|  |
| --- |
| Student: Mr James Biswell  OU PI Number: Y3258789  Tutor: Dr Adrian Pullin  TM470-17B Project Title:  Design and Development of a Player and Fixture Management Software System for Long Marston Cricket Club  EMA:  The Final Project Report  Date: 2nd September 2017 |

Table of Contents

[1 Problem Description 5](#_Toc492937042)

[2 Account of Related Literature 9](#_Toc492937043)

[2.1 Literature Search 9](#_Toc492937044)

[2.2 Literature Review 9](#_Toc492937045)

[3 Account of Project Work and its Outcome 12](#_Toc492937046)

[3.1 Business Domain Description 14](#_Toc492937047)

[3.2 Domain Structural Model 14](#_Toc492937048)

[3.3 Use Case Model 15](#_Toc492937049)

[3.4 Use Case List 15](#_Toc492937050)

[3.5 Use Case Elaboration 16](#_Toc492937051)

[3.6 Requirements Document - Volere Template 16](#_Toc492937052)

[3.7 Requirements Prioritisation 16](#_Toc492937053)

[3.8 Initial Software Architecture Choice 17](#_Toc492937054)

[3.9 Analysis Structural Model 17](#_Toc492937055)

[3.10 Design Structural Model 18](#_Toc492937056)

[3.11 Dynamic Behavioural Models 18](#_Toc492937057)

[3.12 Review Software Architecture Choice 19](#_Toc492937058)

[3.13 Initial Software Construction of First Evolutionary Prototype 20](#_Toc492937059)

[3.14 Implement/Test UC1 “Add Player” in Business Layer 23](#_Toc492937060)

[3.15 Design of User Interface and Integration with Business Layer 23](#_Toc492937061)

[3.16 Implement/Test UC1 “Add Player” in User Interface Layer 24](#_Toc492937062)

[3.17 Implement/Test UC3 “View Player” and UC8 “Amend Player” 24](#_Toc492937063)

[3.18 Implement/Test UC2 “View Player List” 25](#_Toc492937064)

[3.19 Code and Test Persistence of First Evolutionary Prototype 25](#_Toc492937065)

[3.20 Refactor/Test UC1 “Add Player” in Business Layer to Solve Persistence Issue 26](#_Toc492937066)

[3.21 Evaluate First Evolutionary Prototype with Stakeholder 26](#_Toc492937067)

[3.22 Deploy/Test First Evolutionary Prototype with Stakeholder 27](#_Toc492937068)

[3.23 Prioritise Requirements and Use Cases for Second Evolutionary Prototype 28](#_Toc492937069)

[3.24 Review Existing Structural and Behavioural Models 28](#_Toc492937070)

[3.25 Create NetBeans Project for Second Evolutionary Prototype 29](#_Toc492937071)

[3.26 Implement/Test UC13 “Add Fixture” 30](#_Toc492937072)

[3.27 Implement/Test UC14 “View Fixture List” 31](#_Toc492937073)

[3.28 Implement/Test UC15 “View Fixture” and UC18 “Amend Fixture” 31](#_Toc492937074)

[3.29 Implement/Test UC19 “Amend Availability of Players for Specific Fixture” 31](#_Toc492937075)

[3.30 Implement/Test UC20 “Select Players for Fixture” 32](#_Toc492937076)

[3.31 Implement/Test UC6 “View Fixtures Available For” 32](#_Toc492937077)

[3.32 Implement/Test UC7 “View Fixtures Selected For” 33](#_Toc492937078)

[3.33 Refactor/Test UC12 “Delete Player” and UC21 “Delete Fixture” in Business Layer 33](#_Toc492937079)

[3.34 Design UC22 “Add Fixture Results” in User Interface Layer 34](#_Toc492937080)

[3.35 Evaluate Second Evolutionary Prototype with Stakeholder 34](#_Toc492937081)

[3.36 Deploy/Test Second Evolutionary Prototype with Stakeholder 35](#_Toc492937082)

[4 Legal, Social, Ethical and Professional Issues 36](#_Toc492937083)

[4.1 Legal Issues 36](#_Toc492937084)

[4.2 Social Issues 36](#_Toc492937085)

[4.3 Ethical Issues 36](#_Toc492937086)

[4.4 Professional Issues 36](#_Toc492937087)

[5 Review of Current Stage of Project Work 37](#_Toc492937088)

[6 Review of Project Management 40](#_Toc492937089)

[6.1 Project Planning & Management 40](#_Toc492937090)

[6.2 Resources, Activities & Skills Needed 48](#_Toc492937091)

[6.3 Risk Management 48](#_Toc492937092)

[7 Review of Personal Development 50](#_Toc492937093)

[8 References 52](#_Toc492937094)

[9 Bibliography 54](#_Toc492937095)

[9.1 Project Management 54](#_Toc492937096)

[9.2 Software Development 54](#_Toc492937097)

[9.3 UML & Modelling 54](#_Toc492937098)

[9.4 Java Programming 55](#_Toc492937099)

[9.5 Software Testing 55](#_Toc492937100)

[9.6 Legal, Social, Ethical and Professional Issues in Computing 56](#_Toc492937101)

[10 Appendices 57](#_Toc492937102)

[10.1 Appendix 1 – Business Domain Description 57](#_Toc492937103)

[10.2 Appendix 2 – Domain Structural Model 59](#_Toc492937104)

[10.3 Appendix 3 – Use Case Model 60](#_Toc492937105)

[10.4 Appendix 4 – UC1 “Add Player” Use Case 62](#_Toc492937106)

[10.5 Appendix 5 – UC8 “Amend Player” Use Case 63](#_Toc492937107)

[10.6 Appendix 6 – Requirements Document - Volere Template 64](#_Toc492937108)

[10.7 Appendix 7 – Analysis Structural Model 67](#_Toc492937109)

[10.8 Appendix 8 – Design Structural Model 69](#_Toc492937110)

[10.9 Appendix 9 – UC1 “Add Player” Sequence Diagram 70](#_Toc492937111)

[10.10 Appendix 10 – UC8 “Amend Player” Sequence Diagram 72](#_Toc492937112)

[10.11 Appendix 11 – Player Class Constructor 73](#_Toc492937113)

[10.12 Appendix 12 – PlayerPerformance Class Helper Method 73](#_Toc492937114)

[10.13 Appendix 13 – Implementing the “Played For” Association 74](#_Toc492937115)

[10.14 Appendix 14 – CricketClub Class addPlayer Method 75](#_Toc492937116)

[10.15 Appendix 15 – User Interface Designs 76](#_Toc492937117)

[10.16 Appendix 16 – addPlayerButtonActionPerformed Method 78](#_Toc492937118)

[10.17 Appendix 17 – UC1 “Add Player” in Action 79](#_Toc492937119)

[10.18 Appendix 18 – UC3 “View Player” and UC8 “Amend Player” in Action 81](#_Toc492937120)

[10.19 Appendix 19 – UC2 “View Player List” in Action 82](#_Toc492937121)

[10.20 Appendix 20 – openMenuItemActionPerformed (and saveMenuItem) Methods 83](#_Toc492937122)

[10.21 Appendix 21 – CricketClub Class addFixture Method 84](#_Toc492937123)

[10.22 Appendix 22 – UC13 “Add Fixture” in Action 85](#_Toc492937124)

[10.23 Appendix 23 – Fixture Class toStringLong Method 85](#_Toc492937125)

[10.24 Appendix 24 – UC14 “View Fixture List” in Action 86](#_Toc492937126)

[10.25 Appendix 25 – UC15 “View Fixture” and UC18 “Amend Fixture” in Action 88](#_Toc492937127)

[10.26 Appendix 26 – addAvailablePlayerButtonActionPerformed Method & Methods Called 90](#_Toc492937128)

[10.27 Appendix 27 – removeAvailablePlayerButtonActionPerformed Method 91](#_Toc492937129)

[10.28 Appendix 28 – UC19 “Amend Availability Of Players For Specific Fixture” in Action 93](#_Toc492937130)

[10.29 Appendix 29 – addSelectedPlayerButtonActionPerformed Method 95](#_Toc492937131)

[10.30 Appendix 30 – Player Class viewFixturesAvailableFor Method 96](#_Toc492937132)

[10.31 Appendix 31 – UC22 “Add Fixture Results” User Interface Design 97](#_Toc492937133)

[10.32 Appendix 32 – Project Tutor Correspondence 99](#_Toc492937134)

# 1 Problem Description

The project title did not change over the course of the 33 weeks of the project; it remained a valid summary of the project’s overall aim and scope:

“Design and Development of a Player and Fixture Management Software System for Long Marston Cricket Club.”

The project’s overall aim was to evaluate, select and effectively apply relevant software development tools, techniques and processes in order to understand and model the business domain of the cricket club and capture the requirements of the stakeholder (the club captain); and in order to design and develop a software system that fulfils the pragmatic needs of the stakeholder to manage information regarding players and fixtures, and record results.

Engagement with the stakeholder (a work colleague) right at the project’s outset revealed that the existing means of managing information for players and fixtures was based on ad-hoc spreadsheets and that player availability and selection was communicated and captured in many ways: text messages, emails, face-to-face and telephone conversations, paper notes etc. This engagement also elicited a detailed description of the business domain, see Appendix 1, p.57.

The main practical problem that needed to be solved, therefore, was to put in place a robust and standardised method for information management, because the existing means were relatively inefficient, piecemeal, and diverse.

The main real-world benefits of solving this practical problem by successfully implementing a software system are:

* A measurable reduction in the time taken by the stakeholder to manage player and fixture information
* A quantitative improvement in the accuracy and comprehensiveness of the information that is managed by the stakeholder
* A qualitative improvement in the experience of the players who collaborate with the stakeholder concerning such information

The practical scope of the project was tightly defined. The business domain and its boundary were clearly understood and documented in the Business Domain Description (see Appendix 1, p.57) and in the Domain Structural Model (see Appendix 2, p.59). Subsequent engagement with the stakeholder elicited the creation of a complete Use Case Model (see Appendix 3, p.60) and Requirements Document (see Appendix 6, p.64), both thereafter reviewed and agreed with the stakeholder, such that the scope of the software solution being developed and its boundary were likewise defined, in terms of the functionality that should be provided.

The Requirements Document specifies the following five summary functional requirements:

1. *Functional requirements.* The new system shall:

* F1: Manage information for (i.e. allowing for information to be added, viewed, amended and deleted) all Long Marston Cricket Club players.
* F2: Manage information for (i.e. ibid) all fixtures that are scheduled for the club’s two teams, the 1st XI and 2nd XI.
* F3: Manage information for the availability of players for scheduled fixtures.
* F4: Manage information for the selection of players for scheduled fixtures.
* F5: Manage information for the result of each completed fixture, and for each player’s performance therein.

While the foregoing summarises the overall functionality of the system, further detail regarding its functional requirements is recorded in each of the concrete user-goal level base use cases as they are elaborated in more detail.

The key ICT aspects of this problem were those related to ensuring that the approach taken to developing a software solution meant that the resultant system exhibited the desirable software characteristics of usefulness, and good usability, reliability, and availability (Kruchten, 2004, p.159).

So, the project was fundamentally concerned with applying an engineering approach to software development, within the actual live context of the cricket club business domain, in order that considerations of software quality were built into all stages of the process.

The personal benefit of doing this was to enhance knowledge of, skills for and experience of how to undertake a software development project successfully, both by researching ways of doing so, by actually accomplishing this in a practical context, and, throughout the process, by reflecting critically on the effectiveness of the approach being taken.

At the start of the project, existing knowledge of conducting a software development project was entirely based on OU study. As both the research and practical work of the project progressed, further refinement of the project’s scope was possible, in the context of the software development tools, techniques and processes being chosen and exploited for their efficient and effective contribution to software quality.

Thus, the overarching process used on the project was the Rational Unified Process (RUP) for iterative and incremental software development (IBM Developer Works, 2005). Just as RUP is the basis of the development process taught in OU TM354, so further research reinforced its suitability, given that it “embeds guidance on many modern techniques and approaches: object technology and component-based development, modelling and UML, architecture, iterative development . . .” (Kruchten, 2004, p.34). These techniques were all relevant to the project.

An agile approach was taken to UML and modelling on the project, whereby models were created purposively and just to a sufficient level of accuracy and detail. The main purposes were either to aid my understanding of the domain or think about the system, or to aid communication with stakeholder, since the overall aim was to design and develop a software system, rather than modelling for its own sake (Ambler, 2002, pp.31-2).

Java (Oracle Technology Network, n.d. a) was chosen by default as the programming language for the project, because this is the only language in which there was sufficient expertise. However, Java is a mature language that fundamentally embodies an object-oriented methodology for development, and was therefore appropriate for use within an RUP context.

The final output intended for and actually delivered by the project was the software solution for use by the stakeholder, which has a Windows-look-and-feel user interface. The system was intended to, and indeed does, run as a stand-alone application on any personal laptop or desktop computer (operating in a standard domestic environment) which has the Java Platform Standard Edition 6 Runtime Environment installed (Oracle Technology Network, n.d. b). The virtual machine functionality of Java makes possible this cross-platform capability (Merx and Norman, 2007, pp.553-5).

The system was developed using the NetBeans Integrated Development Environment (IDE), which supports the complete Software Development Lifecycle (SDLC), and the specific version (6.9.1) was that previously used for OU M362 (NetBeans, n.d.). This was already correctly installed and configured for use with the relevant Java SE Development Kit (JDK 6u45) and Java SE Runtime Environment (JRE 6u45), in advance of the project (Oracle Technology Network, n.d. b).

Two evolutionary software prototypes (the second being the final system deployment) were planned for and constructed during the project, such that the software developed in each iteration cumulatively created the eventual system (Dawson, 2009, p.125). This was intended to lead to iterative improvement in the quality of the software and incremental implementation of use case functionality. Larman (2005, p.17) outlines the benefits (for this project) that were therefore envisaged: less project failure, early risk mitigation, early visible progress, early feedback, managed complexity and learning during the process to improve it, iteration by iteration. Kruchten (2004, p.23) reinforces this argument.

Intermediate outputs produced and reviewed over the course of the project were use case and requirements documentation, domain and system modelling artefacts, and the first evolutionary prototype. These provided regular opportunities for feedback with the stakeholder, so that progress was assessed and an appropriate solution continued to be designed and developed.

Some analysis of the likely impact of the project occurred during the evaluation of the two evolutionary prototypes with the stakeholder, in terms of usability testing and requirements-based acceptance testing, for verification and validation of the system. For further discussion of the impact of the system in terms of the cricket club organisational context, see Section 4, p.36.

Further analysis of its impact, in terms of the stakeholder accruing the three real-world benefits described above, will be possible only after the close of the project and this report, now that the system has been deployed and can be used actively within the cricket club.

# 2 Account of Related Literature

## 2.1 Literature Search

The sources chosen for their relevance to the project, and which are presented in the Bibliography (see p.54), were categorised under the following topic headings, which were determined as being areas of particular interest for the project:

* Project Management
* Software Development
* UML & Modelling
* Java Programming
* Software Testing
* Legal, Social, Ethical and Professional Issues in Computing

As the project progressed, the topic heading of Software Testing was added as a required area for research.

All the books listed in the Bibliography are credible sources; each is published by a respected publisher within the computing domain. The two online sources are both official Java documentation published by Oracle, so there is the possible caveat of some manufacturer-produced source partiality, for product promotion purposes. While this is not likely for the formal API specification of how Java works, the suite of online Java Tutorials, albeit that they are primarily of educational intent, are unlikely to highlight any disadvantages of the programming language.

## 2.2 Literature Review

The emphasis here in reviewing those that have been studied in some depth is to show how their subject matter was relevant to the project, and how the knowledge gained from them was applied to carrying out the project’s practical and theoretical work.

Please note that all references in this section refer specifically to the Bibliography, see p.54.

Project Management

Cole and Scotcher (2015) is a concise introduction to agile project management, briefly discussing its evolution from the Manifesto for Agile Software Development (2001), stating its intended outcomes and emphasising its benefits, differentiating it from other project management methodologies, describing three agile frameworks (Lean, Scrum and Kanban), and showing how to put them into practice. Despite the source’s sometimes zealous bias, it benefitted the project’s latter stages, to improve its focus on a Minimum Viable Product / Feature Set (MVP / MFS).

Dawson (2009) provides clear guidance to students on all aspects of computing projects and was therefore of importance throughout the project. Its whole content was reviewed in detail as a sound source of realistic advice for choosing, planning, researching, conducting, controlling, evaluating and reporting on projects.

The Stationery Office (2009) is an exhaustive reference source for PRINCE2 Practitioners (such as myself). While the project management methods, themes and processes of the PRINCE2 methodology should be tailorable to all types and sizes of project, the nature of a single-person, part-time, academic software development project is quite specialised and so, while providing a good starting point, it became clear as the project progressed that their relevance was limited, and indeed created unnecessary work.

Whitehead (2001) is very practically oriented, in terms of how to get a software development project successfully completed in the real world. It provides a summary outline of what needs to be done, when, and how, and its uncomplicated approach was helpful. Worthwhile chapters include those regarding project start-up and planning, task estimation, and contingency, and those concerning, architecture, analysis and design.

Software Development

Kruchten (2004) is a valuable overview of the whole of the RUP SDLC. Given that this is the main method that was used in the project, it was of primary relevance to it. The chapters about RUP’s process disciplines of Project Management, Business Modelling, Requirements, Analysis & Design, Implementation, Test and Deployment informed the fundamental purposes of the various phases of the project, and helped to determine the relevance and usefulness of the artefacts that should be output by the project, and the tools needed to produce them.

Merx and Norman (2007) puts Java at the core of software engineering, showing how an object-oriented methodology fits well with architecture-driven development of components, in the context of the Extended Unified Process, and using UML and design patterns. This source also falls under the topic heading of Java Programming. Its chapters concerning the design and development of Java applications, interfacing with users, implementing Java programs, and making iterative improvements, were especially useful.

UML & Modelling

Ambler (2002) describes a pragmatic approach, in terms of Agile values, principles and practices, to making modelling more effective. This technique was used throughout the project so that models aided understanding of the domain or reasoning about the system i.e. so that its focus was on solving the problem, with software. Part One of the book has permeated my thinking about modelling since studying it on OU TM354 and certainly made those project activities easier and better. Part Two (about Agile Modelling (AM) in an organisational context) and Part Three (about AM and XP) were less relevant, but Part Four, about AM and UP (and specifically about the key benefit of a good Use Case Model) directly informed the project.

Booch et al. (2005) is a thorough reference work on UML. It was used in this way on the project, to clarify and deepen understanding about the meaning and use of aspects of the language’s diagrammatic representations.

Gamma et al. (1995) is generally thought of as a seminal source on the use of patterns in software engineering. However, it is written at quite an abstract level, and did not particularly assist with the project’s practical work, other than to reinforce knowledge of the Model-View-Controller (MVC) design pattern and of the Singleton pattern. The example code is given in C++ rather than Java.

Larman (2005) is a comprehensive reference source on all aspects of object-oriented analysis and design, including UML diagramming notation. Its wide-ranging coverage of working with use cases was very useful, as was its treatment of UML class diagrams, interaction diagrams, GRASP principles, and how to map designs to code.

Java Programming

Barnes (2000) is an authoritative work on Java (1,028 pages). It was thus used purely for reference purposes, to further knowledge of particularities of syntax, where the Java API was not sufficiently explanatory.

Barnes and Kölling (2005), however, although used as a first port of call given its simplicity (being a practical introduction to Java), was not typically detailed enough for the level of programming required by the project.

Java Platform SE 6 API Specification (2015), i.e. the online Java API, was used extensively during software construction and was thus of core relevance to the project. Increasing familiarity with how the API is structured, and therefore how the Java language is structured and can be used, was key to programming correctly.

Oracle Java Documentation (2015) i.e. the online Java Tutorials, were also used during implementation as a good starting point or refresher for understanding how to use the core Java packages.

Riley (2003) is a detailed work on Java programming within the object-oriented paradigm, and with an informative emphasis on good software engineering practice, programming by contract and defensive programming. It was used on the project in order properly to contextualise specific programming challenges as they arose, in a better way than was found to be possible with (Barnes, 2000) or (Java Platform SE 6 API Specification, 2015). Of particular use were the chapters concerning logic and selection, file input and output, and Swing.

# 3 Account of Project Work and its Outcome

It is helpful to reiterate the project’s overall aim for this account:

To evaluate, select and effectively apply relevant software development tools, techniques and processes in order to understand and model the business domain of the cricket club and capture the requirements of the stakeholder (the club captain); and in order to design and develop a software system that fulfils the pragmatic needs of the stakeholder to manage information regarding players and fixtures, and record results.

