The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) ______________________
Summary

This project aims to give the author a deeper understanding of the latest technologies for web-application development, by utilising an appropriate web development framework to implement the event booking system for a real-world company, known as Driving Force UK.

Initially research will be conducted into the origins of web development frameworks, and several will be assessed in various ways to select one for the task of creating the event booking system. Using standard techniques for information gathering, Driving Force UK’s system requirements will be assessed, and the minimum features necessary to create a working system will be implemented in the chosen framework. Finally, several system enhancements will be assessed for future work, and the cross-compatibility of the application created will be tested.

All the work undertaken in the project will be evaluated, to assess the suitability and usefulness of web development frameworks in creating online applications. From this, a practical understanding of the use of these new web technologies should be acquired.
Acknowledgements

I would like to thank Richard Costa and Roger Cohen from Driving Force UK for all of their assistance throughout this project, from attending meetings for initial research, to giving feedback on prototypes. I would also like to extend this thanks to Susan Skinner from United Technologies Corporation for her feedback on the client side of the system, and to Tom McNally for his feedback on the driver side of the system.

I would also like to thank my supervisor Matthew Hubbard for all of his ongoing assistance with the project, and my assessor Nick Efford who was always willing to provide advice and pointers on my software during development.

Finally I would like to thank my parents who were always there for me giving encouragement and guidance, and my housemates for providing me with food during long coding sessions!
Table of Contents

Chapter 1 - Introduction............................................................................................................. - 1 -
  1.1 Aim................................................................................................................................. - 1 -
  1.2 Objectives ...................................................................................................................... - 1 -
  1.3 The Model Company ...................................................................................................... - 1 -
  1.4 The Problem .................................................................................................................. - 2 -
  1.5 Minimum Requirements ............................................................................................... - 2 -
  1.6 Relevance to Degree Programme .................................................................................. - 3 -
  1.7 Deliverables ................................................................................................................... - 3 -
  1.8 Project Structure .......................................................................................................... - 3 -
  1.9 Changes to Project from Mid-Project Report ............................................................... - 4 -

Chapter 2 - Background Research ............................................................................................. - 6 -
  2.1 Company Background .................................................................................................. - 6 -
  2.2 Project Methodologies .................................................................................................. - 8 -
    2.2.1 Methodologies Available ....................................................................................... - 8 -
    2.2.2 Choosing a Suitable Methodology ......................................................................... - 9 -
  2.3 Human-Computer Interaction ....................................................................................... - 9 -

Chapter 3 – Phase 1 ..................................................................................................................... - 11 -
  3.1 Development tools - Framework Research .................................................................... - 11 -
    3.1.1 Building Web Applications ..................................................................................... - 11 -
    3.1.2 Web Development Frameworks ............................................................................. - 12 -
    3.1.3 Choosing a Suitable Framework ............................................................................ - 13 -
  3.2 Implement Prototypes with Example Applications ...................................................... - 14 -
    3.2.1 Tapestry jumpstart ................................................................................................. - 15 -
    3.2.2 Wicket phonebook-1.2 .......................................................................................... - 15 -
  3.3 Test Prototype Applications .......................................................................................... - 15 -
  3.4 Framework Conclusions ............................................................................................... - 16 -
Chapter 4 – Phase 2

4.1 Business Modelling

4.2 Requirements Analysis

4.3 Design

4.3.1 Front-end Design

4.3.2 Back-end Design

4.4 Implementation

4.4.1 Installation of Database Server

4.4.2 Initialisation of Qwicket Project

4.4.3 Displaying Event Data

4.4.4 Storage of Event Data – Requirement 2

4.4.5 Restriction of System Access – Requirement 3

4.4.6 Assigning Drivers to Jobs – Requirement 4

4.5 Testing

4.5.1 Database Population Testing

4.5.2 Introduction to User Feedback

4.5.3 Feedback from MD

4.5.4 Feedback from Company Coordinator

4.5.5 Feedback from Client

4.5.6 Feedback from Driver

Chapter 5 – Phase 3

5.1 Additional Requirements Analysis

5.1.1 Listing of a Driver’s Jobs for Management

5.1.2 Email Driver with Job Details

5.1.3 Providing Driver with Map Views

5.2 Enhancement Design Considerations

5.2.1 Multiple Drivers Assigned to a Single Booking
5.2.2 Booking Alert System for Management .................................................. - 35 -
5.2.3 Printing and Exporting of Bookings ......................................................... - 36 -
5.2.4 Listing of a Driver’s Jobs for Management .............................................. - 36 -
5.2.5 Email Driver with Job Details ..................................................................... - 37 -
5.2.6 Providing Driver with Map Views ............................................................... - 37 -
5.2.7 Cross-Compatibility of System with Database Servers ............................... - 38 -

