**Principles and Practices of Software Production**

**Staffordshire Software Solutions**

David Russell | James Nightingale | Amanda Patterson | Scott Dennison

Table of Contents

Section 1   
1.1 - 2.3.3\_01: Plan 2

1.2 - 2.3.3\_02: Strategy 3

Stage One 3

Review Specification 3

Stage Two 4

Identify Resources and Resource Requirements 4

Available Resources 5

1.3 - 2.3.3\_03: Design Review 6

Overview 6

Minutes 6

1.4 - 2.3.3\_04: Requirements 7

Development 8

Test Case Engineering 8

Testing 8

White Box 9

Black Box 9

Acceptance Testing 9

1.6 - 2.3.3\_06: Requirements Annotated 10

1.7 - 2.3.3\_07: Additional Designs 11

1.8 – 2.3.3\_08 Test Cases 12

1.9 - 2.3.3\_09: Basis Path Tests 0

Section 2   
2.1 – 2.3.1.1\_01: Coding Standards 1

Commenting Style 1

Conventions: 1

Variable Naming Style 1

Conventions: 1

Function and Method Naming Style 2

Conventions: 2

Brace and Indentation Formatting 2

Indentation Conventions: 2

Brace Conventions: 2

Example: 3

2.2 – 2.3.1.1\_02: Testing Standards 4

# Section 1 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. |

# 1.6 - 2.3.3\_06: Requirements Annotated

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 Test Cases

|  |  |  |
| --- | --- | --- |
| **Case** | **Testing Requirements** | **Test Case Completed** |
| 01.001 | To access the system the user must enter their username and password |  |
| 01.002 | To access the admin privileges the user must enter the admin username and password |  |
| 02.001 | Bookings recorded |  |
| 02.002 | Log kept of bookings made by the operator |  |
| 03.001 | Bookings not allowed less than 6 hours before film |  |
| 03.002 | A user must be able to select up to 10 seats to be booked at a time |  |
| 03.003 | Peak times are to be set between specified hours |  |
| 04.001 | User must be able to cancel film bookings at request of customer |  |
| 04.002 | Details required for cancellation: Customer, film times, payment details |  |
| 05.001 | User is able to view the seats currently available for booking in a showing |  |
| 06.001 | User able to see which seats are premium seats, Rows H and I |  |
| 06.002 | Price difference in place for booking premium seats |  |
| 07.001 | User able to add: new film, with details: Film name and age rating |  |
| 07.002 | User able to edit film information currently available |  |
| 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. |  |

# 1.9 - 2.3.3\_09: Basis Path Tests

# Section 2 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

# Section3 - Testing