This account is sub-divided into 36 sections. Analysing the problem mainly occurred in the first stage of the project:

Table 1 – Project First Stage

|  |  |  |
| --- | --- | --- |
| **Section** | **First Stage Major Practical Activity** | **Date** |
| 3.1 | Business Domain Description | 06/03/2017 |
| 3.2 | Domain Structural Model | 15/03/2017 |
| 3.3 | Use Case Model | 17/03/2017 |
| 3.4 | Use Case List | 17/03/2017 |
| 3.5 | Use Case Elaboration | 04/04/2017 |
| 3.6 | Requirements Document - Volere Template | 07/04/2017 |
| 3.7 | Requirements Prioritisation | 07/04/2017 |
| 3.8 | Initial Software Architecture Choice | 10/04/2017 |

Synthesising the software solution to this problem occurred in the second and third stages of the project:

Table 2 – Project Second Stage

|  |  |  |
| --- | --- | --- |
| **Section** | **Second Stage Major Practical Activity** | **Date** |
| 3.9 | Analysis Structural Model | 21/04/2017 |
| 3.10 | Design Structural Model | 23/04/2017 |
| 3.11 | Dynamic Behavioural Models | 14/05/2017 |
| 3.12 | Review Software Architecture Choice | 19/05/2017 |
| 3.13 | Initial Software Construction Of First Evolutionary System Prototype | 19/ to 21/05/2017 |
| 3.14 | Implement/Test UC1 "Add Player" In Business Layer | 26/05/2017 |
| 3.15 | Design Of User Interface and Integration With Business Layer | 26/05/2017 |
| 3.16 | Implement/Test UC1 "Add Player" In User Interface Layer | 27/05/2017 |
| 3.17 | Implement/Test UC3 "View Player" And UC8 "Amend Player" | 28/ to 30/05/2017 |
| 3.18 | Implement/Test UC2 "View Player List" | 25/06/2017 |

Table 3 – Project Third Stage

|  |  |  |
| --- | --- | --- |
| **Section** | **Third Stage Major Practical Activity** | **Date** |
| 3.19 | Code and Test Persistence Of First Evolutionary System Prototype | 03/08/2017 |
| 3.20 | Refactor/Test UC1 "Add Player" In Business Layer To Solve Class Variable Persistence Issue | 04/08/2017 |
| 3.21 | Evaluate First Evolutionary System Prototype With Stakeholder | 05/08/2017 |
| 3.22 | Deploy/Test First Evolutionary System Prototype With Stakeholder | 05/08/2017 |
| 3.23 | Prioritise Requirements And Use Cases For Second Evolutionary System Prototype | 06/08/2017 |
| 3.24 | Review Existing Analysis & Design Structural Models And Behavioural Models | 06/08/2017 |
| 3.25 | Create NetBeans Project For Second Evolutionary System Prototype | 06/08/2017 |
| 3.26 | Implement/Test UC13 "Add Fixture" | 06/08/2017 |
| 3.27 | Implement/Test UC14 "View Fixture List" | 12/08/2017 |
| 3.28 | Implement/Test UC15 "View Fixture" And UC18 "Amend Fixture" | 12/ to 13/08/2017 |
| 3.29 | Implement/Test UC19 "Amend Availability Of Players For Specific Fixture" | 20/08/2017 |
| 3.30 | Implement/Test UC20 "Select Players For Fixture" | 21/08/2017 |
| 3.31 | Implement/Test UC6 "View Fixtures Available For" | 21/08/2017 |
| 3.32 | Implement/Test UC7 "View Fixtures Selected For" | 21/08/2017 |
| 3.33 | Refactor/Test UC12 "Delete Player" and UC21 "Delete Fixture" In Business Layer To Remove All Object References | 22/08/2017 |
| 3.34 | Design UC22 "Add Fixture Results" Tabbed Pane In User Interface | 23/08/2017 |
| 3.35 | Evaluate Second Evolutionary System Prototype With Stakeholder | 02/09/2017 |
| 3.36 | Deploy/Test Second Evolutionary Prototype With Stakeholder | 02/09/2017 |

Evaluating the suitability and effectiveness of the software solution, in terms of verification and validation of the product, occurred with the stakeholder during the third stage of the project, but there were also earlier review meetings with him to ensure complete mutual understanding of the business domain and proposed solution.

There were also reviews of the project’s intermediate outputs during the analysis and design phase, so that the models continued to assist with constructing the correct solution.

Finally, the software was tested throughout its construction to ensure that the solution worked properly, so it was evaluated on an ongoing basis.

The following 36 subsections provide evidence of the activities tabulated above and also comprise discussion of how the relevant software development and object-oriented concepts and principles have been applied specifically in the context of the software solution being developed for the cricket club.

## 3.1 Business Domain Description

This is shown in Appendix 1, p.57. It was written and reviewed in collaboration with stakeholder, also using two data resources for reference.

Firstly, the website of the Wetherby Cricket League, to which Long Marston Cricket Club belongs (Wetherby Cricket League, n.d.). This was used to determine the exact structure of the competitions (with their associated fixtures) in which the club participates each season.

Secondly, the existing spreadsheet used to record the match information the stakeholder is obliged to report to the league after each fixture. The extract below shows how basic the existing method of information management was and, therefore, the value that could be added by a software solution.

Table 4 – Extract from the Existing Spreadsheet Used to Record Players and Fixtures

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **23/04/2016 - Ledsham (Away) - Won** | | | |  |  |  |
| No. | Name | Runs | Wickets | Catches | Stumpings | Keeper |
| 1 | Sankara Subraman | 5 | 0 for 24 | 0 | 0 |  |
| 2 | Ed Rawling | 3 | 0 for 11 | 1 | 0 |  |
| 3 | Kieran Trotter | 6 | 0 for 17 | 0 | 0 |  |
| 4 | Scott Lawniczak | 5 |  | 0 | 0 |  |
| 5 | Simon Lewis | 36 |  | 2 | 0 | Y |
| 6 | Alan Armitage | 55\* |  | 0 | 0 |  |
| 7 | Joe Hall | 3 |  | 0 | 0 |  |
| 8 | Steve Barnitt | 23\* | 3 for 51 | 2 | 0 |  |
| 9 | Chris Lewis | DNB | 4 for 47 | 0 | 0 |  |
| 10 | Michael Schofield | DNB | 0 for 16 | 0 | 0 |  |
| 11 | Vince Hughes | DNB | 1 for 31 | 0 | 0 |  |

It was agreed with the stakeholder that the Business Domain Description was accurate and comprehensive.

## 3.2 Domain Structural Model

This is shown in Appendix 2, p.59. The model was created to provide a clear visual representation in UML of those business objects (taken from the Business Domain Description) that were relevant to the scope of the proposed system, in a framework of conceptual classes, and their attributes (using basic data types where obvious, otherwise using conceptual attributes), associations and multiplicities (Larman, 2005, pp.94-100).

## 3.3 Use Case Model

This is shown in Appendix 3, p.60. The process of its creation, working closely with the stakeholder, provided a shared understanding of what he wanted the system to accomplish, in terms of the two actors of the cricket club using it, and the model shows where the system boundary exists (Larman, 2005, p.51).

Larman (2005, p.73) also suggests that there should be one use case for each user goal, with a common exception being “to collapse CRUD (create, retrieve, update, delete) separate goals into one CRUD use case”. However, the decision was taken to show separate use cases for adding, viewing, amending and deleting (i.e. “CRUD”ing) players and fixtures, so that the model represents in greater detail the required functionality of the system.

## 3.4 Use Case List

This is shown below, and is referred to throughout Section 3 of this report. The Use Case List provided all those shown in the Use Case Model with a number, thereafter used in the project, and applied the terminology of base, subfunction and addition use cases to draw a conceptual distinction between the different types (Larman, 2005, p.249).

Table 5 – Use Case List

|  |  |  |
| --- | --- | --- |
| UC1 | *add player* | base use case |
| UC2 | *view player list* | base use case |
| UC3 | *view player* | base use case |
| UC4 | *view player performance* | subfunction use case of UC3 |
| UC5 | *view wicketkeeper performance* | subfunction use case of UC3 |
| UC6 | *view fixtures available for* | subfunction use case of UC3 |
| UC7 | *view fixtures selected for* | subfunction use case of UC3 |
| UC8 | *amend player* | base use case |
| UC9 | *amend player performance* | subfunction use case of UC8 |
| UC10 | *amend wicketkeeper performance* | subfunction use case of UC8 |
| UC11 | *amend availability of player for specific fixtures* | subfunction use case of UC8 |
| UC12 | *delete player* | addition use case of UC8 |
| UC13 | *add fixture* | base use case |
| UC14 | *view fixture list* | base use case |
| UC15 | *view fixture* | base use case |
| UC16 | *view players available for fixture* | subfunction use case of UC15 |
| UC17 | *view players selected for fixture* | subfunction use case of UC15 |
| UC18 | *amend fixture* | base use case |
| UC19 | *amend availability of players for specific fixture* | subfunction use case of UC18 |
| UC20 | *select players for fixture* | subfunction use case of UC18 |
| UC21 | *delete fixture* | addition use case of UC18 |
| UC22 | *add fixture results* | base use case |

## 3.5 Use Case Elaboration

In order to exchange and agree ideas with the stakeholder, the user-goal base use cases (initiated by the Captain actor in the Use Case Model) relevant to the development of the first evolutionary prototype were elaborated in detail, namely:

* UC1 – *add player*
* UC2 – *view player list*
* UC3 – *view player*
* UC8 – *amend player*
* UC13 – *add fixture*

Two examples are provided in this report.

For UC1, see Appendix 4, p.62.

For UC8, see Appendix 5, p.63.

## 3.6 Requirements Document - Volere Template

This is shown in Appendix 6, p.64. The Volere Template which was used is based on the OU TM354 study material (The Open University, 2014 a, pp.115-8).

While the document reiterated some of the information already generated on the project, its overall benefit was as a single point of reference between the stakeholder and the developer of what the software solution should be like and do.

So, the document provides a systematic set of the system’s functional requirements (in summary form) and its software quality factors, and was therefore intended for use in usability testing and requirements-based acceptance testing during stakeholder evaluation.

## 3.7 Requirements Prioritisation

The decision was taken with the stakeholder that the first evolutionary software prototype should fulfil the following functional requirements from the Requirements Document:

* F1: Manage information for (i.e. allowing for information to be added, viewed, amended and deleted) all Long Marston Cricket Club players.
* F3: Manage information for the availability of players for scheduled fixtures.

Thus, the prototype was intended to realise those user-goal base use cases that had been elaborated, see Section 3.5, above.

## 3.8 Initial Software Architecture Choice

A straightforward Software Architecture was considered prior to the creation of the Analysis Structure Model and Design Structural Model, namely to use the Model-View-Controller (MVC) design pattern for the overall structure of the system (Gamma et al., 1995, pp.4-5) and, within the business layer, to model the CricketClub class as a central class based on the Singleton pattern (Gamma et al., 1995, pp.127-34). This architecture was to be reviewed after the creation of the relevant models.

## 3.9 Analysis Structural Model

This is shown in Appendix 7, p.67. Building on the Domain Structural Model, the intention with creating the Analysis Structural Model was to represent diagrammatically the actual structure of the business layer, in terms of the Java classes, their instance variables and associations, and the ways the system is constrained (Larman, 2005, pp.167-87).

For the classes CricketClub, Player and Fixture, instance variables (typically of a generic Collection type) were added, so that the required associations could be implemented, for example fixturesAvailableFor in the Player class.

The multiplicity and navigability of the associations was finalised. For example, a player can play in League and Evening Cup fixtures for both the 1st XI and 2nd XI, but the Sunday Cup is just for the 1st XI, so a Player object should be linked with a maximum of five PlayerPerformance objects. Both Player and Fixture should know about each other in terms of GRASP principles for all three associations between them and therefore the navigability is bi-directional (Larman, 2005, pp.361-71).

It was simplest to express two constraints as annotations, rather than using Object Constraint Language (OMG, 2017).

The class PlayerSummary was added to factor out various related player attributes shown in the Domain Structural Model, and therefore simplify the Player class. PlayerSummary is one of three classes involved in a composition relationship with Player, so that all the relevant player information is represented.

Four enumeration classes were added so that, where these classes are used as instance variables in others, there is just a pre-defined set of constants (Merx and Norman, 2007, pp.275-6). For example, the team is either the 1st XI or 2nd XI.

The CricketClub class was modelled with the annotation <<singleton>> showing that only one instance of the class should exist so that the system behaves predictably, because it is a central class through which the user interface interacts with the business layer (Gamma et al., 1995, pp.127-34).

The CricketClub class also shows its qualified associations with the Player and Fixture classes, with the value identities of playerNumber and fixtureNumber respectively, which are unique identifiers for instances of those classes.

## 3.10 Design Structural Model

This is shown in Appendix 8, p.69. Building on the Analysis Structural Model, the purpose of the Design Structural Model was to create a series of class diagrams for the most significant Java classes.

The model is not comprehensive, intentionally, because of an agile approach to modelling. It doesn’t repeat information already shown on the Analysis Structural Model. It doesn’t list the simple accessor methods for primitive Java data types. What is shown, however, primarily helped with reasoning about the software and should be accurate, regarding what it focusses on.

For all instance variables, it gives the appropriate access modifier and determines the actual data type to be used during software construction. For each class, it indicates the required instance methods, along with their access modifier and return type, but not the full method signature. The basic principle was to declare all variables as private, and methods as public, to ensure full data encapsulation.

The CricketClub class specifies methods for all of the user-goal base use cases, and their extensions.

After initial drafting, the model was reworked during the creation of the Dynamic Behavioural Models. So, for example, the method addPlayerPerformance was added to the Player class. When the Sequence Diagram for UC1 “Add Player” was created, it became clear that a specific method would be required to ensure that two PlayerPerformance objects for the same team type and fixture type (say, 1st XI and League) are not added to the collection referenced by a Player object. The method addPlayerPerformance in the Player class does this. Likewise, for addWicketkeeperPerformance.

Ambler (2002, p304-7) is insightful regarding the promotion of simplicity with an agile modelling approach on an UP project, and regarding creating models in parallel, actively iterating between them.

## 3.11 Dynamic Behavioural Models

Building on the Use Cases for UC1 “Add Player” and UC8 “Amend Player” (see Appendices 4, p.62 ; 5, p.63), Sequence Diagrams were created for both use cases, so that the necessary postconditions could be achieved, see Appendices 9, p.70 ; 10, p.72.

GRASP principles and responsibility-driven design were used in the creation of the diagrams, so that, for example, a Player object is responsible for creating the Player, PlayerPerformance and WicketkeeperPerformance objects for its composition associations (Larman, 2005, pp.364-71).

Again, the UC1 “Add Player” Sequence Diagram is not comprehensive, deliberately. It doesn’t show the creation of the fixturesAvailableFor, fixturesSelectedFor, or fixturesPlayedFor HashSet objects, when a new Player object is constructed, because this would just clutter a diagram that is being used to further understanding.

Once these two sequence diagrams had been created, it was decided that there was sufficient comprehension of the system’s required dynamic behaviour to move onto software construction without the need for further modelling at this stage. Software is the primary goal (Ambler, 2002, p.29).

## 3.12 Review Software Architecture Choice

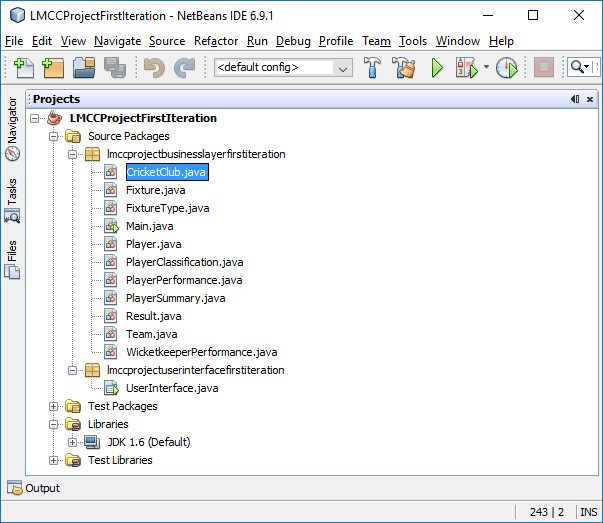
After reviewing the Analysis Structural Model and Design Structural Model, the initial choice of Software Architecture for the first evolutionary prototype was considered to be appropriate, because the models seemed to fit well both with the actual requirements of the system to represent the work of the cricket club, and with the shape of the patterns.

And so, the Model-View-Controller (MVC) design pattern was applied at an architectural level. The business layer contains the data model, while the user interface provides its screen presentation and also responds to user input (Gamma et al., 1995, pp.4-5).

Within the business layer, the CricketClub class was modelled on the Singleton pattern, to act as a central class through which the user interface interacts with the business layer (Gamma et al., 1995, pp.127-34). In so doing, the CricketClub class is responsible for getting the work of the use cases done (The Open University, 2014 b, pp.152-5).

Figure 1 – Software Architecture

The figure is shown below.



## 3.13 Initial Software Construction of First Evolutionary Prototype

Using the Analysis and Design Structural Models, a top-down approach was taken to initial software construction, so that the skeleton structure of each class was created (Whitehead, 2002, pp.262-3). This resulted in the overall business layer structure shown above in Figure 1.

The instance variables of each class were coded first, declaring interface types where possible, for future software flexibility through polymorphism. The automated variable encapsulation functionality of NetBeans was used to create default accessor methods.

Constructors were then coded for each class. The value identity of playerNumber for the Player class was implemented by declaring a class variable numberOfPlayers that is incremented whenever the constructor for Player is invoked, and then assigned to playerNumber, see Appendix 11, p.73.

A particularly interesting helper method was required for the PlayerPerformance class, to maintain the consistency of the instance variables for batting average, bowling average, and all-rounder index. This is called by its constructor and other setter methods, see Appendix 12, p.73.

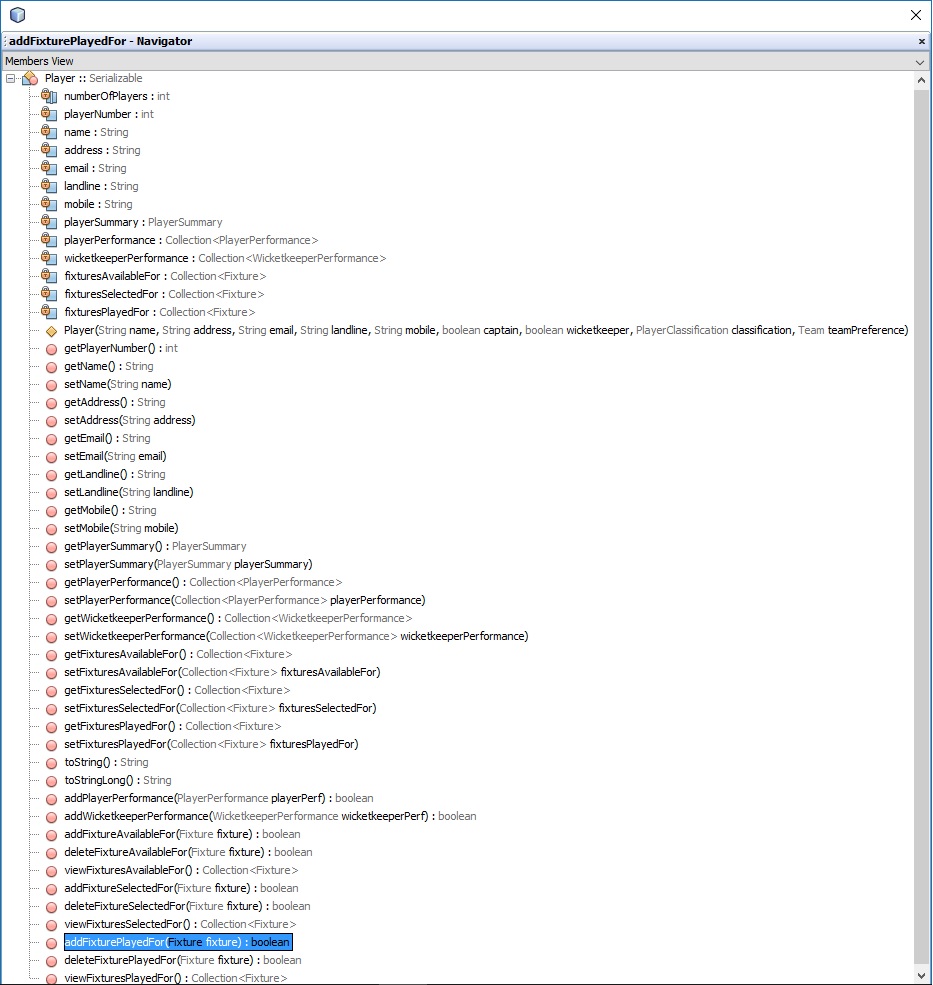
Using the Java Main class, accurate construction of each class was then tested.

Finally, the required methods for the associations between the Player and Fixture classes were coded, so that the required bi-directional navigability and constraints expressed in the Analysis Structural Model were implemented. For the equivalent methods in the Player and Fixture classes for the “played for” association, see Appendix 13, p.74.

This was all then tested, again using the Java Main class.

Figure 2 - The Completed Player Class

The figure is shown below.



## 3.14 Implement/Test UC1 “Add Player” in Business Layer

By following the UC1 “Add Player” Sequence Diagram (see Appendix 9, p.70), this use case was relatively easy to code within the CricketClub class, see Appendix 14, p.75. The Java Main class was used for testing, prior to construction of the user interface.

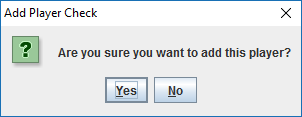
## 3.15 Design of User Interface and Integration with Business Layer

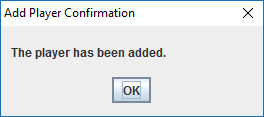
Using the Use Case Model as a panorama, the user interface was designed using the NetBeans GUI Builder, according to the design principles of good visibility, affordance and feedback (The Open University, 2007, pp.21-5).

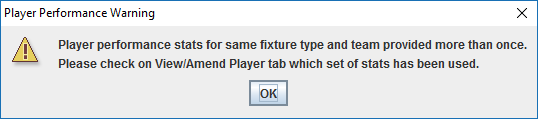
An example of the design for two tabbed panes “Add Player” and “View/Amend Player” is shown in Appendix 15, pp.76-7.

Good visibility is provided by having a consistent layout where possible between the tabbed panes, and by using clear labelling everywhere, most notably on the pane tabs themselves (and on buttons) to indicate how to perform the different use cases. Good affordance is inherent in using a familiar Windows-look-and-feel style. Good feedback has been reinforced by using Dialog windows so that the user must verify significant actions (like adding a player), confirmation is provided of success, and various warnings are provided (Oracle Java Documentation, 2015 b).

Figure 3 – Dialog Windows







The user interface is run from the Java Main method, and integrated with the business layer by means of the singleton instance of the CricketClub class being passed to it.

Main method:

public static void main(String[] args)

{

CricketClub lmcc = CricketClub.getInstance();

UserInterface.run(lmcc);

}

Run method in the user interface:

public static void run(CricketClub cricketClub)

{

final CricketClub cc = cricketClub;

java.awt.EventQueue.invokeLater(new Runnable()

{

public void run()

{

new UserInterface(cc).setVisible(true);

}

});

}

## 3.16 Implement/Test UC1 “Add Player” in User Interface Layer

The approach taken in coding the Add Player button on the Add Player tabbed pane was to get the relevant values from the text fields on the pane, to pass as parameters when the addPlayer method (see Appendix 14, p.75) is invoked on the CricketClub instance in the business layer.

A substantial fragment of the method code is shown in Appendix 16, p.78.

A screenshot of the functionality is shown in Appendix 17, p.79.

## 3.17 Implement/Test UC3 “View Player” and UC8 “Amend Player”

UC3 “View Player” was straightforward to code in the CricketClub class:

public Player viewPlayer(int playerNumber)

{

return this.players.get(playerNumber);

}

So was UC8 “Amend Player”, by following the sequence diagram (see Appendix 10, p.72), and reusing some of the code structure from UC1 “Add Player”.