5.3 Enhancement Implementation and Testing ................................................... - 38 -

Chapter 6 – Evaluation ......................................................................................... - 39 -
6.1 Evaluation Criteria ......................................................................................... - 39 -
6.2 Evaluation of Frameworks ............................................................................. - 39 -
6.2.1 Evaluation of Tapestry ............................................................................... - 40 -
6.2.2 Evaluation of Wicket .................................................................................. - 40 -
6.3 Evaluation of Software Solution ..................................................................... - 41 -
6.3.1 System Correctness ................................................................................... - 42 -
6.3.2 System Maintainability ............................................................................. - 43 -
6.3.3 System Integrity ....................................................................................... - 43 -
6.3.4 System Usability ....................................................................................... - 44 -
6.4 Evaluation of Project ..................................................................................... - 44 -

Chapter 7 – References ....................................................................................... - 46 -

Chapter 8 - Appendices ....................................................................................... - 49 -
8.1 Appendix A – Project Reflection ..................................................................... - 49 -
8.2 Appendix B – Jakob Nielsen’s Ten Usability Heuristics ................................. - 50 -
8.3 Appendix C – Phonebook Application ............................................................ - 52 -
8.4 Appendix D – Use Case Diagram for Phase 2 ............................................... - 53 -
8.5 Appendix E – Activity Diagram for Phase 2 .................................................. - 54 -
Chapter 1 - Introduction

1.1 Aim

The aim of this project is to investigate a set of web development frameworks, and evaluate one in detail by using an appropriate software-engineering methodology to construct a model application. The key objective is to gain a comprehensive understanding of the range of frameworks available, and assess the suitability of one for following a standard system development process.

1.2 Objectives

To fully satisfy the aim of this project, as a minimum the following objectives must be met:

- Analysis of a set of web development frameworks;
- Selection of one framework based on the analysis performed and other design considerations;
- Design and implementation of a software solution in the chosen framework, to fulfil the requirements of the model company;
- Testing of the solution with potential real users of the system;
- Evaluation of the framework which was chosen;
- Evaluation of the software solution produced.

1.3 The Model Company

The model company known as Driving Force UK will be used in this project as it is typical of most event transportation companies. By basing the software solution around the requirements of this company, the solution produced should be appropriate for most event transportation companies’ data requirements.

The company’s primary purpose is to provide an executive transportation service to clients at business events, such as corporate board meetings and Airshows. Company management maintain a list of self-employed drivers and cars, and contract them to provide transportation on a per-event basis. Management must also maintain a list of all bookings required by the client, and ensure that a driver is assigned to each, and that the necessary information is relayed to all the relevant parties.
1.4 The Problem

The initial problem this project set out to solve involved improving the data management at Driving Force UK, however over the course of conducting the project, another problem was discovered regarding the choice and use of a web development framework.

There are now many web development frameworks available [1] freely on the internet to assist in web application development. For the purposes of solving Driving Force UK’s problem, only one framework is required, so it is necessary to evaluate the choices available and select one based on reasoned comparisons.

The problem (see section 2.1 for a detailed analysis) faced by Driving Force UK is in the storage of data. Currently event data is stored within several excel spreadsheets, and company management update them when new information arrives from clients or drivers through phone calls, emails, or direct contact. This is a slow and error-prone process, as inconsistencies in data can arise from a number of sources.

1.5 Minimum Requirements

The minimum requirements detailed below must be met to achieve the aims set out in this project, and solve the problems stated above. Any further enhancements to these minimum requirements will add to the system’s functionality, but are not necessary for a proper evaluation of the framework and project.

