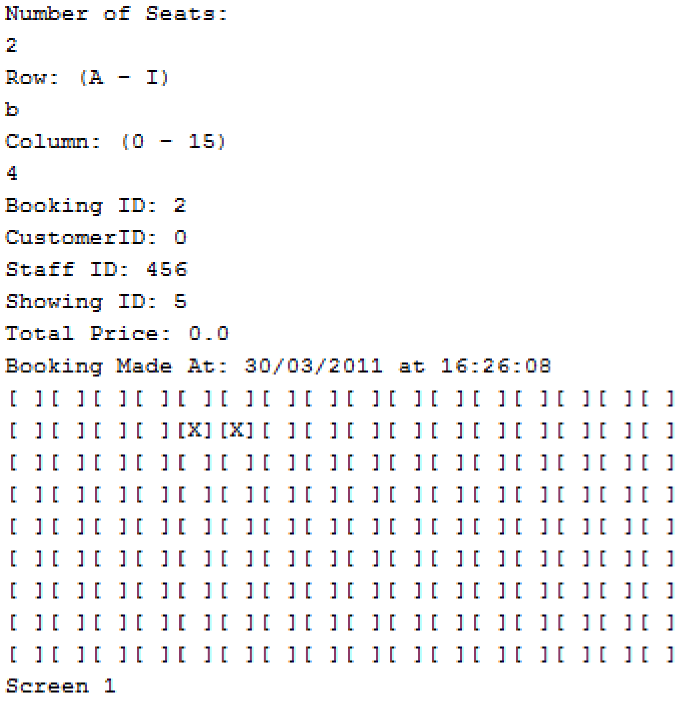
```
-
Enter Customer Details
Customer First Name
Fred
Customer Last Name
Pratt
Customer Age
38
Customer Phone Number
07837456483
```

- After you have entered your details a list of film showings will appear and you will be asked to enter the ID of the showing you wish to book.



```
Showing ID: 5
Film ID: 7
Rating: 15
Date & Time: 30/03/2011 16:16
Screen: Screen 2
Price: 1.55

ID Of Showing
```

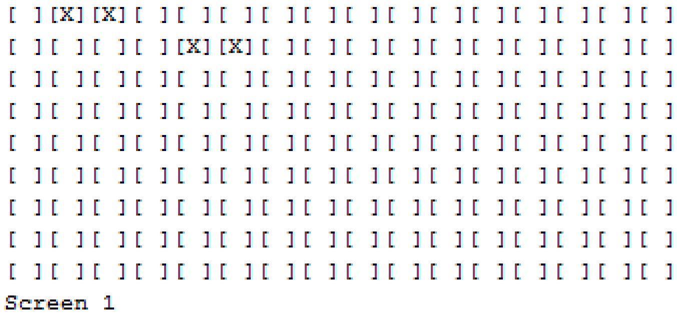
- You will then be asked to enter the number of seats you wish to book and select which row and column you wish to have the seats in. It will then display the total price of the booking and then a visual display of the seats being booked with date and time of when the booking was made with the screen number.



```
Number of Seats:
2
Row: (A - I)
b
Column: (0 - 15)
4
Booking ID: 2
CustomerID: 0
Staff ID: 456
Showing ID: 5
Total Price: 0.0
Booking Made At: 30/03/2011 at 16:26:08
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][X][X][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
Screen 1
```

- To complete a normal booking you must select booking from the main menu by entering 2, the process is the same as advanced booking however you will not be required to enter the customer details.

- In order to see the bookings that have already been made it is possible to select the show bookings option from the main menu, this will display the seats that have been booked for each screen of the cinema.

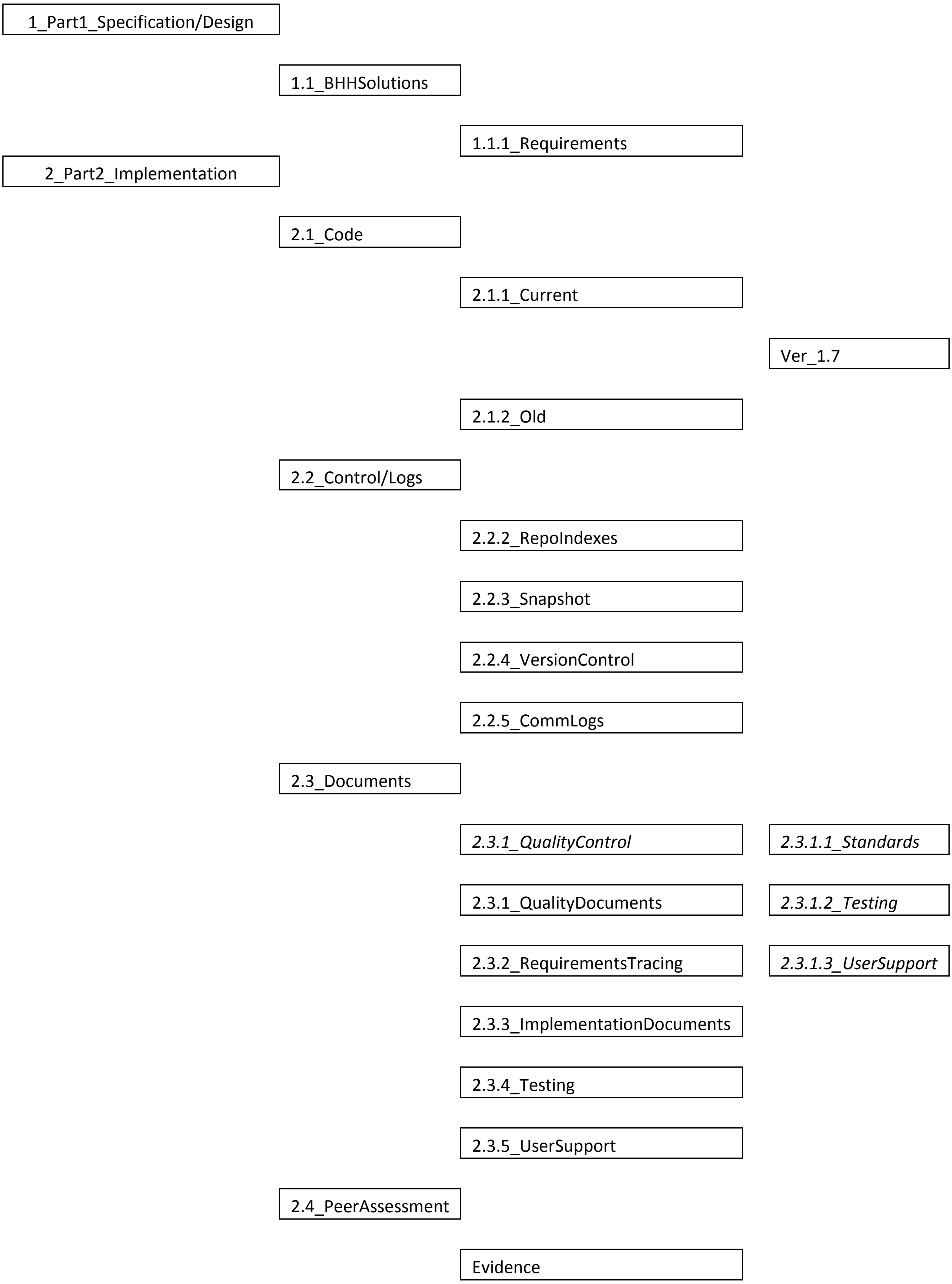


```
[ ][X][X][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][X][X][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
[ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ][ ]
Screen 1
```