The design decision was taken to realise both use cases through a single tabbed pane in the user interface. This is because once a player’s information has been displayed on the screen in text fields using the “View Player” button, then these are editable, and the data can be passed back to the business layer to amend the player’s details, when the “Amend Player” button is pressed.

The Use Case Model also shows UC8 “Amend Player” as being an extension of UC3 View Player. Similarly, UC12 “Delete Player” is shown as an extension of UC8 “Amend Player” and so there is also a “Delete Player button”. This functionality was also implemented.

A screenshot of the functionality is shown in Appendix 18, p.81.

## 3.18 Implement/Test UC2 “View Player List”

This was trivial to code in the CricketClub class:

public Map<Integer, Player> viewPlayerList()

{

return this.players;

}

But more complicated in the user interface. Once the collection is returned from the business layer, it is unpicked for display, using a StringBuilder:

private void

viewPlayerListButtonActionPerformed(java.awt.event.ActionEvent evt)

{

this.playerListTextArea.setText("");

Map<Integer, Player> playerList = new HashMap<Integer, Player>();

playerList = this.cricketClub.viewPlayerList();

Collection playerListArray = new ArrayList(playerList.values());

Iterator<Player> ite = playerListArray.iterator();

StringBuilder sb = new StringBuilder();

while (ite.hasNext())

{

Player player = ite.next();

sb.append(player.toStringLong());

sb.append("\n");

}

this.playerListTextArea.setText(sb.toString());

A screenshot of the functionality is shown in Appendix 19, p.82.

## 3.19 Code and Test Persistence of First Evolutionary Prototype

Persistence of the system prototype was achieved by ensuring that all of the Java classes in the business layer implement the Serializable interface, and import the java.io.Serializable package (Merx and Norman, 2007, p.308). This then means that the serialised form of the singleton instance of the CricketClub class can be written to file, and consequently all objects referenced directly by the CricketClub object, and the objects that they themselves reference. So, in effect, all instantiated objects making up the whole of the system’s state can also be serialised for later retrieval from file and reconstitution (Merx and Norman, 2007, p.461).

In the user interface, this was implemented practically by putting in place appropriate behaviour for the File Menu’s Open and Save commands. Effectively, to write the CricketClub singleton object to file on saving and to read it from file on opening. This required, respectively, use of the FileOutputStream and ObjectOutputStream Java classes, of the FileInputStream and ObjectInputStream ones, and of the necessary exception handling (Riley, 2003, pp.531-4).

The method code is shown in Appendix 20, p.83.

## 3.20 Refactor/Test UC1 “Add Player” in Business Layer to Solve Persistence Issue

The issue that arose on thorough testing of system persistence using object serialisation was that the value of the class variable numberOfPlayers (which is incremented whenever the constructor for Player is invoked, and then assigned to playerNumber, in order to create a value identity for objects of the Player class, see Section 3.13, p.20) was not saved. The value of static variables is not persistent with serialisation (Barnes, 2000, p.407).

The code was therefore refactored in the Player constructor not to use this static variable for creating a unique value for playerNumber for each Player object. Rather, the responsibility for doing this was shifted to the addPlayer method in the CricketClub class, with the following code:

int playerNumber = 0;

if (this.players.isEmpty())

{

playerNumber = 1;

}

else

{

Set<Integer> playerNumberSet = new HashSet<Integer>();

playerNumberSet = this.players.keySet();

int i = 1;

while (playerNumber == 0)

{

if (!(playerNumberSet.contains(i)))

{

playerNumber = i;

}

i++;

}

}

Player player = new Player(playerNumber, name, address, email,

landline, mobile, captain, wicketkeeper,

classification, teamPreference);

## 3.21 Evaluate First Evolutionary Prototype with Stakeholder

A demonstration of the first prototype was provided to the stakeholder, providing a walkthrough of all the user-goal base use cases that had been elaborated and then implemented with their subfunction use cases, namely:

|  |  |  |
| --- | --- | --- |
| UC1 | *add player* | base use case |
| UC2 | *view player list* | base use case |
| UC3 | *view player* | base use case |
| UC4 | *view player performance* | subfunction use case of UC3 |
| UC5 | *view wicketkeeper performance* | subfunction use case of UC3 |
| UC8 | *amend player* | base use case |
| UC9 | *amend player performance* | subfunction use case of UC8 |
| UC10 | *amend wicketkeeper performance* | subfunction use case of UC8 |
| UC12 | *delete player* | addition use case of UC8 |

This requirements-based acceptance testing satisfied the expectations of the stakeholder, both in terms of validation that the right product was being constructed and in terms of verifying that the first prototype had been built correctly (Dawson, 2009, p.137).

Usability testing was also performed to evaluate the following non-functional requirements from the Requirements Document (see Appendix 6, p.64):

LF1: The system does not require a colourful and pretty appearance, but rather should appear functional and uncomplicated.

Fit criterion: System appearance will be evaluated by the club captain.

U1: The system shall be easy to learn for the club captain.

Fit criterion: The club captain (who is very comfortable with IT) should be able

to learn to use 90 percent of the functionality of the system (relevant to him) in 2

hours.

P1: The system shall respond to most user input within 5 seconds. Fit criterion

self-contained.

The stakeholder was also satisfied that these fit criteria had been met.

## 3.22 Deploy/Test First Evolutionary Prototype with Stakeholder

Deployment required the Java SE Runtime Environment (JRE 6u45) to be installed on the stakeholder’s laptop, having been downloaded from the internet (Oracle Technology Network, n.d. b). Once this had been done, the “LMCCProjectFirstIteration.jar” Executable Jar File, taken from the NetBeans project, was transferred to his laptop and the prototype was started and tested in its operational environment, unproblematically. It was further explained that the “cricketclub.obj” file should remain in the same location as the .jar file, so that the cricket club data could be saved and loaded correctly.

## 3.23 Prioritise Requirements and Use Cases for Second Evolutionary Prototype

The decision was taken with the stakeholder that, with the first prototype having fulfilled the following functional requirement:

* F1: Manage information for (i.e. allowing for information to be added, viewed, amended and deleted) all Long Marston Cricket Club players.

the second evolutionary software prototype should also fulfil the following:

* F2: Manage information for (i.e. ibid) all fixtures that are scheduled for the club’s two teams, the 1st XI and 2nd XI.
* F3: Manage information for the availability of players for scheduled fixtures.
* F4: Manage information for the selection of players for scheduled fixtures.
* F5: Manage information for the result of each completed fixture, and for each player’s performance therein.

Thus, the second prototype was intended to realise the following user-goal base use cases and their subfunction use cases:

* UC13 – *add fixture*
* UC14 – *view fixture list*
* UC15 – *view fixture*
* UC18 – *amend fixture*
* UC22 – *add fixture results*

Following the evaluation of the first prototype, it was further agreed with the stakeholder that it was unnecessary to elaborate these to the same level as in the first iteration of modelling, as there was a mutual confidence from the prototype evaluation that what was actually needed would be implemented.

## 3.24 Review Existing Structural and Behavioural Models

On reviewing the existing Analysis Structural Model, Design Structural Model, and Sequence Diagrams, it was decided that sufficient modelling had been completed to be able to continue with software construction for the second prototype.

The structural models had been a constant and adequate source of reference to visualise the structure of the software during implementation and testing of the first prototype, so it was assumed that their fundamental usefulness to the project would continue. No modifications were needed to the classes or associations shown in the models, to realise the required use cases.

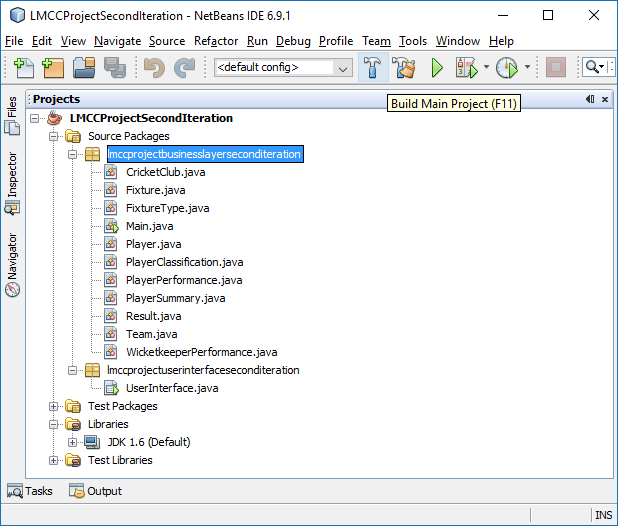
To reiterate, software is the primary goal (Ambler, 2002, p.29).

Regarding the existing Sequence Diagrams, while these only showed the object interactions needed to fulfil two already realised use cases, it was thought that enough was understood about how the dynamic behaviour of the system should be engineered, in order to implement the required, unrealised ones, especially given that the behaviour of some of these was similar to those already realised - both in the business domain, in terms of goal, precondition, postcondition and scenario, and in the software. For example, UC1 “Add Player” and UC13 “Add Fixture”, and UC2 “View Player List” and UC14 “View Fixture List”.

## 3.25 Create NetBeans Project for Second Evolutionary Prototype

For good basic configuration and change management practice, a separate NetBeans project was created. The automated refactoring functionality of NetBeans was used so that the renaming of packages did not affect the necessary package import statements for the classes.

Figure 4 – NetBeans Project



## 3.26 Implement/Test UC13 “Add Fixture”

The behaviour of this being very similar to that for UC1 “Add Player”, some of the code structure was reused in the business layer from the addPlayer method, for the addFixture method in the CricketClub class, except that the system also needs to check that a fixture for that date and team (e.g. 1st XI) does not already exist. See Appendix 21, p.84.

It was also easier to design and implement in the user interface, as the Fixture class is not involved in a complicated set of composition relationships like the Player class, so there is much less information to capture from text fields and marshal into the correct form to pass as parameters to the addFixture method.

A screenshot of the functionality is shown in Appendix 22, p.85.

## 3.27 Implement/Test UC14 “View Fixture List”

The code from UC2 “View Player” List both in the business layer and user interface, as well as the design of the tabbed pane, were almost entirely reused for implementation of this use case, see Section 3.18, p.25.

This code and design reuse was felt to have several benefits (Hunt and Thomas, 2000, p.33):

* Increased project efficiency
* Improved software quality by reusing reliable, tested code
* A consistent look-and-feel for the interface, enhancing the user experience

Again, Fixture being a simpler class than Player, the required toStringLong method in Fixture was more straightforward to code, see Appendix 23, p.85.

A screenshot of the functionality is shown in Appendix 24, p.86.

## 3.28 Implement/Test UC15 “View Fixture” and UC18 “Amend Fixture”

Substantial code and design reuse was again possible, from implementing UC3 “View Player” and UC8 “Amend Player” (see Section 3.17, p.24); and design reuse from UC13 “Add Fixture” (see Section 3.26, above) for consistency of interface layout.

Both use cases were realised through a single tabbed pane. Once the fixture information is displayed in text fields these are editable, and the data is passed back to the business layer for amendment. UC18 is an extension of UC15.

Furthermore, UC21 “Delete Fixture” is an extension of UC18 and so was also implemented.

A screenshot of the functionality is shown in Appendix 25, p.88.

## 3.29 Implement/Test UC19 “Amend Availability of Players for Specific Fixture”

This required coding from scratch in the user interface, in part calling the appropriate methods in the business layer for correctly manipulating bi-directional associations between the Player and Fixture classes, already coded during initial software construction, see Section 3.13, p.20.

To add an available player, the principle of design by contract was used, so that the user interface passes the selected Player to the addPlayerAvailableFor method invoked on the appropriate Fixture, and, if this returns true by passing the single conditional statement, then the user interface is updated and confirmation feedback is returned, otherwise an appropriate warning is returned (Hunt and Thomas, 2000, 109-19). See Appendix 26, p.90.

To remove an available player, the principle of defensive programming was additionally required, so that the user interface is also able to provide relevant text for an alternative warning if the player has already been selected for the fixture, although this is already accounted for as one of the two conditionals of the deletePlayerAvailableFor method of Fixture. See Appendix 27, p.91.

UC16 “View Players Available For Fixture” was also, concomitantly, implemented.

A screenshot of the functionality is shown in Appendix 28, p.93.

## 3.30 Implement/Test UC20 “Select Players for Fixture”

A similar strategy was used as for the implementation and testing for UC19, in the user interface.

A complicated set of nested conditionals was required for the addSelectedPlayerButtonActionPerformed method, and Riley (2003, pp.233-68) helped with thinking logically about selection instructions. See Appendix 29, p.95.

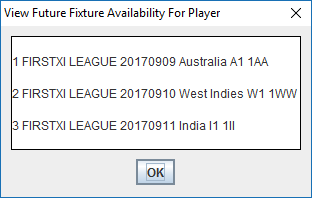
UC17 “View Players Selected For Fixture” also realised.

A screenshot of the functionality is shown in Appendix 28, p.93.

## 3.31 Implement/Test UC6 “View Fixtures Available For”

With UC19 having been realised as above, it was now possible to implement this subfunction use case of UC3 “View Player” as a popup box, invoking the already coded viewFixturesAvailableFor method in the Player class, to view future fixtures for which the player is available – which required use of the Java Calendar class. See Appendix 30, p.96.

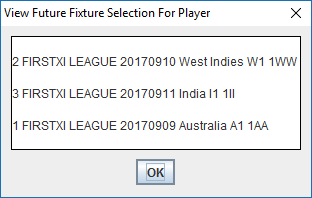
Figure 5 – UC6 “View Fixtures Available For” In Action



## 3.32 Implement/Test UC7 “View Fixtures Selected For”

The only remaining subfunction use case of UC3 that was still to be implemented, as indeed it was after UC20 had been.

Figure 6 – UC7 “View Fixtures Selected For” In Action



## 3.33 Refactor/Test UC12 “Delete Player” and UC21 “Delete Fixture” in Business Layer

Once Player and Fixtures objects started to be referenced by each other (via the “available for” and “selected for” associations), as well as just by the CricketClub object, it became clear that all such references needed to be removed on their deletion, so that the objects could be garbage collected, and no unpredictable system behaviour would occur. The code in the CricketClub class is similar for deleting both players and fixtures:

public void deletePlayer(int playerNumber)

{

for (Player play : this.players.values())

{

if (play.getPlayerNumber() == playerNumber)

{

for (Fixture fixt : this.fixtures.values())

{

if (fixt.getPlayersAvailableFor().contains(play))

{

fixt.getPlayersAvailableFor().remove(play);

}

if (fixt.getPlayersSelectedFor().contains(play))

{

fixt.getPlayersSelectedFor().remove(play);

}

if (fixt.getPlayersPlayedFor().contains(play))

{

fixt.getPlayersPlayedFor().remove(play);

}

}

}

}

this.players.remove(playerNumber);

}

## 3.34 Design UC22 “Add Fixture Results” in User Interface Layer

In the remaining project time available, it was only possible to design UC22 in the user interface, and partially implement its functionality, to record the result of the fixture.

The design combines a familiar look-and-feel regarding both fixtures and players, from other tabbed panes. See Appendix 31, p.97.

The design would allow for the performance of each player selected for the fixture (pre-populated) to be recorded, in the following terms (as per the Business Domain Description):

* Batting (runs, duck or not, out or not out, or did not bat)
* Bowling (wickets taken and runs conceded)
* Wicketkeeping (catches and stumpings)
* Number of outfield catches

This data would then be passed back to the business layer, so that the relevant player and wicketkeeper performance statistics could be updated for that particular team type and fixture type.

## 3.35 Evaluate Second Evolutionary Prototype with Stakeholder

A demonstration of the second prototype was provided to the stakeholder, providing a walkthrough of all the user-goal base use cases that had been implemented with their subfunction use cases, namely:

|  |  |  |
| --- | --- | --- |
| UC13 | *add fixture* | base use case |
| UC14 | *view fixture list* | base use case |
| UC15 | *view fixture* | base use case |
| UC16 | *view players available for fixture* | subfunction use case of UC15 |
| UC17 | *view players selected for fixture* | subfunction use case of UC15 |
| UC18 | *amend fixture* | base use case |
| UC19 | *amend availability of players for specific fixture* | subfunction use case of UC18 |
| UC20 | *select players for fixture* | subfunction use case of UC18 |
| UC21 | *delete fixture* | addition use case of UC18 |
| UC22 | *add fixture results* | base use case |

Again, this requirements-based acceptance testing satisfied the expectations of the stakeholder, in terms of validation and verification.

Usability testing was also performed to evaluate the following non-functional requirements from the Requirements Document (see Appendix 6, p.64):

P2: The system shall respond to a complex user request within 20 seconds. Fit

criterion self-contained.

P3: The system shall be reasonably reliable and available.

Fit criterion: Reliability and availability will continue to be evaluated by the club

captain after the system is first released, with maintenance and support provided

as below.

P4: The system should operate with a very high level of accuracy in relation to all

of its record keeping functional requirements.

Fit criterion: Accuracy will continue to be evaluated by the club captain after the

system is first released, with maintenance and support provided as below.

M1: The system will be maintained and supported after its first release on an

informal and ad hoc basis.

From having used the first prototype for a month since its deployment, the stakeholder was satisfied that it had met the fit criteria for P2, P3 and P4, and would monitor the second prototype for continued adherence to these criteria, with further collaboration to be agreed should maintenance be required.

## 3.36 Deploy/Test Second Evolutionary Prototype with Stakeholder

The Java SE Runtime Environment (JRE 6u45) already having been installed on the stakeholder’s laptop, the “LMCCProjectSecondIteration.jar” Executable Jar File, was uploaded to his laptop and the prototype was successfully started and tested in its operational environment.

# 4 Legal, Social, Ethical and Professional Issues

Not written.

## 4.1 Legal Issues

Not written.

## 4.2 Social Issues

Not written.

## 4.3 Ethical Issues

Not written.

## 4.4 Professional Issues

Not written.

# 5 Review of Current Stage of Project Work

The final project output deployed with the stakeholder realised all of the use cases apart from the full functionality of UC22 “Add Fixtures Results”, as discussed above in Section 3.34, p.34.

So, the next incremental implementation of system functionality would realise UC22, to update the relevant player and wicketkeeper performance statistics (for that fixture’s particular fixture type and team type, e.g. League and 1st XI), for each player that played in that fixture. Based on previous project work, this is estimated at 15 hours, with testing.

The additional construction work needed in the business layer would be to complete the method code of addFixtureResults in the CricketClub class for this user-goal base use case, using existing methods of Player to manipulate the collections of PlayerPerformance and WicketkeeperPerformance objects with which it has composition relationships.

The tabbed pane having already been designed for this (see Appendix 31, p.97), the necessary work in the user interface would be to pre-populate the screen with the players selected for that fixture. Also, to implement the button functionality to update their match statistics (see Figure 7, below) – this would entail substantial code reuse, because the same functionality is required for each of the eleven players, displayed in separate panels. So, all of the text fields and boolean boxes have (laboriously) been named logically within each panel, see Figure 8, below.

Figure 7 – Work to do in the User Interface

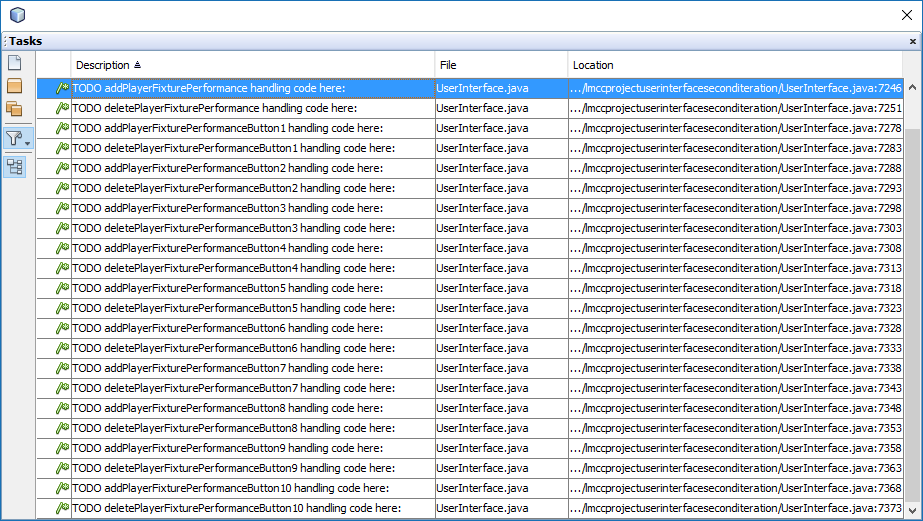
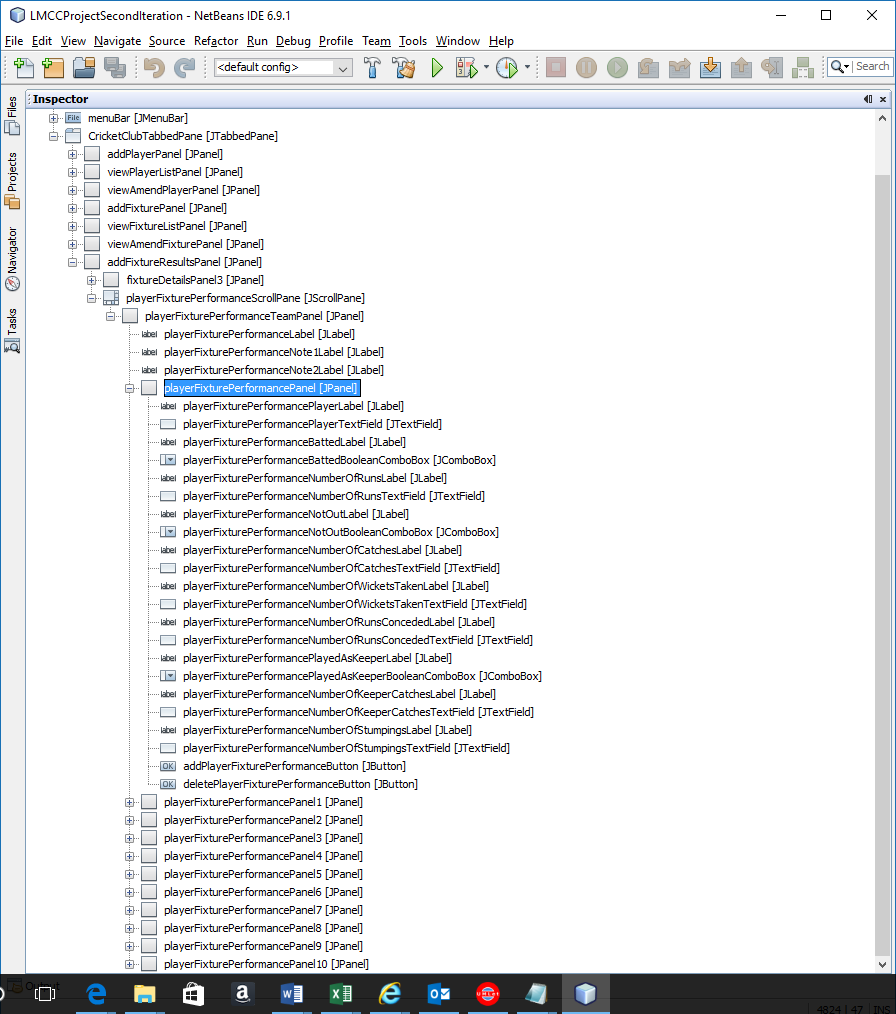


Figure 8 – The “Add Fixture Results” Tabbed Pane Components

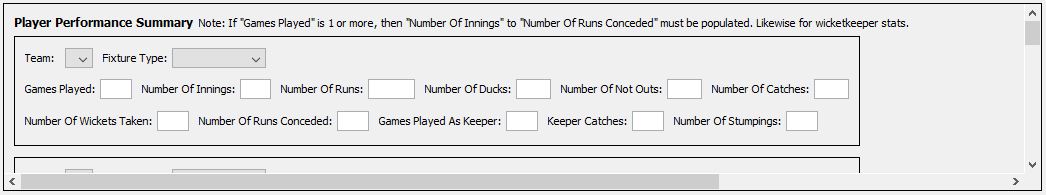


While the stakeholder only needs to store one cricket club’s data (see Section 3.19, p.25), another incremental implementation of system functionality would be to enable the File Menu’s Save As command, to allow different .obj files to be saved. The File Menu’s Open command could also be iteratively improved, to allow a selected .obj file to be loaded. Using the FileChooser functionality of Java (Merx and Norman, 2007, pp.462-466), this is estimated at 3 hours.