1. **Research and utilise an appropriate web development framework:**
   By researching and assessing the available frameworks in phase 1 of the project, an educated choice can be made for which to utilise for the implementation of the model application.

2. **Provide means for storage and editing of event bookings on the system by clients and management:**
   The model application must allow for event and booking data to be stored in a database, and allow all users with the correct access to view and edit the bookings already made.
3. **Utilise security system for delegation of use and to prevent disclosure of confidential data, by assigning of users to groups:**
   As the model application will be used by various groups of users, it is important to limit system functionality for secondary users outside of the host company, and prevent clients and drivers from viewing event and booking data for events they are not registered to.

4. **Provide means for allocated job information to be shown to drivers using the system:**
   Once a booking has been made on the system, and company management have assigned a driver to the job, the allocated driver should be able to login to the system and view their current jobs.

1.6 Relevance to Degree Programme

This project draws upon knowledge gained in several modules taken over the last three years. This includes software-engineering modules (SE10, SE14, SE20, SE24), database modules (DB11, DB21), human-computer interaction modules (GI11), and web programming modules (SY23, SY33).

1.7 Deliverables

The deliverables necessary for the project, in addition to this report, will be the online booking system created for Driving Force UK. It should serve as a way for clients to specify their requirements, company management to process all bookings and store event data, and provide a means for drivers to obtain up-to-date job information.

1.8 Project Structure

The project will be structured into three phases:
- An investigation into the available frameworks;
- The creation of a software prototype to meet the minimum requirements specified above;
- A discussion of possible enhancements to the prototype, and implementation of as many features as possible
The first phase of the project will focus on finding the most appropriate framework for implementing the model company’s prototype solution. A selection of frameworks will be chosen from the web, and then disposable prototype systems will be created in the two frameworks deemed most suitable for the second phase of the project. Based on the experiences with these two disposable prototypes, a decision will be made for which framework will be utilised for creating the prototype solution for the next phase.

In the second phase, Driving Force UK’s data management processes will be modelled, and a requirement specification created for a web-based booking system. The design for the system will be produced, and implemented in the framework selected in phase 1. The system will then be tested by the hosting company’s management team, an event coordinator from a client organisation, and a driver contracted to the host company.

Finally in the third phase, some system enhancements will be detailed and appropriate solutions to problems arising from user testing will also be described. Design issues in implementing the new features will be considered, and the prototype from phase 2 will be enhanced as much as time permits.

1.9 Changes to Project from Mid-Project Report

There have been several changes to the project from the goals set out in the mid-project report, for various reasons, and these will be detailed below.

The following changes were made to the minimum requirements from those submitted as part of the mid-project report:

- The ‘providing means for storage of event bookings’ requirement has been extended to ‘providing means for storage and editing of event bookings’. This is because it was determined that within the system, it is essential that data currently stored is editable if for any reason data already stored is incorrect, or the requirements change. This feature was stated in the mid-project report, under the ‘Allow logistic coordinators to amend stored data for drivers/events/bookings’ requirement in the ‘should-have’ functionality list. It should of course be possible for any management user or client, to modify any type of data within the system that they have access to. Thus this functionality will be implemented in phase 2 of the project.
- The ‘storage of driver’s personal details’ was initially specified as a minimum requirement of the system. Since submitting the mid-project report however, it was decided that this was not a critical requirement of the system since drivers can still be stored in the system as users, and allocated to bookings in this way. The implementation of the feature will be a large task in itself, since an additional driver table will be required to store the details, and several additional input and output screens will be necessary to manipulate the table’s data. Several modifications will also need to be made to the login procedures to check for valid driver login credentials. If possible this feature will be incorporated into the system as an enhancement in phase 3.

It was stated in the mid-project report that the deliverables submitted would be:

- Report on the software solution created, detailing all design considerations and evaluating the solution produced.
- Software solution, delivered within a suitable deployable application.
- A brief user guide for company management, explaining key features of the system.