- It is then possible to logoff the system by entering 0 at the main menu; from here you can select to log back in as admin or user.

SECTION 6 - CONFIGURATION MANAGEMENT

https://github.com/staffs-ppsp/staffs-ppsp-repo



- 1_Part1_Specification/Design
  - 1.1_BHHSolutions
    - 1.1.1_Requirements
- 2_Part2_Implementation
  - 2.1_Code
    - 2.1.1_Current
      - Ver_1.7
    - 2.1.2_Old
  - 2.2_Control/Logs
    - 2.2.2_RepoIndexes
    - 2.2.3_Snapshot
    - 2.2.4_VersionControl
    - 2.2.5_CommLogs
  - 2.3_Documents
    - *2.3.1_QualityControl*
      - *2.3.1.1_Standards*
      - *2.3.1.2_Testing*
      - *2.3.1.3_UserSupport*
    - 2.3.1_QualityDocuments
    - 2.3.2_RequirementsTracing
    - 2.3.3_ImplementationDocuments
    - 2.3.4_Testing
    - 2.3.5_UserSupport
  - 2.4_PeerAssessment
    - Evidence

Index ID	Name of Document	Author(s)	Date of Creation	Location	Description	Current Version	Version ID	SWVersion ID
I1	Index.xlsx	David Russell	18/03/2011	2.2.2_RepoIndexes	Index log of all directories and content for the project	2.5	V1	VC04
I2	*SWVersionControl.xlsx*	*David Russell*	*18/03/2011*	*2.2.4_VersionControl*	*Index log of all software/operating system versions used for the project*	*1.1*	*V2*	VC04
I3	DocVersionControl.xlsx	David Russell	18/03/2011	2.2.1_VersionControl	Index log of all changes to all directories and content for the project	1.1	V3	VC04
I4	SoftwareRequirements.docx	Partner Group	18/03/2011	1.1.1_Requirements	Partner group requirements specification	1.0	V4	VC04
I5	SoftwareRequirementsRevised.jpeg	Partner Group	18/03/2011	1.1.1_Requirements	Partner groups revised requirements specification	1.0	V5	N/A
I6	01_Plan.docx	David Russell	18/03/2011	2.3.3_ImplementationDocuments	Document outlining the plan of the implementation phase	1.0	V6	VC04
I7	*QualityAssurance.docx*	*Group*	*18/03/2011*	*2.3.3_ImplementationDocuments*	*Quality assurance document*	*1.0*	*V7*	VC04
I8	04_Requirements.docx	James Nightingale	18/03/2011	2.3.3_ImplementationDocuments	Clear and concise requirement specification	1.0	V8	VC04
I9	05_ResourceAllocation.docx	Group	18/03/2011	2.3.3_ImplementationDocuments	Document describing the break down of tasks and their allocated resources	1.1	V9	VC04
I10	02_Strategy.docx	Group	18/03/2011	2.3.3_ImplementationDocuments	Document outlining the strategies for the implementation phase	1.0	V10	VC04
I11	01_CodingStandards.docx	Scott Dennison, David Russell	19/03/2011	2.3.1_QualityDocuments	Document establish coding standards for all developers	1.1	V11	VC04
I12	CommLog01.docx	N/A	20/03/2011	2.2.5_CommLogs	Document discussing amendments to CodingStandards.docx	1.1	N/A	VC04, VC10
I13	*ppsp-davidrussell (project folder - various date/time stamps)*	*David Russell*	*21/03/2011*	*2.1.2_Old*	*Project folder for code contribution to project*	*1.1*	*V12*	VC06, VC07, VC08
I14	*ppsp-jamesnightingale (project folder - various date/time stamps)*	*James Nightingale*	*21/03/2011*	*2.1.2_Old*	*Project folder for code contribution to project*	*1.2*	*V13*	VC06, VC07, VC08
I7	08_QualityAssurance.docx	Amanda Patterson	22/03/2011	2.3.3_ImplementationDocuments	Quality assurance document	1.1	V7	VC04