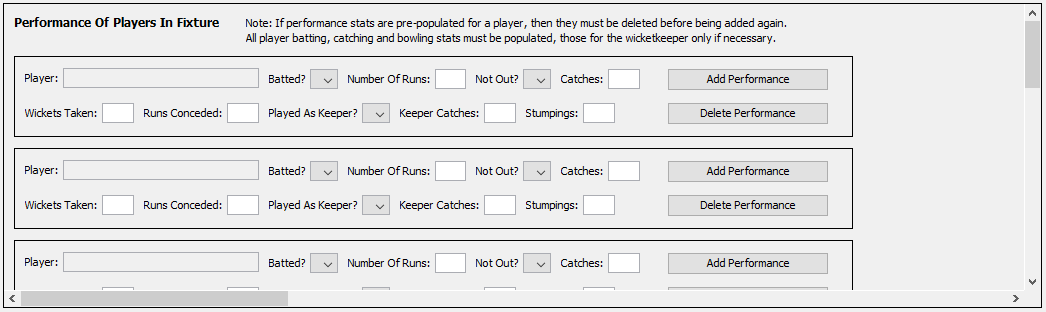
However, the major iterative improvement of system functionality would be to validate user input to a greater extent. Currently, the user interface displays the notices shown in Figure 9, below.

Figure 9 – User Interface Notices

From the “Add Player” tabbed pane:



From the “Add Fixture Results” tabbed pane:



If these notices are not obeyed, then that particular PlayerPerformance or WicketkeeperPerformance object is not updated, likewise if non-integer data is input to the text boxes. Furthermore, invalid combinations of integer data can currently be input. For example, the number of innings can exceed games played, the number of not outs can exceed innings, wickets taken can exceed games played times ten etc. The stakeholder is aware of this and care is taken.

Merx and Norman (2007, pp.337-352) suggest a strategy for using Java regular expressions to validate that user input adheres to the correct format, by using pattern-matching code, otherwise providing dynamically created feedback on errors. This would be a good starting point for research and implementation, but to estimate duration for implementation of this is more difficult.

# 6 Review of Project Management

## 6.1 Project Planning & Management

It is helpful to reiterate the project’s overall aim for this review:

To evaluate, select and effectively apply relevant software development tools, techniques and processes in order to understand and model the business domain of the cricket club and capture the requirements of the stakeholder (the club captain); and in order to design and develop a software system that fulfils the pragmatic needs of the stakeholder to manage information regarding players and fixtures, and record results.

The Iterative Waterfall Lifecycle Model was chosen for the project, for iteration of the analysis, design, implementation and evaluation phases, and using an iterative and incremental software development process based on RUP, with evolutionary prototypes cumulatively creating the eventual system (Dawson, 2009, p.125). Overall, however, attention was focussed on making forward progress to the final project output, and not iterating back unnecessarily, see Section 3.24, p.28.

Dawson (2009, p.119) emphasises the suitability of Waterfall Models for software development projects. Using the Iterative Waterfall Lifecycle Model allowed the project output to evolve in complexity as knowledge increased and greater understanding of the problem and solution emerged over time, both for the stakeholder and the developer.

Applying the Iterative Waterfall Lifecycle Model, and using RUP, provided the following positive aspects for this project:

* The first iteration of Business Modelling and Requirements engaged the stakeholder without overloading him.
* The first iteration of Analysis & Design and Implementation ensured that a system prototype was developed earlier in the project, rather than leaving development to its later stages, confronting head-on a skills gap of converting models to software.
* Evaluating the first evolutionary prototype inspired the stakeholder, when a partially working piece of software was demonstrated, and provided useful feedback for the second iteration.
* The second iteration delivered incremental functionality for the prototyped solution.
* The second iteration increased the quality of the prototyped software.
* Because the functionality intended for implementation in each iteration was carefully chosen, at project closure the stakeholder was provided with a working system, albeit without all possible use cases implemented (Dawson, 2009, p.130).
* Greater flexibility was possible in project planning for rescheduling and reacting to change.

Using RUP disciplines as a framework (Kruchten, 2004, p.34), this review now considers in turn how and whether the project’s practical work achieved the purpose of each of them, in the cricket club context.

Business Modelling is about understanding the current organisational structure and dynamics and identifying problems and potential improvements (Kruchten, 2004, p.141). Accurate modelling of the problem domain took place. The Business Domain Description and Domain Structural Model were created and reviewed with the stakeholder and the collaboration was close, enabling a clear description of the problem.

The purpose of the Requirements discipline is to agree the system’s full functionality (what it should do and why) with the stakeholder, and to define the system’s boundary (Kruchten, 2004, p.157). For this project, the Requirements Document was agreed as a shared reference point to a systematic set of the system’s overall functional requirements. Furthermore, the Use Case Model gives a comprehensive overview, while those user-goal base use cases that were elaborated further (UC1, UC2, UC3, UC8 & UC13) provided an accurate basis for what detailed functionality was to be implemented.

Analysis & Design concerns understanding the system requirements and translating them into a system design which specifies an appropriate implementation strategy (Kruchten, 2004, p.173). Focussing on an object-oriented methodology, the Analysis Structural Model, Design Structural Model and Dynamic Behavioural Models were created for the project at the right level of detail and completeness for effective Java coding. Unnecessary additional sequence diagrams were not created. Time was not wasted.

The Implementation discipline is about organising the code by defining layered subsystems and implementing classes and objects as components. (Kruchten, 2004, p.187). An early choice of Software Architecture was beneficial to coding. Because the process of Initial Software Construction was laborious it would be worthwhile investigating the use of a CASE tool for future projects, to automate the generation of Java source code from class diagrams (Ambler, 2002, pp.284-6). However, CASE tools also require complete accuracy with class diagrams to ensure the right code is generated, so the saving in time is debatable. For this project, it was useful to write the code manually, to ensure a full understanding of the code base and refresh Java skills, for further application of these during use case coding in the business layer and user interface. Techniques of code reuse, refactoring, defensive programming and design by contract were applied effectively.

The purposes of the Testing discipline are to assess the product to find and document defects, evaluate the quality of the software, verifying that the product works as designed, and validating that it meets its requirements (Kruchten, 2004, p.197). Testing and fixing of problems naturally took place throughout coding, but no formal testing technique was employed. Usability testing and requirements-based acceptance testing occurred when the first and second prototypes were evaluated with the stakeholder.

Deployment concerns packaging the software for delivery, distributing and installing it, testing it in its final operational environment, and training the end users (Kruchten, 2004, p.237). This happened when both prototypes were deployed with the stakeholder.

As the project progressed, a shift in project management style occurred. Where, initially, the emphasis was on detailed planning, from previous experience with PRINCE2, in the project’s latter stages a more agile approach was adopted. Both were of benefit. A robust plan at the outset meant that the overall direction of the project for the following thirty weeks was evident, but did hamper flexibility until a change of mindset. Using agile techniques later on meant that the project was able to focus on minimising the product backlog, near its close (Cole and Scotcher, 2015, p.51).

Personal circumstances (registered with the OU), and wholly beyond the control of the project, finally led to the loss of five complete weeks of study, so at least 50 hours of project time was lost. However, the project plan and SDLC were followed as closely possible, and the clear and understandable structure of RUP provided control, during this time. The fact that a clear plan was in place was reassuring, and all of the contingency built into it was used, preventing the project from having to be rescoped.

Table 6 – The Full Project Plan

This is shown below.











## 6.2 Resources, Activities & Skills Needed

Resources

The main technical resource required was the NetBeans Integrated Development Environment (IDE), and the specific version (6.9.1) was that previously used for OU M362 (NetBeans, n.d.). This was already correctly installed and configured for use with the relevant Java SE Development Kit (JDK 6u45) and Java SE Runtime Environment (JRE 6u45), in advance of the project (Oracle Technology Network, n.d. b).

UMLet (version 14.2) was used as the UML diagramming tool (UMLet, n.d.). This was chosen based on its simplicity and ease-of-use, and because of pre-existing familiarity with its functionality, from its recommended use on OU TM354.

The two business domain data resources used were the website of the Wetherby Cricket League (Wetherby Cricket League, n.d.), to help understand the structure of the fixtures in which the club participates, and the existing spreadsheet used to record the match information the stakeholder reports back to the league (see Section 3.1, p,14).

And, of course, the stakeholder himself should be considered as a project resource, in terms of his having been required for information about the business domain, eliciting his requirements for the software, and collaborating with him for useful product feedback during evaluation.

Activities

The overarching process used on the project was the Rational Unified Process (RUP) for iterative and incremental software development (IBM Developer Works, 2005). This provided the overall framework (see Section 6.1, pp.40-1) for the software development activities and techniques that were then used. For a detailed discussion of which, see all of Section 3, pp.12-35.

Skills

The main skill required by the project for which acknowledged improvement was required was that of software construction i.e. how to write the correct code when mapping from UML models to Java, due to the main study of Java programming being in 2008, 9 & 10 for OU M255, M256 and M257 respectively, and because the Java programming required for OU M362 in 2015 mainly concerned concurrency and distribution, while OU TM354 in 2016 was primarily concerned with UML modelling. The nature of doing a part-time degree.

## 6.3 Risk Management

The deficiency in software construction skills was categorised as a people risk, with a high impact to the final project output and with a medium likelihood of occurring. It was addressed as a key focus of the project during the first iteration of the SDLC, by using the Java Tutorials (Oracle Java Documentation, 2015 a) and by much other project research, so that the risk was avoided.

Loss of the stakeholder was categorised as an ongoing people risk with a high impact to the success of the project. However, this risk was avoided, and only a low likelihood of his pulling out pertained throughout the project.

A series of six meetings with him in February, March and April, when the Business Modelling and Requirements work was conducted, cemented his interest in the project. Regular contact was maintained thereafter, during the Analysis & Design and Implementation stages of the project, in order briefly to report progress, while his concrete involvement was less necessary.

His full participation in the evaluation and deployment of the first prototype at the start of August reinforced his interest, meaning that the final deployment of the second prototype at the beginning of September was successful.

# 7 Review of Personal Development

Overall, I have found the experiential learning process of conducting an individual student project to be positive in a number of ways, and most useful in respect of the improvement of the more academic and technical skill sets, rather than those of a professional nature, where more consolidation of pre-existing skills occurred.

Looking back on it, I can see just how valuable the experience of working and learning independently on the project has been. Delivering a bug-free piece of software of 4,600 lines of manual code (NetBeans GUI Builder generated code not included) –

which actually works – as the final project output has really boosted my confidence that I can be a competent software developer, as I look to change career. Moreover, the fact that this was possible without any technical assistance, and purely through my own research and initiative, has reinforced the sense of achievement.

I became better at working independently as the project progressed. In the first month of the project, I stuck with what I knew (i.e. planning, as a qualified PRINCE2 Practitioner), and while this was useful in itself, I had then to adjust to the fact that this alone would not get the job done on an individual project. So, I forced myself to focus much more on the research and technical aspects of the project. In so doing, and in not being directed in what to do, the lessons learned have become more embedded.

As I initially undertook the literature survey, it became clear (initially from Dawson (2009), Whitehead (2001), and Kruchten (2004)) from an academic and practical perspective, what was required successfully to undertake a student software development project. Defining a clear aim and objectives, how to start up and plan a project, what needs to be done to achieve a quality output. Had these sources been studied a little earlier, and the knowledge applied, project management at the start of the project would have gone better.

Once the project had started on a firmer footing, the analysis of a practical problem went particularly well. This was because I had chosen a business domain I was interested in, because I was able to apply the high level of knowledge I had attained with OU TM354 to the problem, and because liaison with clients is a fundamental part of my professional life. Working with the stakeholder was a highlight of the project.

I have described above (in Section 6.1, p.41) how RUP was the overarching software development process used on the project, and discussed how well the project’s practical work and intermediate outputs at all stages of the SDLC fulfilled the purposes of the RUP disciplines. Performing two complete iterations of the SDLC has given me a lot of practical experience about the overall process.

I have also described (in Section 3, pp.15-18) how skills of object-oriented analysis and design were applied to creating useful models of the software system, and how agile modelling improved the effectiveness and efficiency of this work. Although my skills in this respect were already good from OU TM354, they have been further enhanced by practice.

I have discussed in detail (in Section 3, pp.20-26 and pp.30-34) the software construction, Java coding and GUI building activities that were required. It is fair to say that this is the area where my skills were most in need of refreshing, where the greatest advances have been made, and where I have gained most satisfaction from doing the project, by applying all of my research to actually doing something very practical.

While my skills of technical writing were already reasonably advanced from previous OU study, they have been honed regarding their specificity, in terms of describing how to apply knowledge of a software development technique (across the SDLC), or coding strategy, to a particular context.

My skills of academic report writing have been enhanced significantly over the course of the project, and there has been a cumulative improvement in the quality of the three TMAs. Greater coherency of structure, conciseness of content, extensive and accurate academic referencing, sectional cross-referencing, and the use of the passive voice for scientific detachment (everywhere else in this report apart from in this section on personal development), have been due to the interaction with my tutor and to then putting his constructive feedback into practice, see Appendix 32, p.99. I am grateful to him for his assistance.

I have already alluded to personal circumstances (registered with the OU), and wholly beyond the control of the project which led to the loss of at least 50 hours of project time. I nevertheless managed to maintain regular contact with my tutor and he was supportive.

It would have been interesting to be able to study in detail Legal, Social, Ethical and Professional Issues in relation to this project, especially given the fact that I have previously studied moral philosophy at undergraduate level, and law for a professional qualification, but personal time constraints meant that something had to give on the project, as all of its contingency was used. LSEPIs were therefore, unfortunately, neglected as they were not on the critical path for completion of the project. What I did learn about professional issues concerned applying the principles of the Manifesto For Agile Software Development to the project’s practical work (Manifesto For Agile Software Development, 2001).

Future work which I would like to carry out for the software would be to turn it into a distributed system so that players are able to interact with it via an Android app, providing their availability for future fixtures and being able to see the future fixtures for which they have been selected. This would require revision and enhancement of all of the knowledge I have about implementing distributed systems in Java, using Remote Method Invocation, when I studied OU M362 in 2015.

The project and this report are dedicated to the memory of my mother, Valerie Biswell, who passed away on 30th July 2017.

11,760 words (not including Table of Contents, References, Bibliography and Appendices).

# 8 References

Ambler, S. (2002) *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, New York, NY, United States, John Wiley & Sons.

Barnes, D. (2000) *Object-Oriented Programming with Java: An Introduction*, Upper Saddle River, NJ, United States, Prentice-Hall.

Cole, R. and Scotcher, E. (2015) *Brilliant Agile Project Management*, Harlow, England, Pearson Education.

Dawson, C. (2009) *Projects in Computing and Information Systems*, 2nd edition, Harlow, England, Pearson Education.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design Patterns: Elements of Reusable Object­Oriented Software*, Upper Saddle River, NJ, United States, Addison­Wesley.

Hunt, A. and Thomas,D. (2000) *The Pragmatic Programmer: From Journeyman to Master*, Upper Saddle River, NJ, United States, Addison-Wesley.

IBM Developer Works (2005) *Rational Unified Process: Best Practices for Software Development Teams* [Online]. Available at <https://www.ibm.com/developerworks/rational/library/253.html> (Accessed 3rd September 2017).

Kruchten, P. (2004) *The Rational Unified Process: An Introduction*, 3rd edition, Upper Saddle River, NJ, United States, Addison Wesley.

Larman, C. (2005) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd edition, Upper Saddle River, NJ, United States, Addison Wesley.

Manifesto For Agile Software Development (2001) *Manifesto for Agile Software Development* [Online]. Available at <http://agilemanifesto.org/> (Accessed 3rd September 2017).

Merx, G. and Norman, R. (2007) *Unified Software Engineering with Java*, Upper Saddle River, NJ, United States, Pearson Prentice Hall.

NetBeans (n.d.) *NetBeans IDE 6.9.1 Download* [Online]. Available at <https://netbeans.org/downloads/6.9.1> (Accessed 3rd September 2017).

OMG (2017) *Object Constraint Language* [Online]. Available at <http://www.omg.org/spec/OCL/> (Accessed 4th September 2017).

The Open University (2014 a) *Module TM354; Unit 2: Requirements Concepts; Section 6: Representing the Requirements*, Milton Keynes, England, The Open University.

The Open University (2014 b) *Module TM354; Unit 7: More on Dynamic Modelling – States and Activities; Section 3: Design Issues*, Milton Keynes, England, The Open University.

The Open University (2007) *Module M256; Unit 11: Human-Computer Interaction; Section 4: Introduction to User Interface Design*, Milton Keynes, England, The Open University.

Oracle Java Documentation (2015 a) *The Java Tutorials* [Online]. Available at <http://docs.oracle.com/javase/tutorial/> (Accessed 3rd September 2017).

Oracle Java Documentation (2015 b) *How to Make Dialogs* [Online]. Available at <http://docs.oracle.com/javase/tutorial/uiswing/components/dialog.html> (Accessed 3rd September 2017).

Oracle Technology Network (n.d. a) *Oracle Technology Network for Java Developers* [Online]. Available at <http://www.oracle.com/technetwork/java/index.html> (Accessed 3rd September 2017).

Oracle Technology Network (n.d. b) *Java Archive Downloads - Java SE 6* [Online]. Available at <http://www.oracle.com/technetwork/java/javase/downloads/java-archive-downloads-javase6-419409.html> (Accessed 3rd September 2017).

Riley, D. (2003) *The Object of Java: Introduction to Java Programming Using Software Engineering Principles*, Upper Saddle River, NJ, United States, Addison-Wesley.

UMLet (n.d.) *UMLet 14.2 – Free UML Tool for Fast UML Diagrams* [Online]. Available at <http://umlet.com/> (Accessed 8th September 2017).

Wetherby Cricket League (n.d.) *Wetherby Cricket League* [Online]. Available at <http://wetherbyleague.co.uk/home.html> (Accessed 4th September 2017).

Whitehead, R. (2001) *Leading a Software Development Team: A Developer’s Guide to Successfully Leading People & Projects*, Harlow, England, Pearson Education.

# 9 Bibliography

## 9.1 Project Management

Cole, R. and Scotcher, E. (2015) *Brilliant Agile Project Management*, Harlow, England, Pearson Education.

Dawson, C. (2009) *Projects in Computing and Information Systems*, 2nd edition, Harlow, England, Pearson Education.

Manifesto For Agile Software Development (2001) *Manifesto for Agile Software Development* [Online]. Available at <http://agilemanifesto.org/> (Accessed 3rd September 2017).

Murray, R. (2002) *How to Write a Thesis*, Maidenhead, England, Open University Press.

Schön, D. (1991) *The Reflective Practitioner: How Professionals Think in Action*, Farnham, England, Ashgate Publishing Group.

The Stationery Office (2009) *Managing Successful Projects with PRINCE2*, 5th edition, Norwich, England, The Stationery Office.

Whitehead, R. (2001) *Leading a Software Development Team: A Developer’s Guide to Successfully Leading People & Projects*, Harlow, England, Pearson Education.

## 9.2 Software Development

Kruchten, P. (2004) *The Rational Unified Process: An Introduction*, 3rd edition, Upper Saddle River, NJ, United States, Addison Wesley.

Merx, G. and Norman, R. (2007) *Unified Software Engineering with Java*, Upper Saddle River, NJ, United States, Pearson Prentice Hall.

Sommerville, I. (2007) *Software Engineering*, 8th edition, Harlow, England, Pearson Education.

## 9.3 UML & Modelling

Ambler, S. (2002) *Agile Modeling: Effective Practices for Extreme Programming and the Unified Process*, New York, NY, United States, John Wiley & Sons.

Booch, G., Rumbaugh, J. and Jacobson, I. (2005) *The Unified Modelling Language User Guide*, 2nd edition, Upper Saddle River, NJ, United States, Addison Wesley.

Gamma, E., Helm, R., Johnson, R. and Vlissides, J. (1995) *Design Patterns: Elements of Reusable Object­Oriented Software*, Upper Saddle River, NJ, United States, Addison­Wesley.

Larman, C. (2005) *Applying UML and Patterns: An Introduction to Object-Oriented Analysis and Design and Iterative Development*, 3rd edition, Upper Saddle River, NJ, United States, Addison Wesley.

Maciaszek, L.A. (2005) *Requirements Analysis and System Design*, 2nd edition, Harlow, England, Pearson Education.

## 9.4 Java Programming

Barnes, D. (2000) *Object-Oriented Programming with Java: An Introduction*, Upper Saddle River, NJ, United States, Prentice-Hall.