Since this was written however, it was found that the Oracle database server claimed to be in use on Driving Force UK’s systems was not actually present. Also as phase 2 of the project took longer than initially expected, it was decided that the deployment stage would be removed from the project, so that other project aspects could be concentrated on. This led to the alteration of the project deliverables, as now the software solution will be delivered as source code in a suitable package, such as an archive written to a CD. Since deployment will now not take place, it was also decided to remove the brief user guide from the deliverables, as company management will have no use for it at this time.

As mentioned above, phase 2 of the project took longer than was initially predicted. The reasons for this were the unexpected complexity of some components required in the system, and the intended completion of the design and implementation stages during the January exam revision and examination periods. This led to the phase 2 implementation stage being completed approximately one month behind the initial schedule, so user testing was not started until the end of March. Feedback from testing was received during the Easter break, at a similar time to the report write-up commencing. At this stage it was decided to not implement any new features into the system, and to focus on the report, hence why phase 3 of the project is mostly only a discussion of new features and functionality.
Chapter 2 - Background Research

Before work on this project can commence, an investigation into the current data management practices at Driving Force UK will be made. A suitable methodology will be chosen for conducting the project, and some research into Human-Computer interaction will be made to ensure good practices are followed when designing and implementing the system.

2.1 Company Background

To assess the requirements of the company, it is first necessary to understand the current system of data storage for events. On November 25th 2006, a meeting was conducted with the Managing Director of Driving Force UK, to discuss the current system and assess what problems are experienced with it. At this meeting, sample documents from previous events were also supplied, to aid in comprehension of the data management practices.

After the meeting and following assessment of the sample documents, it was understood that data has previously been stored in multiple formats and locations, from spreadsheet and document files, to pieces of paper and notes. Whilst much effort is made to collate all data into electronic files, as different files were used for various purposes, problems arose as data became duplicated between files, and copying errors produced incorrect information. For example, when copying telephone numbers back and forth from emails, occasionally digits were transposed which, if left undiscovered, led to problems with communication during the event.

By creating a database application with each data item being stored atomically, commonly known as Boyce-Codd Normal Form (BCNF) [2], and used within a system, the possibility of semantic copying errors is eliminated. By then putting the application online, clients and suppliers can directly input data into the system, negating the need for paper copies of data and preventing inaccurate copying of information. All event data can be added directly to the database by a user, and retrieved by all other relevant users, without the risk of duplication errors. This ensures the cohesion of all data used within the event transportation company, and allows the management team to effectively manage the transport logistics without worrying about the consistency of data.
Another main benefit of creating an online database application for information storage and retrieval is the ability of the system to provide customised views of the data stored. When using separate data files and multiple sheets of paper, the coordinator has to frequently switch between them to get a proper overview of what is happening at any time. With a database application, it is sensible to create standard templates for data views which will be needed most frequently, but as an enhancement to the system, it could also be possible to allow the management users to refine the views to customise data output for individual events. Further to this point, it is important to understand the hierarchy of users within the system, something which will be critical in an access-rights system if implemented. The diagram below summarises the main stakeholders in a transportation booking system.

![Diagram](image)

Figure 1: Potential user hierarchy for a transportation booking system

One important constraint which must be considered is if a client or driver does not have access to a computer or the internet. Coordinators must be able to enter the appropriate information themselves, meaning that the system must be designed to allow this, by allowing a user at a higher level to perform all the functions available to the user level(s) below.
2.2 Project Methodologies

Many methodologies now exist for developing applications on the web, so to choose an appropriate style for this project, research into the various options is necessary. A relevant summary of this research will be presented below, followed by a justified choice, and a description of how the methodology will be applied in this project.

2.2.1 Methodologies Available

In [3] the first methodologies created are described as being based on the ‘waterfall’ model, which instructed developers to create systems in a linear step-by-step manner. The model states that before any work can begin on the actual software solution, a detailed feasibility requirements analysis must be conducted, and converted into a specification for the system. The software and all its features should then be thoroughly designed before implementation can occur. The finished product should then be tested, and any problems rectified, before being deployed in the appropriate environment.

This model was used successfully for many years, however eventually the limitations of the model became clear. The requirements and specifications for a project aren’t always fully known initially, so it can become apparent later during the implementation phase that the system design is flawed, which can often lead to project failure. These problems were addressed in later years, when evolutionary and incremental methodologies [3] were devised to develop systems faster, and with greater flexibility.