I15	*ppsp-project (project folder)*	*James Nightingale, David Russell*	*23/03/2011*	*2.1.2_Old*	*Project folder for code contribution to project*	*1.2*	*V14*	VC06, VC07, VC08
I15	*Ver_1.6*	*James Nightingale, David Russell*	*24/03/2011*	*2.1.1_Current*	*Current Version Project folder*	*1.6*	*V14*	VC06, VC07, VC08
I16	SWVersionControl_NEW.xlsx	Scott Dennison, David Russell	23/03/2011	2.2.4_VersionControl	Index log of all software/operating system versions used for the project	1.1	V15	VC04, VC10
I17	01_RequirementsTracing.docx	David Russell	24/03/2011	2.3.2_RequirementsTracing	Document to map the implemented code to the specified requirements	1.2	V16	VC04
I18	03_DesignReview.docx	David Russell	28/03/2011	2.3.3_ImplementationDocuments	Digital write up copy of the minutes for the design review meeting with BHH	1.0	N/A	VC04
I19	02_BasisPathTesting.docx	David Russell, James Nightingale	27/03/2011	2.3.4_Testing	Document containing the basis path testing flow diagrams	1.3	V17	VC04
I20	01_TestCases.docx	James Nightingale	24/03/2011	2.3.4_Testing	Document containing the test cases	1.2	V18	VC04
I21	07_NewDesigns.vsd	James Nightingale	24/03/2011	2.3.3_ImplementationDocuments	Visio drawing containing some simple screen designs which were not included in the original spec	1.0	V19	VC04
I22	06_AnnotatedRequirements.docx	Scott Dennison	24/03/2011	2.3.3_ImplementationDocuments	Requirements with some annotations which we must design for	1.0	V20	VC04
I23	07_NewDesigns.png	James Nightingale	28/03/2011	2.3.3_ImplementationDocuments	PNG format of the simple new designs	1.0	V21	N/A
I24	QualityAuditTemplate.docx	David Russell	28/03/2011	2.3.1_QualityDocuments	Template for the quality audit doucment	1.1	V22	VC04
I25	FinalDocument.docx	Group	27/03/2011	2.3_Documents	Final Document	1.5	V23	VC04
I26	basis testing functions.docx	James Nightingale	29/03/2011	2.3.4_Testing	James's basis path test flow graphs	1.1	N/A	VC04
I27	TestTemplate.docx	David Russell	29/03/2011	2.3.4_Testing	Template for tests to be recorded on	1.0	N/A	VC04
I28	01_UserManual.docx	James Nightingale, Amanda Patterson	30/03/2011	2.3.5_UserSupport	The user manual for the system	1.0	V24	VC04
I29	Ver_1.7	David Russell, James Nightingale	30/03/2011	2.1.1_Current	New version of the implementation	1.8	V25	VC06, VC07, VC08

I30	02_TestingStandards.docx	David Russell	30/03/2011	2.3.1_QualityDocuments	Testing standards. These set a base line for testing standards for the project	1.0	V26	VC04
I31	03_UnitTests.docx	David Russell	30/03/2011	2.3.4_Testing	Recorded unit tests of all classes in the implementation	1.0	V27	VC04
I32	04_BlackBoxTesting	Amanda Patterson, James Nightingale	30/03/2011	2.3.4_Testing	Recorded black box test results	1.1	V28	VC04
I33	black box u.docx	Amanda Patterson, James Nightingale	30/03/2011	2.3.4_Testing	Document storing some black box test results, to be added to the final document	N/A	N/A	VC04
I34	blackboxtestdata.docx	Amanda Patterson, James Nightingale	30/03/2011	2.3.4_Testing	Document storing some black box test results, to be added to the final document	N/A	N/A	VC04
i35	Audit0,1,2,3.jpg	Group	30/03/2011	2.3.1_QualityDocuments	Scanned audits of the group	N/A	N/A	N/A
I36	02_TestingStandards.docx	David Russell	31/03/2011	2.3.1_QualityDocuments	Testing standards. These set a base line for testing standards for the project	1.0	V29	VC04
I37	Student Peer Assessment.docx	Amanda Patterson	31/03/2011	2.4_PeerAssessment	Peer assessment form	N/A	N/A	VC04

6.2 - 2.2.4: VERSION CONTROL

Category	Version Control ID	Software/OS Title	Description
Operating Sytems			
	VCOS01	Microsoft Windows 7	Microsoft operating system.
	VCOS02	Mac OSX	Apple Mac operating system.
	VCOS03	Ubuntu Linux	Linux operating system.

VCID	(OS (Formula))	User	Version
VCOS01	Microsoft Windows 7	David Russell	6.1 Build 7600
		Amanda Patterson	6.1 Build 7600
		James Nightingale	6.1 Build 7600
VCOS02	Mac OSX	David Russell	10.6.6
VCOS03	Ubuntu Linux	David Russell	10.10
		James Nightingale	10.10
		Scott Dennison	10.04.2 LTS

Category	Version Control ID	Software/OS Title	Description
Document Editors			
	VC04	Microsoft Office EE	Package of Office products by Microsoft. Includes: Excel & Word.
	VC05	OpenOffice.org	Free Office products based upon the Microsoft product
Code Editors			
	VC06	Netbeans	Netbeans Java Development Enviroment
	VC07	Eclipse	Eclipse java development environment
Code Compilers			
	VC08	Java	SDK to develop and deploy java applications
Internet Browsers			
	VC09	Mozilla Firefox	Internet browser by Mozilla
	VC10	Google Chrome	Internet browser by Google