Barnes, D. and Kölling, M. (2005) *Objects First with Java: A Practical Introduction Using BlueJ*, 2nd edition, Harlow, England, Pearson Education.

Java Platform SE 6 API Specification (2015) *Java Platform Standard Edition 6 API Specification* [Online]. Available at <http://docs.oracle.com/javase/6/docs/api/> (Accessed 3rd September 2017).

Flanagan, D. (2002) *Java in a Nutshell: A Desktop Quick Reference*, 4th edition, Sebastopol, CA, United States, O’Reilly & Associates.

Flanagan, D., Farley, J., Crawford, W. and Magnusson, K. (1999) *Java Enterprise in a Nutshell: A Desktop Quick Reference*, Sebastopol, CA, United States, O’Reilly & Associates.

Hunt, A. and Thomas,D. (2000) *The Pragmatic Programmer: From Journeyman to Master*, Upper Saddle River, NJ, United States, Addison-Wesley.

Oracle Java Documentation (2015) *The Java Tutorials* [Online]. Available at <http://docs.oracle.com/javase/tutorial/> (Accessed 3rd September 2017).

Riley, D. (2003) *The Object of Java: Introduction to Java Programming Using Software Engineering Principles*, Upper Saddle River, NJ, United States, Addison-Wesley.

## 9.5 Software Testing

Laboon, B. (2016) *A Friendly Introduction to Software Testing*, North Charleston, SC, United States, CreateSpace Publishing.

Richardson, A. (2015) *Java for Testers: Learn Java Fundamentals Fast*, St Albans, England, Compendium Developments.

## 9.6 Legal, Social, Ethical and Professional Issues in Computing

Ermann, D., Williams, M. and Shauf, M. (1997) *Computers, Ethics, and Society*, 2nd edition, Oxford, England, Oxford University Press.

# 10 Appendices

Each appendix is cross-referenced with the Section(s) that refers to it.

## 10.1 Appendix 1 – Business Domain Description

See Section 1, p.5. See Section 3.1, p.14.

***Long Marston Cricket Club Business Domain Description – 06/03/2017***

Based in Yorkshire, Long Marston Cricket Club fields both a 1st XI and a 2nd XI team in the Wetherby Cricket League, which consists of four divisions.

The current method of record-keeping for players and fixtures is based on a spreadsheet and the Wetherby Cricket League website is used as the overall framework to keep track of the fixtures. Player future availability is currently captured and recorded by a variety of means: text messages, face-to-face and telephone conversations, emails, paper etc.

The club consists of players, as well as the roles of President, Chairman, Secretary, Treasurer, 1st XI Captain, 1st XI Vice Captain, 2nd XI Captain and 2nd XI Vice Captain.

The League’s fixtures for the 2017 season take place every Saturday starting on 22nd April and finishing on 16th September, with the 1st XI to play in Division 2, and the 2nd XI in Division 3. Both of these divisions consist of 12 teams. Each fixture starts at 1pm.

As in previous seasons, both of Long Marston’s teams play 22 fixtures (home and away against each side in the same division) in the 2017 season, with one team always playing away each Saturday and the other playing at home. Each fixture consists of a 45 over innings per side – the winning side being that to score the most runs.

During the 2017 season, the cricket club will also participate in three knockout cup competitions, also organised by the Wetherby Cricket League:

The 1st XI will play in the Fred Fleetwood Cup, with the four rounds taking place on Sundays (14th May, 18th June, 9th July & 6th August). Each round consists of a 40 over innings per side. Each fixture starts at 1pm.

The 1st XI will also play in the Senior Evening Cup, with three of the rounds taking place at 6.30pm on a Tuesday (13th June, 4th July & 25th July), and the final (round) taking place on Sunday 20th August at 1pm. Each round consists of a 20 over innings per side.

The 2nd XI will play in the Reserve Evening Cup, with three of the rounds taking place at 6.30pm on a Tuesday (30th May, 27th June & 18th July), and the final (round) taking place on Sunday 20th August at 1pm. Each round consists of a 20 over innings per side.

So, there is never any clash of fixtures for the 1st XI or 2nd XI teams, looking at each one in isolation. However, both teams play every Saturday during the 2017 season, and both teams could potentially play on the same day in the Evening Cups’ combined Finals Day on Sunday 20th August.

Information that should be recorded about players is as follows:

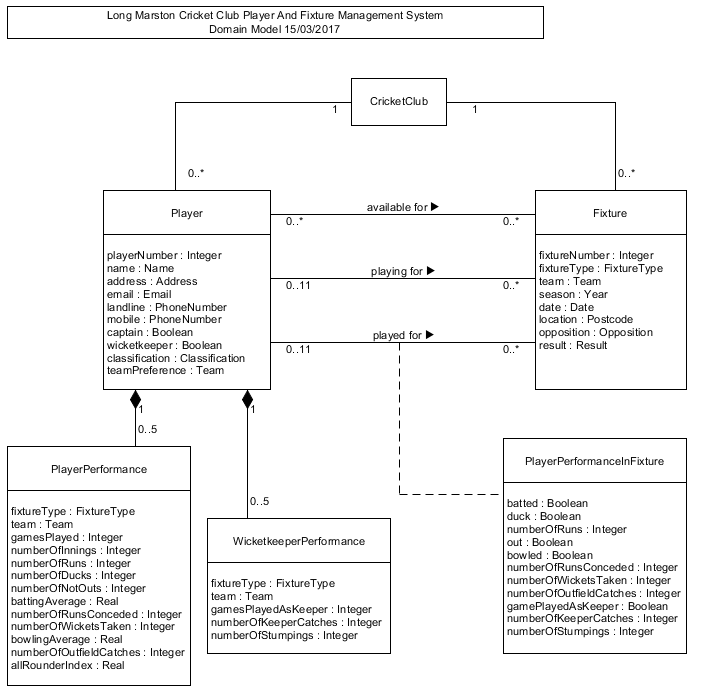
* Name
* Email address
* Physical address
* Landline and Mobile number
* Future availability for fixtures
* The fixtures that the player has played in and his performance therein
* If the player can act as captain
* If the player can act as wicketkeeper
* If the player is classified as a batsman, bowler, all-rounder, or wicketkeeper
* Whether the player prefers to play in the 1st XI or 2nd XI
* Number of games played (separately, for 1st XI and for 2nd XI, and for League, Sunday Cup, or Evening Cup games)
* Number of innings batted (ibid)
* Number of runs scored while batting (ibid)
* Number of “ducks” while batting (ibid)
* Number of “not outs” while batting (ibid)
* Batting average (ibid)
* Number of runs conceded while bowling (ibid)
* Number of wickets taken while bowling (ibid)
* Bowling average (ibid)
* Number of outfield catches (ibid)
* Number of catches made as wicketkeeper (ibid)
* Number of stumpings made as wicketkeeper (ibid)
* All-rounder index (ibid). Calculated by dividing batting average by bowling average.

Information that should be recorded about fixtures is as follows:

* Whether the fixture is for the 1st XI or 2nd XI
* Whether the fixture is a League, Sunday Cup, or Evening Cup game
* Year of season
* Date of fixture
* Postcode of location of fixture
* Name of opposing side
* The players that are playing for each fixture
* Result of the fixture once it is completed (win, loss, tie, abandoned before starting, abandoned after starting, concession by either side)
* Once the fixture is completed, the performance of each player in terms of:
* Batting (runs, duck or not, out or not out, or did not bat)
* Bowling (wickets taken and runs conceded)
* Wicketkeeping (catches and stumpings)
* Number of outfield catches

## 10.2 Appendix 2 – Domain Structural Model

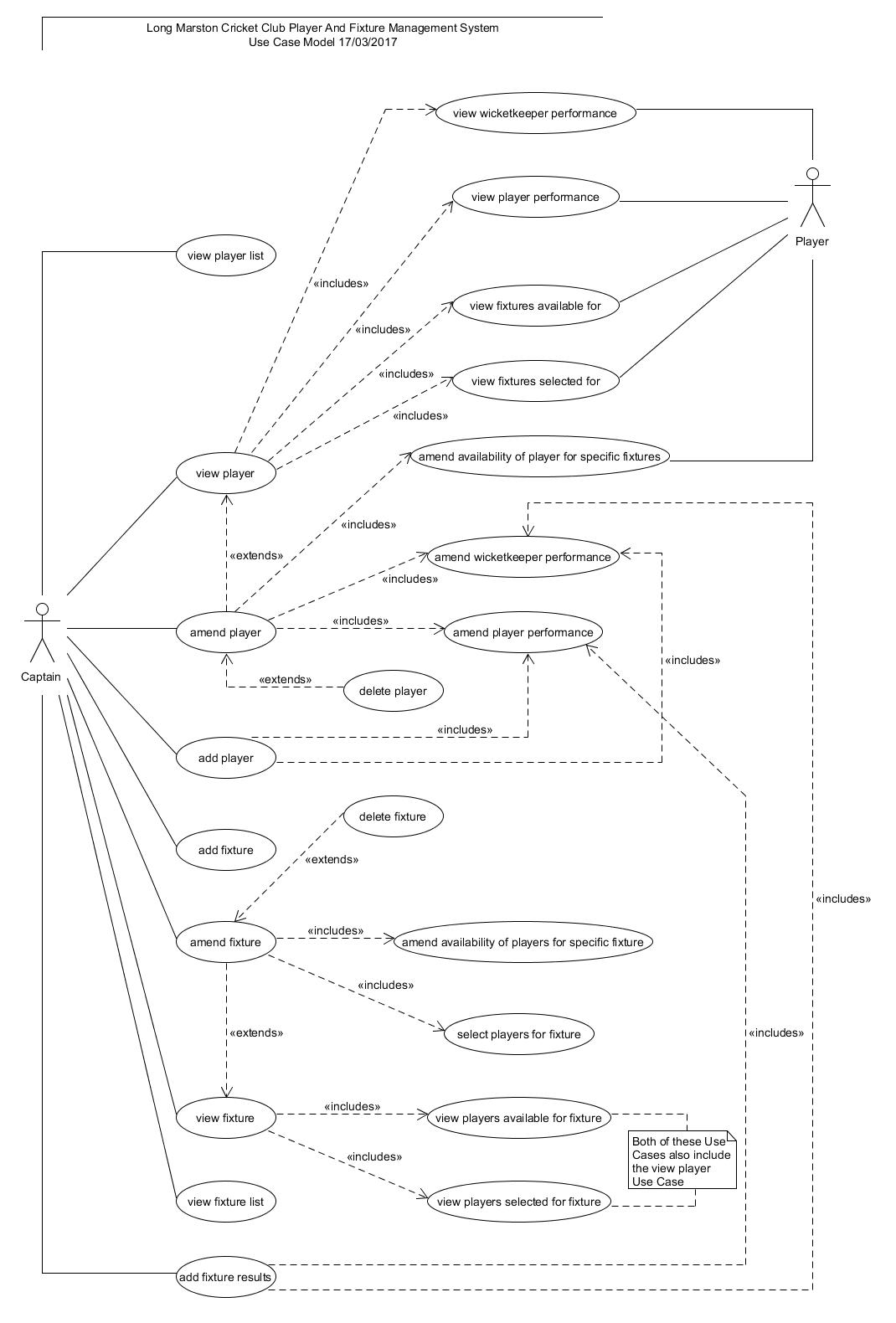
See Section 1, p.5. See Section 3.2, p.14.



## 10.3 Appendix 3 – Use Case Model

See Section 1, p.5. See Section 3.3, p.15.

See next page for model as a full page is required.



## 10.4 Appendix 4 – UC1 “Add Player” Use Case

See Section 3.5, p.16. See Section 3.11, p.18.

|  |  |
| --- | --- |
| Identifier & Name | UC1 *add player* |
|
| Initiator | *Captain* |
|
| Goal | A player is created on the system and all of the required information is recorded. |
|
| Precondition | None |
|
| Postcondition | The player will have been created on the system and allocated a unique player number. All of the personal contact information and summary player classification information, and, where necessary, player and wicketkeeper performance information, will have been recorded. |
|
| Assumptions | The expected initiator is the captain. |
|
| Main Success Scenario | 1/ The captain requests to add a player. |
|
| 2/ The system requests the player's personal contact and summary player classification information, prompting what default value should be provided if a particular piece of information is not known. |
|
| 3/ The captain provides the required information. |
|
| 4/ The system creates a player and allocates a unique player number. |
|
| 5/ The system requests whether player performance is to be recorded. |
|
| 6/ The captain confirms this is the case. |
|
| 7/ The system requests for what teams/fixture types player performance is to be recorded. |
|
| 8/ The captain confirms these details. |
|
| 9/ UC9 *amend player performance* is initiated, the appropriate number of times. |
|
| 10/ The system requests whether wicketkeeper performance is to be recorded. |
|
| 11/ The captain confirms this is the case. |
|
| 12/ The system requests for what teams/fixture types wicketkeeper performance is to be recorded. |
|
| 13/ The captain confirms these details. |
|
| 14/ UC10 *amend wicketkeeper performance* is initiated, the appropriate number of times. |
|
| 15/ The system confirms that the player has been added to the system. |
|
| Extensions | 6.a.1/ The captain confirms that this is not the case and the scenario continues at step 15. |
|
| 11.a.1/ The captain confirms that this is not the case and the scenario continues at step 15. |
|

## 10.5 Appendix 5 – UC8 “Amend Player” Use Case

See Section 3.5, p.16. See Section 3.11, p.18.

|  |  |
| --- | --- |
| Identifier & Name | UC8 *amend player* |
|
| Initiator | *Captain* |
|
| Goal | Any or all of the information pertaining to a player can be amended. |
|
| Precondition | A player has already been created on the system. |
|
| Postcondition | The personal contact information, summary player classification information, player or wicketkeeper performance information, or fixture availability information for a specific player can be amended. |
|
| Assumptions | The expected initiator is the captain. |
|
| Main Success Scenario | 1/ Personal contact information can be amended. |
|
| 2/ Summary player classification information can be amended. |
|
| 3/ UC9 *amend player performance* can be initiated. |
|
| 4/ UC10 *amend wicketkeeper performance* can be initiated. |
|
| 5/ UC11 *amend availability of player for specific fixtures* can be initiated. |
|
| Extensions | None |
|

## 10.6 Appendix 6 – Requirements Document - Volere Template

See Section 1, p.5. See Section 3.6, p.16. See Section 3.21, p.26. See Section 3.35, p.34.

***Long Marston Cricket Club System User Requirements Volere Template – 07/04/2017***

**Product constraints**

1. *The purpose of the project.* Long Marston Cricket Club wants to move from an existing ad hoc system to a new computerised system in order to administer certain aspects of its operations.

The club captain requires a system which will provide an improved method for him to manage players and fixtures, and record results. The main benefits of a successful system will be as follows:

* + A measurable reduction in the time taken by the club captain to manage player and fixture information
  + A quantitative improvement in the accuracy and comprehensiveness of the information that is managed by the club captain
  + A qualitative improvement in the experience of the players who collaborate with the club captain concerning such information

1. *The stakeholders.* The customer is Long Marston Cricket Club. The main stakeholders are the club’s captain and its players. The users of the new system are the club’s captain and its players.
2. *Mandated constraints.* The first release of the system should be completed by August 2017.
3. *Naming conventions and definitions.* All domain-specific terms used so far in communications with the club captain have been precisely and completely understood.
4. *Relevant facts and assumptions.* None pertaining at present.

**Functional requirements**

1. *Scope of the work.* The new system will be developed just for the cricket club.
2. *Business data model and data dictionary.* A description of the data that needs to be manipulated has been captured and accurately recorded with the writing of the Business Domain Description and specified correctly in the Domain Structural Model which has been created.
3. *Scope of the product.* Within the context of the scope of the work determined above, the boundaries of the product have been defined in the Use Case Model, which represents the users of the product and how they interact with the specified use cases.
4. *Functional requirements.* The new system shall:

* F1: Manage information for (i.e. allowing for information to be added, viewed, amended and deleted) all Long Marston Cricket Club players.
* F2: Manage information for (i.e. ibid) all fixtures that are scheduled for the club’s two teams, the 1st XI and 2nd XI.
* F3: Manage information for the availability of players for scheduled fixtures.
* F4: Manage information for the selection of players for scheduled fixtures.
* F5: Manage information for the result of each completed fixture, and for each player’s performance therein.

While the foregoing summarises the overall functionality of the system, further detail regarding its functional requirements is recorded in each of the concrete user-goal level base use cases as they are elaborated in more detail.

**Non-functional requirements**

1. *Look-and-feel requirements.*

LF1: The system does not require a colourful and pretty appearance, but rather should appear functional and uncomplicated.

Fit criterion: System appearance will be evaluated by the club captain.

1. *Usability and humanity requirements.*

U1: The system shall be easy to learn for the club captain.

Fit criterion: The club captain (who is very comfortable with IT) should be able

to learn to use 90 percent of the functionality of the system (relevant to him) in 2

hours.

U2: The system shall be easy to learn for the players.

Fit criterion: The players (whose ability with IT is unknown and variable) should

be able to use 90 percent of the functionality of the system (relevant to them) in 2

hours.

1. *Performance requirements.*

P1: The system shall respond to most user input within 5 seconds. Fit criterion

self-contained.

P2: The system shall respond to a complex user request within 20 seconds. Fit

criterion self-contained.

P3: The system shall be reasonably reliable and available.

Fit criterion: Reliability and availability will continue to be evaluated by the club

captain after the system is first released, with maintenance and support provided

as below.

P4: The system should operate with a very high level of accuracy in relation to all

of its record keeping functional requirements.

Fit criterion: Accuracy will continue to be evaluated by the club captain after the

system is first released, with maintenance and support provided as below.

1. *Operational and environmental requirements.*

O1: The system shall operate in a standard domestic environment.

1. *Maintainability and support requirements.*

M1: The system will be maintained and supported after its first release on an

informal and ad hoc basis.

1. *Security requirements.*

S1: Only the club captain shall be able to perform those operations specific to

him, as shown in the Use Case Model. Fit criterion self-contained.

S2: Personal information (name, address, email, landline, mobile) relating to the

players shall be securely managed by the system, so that this data is only

available to those who are authorised to access it.

Fit criterion: No breaches shall occur.

Should the system become a distributed computing system, then further security

requirements need to be considered.

1. *Cultural requirements.*

C1: The language required for the interface is English, with this being clear and

straightforward. Fit criterion self-contained.

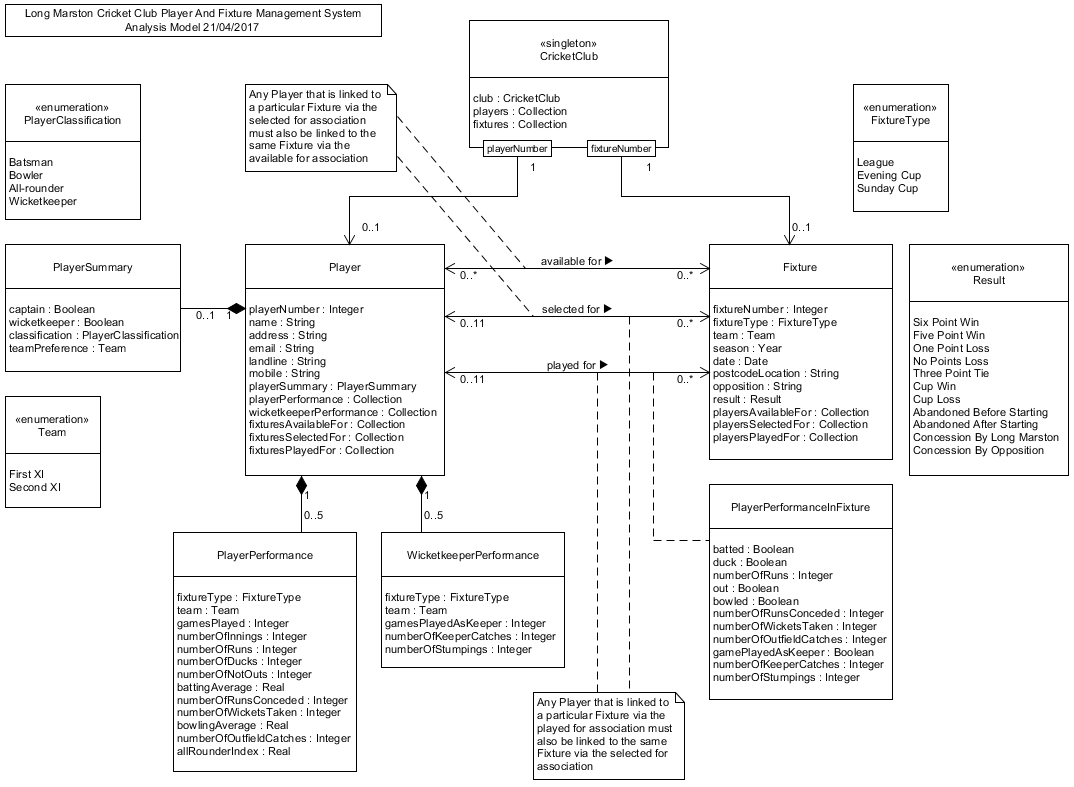
1. *Legal requirements.*

L1: The system shall comply with European and UK law.