In [3] evolutionary methodologies are described as those where “the system is developed using a prototype and refined through user feedback of the system in use and changes in the application itself.” The main advantage to this is the constant improvement of the system to fulfil the requirements of the users. Source [4] however describes how prototypes can often be inefficient and difficult to maintain, and may not scale well to large systems.

Incremental methodologies were first applied in iterative rapid application development (RAD) models [3], whereby the waterfall model was repeatedly applied. The primary objectives of RAD include fast development of high quality solutions at a low cost [4]. This methodology is now in widespread use, and continues to be a highly regarded approach.
2.2.2 Choosing a Suitable Methodology

After conducting the research described above into methodologies, and understanding the requirements from the model company, it was decided to follow an evolutionary methodology with prototyping similar to the RAD model, to complete the project. Some UML will be used in the system analysis, and feedback on the prototype will be acquired from real users of the system. This methodology was chosen since the web development frameworks which will be used are relatively unfamiliar to the developer, and so an iterative approach will allow for comprehension of the framework features between iterations, and project risk will be controlled by implementing features on a priority basis. By creating prototypes at each stage, user feedback can be incorporated, and human-computer interaction issues can be addressed. The issues raised by [3] regarding prototypes is relevant, however as this project will only be undertaken by a single developer, the scalability issues can be disregarded.

The project will be broken down into 3 iterations necessary to achieve the aim. Before design of the booking system can commence, two pre-requisites must be fulfilled. Firstly the development environment must be decided upon, based upon research into web development frameworks, as detailed in phase 1 below. Then requirements analysis will be conducted on the model company, which will be used to create a specification and design for the software solution. Phase 1 will involve the research into web development frameworks, and disposable prototypes will be created to assess which framework is most suitable for the next phase. Phase 2 will involve requirements analysis, design and implementation of a prototype for use in testing, and to obtain user feedback. Phase 3 will entail further requirements analysis and enhancement design for developing the prototype from iteration 2, to improve functionality and to assess the expansibility of the system.

2.3 Human-Computer Interaction

As shown from the hierarchy established above, users with a large range of technical expertise will have the potential to use the system. As such it is important that the system is easy and quick to use, to encourage system reuse and effectiveness as a solution. To ensure this attitude is applied to the design of the system, Nielsen’s ‘10 usability heuristic principles’ [5] as shown in appendix B will be used as a basis for HCI design.
The 10 points stated have been considered and the principles of each point will be applied to the system produced in the following ways:

1/3 - Utilise widgets and features from a suitable web development framework;
2 - Effective research into model company’s terminology and business practices through meetings and studying of documentation;
4/9 - Modify system terminology if necessary after evaluating feedback from user testing;
5 - Utilise input validation features from a suitable web development framework;
6/8 - Ensure sound system design and self-descriptive features;
7 - Potentially implement wizards for routine tasks;
10 - Provide brief user guide for setup and use of system by company management.

The prototype produced in the second phase of the project will be tested with real stakeholders of the system, and their feedback analysed and considered in feature design for the third phase. Testers will be asked to give feedback on the usability of the system, and how self-explanatory the system’s features are to understand.
Chapter 3 – Phase 1

In this primary phase of the project, the processes behind building online systems will be researched and a small selection of web development frameworks will be chosen to investigate further. The two preferred choices from this selection will be utilised to create disposable prototypes, to gain an idea of how to code using the frameworks.

Once these prototypes have been created, the frameworks will be assessed, and the conclusions made will lead to a single choice of framework for use in the next phase of the project.

3.1 Development tools - Framework Research

In order to select two frameworks for creating the disposable prototypes, several frameworks must first be researched, to ensure that reasoned justification supports their use. For this it is helpful to understand how web development frameworks have evolved, and to find criteria for the selection process.

3.1.1 Building Web Applications

When the internet was first established as a public resource, websites merely contained static data which was of limited use. As it grew, it quickly became apparent that dynamic content could hugely increase the usefulness of the internet. Programming languages were already quite developed by this point, and so the first dynamic web applications tended to use familiar languages such as C and Perl [6], to generate HTML mark-up for web browsers to display.