Software VCID	(Software (Formula))	User	OS VCID	(OS (Formula))	Version
VC04	Microsoft Office EE	Amanda Paterson	VCOS01	Microsoft Windows 7	12.0.6425.1000
		James Nightingale			
		David Russell	VCOS02	Mac OSX	12.2.8
VC05	OpenOffice.org		VCOS03	Ubuntu Linux	3.2.1 Build 9505
		Scott Dennison			
VC06	Netbeans		VCOS01	Microsoft Windows 7	6.9.1
		James Nightingale	VCOS03	Ubuntu Linux	6.8 Build 200912041610
		Scott Dennison			
VC07	Eclipse		VCOS01	Microsoft Windows 7	1.3.1
		David Russell	VCOS02	Mac OSX	1.3.1
VC08	Java		VCOS01	Microsoft Windows 7	1.6
		David Russell	VCOS02	Mac OSX	1.6
			VCOS03	Ubuntu Linux	1.6
			VCOS01	Microsoft Windows 7	1.6
		James Nightingale	VCOS03	Ubuntu Linux	1.6
			VCOS03	Ubuntu Linux	1.6.0_20
		Scott Dennison			
VC09	Mozilla Firefox		VCOS01	Microsoft Windows 7	3.6.15
		David Russell	VCOS02	Mac OSX	3.6.15
			VCOS03	Ubuntu Linux	3.6.15
			VCOS01	Microsoft Windows 7	3.6.15
		James Nightingale	VCOS03	Ubuntu Linux	3.6.15
			VCOS03	Ubuntu Linux	3.6.15
		Scott Dennison			
VC10	Google Chrome		VCOS01	Microsoft Windows 7	10.0.648.151
		David Russell	VCOS02	Mac OSX	10.0.648.151
			VCOS03	Ubuntu Linux	10.0.648.151
			VCOS01	Microsoft Windows 7	10.0.648.151
		James Nightingale	VCOS03	Ubuntu Linux	10.0.648.151
		Scott Dennison	VCOS03	Ubuntu Linux	10.0.648.151

6.3 – 2.2.4: CHANGE LOG

Version ID	Index ID	Location	Date of Change	User to make changes	Description of Changes	Old Version	Current Version
V0	N/A	staffs_ppsp_repo	18/03/2011	David Russell	Added directory 2.3.3_ImplementationDocuments	N/A	1.0
			20/03/2011	David Russell	Added directory 2.2.5_CommLogs	1.1	1.2
			20/03/2011	David Russell	Removed Directory 2.2.1_ChangeControl, Moved and renamed document ChangeControl.xlsx -> DocVersionControl.xlsx to 2.2.4_Version Control. Renamed document VersionControl.xlsx -> SWVersionControl.xls	1.2	1.3
			23/03/2011	David Russell, James Nightingale	Added directory ppsp-project to 2.1.1_Current, moved redundant code to 2.1.2_Old	1.3	1.4
			29/03/2011	David Russell	Renamed 2.3.1_QualityControl to QualityDocuments. Removed directory 2.3.1.1_Standards. Moved 2.3.1.2_Testing to 2.3_Documents, saved as 2.3.4_Testing. Moved 2.3.1.2_UserSupport to 2.3_Documents, saved as 2.3.5_UserSupport	1.4	1.5
			31/03/2011	David Russell	Added new directory 2.4_PeerAssessment	1.5	1.6

V1	I1	2.2.2_RepoIndexes	18/03/2011	David Russell	Updated file: changed formatting and content	N/A	1.1
			18/03/2011	David Russell	Added Dir: 1.1.1_Requirements. Added file: SoftwareRequirements and SoftwareRequirementsRevised. Removed: 1.1_SSS	1.1	1.2
			18/03/2011	David Russell	Added files to 2.3.3_ImplementationDocuments	1.2	1.3
			19/03/2011	David Russell	Updated index for daily change	1.3	1.4
			20/03/2011	David Russell	Updated index for daily change	1.4	1.5
			21/03/2011	David Russell	Updated index for daily change	1.5	1.6
			22/03/2011	David Russell	Updated index for daily change	1.6	1.7
			23/03/2011	David Russell	Updated index for daily change	1.7	1.8
			24/03/2011	David Russell	Updated index for daily change	1.8	1.9
			25/03/2011	David Russell	Updated index for daily change	1.9	2.0

			26/03/2011	David Russell	Updated index for daily change	2.0	2.1
			27/03/2011	David Russell	Updated index for daily change	2.1	2.2
			28/03/2011	David Russell	Updated index for daily change	2.2	2.3
			29/03/2011	David Russell	Updated index for daily change	2.3	2.4
			30/03/2011	David Russell	Updated index for daily change	2.4	2.5
			31/03/2011	David Russell	Updated index for daily change	2.5	2.6

V2	I2	2.2.4_VersionControl	20/03/2011	David Russell	Rename: VersionControl.xlsx -> SWVersionControl.xlsx	1.0	1.1
V15	I16		23/03/2011	Scott Dennison	New document for software version logs	N/A	1.0

V3	I3	2.2.1_ChangeControl(Removed)	20/03/2011	David Russell	Removed Directory, moved file to 2.2.4_VersionControl, renamed file to DocVersionControl	1.0	1.1
		2.2.4_VersionControl	20/03/2011	David Russell			

V4	I4	1.1.1_Requirements	18/03/2011	David Russell	Added document to repository	N/A	1.0

V5	I5	1.1.1_Requirements	18/03/2011	David Russell	Added document to repository	N/A	1.0

V6	I6	2.3.3_ImplementationDocuments	18/03/2011	David Russell	Added document to repository	N/A	1.0

V7	I7	2.3.3_ImplementationDocuments	18/03/2011	Amanda Patterson	Added document to repository	N/A	1.0