## 10.7 Appendix 7 – Analysis Structural Model

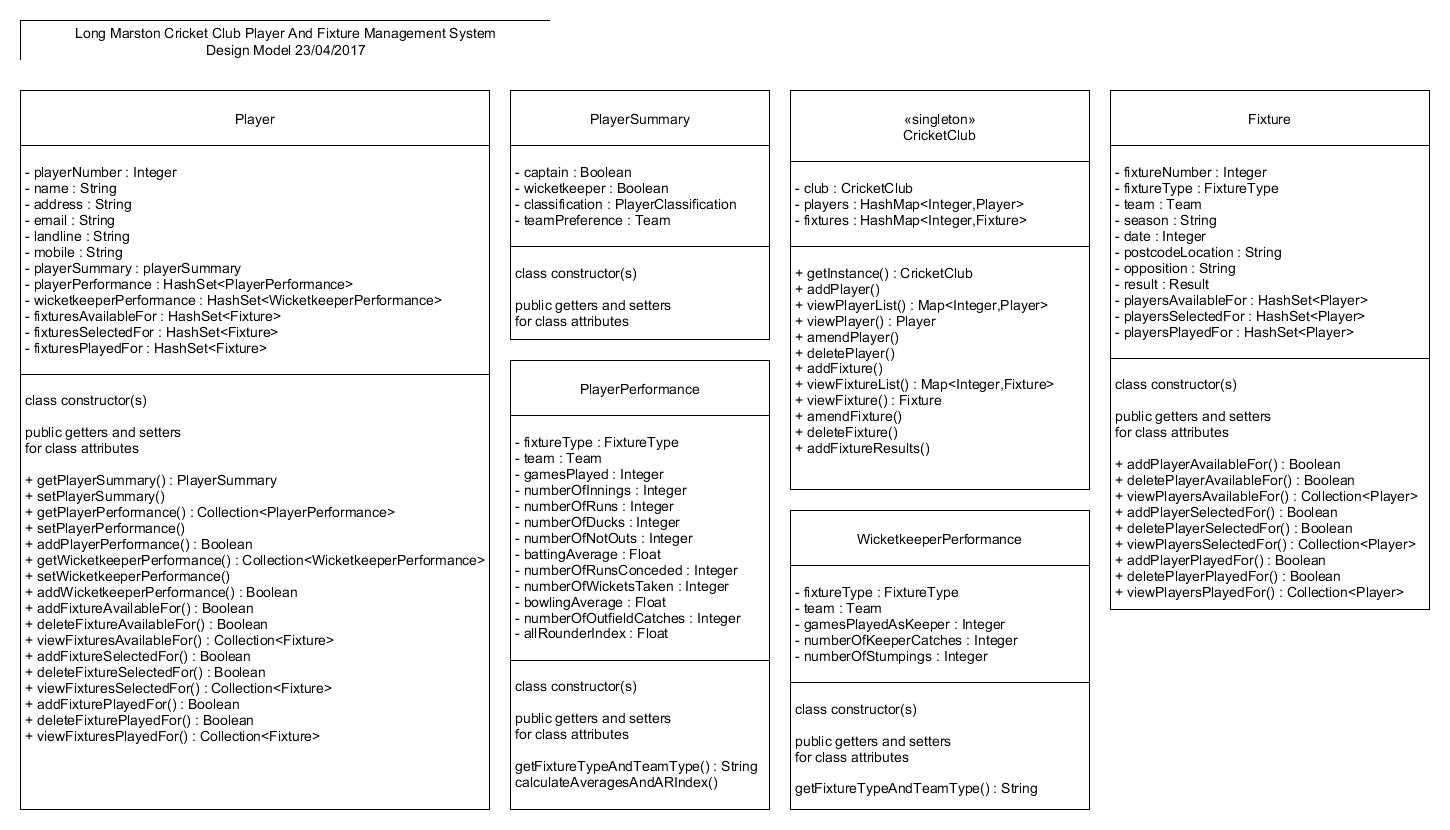
See Section 3.9, p.17.

See next page for model as a full page is required.



## 10.8 Appendix 8 – Design Structural Model

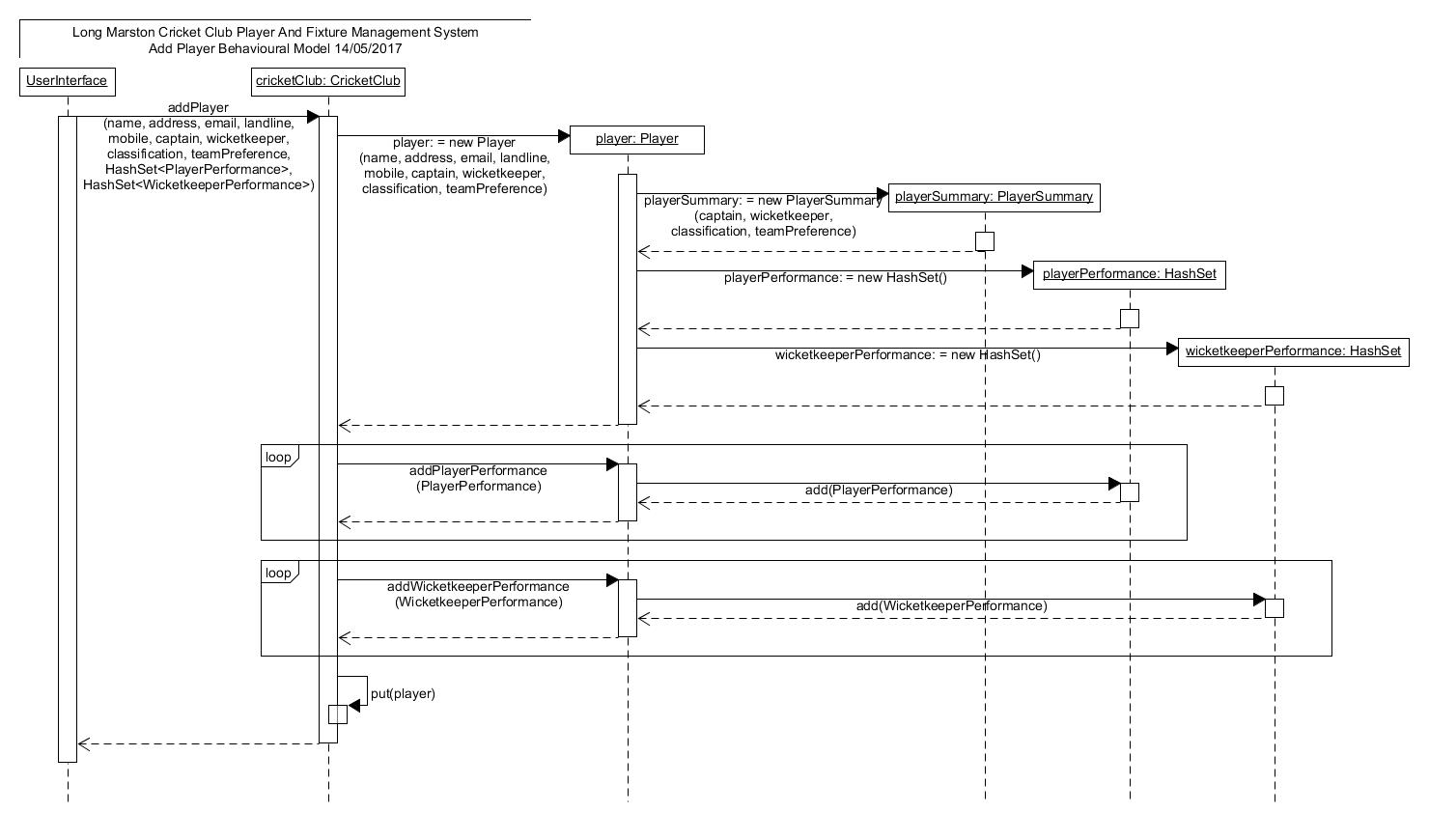
See Section 3.10, p.18.



## 10.9 Appendix 9 – UC1 “Add Player” Sequence Diagram

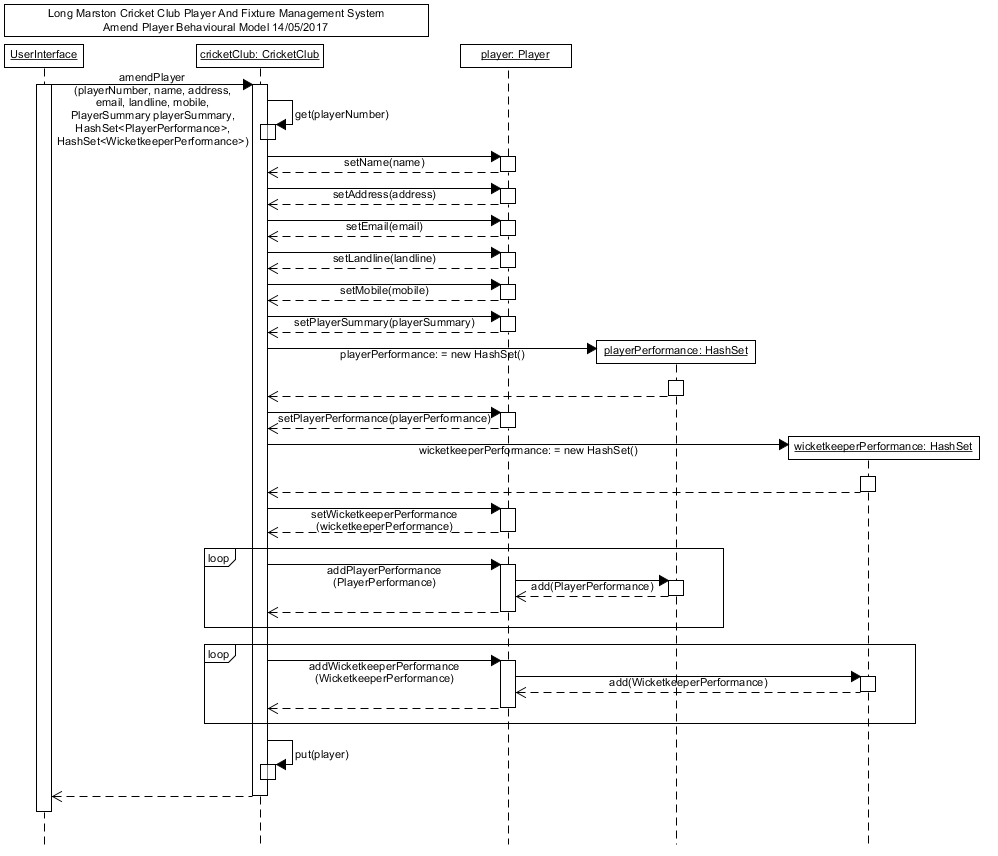
See Section 3.11, p.18. See Section 3.14, p.23.

See next page for model as a full page is required.



## 10.10 Appendix 10 – UC8 “Amend Player” Sequence Diagram

See Section 3.11, p.18.



## 10.11 Appendix 11 – Player Class Constructor

See Section 3.13, p.20.

public class Player implements Serializable

{

private static int numberOfPlayers = 0;

private int playerNumber;

private String name;

private String address;

private String email;

private String landline;

private String mobile;

private PlayerSummary playerSummary;

private Collection<PlayerPerformance> playerPerformance;

private Collection<WicketkeeperPerformance> wicketkeeperPerformance;

private Collection<Fixture> fixturesAvailableFor;

private Collection<Fixture> fixturesSelectedFor;

private Collection<Fixture> fixturesPlayedFor;

public Player(String name, String address, String email,

String landline, String mobile, boolean captain,

boolean wicketkeeper, PlayerClassification classification,

Team teamPreference)

{

numberOfPlayers++;

this.playerNumber = numberOfPlayers;

this.name = name;

this.address = address;

this.email = email;

this.landline = landline;

this.mobile = mobile;

this.playerSummary = new PlayerSummary(captain, wicketkeeper,

classification, teamPreference);

this.playerPerformance = new HashSet<PlayerPerformance>();

this.wicketkeeperPerformance = new HashSet<WicketkeeperPerformance>();

this.fixturesAvailableFor = new HashSet<Fixture>();

this.fixturesSelectedFor = new HashSet<Fixture>();

this.fixturesPlayedFor = new HashSet<Fixture>();

}

. . .

. . .

. . .

}

## 10.12 Appendix 12 – PlayerPerformance Class Helper Method

See Section 3.13, p.20.

/\*\*

\* Calculate averages and all-rounder index helper method

\* for setNumberOfInnings, setNumberOfNotOuts, setNumberOfRuns,

\* setNumberOfRunsConceded, SetNumberOfWicketsTaken

\* and class constructor

\*/

private void calculateAveragesAndARIndex()

{

// batting average is calculated if there are some runs

// and some innings where the player is out

if ((this.numberOfRuns > 0)

&& ((this.numberOfInnings - this.numberOfNotOuts) > 0))

{

this.battingAverage = ((float) this.numberOfRuns

/ (float) (this.numberOfInnings - this.numberOfNotOuts));

}

// bowling average is calculated if there are some runs

// conceded and some wickets taken

if ((this.numberOfRunsConceded > 0)

&& (this.numberOfWicketsTaken > 0))

{

this.bowlingAverage = ((float) this.numberOfRunsConceded

/ (float) this.numberOfWicketsTaken);

}

// all-rounder index is calculated if there is both

// a batting average and a bowling average

if ((this.battingAverage > 0)

&& (this.bowlingAverage > 0))

{

this.allRounderIndex = (this.battingAverage

/ this.bowlingAverage);

}

}

## 10.13 Appendix 13 – Implementing the “Played For” Association

See Section 3.13, p.20.

Method in Player class:

/\*\*

\* Add fixture played for

\*

\* @param fixture - the fixture for which the player played

\*

\* @return whether or not it is added

\*/

public boolean addFixturePlayedFor(Fixture fixture)

{

if (this.fixturesSelectedFor.contains(fixture)

&& !(this.fixturesPlayedFor.contains(fixture)))

{

this.fixturesPlayedFor.add(fixture);

if (!(fixture.getPlayersPlayedFor().contains(this)))

{

fixture.addPlayerPlayedFor(this);

}

return true;

}

return false;

}

Method in Fixture class:

/\*\*

\* Add player played for

\*

\* @param player - the player who played for the fixture

\*

\* @return whether or not he is added

\*/

public boolean addPlayerPlayedFor(Player player)

{

if ((this.playersSelectedFor.contains(player))

&& (this.playersPlayedFor.size() < 11)

&& !(this.playersPlayedFor.contains(player)))

{

this.playersPlayedFor.add(player);

if (!(player.getFixturesPlayedFor().contains(this)))

{

player.addFixturePlayedFor(this);

}

return true;

}

return false;

}

## 10.14 Appendix 14 – CricketClub Class addPlayer Method

See Section 3.14, p.23. See Section 3.16, p.24.

/\*\*

\* Add a player to the system

\*

\* @param name - name of player

\* @param address - home address of player

\* @param email - email address of player

\* @param landline - landline number of player

\* @param mobile - mobile number of player

\* @param captain - whether or not the player can act as captain

\* @param wicketkeeper - whether or not the player can act as wicketkeeper

\* @param classification - the classification of player (batsman, bowler

\* etc.)

\* @param teamPreference - the team (1st XI or 2nd XI) for which the

\* player prefers to play

\* @param playerPerformance - PlayerPerformance object

\* @param wicketkeeperPerformance - WicketkeeperPerformance object

\*/

public void addPlayer(String name, String address, String email,

String landline, String mobile, boolean captain,

boolean wicketkeeper, PlayerClassification classification,

Team teamPreference, Collection<PlayerPerformance>

playerPerformance,

Collection<WicketkeeperPerformance> wicketkeeperPerformance)

{

Player player = new Player(name, address, email, landline, mobile,

captain, wicketkeeper, classification, teamPreference);

Iterator<PlayerPerformance> setIte = playerPerformance.iterator();

while (setIte.hasNext())

{

player.addPlayerPerformance(setIte.next());

}

Iterator<WicketkeeperPerformance> setIt =

wicketkeeperPerformance.iterator();

while (setIt.hasNext())

{

player.addWicketkeeperPerformance(setIt.next());

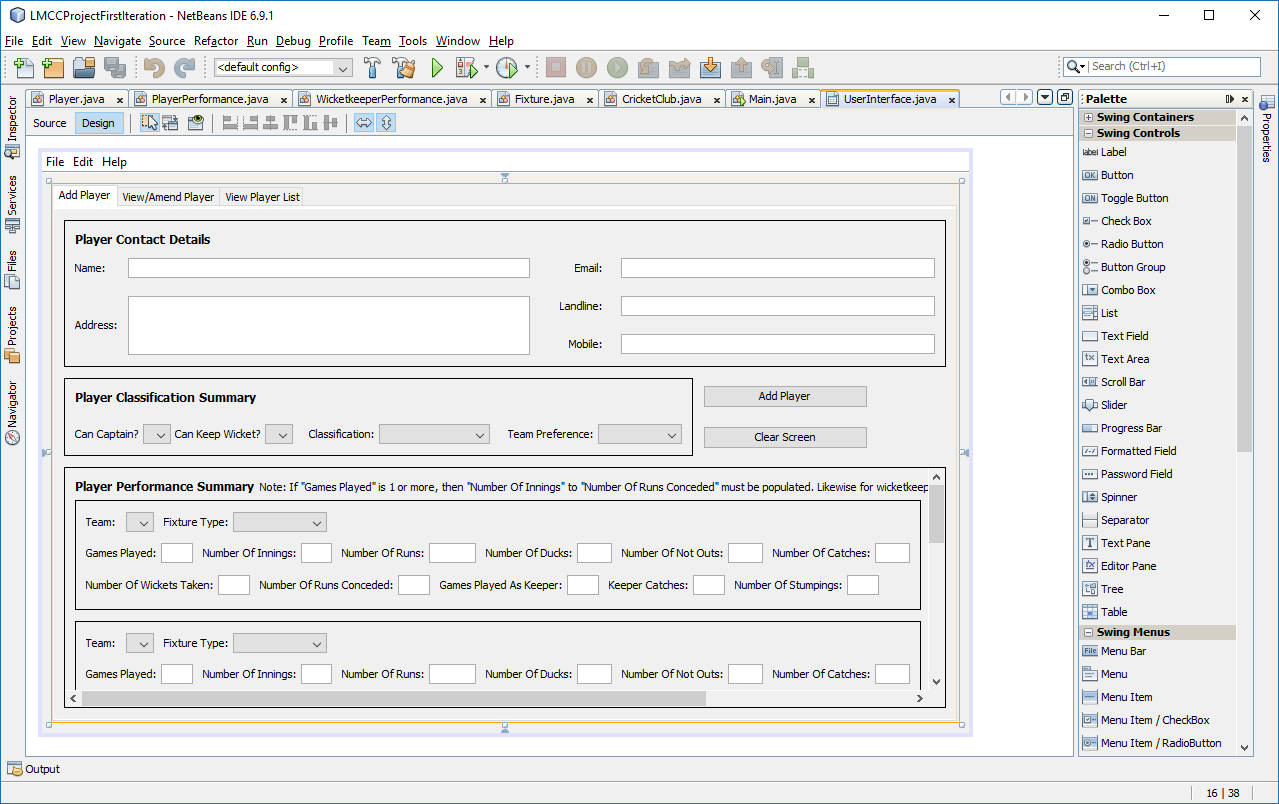
}

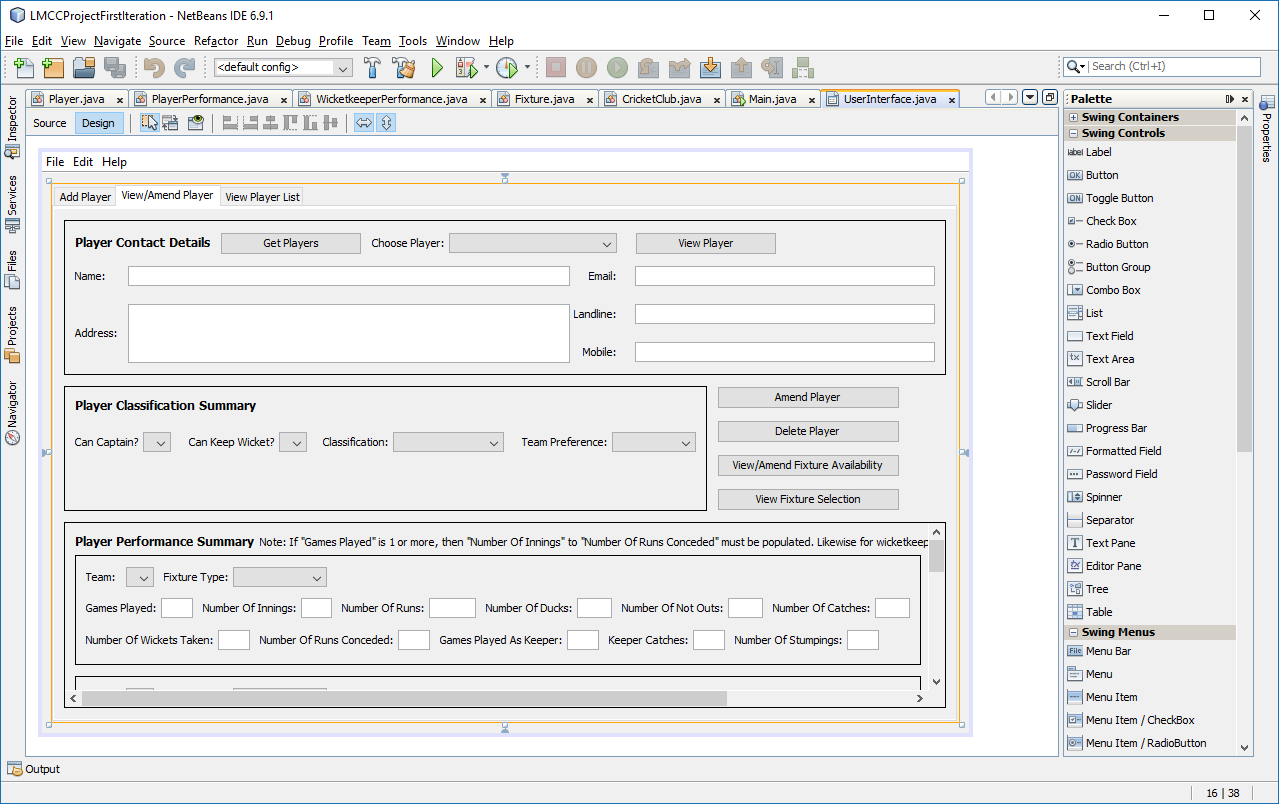
this.players.put(player.getPlayerNumber(), player);

}

## 10.15 Appendix 15 – User Interface Designs

See Section 3.15, p.23.





## 10.16 Appendix 16 – addPlayerButtonActionPerformed Method

See Section 3.16, p.24.

private void addPlayerButtonActionPerformed(java.awt.event.ActionEvent

evt)

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to add this player?",

"Add Player Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

// get relevant values from addPlayer panel for player information

// to pass to CricketClub.addPlayer method called below i.e.