As the complexity of web applications increased, this way of creating applications started to show its limitations. At this point Application Programming Interfaces (APIs) sprang into existence which built and improved upon preceding technologies. Eventually the limit of what could easily be done with APIs was reached by developers, and so a new way of publishing dynamic web content was required. Amongst many others, Java was a developed language by this point, and had proven itself in desktop applications, so it was extended to the Java Servlet API and eventually JavaServer Pages (JSPs) as a way of simplifying the building of dynamic content (see chapter 1.2 of [7] for a detailed explanation). These technologies were initially adopted by developers as they made building
applications simpler, however the main drawback was the lack of separation between template code and dynamic content generating code.

Around this time it became apparent that design guidelines were required to give separation of the user interface from the underlying data model. The Model-View-Controller (MVC) design pattern originally used in the ‘Smalltalk’ language for non-web applications [6 pp. 7], was adopted by web developers and modified to mesh better with the web application world, and emerged as what is popularly called Model 2. When Model 2 is applied to Java web applications, the MVC pattern is represented as such: the Model is represented by entity beans; the View represented by JSP or a similar HTML code generating system; the Controller represented by the Servlet.

“Model 2’s separation of responsibilities allows the various aspects of the application to be built concurrently. Thus, the Java developers can concentrate on building the models and controllers and the UI designers can build user interfaces.” [6 pp. 93]

3.1.2 Web Development Frameworks

The Model 2 design pattern is a perfectly suitable foundation for building scalable web applications, however as time progressed it became apparent that most web applications used the same key building blocks again and again. Design patterns for components and artefacts were drawn together into frameworks, which aided developers by supplying sets of pre-built parts, and allowing them to use them at a higher level of programming abstraction. Many of these web development frameworks have now been created, and most are open source and free.

For the building of the application in the next phase, a web development framework will be used to speed up application development, however a unique framework must be decided upon before design and implementation can begin. Several criteria were selected to assess the frameworks chosen, and the summary from the research into 5 popular frameworks is shown below.
<table>
<thead>
<tr>
<th>Evaluation Criteria</th>
<th>Django</th>
<th>Struts</th>
<th>Tapestry</th>
<th>TurboGears</th>
<th>Wicket</th>
</tr>
</thead>
<tbody>
<tr>
<td>DBMS support</td>
<td>Internal – Oracle/ MSSQL experimental</td>
<td>Hibernate support through J2EE environment</td>
<td>Hibernate support through J2EE environment</td>
<td>Internal – Uses SQLObject engine</td>
<td>Hibernate support through J2EE environment</td>
</tr>
<tr>
<td>IDE integration</td>
<td>Yes - ActiveGrid [8]</td>
<td>Yes - Eclipse</td>
<td>Yes - Eclipse</td>
<td>Yes - ActiveGrid</td>
<td>Yes - Eclipse</td>
</tr>
<tr>
<td>Hosting Technology</td>
<td>HTTP Web Server</td>
<td>Servlet</td>
<td>Servlet</td>
<td>HTTP CherryPy</td>
<td>Servlet</td>
</tr>
<tr>
<td>Language used</td>
<td>Python</td>
<td>Java</td>
<td>Java</td>
<td>Python</td>
<td>Java</td>
</tr>
<tr>
<td>Documentation</td>
<td>Thorough</td>
<td>Thorough</td>
<td>Thorough</td>
<td>Thorough</td>
<td>Thorough</td>
</tr>
<tr>
<td>Component Examples</td>
<td>Many</td>
<td>Many</td>
<td>Some</td>
<td>Few</td>
<td>Many</td>
</tr>
<tr>
<td>Initial application project provided by publisher</td>
<td>Yes - Including admin interface</td>
<td>No – only 3rd party available</td>
<td>Yes - jumpstart</td>
<td>Yes - quickstart (3rd party) [9]</td>
<td></td>
</tr>
</tbody>
</table>

Table 1: Summary of evaluation criteria for selection of a web development framework

3.1.3 Choosing a Suitable Framework

Following the research described above, it was concluded that Tapestry and Wicket are the two best contenders for implementing the software solution. The main reasons for this choice are detailed below:

- Django lacks full Oracle and Microsoft SQL Server compatibility, and as the model company had stated that their current database system is an Oracle DBMS, Django was not used, as problems may have surfaced later.
- The author has a strong background with Java, and the ‘Building Distributed Systems’ [10] module has given a good understanding of Java Servlets and JSP. This favours Struts, Tapestry and Wicket over Django and TurboGears, as the implementation and deployment techniques are already familiar.
Struts, Tapestry and Wicket can all be developed using the IDE Eclipse [11], which is familiar to the author, which aids greatly in coding during development. Web servers such as Tomcat can also be integrated into the Eclipse, allowing real-time testing of the application during development. Django and TurboGears can also use an IDE such as ActiveGrid, however the author has no prior experience with this.

An initial application project is very useful when using an unfamiliar coding system, and all framework publishers except Struts provide ready made projects to serve as a starting point for coding. There are third party ready made projects for Struts available however such as AppFuse [12], which can also be used for Tapestry.

To decide between Tapestry and Wicket, disposable prototypes were created in both frameworks to gauge ease of use and speed of development. Once these two prototypes were completed, the applications were tested to ensure smooth operation, and various modifications were made to assess the coding structures. Then based on the experience with both frameworks, one was selected to implement the booking system for Driving Force UK.

3.2 Implement Prototypes with Example Applications

The first step in the implementation of the two disposable prototypes was the setting up of the Eclipse IDE and Tomcat web server. To integrate Eclipse with Tomcat properly so that a ‘Dynamic Web Project’ can be created to mirror the folder hierarchy required, the Eclipse package complete with Web Tools Platform [13] was downloaded and installed. The Tomcat server [14] was also downloaded, and installed into the default directory. The steps outlined in the Tapestry Wiki [15] were followed to define the Tomcat installation within Eclipse, so that changes to project code would be automatically applied to the pages served by Tomcat, without requiring a server restart.

To assist in creating the prototypes, example projects for each framework were downloaded to assess the semantics of the programming styles. The example projects selected were:

- Tapestry jumpstart-min application [16];
- Wicket phonebook application [17].

The steps necessary to import these projects into Eclipse are detailed below.
3.2.1 Tapestry jumpstart

The Tapestry jumpstart project was downloaded as a compressed archive, and the contents extracted into a new folder. Within Eclipse a new ‘Dynamic Web Project’ was created to host the prototype, and named ‘jumpstart-min’. The files from the extracted ‘jumpstart-min\exploded\jumpstart-min.ear\jumpstart-min.war’ directory were added into the ‘WebContent’ folder under the new project, and files from the ‘jumpstart-min\web\src\java’ directory were added to the ‘src’ folder. It was then necessary to add the relevant jar libraries to the project, which was accomplished by following the instructions on [18], which uses Ant to retrieve the libraries from the internet. This completed the import of the project into Eclipse.

3.2.2 Wicket phonebook-1.2

The Wicket phonebook project was downloaded in an archive similar to the Tapestry jumpstart application, and the contents were extracted to a new folder. Within Eclipse another new ‘Dynamic Web Project’ was created to host the Wicket prototype, and named ‘wicket-phonebook-1.2’. The ‘images’ directory was added to the ‘WebContent’ folder, and the ‘web.xml’ file from the ‘WEB-INF’ directory was added to the ‘WebContent/WEB-INF’ folder. All files from the ‘WEB-INF\classes’ directory were added into the ‘src’ folder, and finally since the archive already contained all the jar libraries necessary for the project, all files from within the ‘WEB-INF\lib’ directory were added into the ‘WEB-INF\lib’ folder of the project. This completed the import of the project into Eclipse.

3.3 Test Prototype Applications

Once the Tomcat server was started with both projects configured to run on it, a web browser was started, and each project was accessed through the URL: http://localhost:8080/[projectname]. Correct operation of both prototypes was tested by logging into the Tapestry prototype and the creation of a new user account, and with the editing and filtering of data in the Wicket prototype. Both systems were fully functional as was expected, and worked suitably and quickly.
The next step in evaluating the prototypes was some modification of the application code, to gain a measure of the speed of development using the frameworks, and the simplicity of the coding structures used. Several types of modifications were made to each application, such as:

- Alterations to page template code to modify page output;
- Substituting form components for various types to assess the ease of component use;
- Modifying java classes to ‘break’ the application, to observe the complexity level of error messages produced.