V8	I8	2.3.3_ImplementationDocuments	18/03/2011	David Russell	Added document to repository	N/A	1.0

V9	I9	2.3.3_ImplementationDocuments	18/03/2011	David Russell	Added document to repository	N/A	1.0
			28/03/2011	David Russell	Updated document - added new content	1.0	1.1

V10	I10	2.3.3_ImplementationDocuments	18/03/2011	David Russell	Added document to repository	N/A	1.0

V11	I11	2.3.1.1_Standards	20/03/2011	David Russell	Amended changes discussed in email correspondence (Index I12)	1.0	1.1
		2.3.1_QualityDocuments	29/03/2011	David Russell	Moved location.	1.1	1.2

V12	I13	2.1.1_Current	21/03/2011	David Russell	Added project folder to 2.1.1_Current	N/A	1.0
		2.1.2_Old	23/03/2011	David Russell	Moved folder to 2.1.1_Old	1.0	1.1

V13	I14	2.1.1_Current	22/03/2011	James Nightingale	Added project folder to 2.1.1_Current	N/A	1.0
		2.1.1_Current	23/03/2011	James Nightingale	Ammended code to meet quality specifications	1.0	1.1
		2.1.2_Old	23/03/2011	David Russell	Moved folder to 2.1.2_Old	1.1	1.2

V14	I15	2.1.1_Current	23/03/2011	David Russell, James Nightingale	Created project folder for all code	N/A	1.0
			23/03/2011	James Nightingale	Added Film class	1.0	1.1
			23/03/2011	David Russell, James Nightingale	Created main function with CLI menu driver	1.1	1.2
			23/03/2011	David Russell, James Nightingale	Added further functionality	1.2	1.3
			24/03/2011	David Russell	Updated main functions and menu driver	1.3	1.4
			24/03/2011	David Russell	Updated main functions and menu driver	1.4	1.5
			27/03/2011	David Russell	Updated main program - see commit details	1.5	1.6

V16	I17	2.3.2_RequirementsTracing	24/03/2011	David Russell	Created document	N/A	1.0
			28/03/2011	David Russell	Updated document: revised location of code as code has changed significantly. More methods have been implemented and these have been traced to the requirement.	1.0	1.1
			31/03/2011	David Russell	Updated document with final revision	1.1	1.2

V17	I19	2.3.3_ImplementationDocuments	24/03/2011	David Russell	Created document	N/A	1.0
			25/03/2011	David Russell	Updated document - added a new test	1.0	1.1
			27/03/2011	David Russell	Updated document - added a new test	1.1	1.2
		2.3.4_Testing	29/03/2011	David Russell	Moved document	1.2	1.3

V18	I20	2.3.3_ImplementationDocuments	24/03/2011	James Nightingale	Created document	N/A	1.0
			28/03/2011	David Russell	Renamed doucment, added title	1.0	1.1
		2.3.4_Testing	29/03/2011	David Russell	Moved document	1.1	1.2

V19	I21	2.3.3_ImplementationDocuments	24/03/2011	James Nightingale	Created document	N/A	1.0

V20	I22	2.3.3_ImplementationDocuments	24/03/2011	Scott Dennison	Created document, copy of requirements with some annotations which we need to consider	N/A	1.0

V21	I23	2.3.3_ImplementationDocuments	28/03/2011	James Nightingale	Created document	N/A	1.0

V22	I24	2.3.1.1_Standards	28/03/2011	David Russell	Created document	N/A	1.0
		2.3.1_QualityDocuments	29/03/2011	David Russell	Moved Location	1.0	1.1

V23	I25	2.3_Documents	27/03/2011	David Russell	Created document, added work so far	N/A	1.0
			28/03/2011	David Russell	Updated document	1.0	1.1
			29/03/2011	James Nightingale	Added basis path tests to document	1.1	1.2
			30/03/2011	James Nightingale, David Russell, Amanda Patterson	Added user manual, quality assurance, testing documents	1.2	1.3
			30/03/2011	David Russell, James Nightingale, Amanda Patterson	Updated document	1.3	1.4
			31/03/2011	David Russell, James Nightingale, Amanda Patterson	Added final testing to document, testing standards and audits	1.5	1.5

V24	I26		30/03/2011	James Nightingale, Amanda Patterson	Created the document	N/A	1.0

V25	I28		30/03/2011	David Russell, James Nightingale	Created the project - significant changes since last update	1.6	1.7