//

// this.cricketClub.addPlayer

// (this.playerNameTextField.getText(),

// this.playerAddressTextField.getText(),

// this.playerEmailAddressLabel.getText(),

// this.playerLandlineTextField.getText(),

// this.playerMobileTextField.getText(),

// canCaptain, canWicketkeeper, classification, team,

// playerPerformance, wicketkeeperPerformance);

Boolean canCaptain = (Boolean)

this.playerCanCaptainBooleanComboBox.getSelectedItem();

Boolean canWicketkeeper = (Boolean)

this.playerCanWicketkeeperComboBox.getSelectedItem();

PlayerClassification classification = (PlayerClassification)

this.playerClassificationComboBox.getSelectedItem();

Team team = (Team)

this.playerTeamPreferenceComboBox.getSelectedItem();

// build playerPerformance HashSet to pass to

// CricketClub.addPlayer method called below

Collection<PlayerPerformance> playerPerformance = new

HashSet<PlayerPerformance>();

if

(!(this.playerPerformanceGamesPlayed1TextField.getText().isEmpty()

))

{

Integer pfixtTeamAndType1a = Integer.valueOf(

this.playerPerformanceGamesPlayed1TextField.getText());

if (pfixtTeamAndType1a > 0)

{

Integer pfTAT1b = Integer.valueOf(

this.playerPerformanceNumberOfInnings1TextField.getText());

. . .

. . .

. . .

PlayerPerformance pp1 = new PlayerPerformance(

pfTAT1e, pfTAT1d, pfixtTeamAndType1a, pfTAT1b,

pfTAT1c, pfTAT1f, pfTAT1g, pfTAT1h, pfTAT1i,

pfTAT1j);

playerPerformance.add(pp1);

}

}

if

(!(this.playerPerformanceGamesPlayed2TextField.getText().isEmpty()

))

. . .

. . .

. . .

// build wicketkeeperPerformance HashSet to pass

// to CricketClub.addPlayer method called below

Collection<WicketkeeperPerformance> wicketkeeperPerformance = new

HashSet<WicketkeeperPerformance>();

if (!(this.playerPerformanceGamesPlayedAsKeeper1TextField

.getText().isEmpty()))

{

Integer wfixtTeamAndType1a = Integer.valueOf(

this.playerPerformanceGamesPlayedAsKeeper1TextField.getText());

if (wfixtTeamAndType1a > 0)

{

Integer wfTAT1b = Integer.valueOf(

this.playerPerformanceNumberOfKeeperCatches1TextField.getText());

. . .

. . .

. . .

WicketkeeperPerformance wkp1 = new

WicketkeeperPerformance(wfTAT1e,

wfTAT1d, wfixtTeamAndType1a, wfTAT1b, wfTAT1c);

wicketkeeperPerformance.add(wkp1);

}

}

if (!(this.playerPerformanceGamesPlayedAsKeeper2TextField

.getText().isEmpty()))

. . .

. . .

. . .

this.cricketClub.addPlayer(this.playerNameTextField.getText(),

this.playerAddressTextField.getText(),

this.playerEmailAddressTextField.getText(),

this.playerLandlineTextField.getText(),

this.playerMobileTextField.getText(),

canCaptain, canWicketkeeper, classification, team,

playerPerformance, wicketkeeperPerformance);

javax.swing.JOptionPane pane2 = new javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"The player has been added.",

"Add Player Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

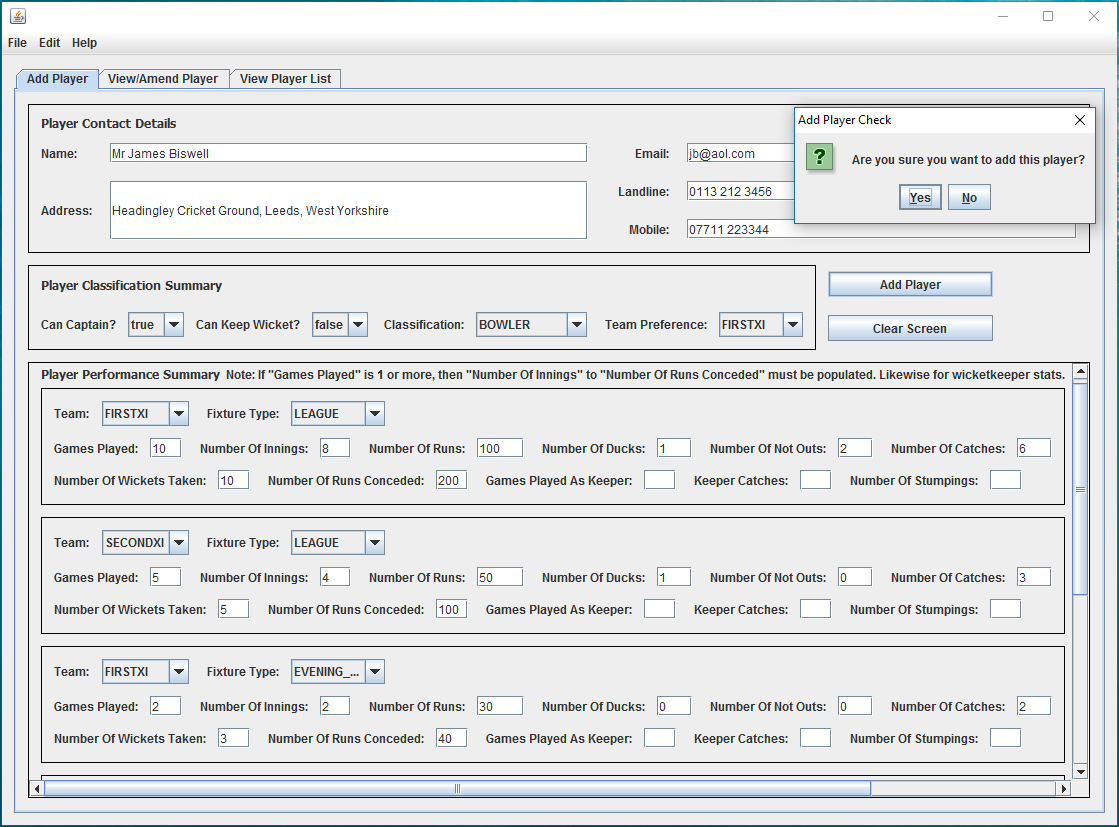
}

}

## 10.17 Appendix 17 – UC1 “Add Player” in Action

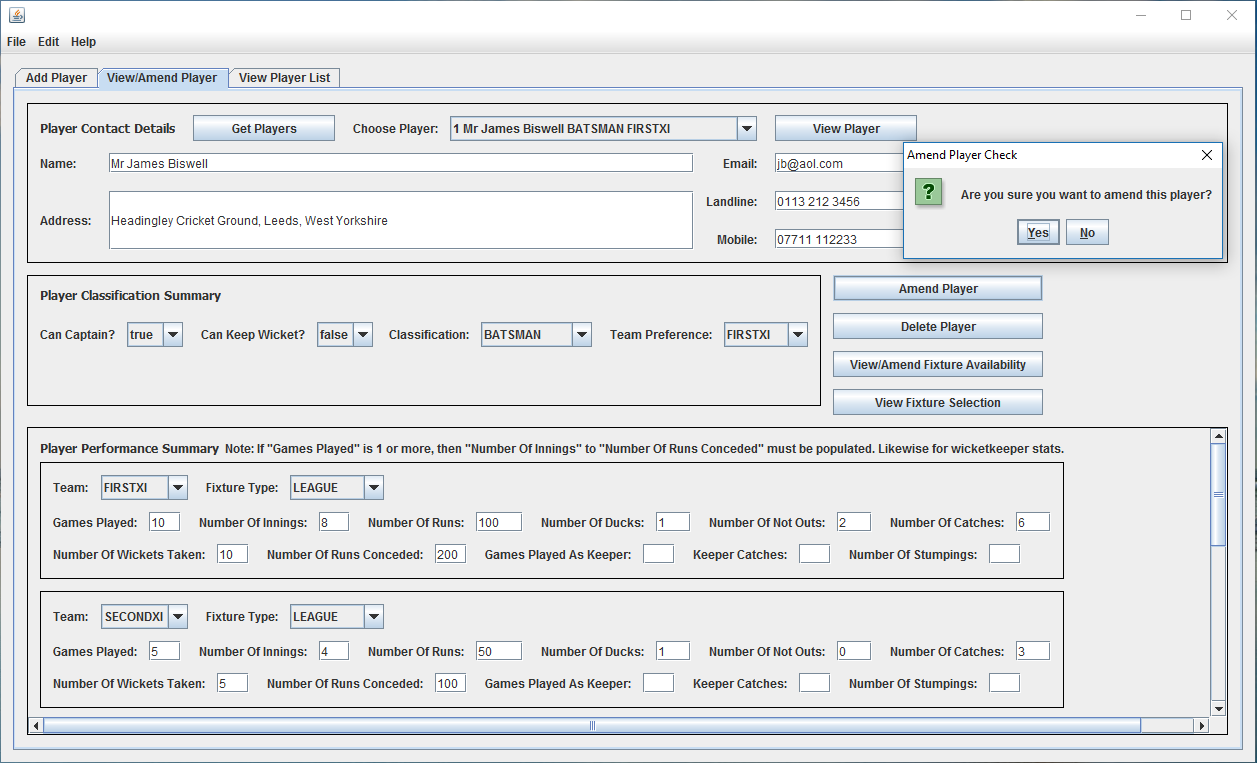
See Section 3.16, p.24.

See next page as a full page is required.



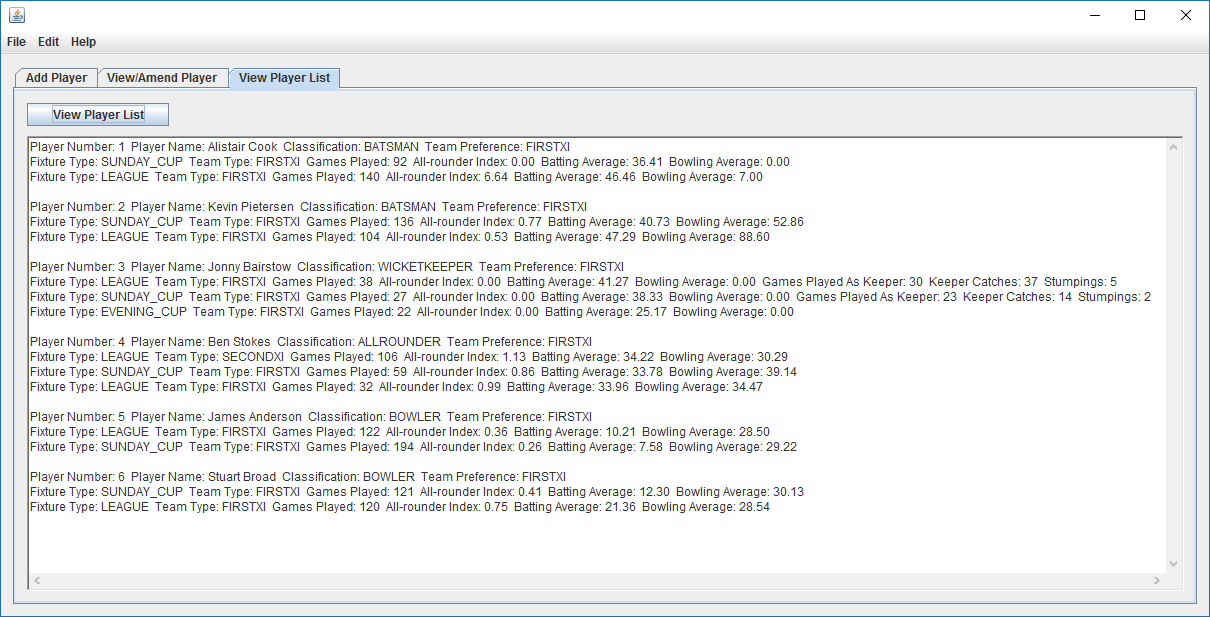
## 10.18 Appendix 18 – UC3 “View Player” and UC8 “Amend Player” in Action

See Section 3.17, p.24.



## 10.19 Appendix 19 – UC2 “View Player List” in Action

See Section 3.18, p.25.



## 10.20 Appendix 20 – openMenuItemActionPerformed (and saveMenuItem) Methods

See Section 3.19, p.25.

private void openMenuItemActionPerformed(java.awt.event.ActionEvent evt)

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to load the previously saved cricket

club?",

"Load Cricket Club Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

// System.out.println("Loading Cricket Club");

try

{

FileInputStream inStream;

ObjectInputStream objStream;

inStream = new FileInputStream("cricketclub.obj");

objStream = new ObjectInputStream(inStream);

this.cricketClub = (CricketClub) objStream.readObject();

objStream.close();

javax.swing.JOptionPane pane2 = new javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"The cricket club has been loaded.",

"Load Cricket Club Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

}

catch (Exception e)

{

// System.out.println("Error Loading Cricket Club");

javax.swing.JOptionPane pane3 = new javax.swing.JOptionPane();

pane3.showMessageDialog(null,

"It has not been possible to load the cricket club"

+ "\n" + "as none has previously been saved.",

"Error Loading Cricket Club Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

}

}

private void saveMenuItemActionPerformed(java.awt.event.ActionEvent evt)

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to save the cricket club?",

"Save Cricket Club Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

// System.out.println("Saving Cricket Club");

try

{

FileOutputStream outStream;

ObjectOutputStream objStream;

outStream = new FileOutputStream("cricketclub.obj");

objStream = new ObjectOutputStream(outStream);

objStream.writeObject(this.cricketClub);

objStream.flush();

objStream.close();

javax.swing.JOptionPane pane2 = new javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"The cricket club has been saved.",

"Save Cricket Club Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

}

catch (IOException e)

{

// System.out.println("Error Saving Cricket Club");

javax.swing.JOptionPane pane3 = new javax.swing.JOptionPane();

pane3.showMessageDialog(null,

"It has not been possible to save the cricket club.",

"Error Saving Cricket Club Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

}

}

## 10.21 Appendix 21 – CricketClub Class addFixture Method

See Section 3.26, p.30.

/\*\*

\* Add a fixture to the system

\*

\* @param fixtureType - type of fixture (i.e. league, evening cup etc.)

\* @param team - for which team (i.e. 1st XI or 2nd XI) it is a fixture

\* @param date - date of fixture

\* @param postcodeLocation - postcode of location where fixture is played

\* @param opposition - against whom the fixture is being played

\*

\* @return whether the fixture is added

\*/

public boolean addFixture(FixtureType fixtureType, Team team,

int date, String postcodeLocation, String opposition)

{

for (Fixture fixt : this.fixtures.values())

{

if (fixt.getTeam().equals(team)

&& (fixt.getDate() == date))

{

javax.swing.JOptionPane pane = new javax.swing.JOptionPane();

pane.showMessageDialog(null,

"Fixture for team and date already exists.",

"Add Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

// System.out.println("Fixture already exists!");

return false;

}

}

int fixtureNumber = 0;

if (this.fixtures.isEmpty())

{

fixtureNumber = 1;

}

else

{

Set<Integer> fixtureNumberSet = new HashSet<Integer>();

fixtureNumberSet = this.fixtures.keySet();

int i = 1;

while (fixtureNumber == 0)

{

if (!(fixtureNumberSet.contains(i)))

{

fixtureNumber = i;

}

i++;

}

}

Fixture fixture = new Fixture(fixtureNumber, fixtureType, team, date,

postcodeLocation,

opposition);

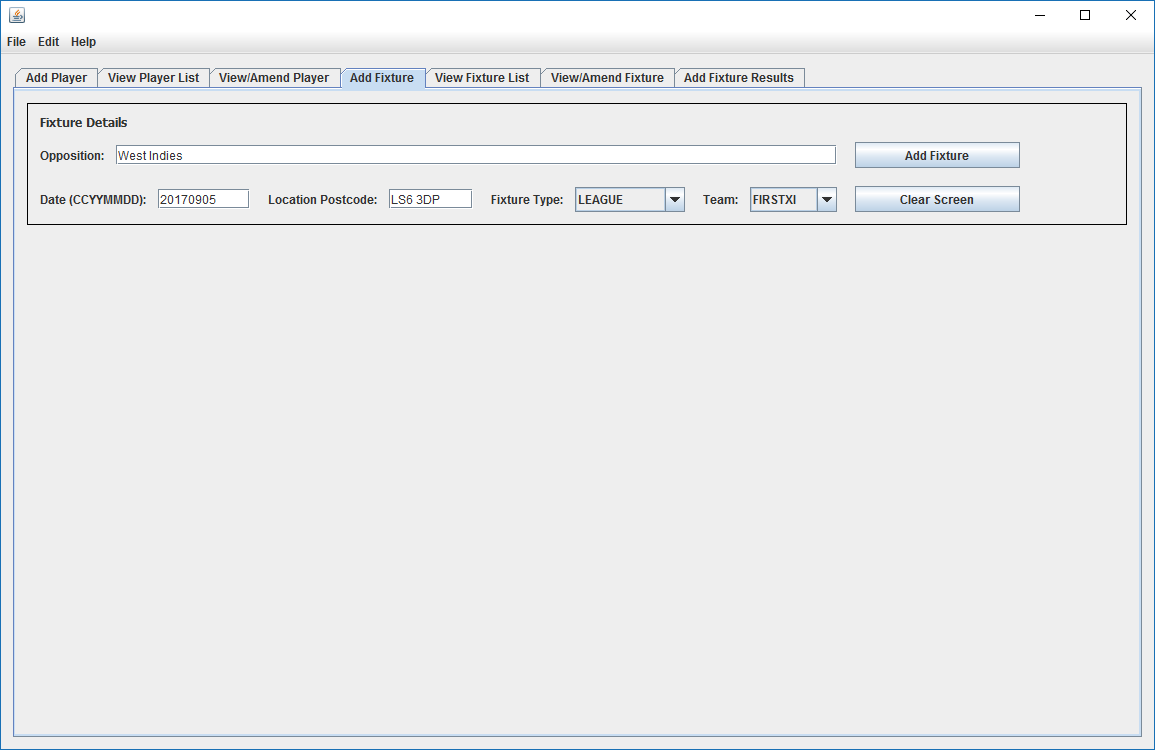
this.fixtures.put(fixture.getFixtureNumber(), fixture);

return true;

}

## 10.22 Appendix 22 – UC13 “Add Fixture” in Action

See Section 3.26, p.30.



## 10.23 Appendix 23 – Fixture Class toStringLong Method

See Section 3.27, p.31.

/\*\*

\* Return a String with various fixture details

\*

\* @return the String

\*/

public String toStringLong()

{

StringBuilder sb = new StringBuilder();

sb.append("Fixture Number: " + this.fixtureNumber + " ");

sb.append("Team: " + this.team + " ");

sb.append("Fixture Type: " + this.fixtureType + " ");

sb.append("Date: " + this.date + " ");

sb.append("Opposition: " + this.opposition + " ");

sb.append("Location: " + this.postcodeLocation + " ");

if (this.result != null)

{

sb.append("Result: " + this.result + " ");

}

sb.append("\n");

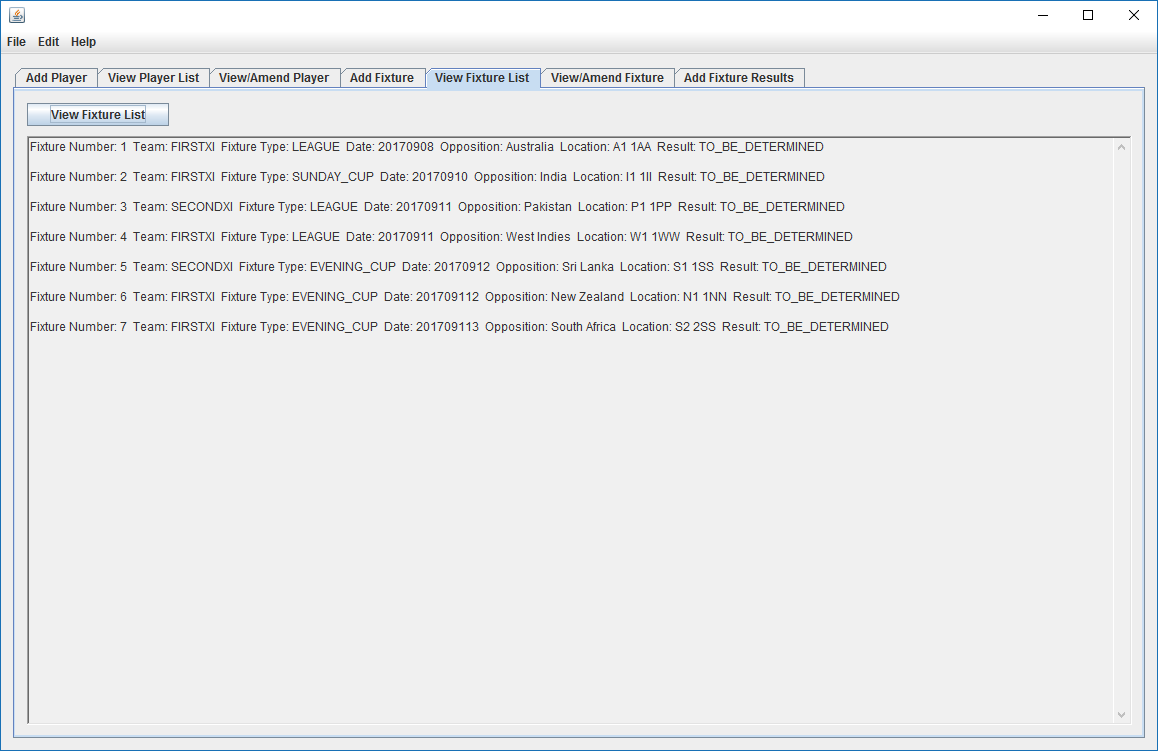
return sb.toString();

}

## 10.24 Appendix 24 – UC14 “View Fixture List” in Action

See Section 3.27, p.31.

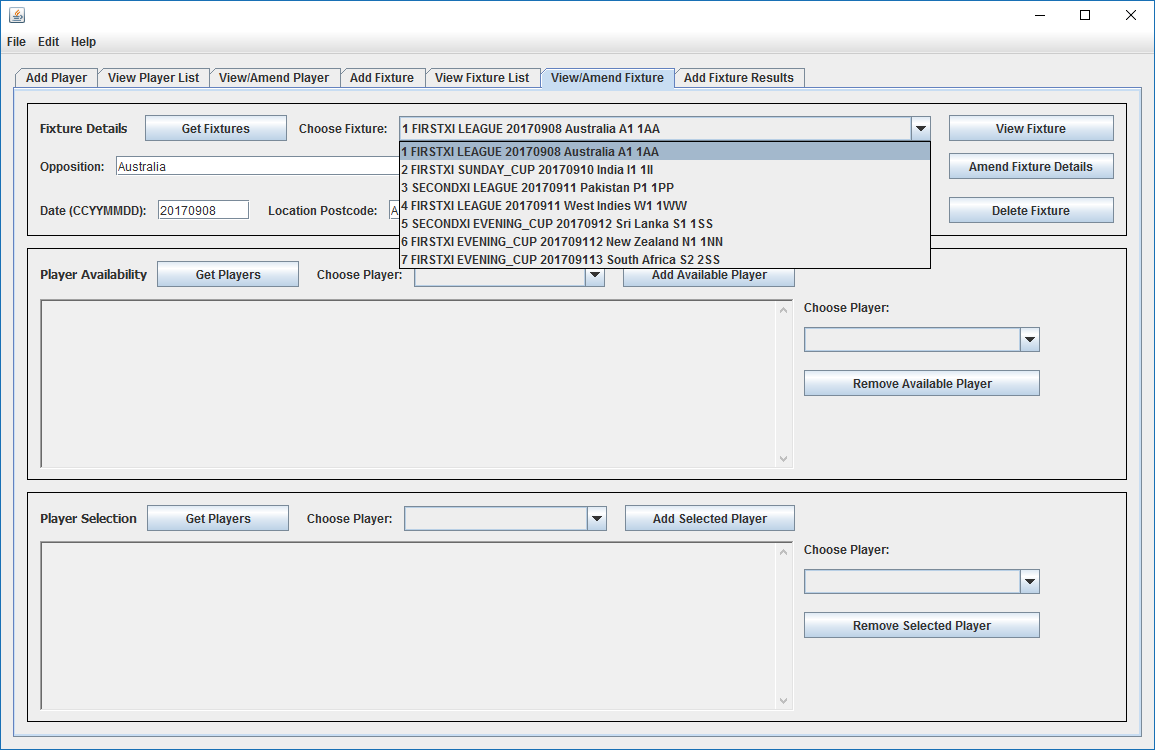
See next page as a full page is required.



## 10.25 Appendix 25 – UC15 “View Fixture” and UC18 “Amend Fixture” in Action

See Section 3.28, p.31.

See next page as a full page is required.



## 10.26 Appendix 26 – addAvailablePlayerButtonActionPerformed Method & Methods Called

See Section 3.29, p.31.

Method in User Interface:

private void

addAvailablePlayerButtonActionPerformed(java.awt.event.ActionEvent evt)

{

if ((this.chooseFixtureComboBox.getSelectedItem() != null)

&& (this.chooseAvailablePlayerComboBox.getSelectedItem() !=

null))

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to make this player available for

this fixture?",

"Add Player Available For Fixture Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

Fixture fixture = (Fixture)

this.chooseFixtureComboBox.getSelectedItem();

Player player = (Player)

this.chooseAvailablePlayerComboBox.getSelectedItem();

if (fixture.addPlayerAvailableFor(player))

{

javax.swing.JOptionPane pane2 = new

javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"The player has been made available for this

fixture.",

"Add Player Available For Fixture Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

// populate available players text area

this.availablePlayersTextArea.setText("");

Collection<Player> playerList = new HashSet<Player>();

playerList = fixture.viewPlayersAvailableFor();

Iterator<Player> ite = playerList.iterator();

StringBuilder sb = new StringBuilder();

while (ite.hasNext())

{

Player play = ite.next();

sb.append(play.toString());

sb.append("\n");

sb.append("\n");

}

this.availablePlayersTextArea.setText(sb.toString());

// populate remove available player combo box

this.removeAvailablePlayerComboBox.setModel(

new javax.swing.DefaultComboBoxModel(

fixture.viewPlayersAvailableFor().toArray()));

// populate choose selected player combo box

this.chooseSelectedPlayerComboBox.setModel(

new javax.swing.DefaultComboBoxModel(

fixture.viewPlayersAvailableFor().toArray()));

}

else

{

javax.swing.JOptionPane pane3 = new

javax.swing.JOptionPane();

pane3.showMessageDialog(null,

"The player is already available for this

fixture.",

"Add Player Available For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

}

}

}

Method in Fixture Class:

public boolean addPlayerAvailableFor(Player player)

{

if (!(this.playersAvailableFor.contains(player)))

{

this.playersAvailableFor.add(player);

if (!(player.getFixturesAvailableFor().contains(this)))

{

player.addFixtureAvailableFor(this);

}

return true;

}

return false;

}

Method in Player Class:

public boolean addFixtureAvailableFor(Fixture fixture)

{

if (!(this.fixturesAvailableFor.contains(fixture)))

{

this.fixturesAvailableFor.add(fixture);

if (!(fixture.getPlayersAvailableFor().contains(this)))

{

fixture.addPlayerAvailableFor(this);

}

return true;

}

return false;

}

## 10.27 Appendix 27 – removeAvailablePlayerButtonActionPerformed Method

See Section 3.29, p.31.

Method in User Interface:

private void

removeAvailablePlayerButtonActionPerformed(java.awt.event.ActionEvent evt)

{

if ((this.chooseFixtureComboBox.getSelectedItem() != null)

&& (this.removeAvailablePlayerComboBox.getSelectedItem() !=

null))

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to remove this player from being

available for this fixture?",

"Remove Player Available For Fixture Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

Fixture fixture = (Fixture)

this.chooseFixtureComboBox.getSelectedItem();

Player player = (Player)

this.removeAvailablePlayerComboBox.getSelectedItem();

if ((fixture.getPlayersSelectedFor().contains(player)))

{

javax.swing.JOptionPane pane2 = new

javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"The player has already been selected for this

fixture. Please unselect him first.",

"Remove Player Available For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

else

{

if (fixture.deletePlayerAvailableFor(player))

{

javax.swing.JOptionPane pane3 = new

javax.swing.JOptionPane();

pane3.showMessageDialog(null,

"The player has been made unavailable for this

fixture.",

"Remove Player Available For Fixture

Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

// populate available players text area

this.availablePlayersTextArea.setText("");

Collection<Player> playerList = new HashSet<Player>();

playerList = fixture.viewPlayersAvailableFor();

Iterator<Player> ite = playerList.iterator();

StringBuilder sb = new StringBuilder();

while (ite.hasNext())

{

Player play = ite.next();

sb.append(play.toString());

sb.append("\n");

sb.append("\n");

}

this.availablePlayersTextArea.setText(sb.toString());

// populate remove available player combo box

this.removeAvailablePlayerComboBox.setModel(

new javax.swing.DefaultComboBoxModel(

fixture.viewPlayersAvailableFor().toArray()));

// populate choose selected player combo box

this.chooseSelectedPlayerComboBox.setModel(

new javax.swing.DefaultComboBoxModel(

fixture.viewPlayersAvailableFor().toArray()));

}

else

{

javax.swing.JOptionPane pane4 = new

javax.swing.JOptionPane();

pane4.showMessageDialog(null,

"The player has not before been made available

for this fixture.",

"Remove Player Available For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

}

}

}

}

Method in Fixture Class:

public boolean deletePlayerAvailableFor(Player player)

{

if (this.playersAvailableFor.contains(player)

&& !(this.playersSelectedFor.contains(player)))

{

this.playersAvailableFor.remove(player);

if (player.getFixturesAvailableFor().contains(this))

{

player.deleteFixtureAvailableFor(this);

}

return true;

}

return false;

}

Method in Player Class:

public boolean deleteFixtureAvailableFor(Fixture fixture)

{

if (this.fixturesAvailableFor.contains(fixture)

&& !(this.fixturesSelectedFor.contains(fixture)))

{

this.fixturesAvailableFor.remove(fixture);

if (fixture.getPlayersAvailableFor().contains(this))

{

fixture.deletePlayerAvailableFor(this);

}

return true;

}

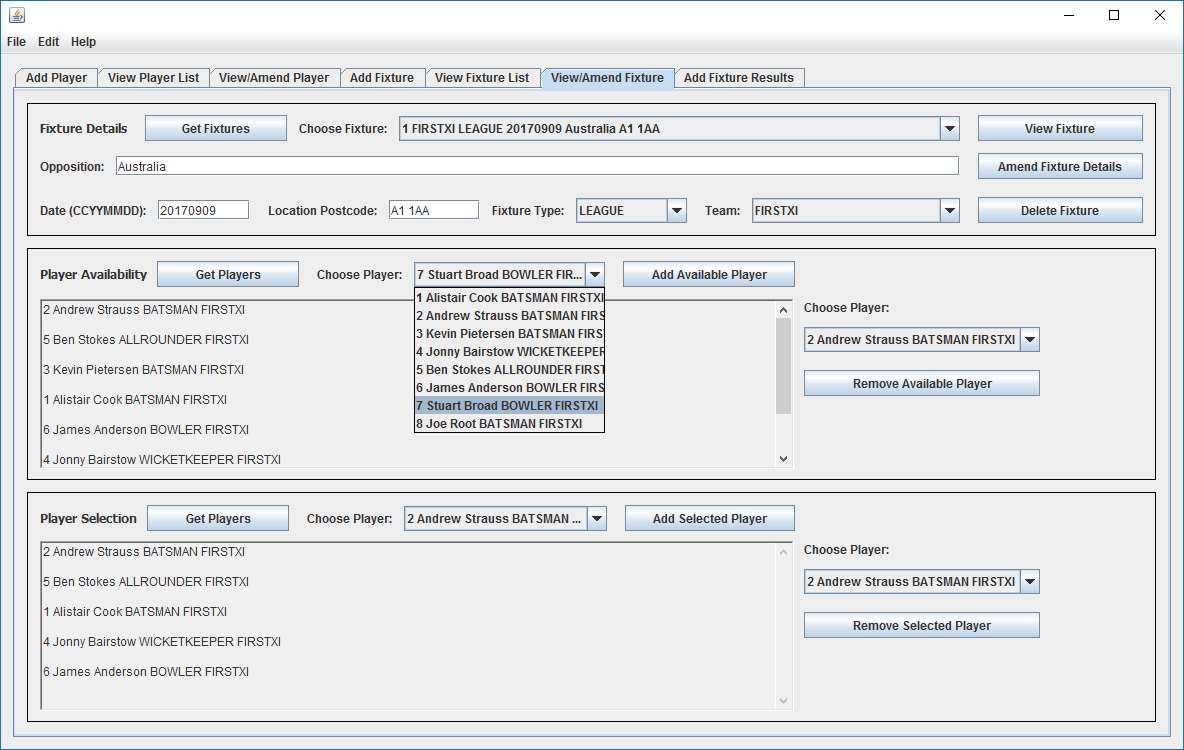
return false;

}

## 10.28 Appendix 28 – UC19 “Amend Availability Of Players For Specific Fixture” in Action

See Section 3.29, p.31.

See next page as a full page is required.



## 10.29 Appendix 29 – addSelectedPlayerButtonActionPerformed Method

See Section 3.30, p.32.

Method in User Interface:

private void

addSelectedPlayerButtonActionPerformed(java.awt.event.ActionEvent evt)

{

if ((this.chooseFixtureComboBox.getSelectedItem() != null)

&& (this.chooseSelectedPlayerComboBox.getSelectedItem() !=

null))

{

javax.swing.JOptionPane pane1 = new javax.swing.JOptionPane();

int n = pane1.showConfirmDialog(null,

"Are you sure you want to select this player for this

fixture?",

"Select Player For Fixture Check",

javax.swing.JOptionPane.YES\_NO\_OPTION);

if (n == 0)

{

Fixture fixture = (Fixture)

this.chooseFixtureComboBox.getSelectedItem();

Player player = (Player)

this.chooseSelectedPlayerComboBox.getSelectedItem();

if (!(fixture.getPlayersSelectedFor().size() < 11))

{

javax.swing.JOptionPane pane2 = new

javax.swing.JOptionPane();

pane2.showMessageDialog(null,

"Eleven players are already selected for this

fixture.",

"Select Player For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

else

{

if (!(fixture.getPlayersAvailableFor().contains(player)))

{

javax.swing.JOptionPane pane3 = new

javax.swing.JOptionPane();

pane3.showMessageDialog(null,

"The player is not available for this

fixture.",

"Select Player For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

else

{

if (fixture.addPlayerSelectedFor(player))

{

javax.swing.JOptionPane pane4 = new

javax.swing.JOptionPane();

pane4.showMessageDialog(null,

"The player has been selected for this

fixture.",

"Select Player For Fixture Confirmation",

javax.swing.JOptionPane.PLAIN\_MESSAGE);

// populate selected players text area

this.selectedPlayersTextArea.setText("");

Collection<Player> playerList = new

HashSet<Player>();

playerList = fixture.viewPlayersSelectedFor();

Iterator<Player> ite = playerList.iterator();

StringBuilder sb = new StringBuilder();

while (ite.hasNext())

{

Player play = ite.next();

sb.append(play.toString());

sb.append("\n");

sb.append("\n");

}

this.selectedPlayersTextArea.setText(sb.toString());

// populate remove available player combo box

this.removeSelectedPlayerComboBox.setModel(

new javax.swing.DefaultComboBoxModel(

fixture.viewPlayersSelectedFor().toArray()));

}

else

{

javax.swing.JOptionPane pane5 = new

javax.swing.JOptionPane();

pane5.showMessageDialog(null,

"The player is already selected for this

fixture.",

"Select Player For Fixture Warning",

javax.swing.JOptionPane.WARNING\_MESSAGE);

}

}

}

}

}

}

Method in Fixture Class:

public boolean addPlayerSelectedFor(Player player)

{

if ((this.playersAvailableFor.contains(player))

&& (this.playersSelectedFor.size() < 11)

&& !(this.playersSelectedFor.contains(player)))

{

this.playersSelectedFor.add(player);

if (!(player.getFixturesSelectedFor().contains(this)))

{

player.addFixtureSelectedFor(this);

}

return true;

}

return false;

}

## 10.30 Appendix 30 – Player Class viewFixturesAvailableFor Method

See Section 3.31, p.32.

/\*\*

\* View fixtures available for

\*

\* @return the future fixtures for which the player is available

\*/

public Collection<Fixture> viewFixturesAvailableFor()

{

Calendar calendar = Calendar.getInstance(TimeZone.getDefault());

Integer year = calendar.get(calendar.YEAR);

Integer month = calendar.get(calendar.MONTH);

month = month + 1;

String monthString = null;

if (month < 10)

{

monthString = "0" + month;

}

else

{

monthString = month.toString();

}

Integer day = calendar.get(calendar.DAY\_OF\_MONTH);

String dayString = null;

if (day < 10)

{

dayString = "0" + day;

}

else

{

dayString = day.toString();

}

String dateString = year + monthString + dayString;

Integer dateInt = Integer.parseInt(dateString);

Collection<Fixture> fixtures = new HashSet<Fixture>();

for (Fixture fixture : this.fixturesAvailableFor)

{

if (fixture.getDate() >= dateInt)

{

fixtures.add(fixture);

}

}

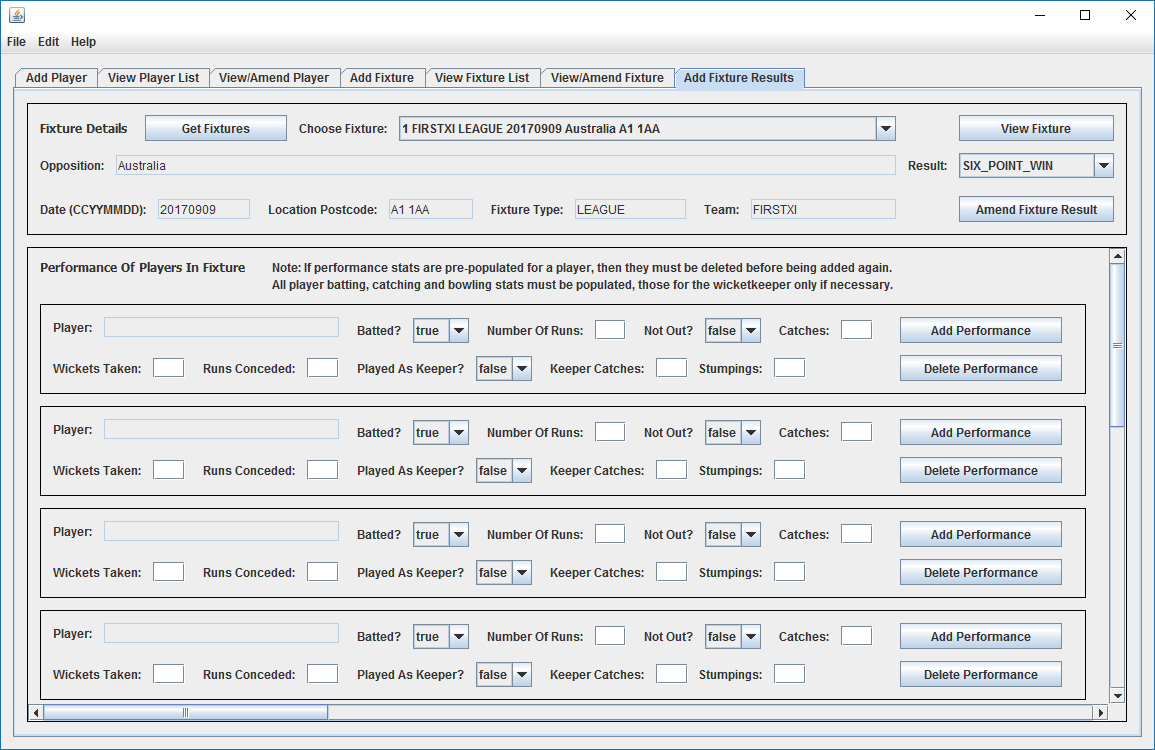
return fixtures;

}

## 10.31 Appendix 31 – UC22 “Add Fixture Results” User Interface Design

See Section 3.34, p.34.

See next page as a full page is required.



## 10.32 Appendix 32 – Project Tutor Correspondence

See Section 7, p.51.

**From:** James Biswell [<mailto:jamesbiswell@aol.com>]   
**Sent:** 25 May 2017 00:10  
**To:** 'a.j.pullin@open.ac.uk' <[a.j.pullin@open.ac.uk](mailto:a.j.pullin@open.ac.uk)>  
**Cc:** 'james.biswell@callcreditgroup.com' <[james.biswell@callcreditgroup.com](mailto:james.biswell@callcreditgroup.com)>  
**Subject:** TM470 Summary Update Email To Tutor 24052017

Hi Adrian,

As I mentioned, here are some things I would like to talk about during our call tomorrow. I think this should take between 30 mins to 45 at most, if that’s okay?

1/ Quick discussion of your feedback to TMA02.

2/ Brief discussion of approach for TMA03. I think I am much clearer on what’s needed now, after reading TMA03 and EMA, and your postings on TMA02 Feedback / TMA03 Advice and on What Should Be In TMA03.

3/ For the problem description I think I just need to synthesise into a cogent whole bits of what were “scattered around” in TMA02, including some of what was in the client-facing problem statement, and some of what was in the “project definition, goals and scope” section i.e. project aim, objectives, scope and outputs, to make that content quite specific to the actual problem being investigated and solution being proposed (and its ICT context).

4/ Should I mention personal issues as a project risk/setback? (I feel a little uncomfortable about doing so etc.)

5/ For the EMA, should I include the full Volere Template? and all of the Use Cases? as appendices. Because both of these contain a lot of words.

6/ LSEPIs. I don’t think any marks are available for these in TMA03 (but they are in EMA) so I’m probably not going to focus on them until I have made further advances with the practical work.

7/ Coding and testing. I’ve made some good progress with the coding so far, but if I am to catch up (over the next four days while not being at work), I need to focus on coding rather than very low-level unit testing. I don’t think learning or applying JUnit in detail is going to add much value to the project.

8/ Would screenshots from NetBeans be acceptable to use as appendix evidence that a large amount of the donkey work has been done to set up the skeleton structure of the code base?

9/ I think I am going to focus on use-case based functional testing, and could this also be evidenced by screenshots once the UI has been designed and built?

It would be useful if I could share my PC screen with you for the above and I can use Glance to do so:

<https://www.glance.net/default.asp?username=jamesbiswelltm470.glance.net>

Cheers

James

**From:** James Biswell [<mailto:jamesbiswell@aol.com>]   
**Sent:** 14 March 2017 00:25  
**To:** 'a.j.pullin@open.ac.uk' <[a.j.pullin@open.ac.uk](mailto:a.j.pullin@open.ac.uk)>  
**Cc:** 'james.biswell@callcreditgroup.com' <[james.biswell@callcreditgroup.com](mailto:james.biswell@callcreditgroup.com)>  
**Subject:** TM470 Weekly Update Email To Tutor 13032017

Hi Adrian,

Hope you are well.

It’s been a couple of weeks since my last email (used to ask questions, make observations, seek advice, raise concerns). The week before last, all of my efforts were on the TMA which you were going to read anyway, so it seemed unnecessary to email you as well. Unfortunately, last week was very busy at work and I was travelling for two days/nights.

**Firstly**, regarding the TMA, thanks for your constructive feedback on it. It took me much longer than I had initially anticipated to write the TMA (roughly 15 hours rather than the 8 I had originally estimated) and it’s clear now from your specific feedback on my script and your (very helpful) forum posting last week that a lot of work needs to go into this part of the project, in terms of structure, content and style. But the tools in Word look a lot better than when I wrote a dissertation twenty years ago, I’m comfortable with how to reference, and now that I am actively learning more about my software development project from the relevant literature, there will be more in future reports.

My weaker points were:

1/ LO3 Resources, Skills, Activities: agree with your comments (as attached, more discussion needed on resources) and I am focussing on this as the project now progresses.

2/ LO4 Gather, Analyse, Evaluate Information: agree with your comments, this is my main academic focus for this part of the project up to the next TMA.

3/ LO7 Communication: agree with your comments regarding the quality (especially referencing) of the report and this will be easily addressed, but felt I was undermarked for the quality and timeliness of the contact I maintained with you up to that point (and for prompt posting to the forum) with clearly expressed project ideas - as per the indicative marking scheme. You also mentioned that the phrase “as we all know” was unsuitable for a formal report; I completely agree, and had indicated that this was a quotation from a posting I had made to the forum. I appreciate that you had had to read 14 of these, but a lifetime of reading Classics means I try not to write sloppily.

4/ LO8 Learn, Reflect, Skills, Knowledge: agreed again, and filling the skills gap will come, primarily, from learning from the literature.

Noted well regarding use of “third person, passive” for the progress reporting part of reports, but it seems preferable to use the first person for some of the review and reflection part, what do you think?

I note also that the review and reflection section of TMA02 requires an indication of “how you have responded to tutor feedback on TMA01”. In short, I have found both the specific feedback and your forum posting very useful, thanks again, and acting on the advice in both should make for an overall advancement for TMA02. Indeed, this email shows I have thought carefully about how to improve further.