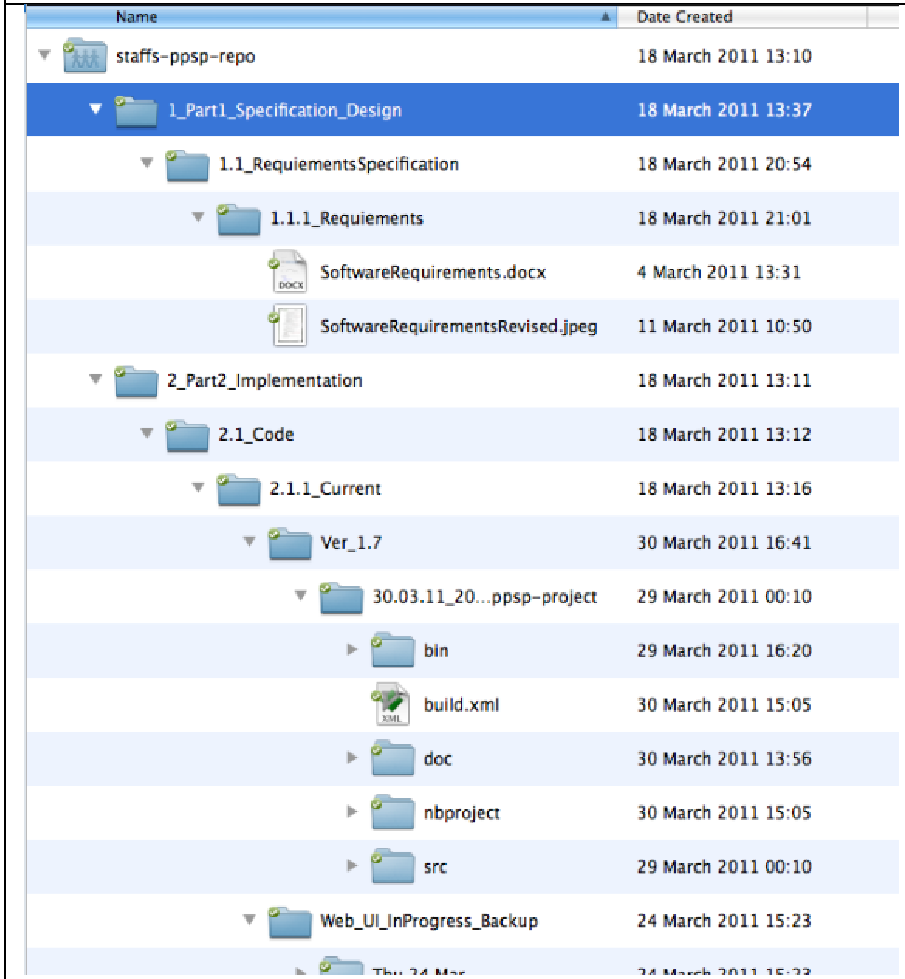
V26	I29		30/03/2011	David Russell	Created the document	N/A	1.0

V27	I30		30/03/2011	David Russell	Created the document	N/A	1.0

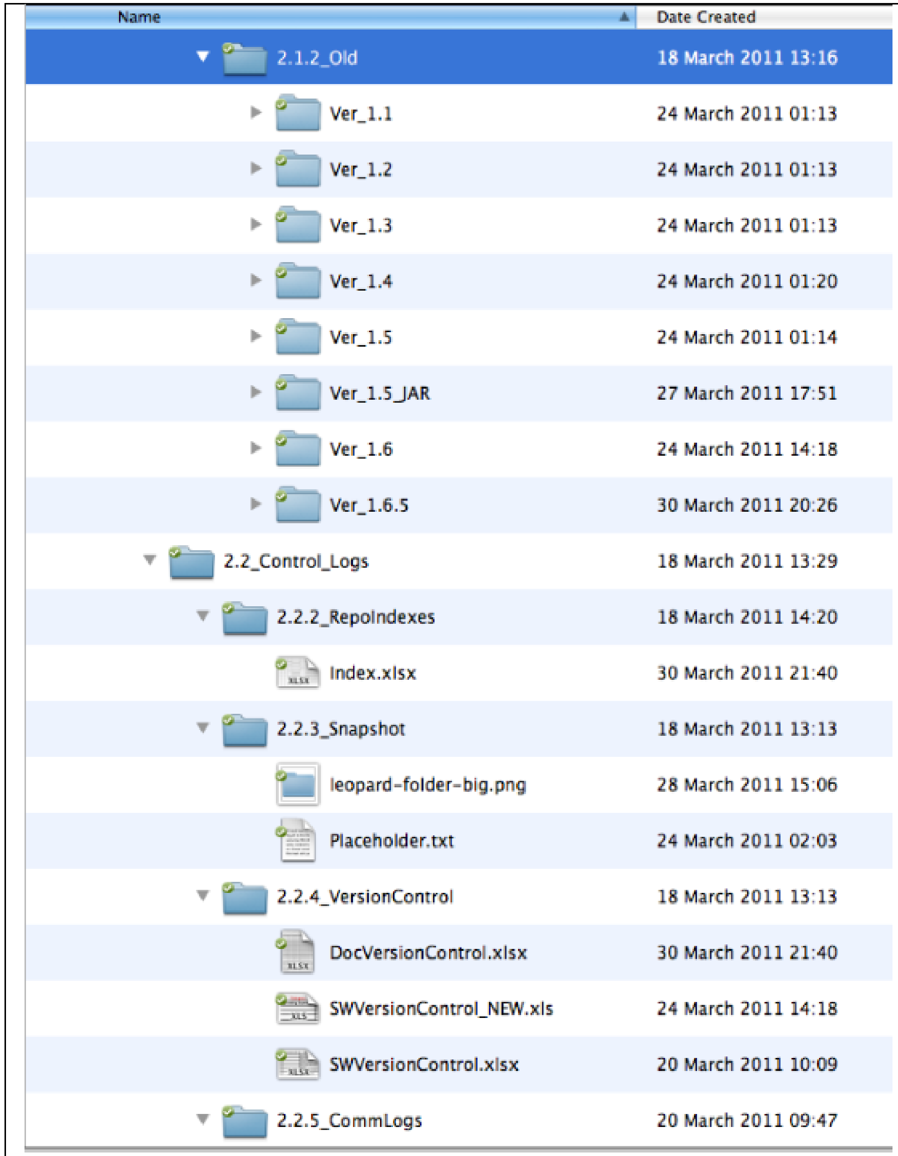
V28	I31		30/03/2011	Amanda Patterson, James Nightingale	Created the document	N/A	1.0
			31/03/2011	Amanda Patterson, James Nightingale	Updated the document	1.0	1.1

V29	I32		30/03/2011	Amanda Patterson, James Nightingale	Created the document	N/A	1.0

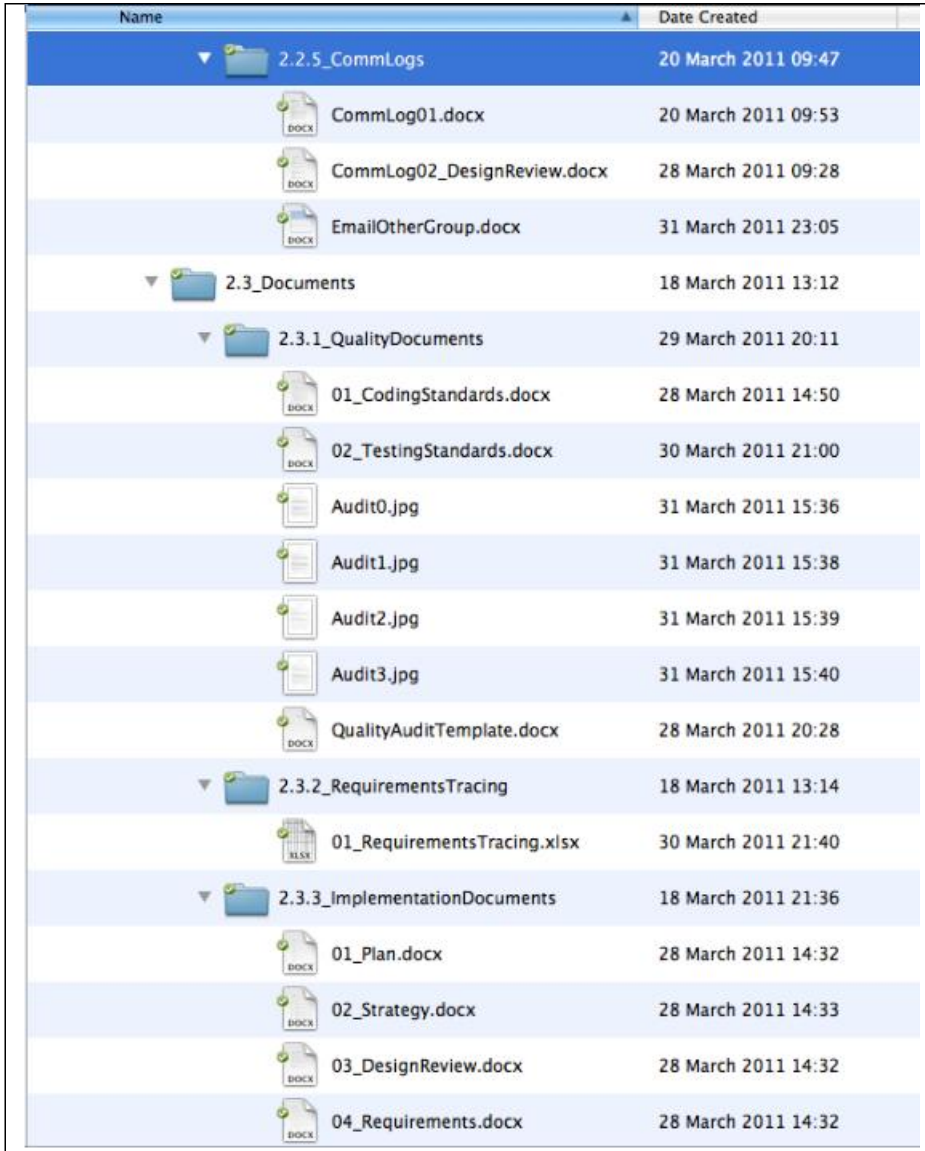
6.4 – 2.2.3: SYSTEM SNAPSHOT

Snapshot	Checked?
	01/04/2011

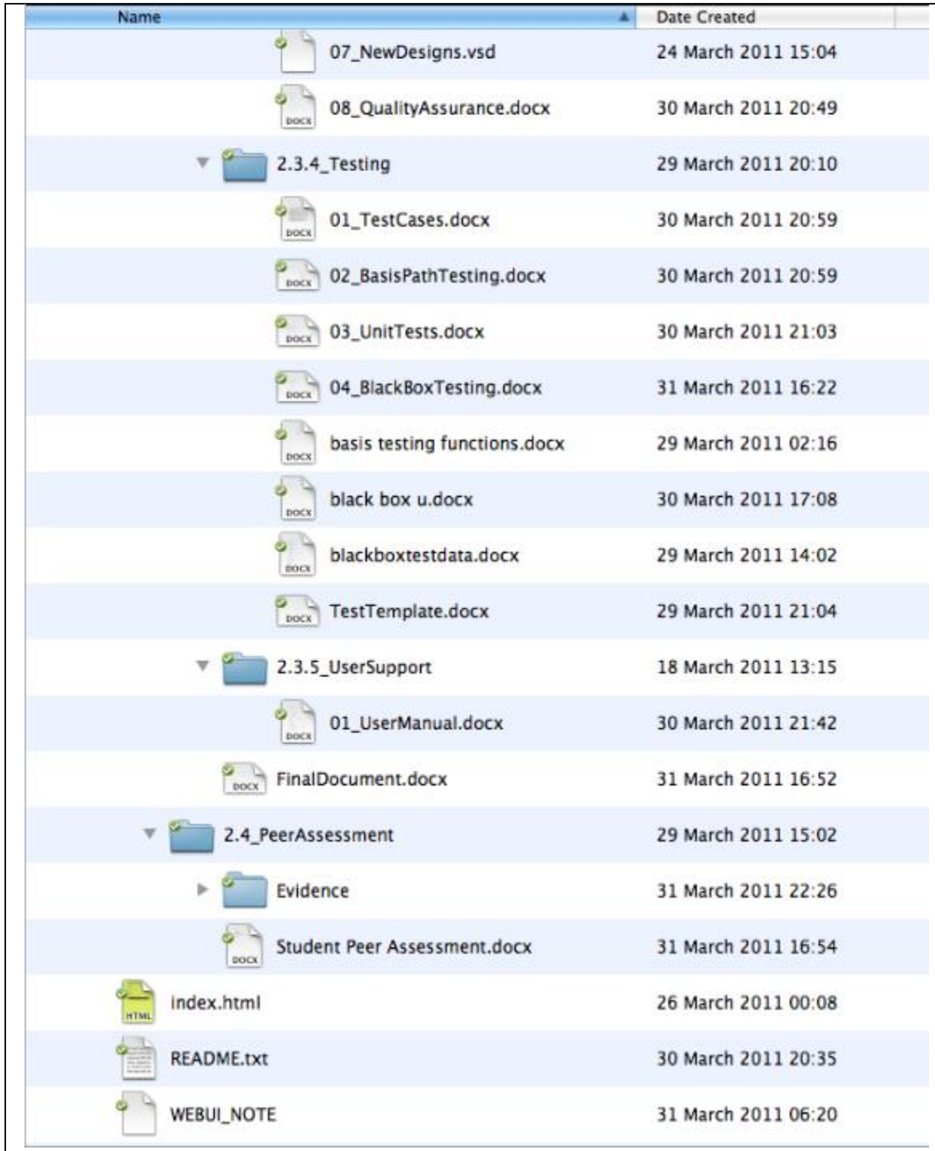
Name	Date Created
staffs-ppsp-repo	18 March 2011 13:10
1_Part1_Specification_Design	18 March 2011 13:37
1.1_RequiementsSpecification	18 March 2011 20:54
1.1.1_Requiements	18 March 2011 21:01
SoftwareRequirements.docx	4 March 2011 13:31
SoftwareRequirementsRevised.jpeg	11 March 2011 10:50
2_Part2_Implementation	18 March 2011 13:11
2.1_Code	18 March 2011 13:12
2.1.1_Current	18 March 2011 13:16
Ver_1.7	30 March 2011 16:41
30.03.11_20...ppsp-project	29 March 2011 00:10
bin	29 March 2011 16:20
build.xml	30 March 2011 15:05
doc	30 March 2011 13:56
nbproject	30 March 2011 15:05
src	29 March 2011 00:10
Web_UI_InProgress_Backup	24 March 2011 15:23
Thu 24 Mar	24 March 2011 15:23

	01/04/2011

Name	Date Created
▼ 2.1.2_Old	18 March 2011 13:16
▶ Ver_1.1	24 March 2011 01:13
▶ Ver_1.2	24 March 2011 01:13
▶ Ver_1.3	24 March 2011 01:13
▶ Ver_1.4	24 March 2011 01:20
▶ Ver_1.5	24 March 2011 01:14
▶ Ver_1.5_JAR	27 March 2011 17:51
▶ Ver_1.6	24 March 2011 14:18
▶ Ver_1.6.5	30 March 2011 20:26
▼ 2.2_Control_Logs	18 March 2011 13:29
▼ 2.2.2_RepoIndexes	18 March 2011 14:20
Index.xlsx	30 March 2011 21:40
▼ 2.2.3_Snapshot	18 March 2011 13:13
leopard-folder-big.png	28 March 2011 15:06
Placeholder.txt	24 March 2011 02:03
▼ 2.2.4_VersionControl	18 March 2011 13:13
DocVersionControl.xlsx	30 March 2011 21:40
SWVersionControl_NEW.xls	24 March 2011 14:18
SWVersionControl.xlsx	20 March 2011 10:09
▼ 2.2.5_CommLogs	20 March 2011 09:47

	01/04/2011

Name	Date Created
2.2.5_CommLogs	20 March 2011 09:47
CommLog01.docx	20 March 2011 09:53
CommLog02_DesignReview.docx	28 March 2011 09:28
EmailOtherGroup.docx	31 March 2011 23:05
2.3_Documents	18 March 2011 13:12
2.3.1_QualityDocuments	29 March 2011 20:11
01_CodingStandards.docx	28 March 2011 14:50
02_TestingStandards.docx	30 March 2011 21:00
Audit0.jpg	31 March 2011 15:36
Audit1.jpg	31 March 2011 15:38
Audit2.jpg	31 March 2011 15:39
Audit3.jpg	31 March 2011 15:40
QualityAuditTemplate.docx	28 March 2011 20:28
2.3.2_RequirementsTracing	18 March 2011 13:14
01_RequirementsTracing.xlsx	30 March 2011 21:40
2.3.3_ImplementationDocuments	18 March 2011 21:36
01_Plan.docx	28 March 2011 14:32
02_Strategy.docx	28 March 2011 14:33
03_DesignReview.docx	28 March 2011 14:32
04_Requirements.docx	28 March 2011 14:32

	01/04/2011

Name	Date Created
07_NewDesigns.vsd	24 March 2011 15:04
08_QualityAssurance.docx	30 March 2011 20:49
2.3.4_Testing	29 March 2011 20:10
01_TestCases.docx	30 March 2011 20:59
02_BasisPathTesting.docx	30 March 2011 20:59
03_UnitTests.docx	30 March 2011 21:03
04_BlackBoxTesting.docx	31 March 2011 16:22
basis testing functions.docx	29 March 2011 02:16
black box u.docx	30 March 2011 17:08
blackboxtestdata.docx	29 March 2011 14:02
TestTemplate.docx	29 March 2011 21:04
2.3.5_UserSupport	18 March 2011 13:15
01_UserManual.docx	30 March 2011 21:42
FinalDocument.docx	31 March 2011 16:52
2.4_PeerAssessment	29 March 2011 15:02
Evidence	31 March 2011 22:26
Student Peer Assessment.docx	31 March 2011 16:54
index.html	26 March 2011 00:08
README.txt	30 March 2011 20:35
WEBUI_NOTE	31 March 2011 06:20

SECTION 7 – APPENDIX