The exact details of all of these modifications will not be discussed here as many minor adjustments were made to fully assess the frameworks, however the conclusions drawn from the testing will be discussed with examples below.

3.4 Framework Conclusions

It became apparent from the implementation of the prototypes that for each page to be implemented in Tapestry, 3 files are required: an HTML template file, an XML descriptor file, and a Java class file containing application logic. Wicket however, requires only 2 files: the HTML template file and the Java class file containing the application logic. This alone does not justify the use of Wicket, however this indicates that development in Tapestry should be somewhat more time consuming, due to the additional XML file required.

When making modifications to the prototypes, it was found that making alterations to the HTML templates was a simple process, since both frameworks render the final pages in a similar way. HTML tags with ‘jwcid’ or ‘wicket:id’ attributes are replaced with dynamic content when pages are rendered. This leads to the conclusion that a template page from either framework is editable with the full range of WYSIWYG HTML editors available.

The next types of modifications made were the substitution of form components within the prototypes, for example the swapping of a textbox for a dropdown box. With Tapestry, this was a fairly simple process of altering the HTML file ‘<input>’ tag to a ‘<select>’ tag, and changing the value of the ‘jwcid’ attribute from ‘@TextField’ to ‘@PropertySelection’. To specify the list of choices, a ‘model’ attribute was specified in the ‘<select>’ tag to use an Array of String objects which was created in a Java class to list the selection options. With Wicket, it was again necessary to swap the ‘<input>’ tag for a ‘<select>’ tag, however the ‘wicket:id’ attribute was left unchanged. Within
the Java code, the corresponding ‘TextField’ object was substituted for a ‘DropDownChoice’ object, which takes an extra parameter of a ‘List’ object to specify selection options. Although the semantics of the changes differs between the frameworks, the complexity level of the changes is equivalent, and it would seem that the basic usage of form components between the frameworks is very similar.

It was also evident that the syntax Tapestry uses in its tags within the HTML is somewhat more complex than equivalent tags in Wicket, since it uses a system known as Object-Graph Navigation Language (OGNL) [19]. This seems to allow Tapestry to be slightly more expandable and configurable, however for the types of components required in phase 2, Wicket seems fully capable of implementing all the required features.

For the final type of modifications made to observe error messages produced by the frameworks on exceptions occurring, several types of amendments to the HTML and Java code were made to view the various types of messages produced. For example tag names were altered within the template files, and getters and setters were removed to prevent proper functioning of the frameworks. Both frameworks produced detailed error messages within the web browser, with Tapestry on the whole producing somewhat longer diagnostic messages. Whilst this can be seen as an advantage in error analysis, the added detail can serve to slow the discovery of the problem. It was also observed that whenever possible, Wicket would provide a description of the problem in the code in real English sentences, allowing quicker resolution of the problem.

From the analysis of the prototypes stated here, it has been concluded that although both frameworks seem capable of similar feature implementations, development with Wicket should be a quicker process, due primarily to fewer files being required to render pages, and easier to comprehend error messages. Although it seems that Tapestry is slightly more configurable, the added complexity required in the coding structures indicate that the learning curve is greater than for Wicket, and as such Wicket will be the framework used for phase 2.

After this verdict, the persistence system used by the phonebook application was analysed further, to aid in the implementation of the back-end in the next phase. The file structure was modelled in a diagram which can be seen in appendix C.
Chapter 4 – Phase 2

In this phase of the project the web development framework Wicket, which was selected in the previous phase, will be used to create a software solution for the model company introduced earlier in chapter 1.3.

The first crucial stage in creating the solution is to model the company’s current dataflow, and gain a thorough understanding of the processes involved in transporting a client on a booked journey. This modelling will allow a requirement specification to be created, which will be used for the prototype design.

The next step involves designing the front and back-ends of the system, with some UML diagrams to describe aspects of the software design. In this phase the only features which will be designed into the system will be the ones necessary to fulfil the minimum requirements laid out in section 1.5. This design specification will then be implemented using the Wicket framework, and once the prototype has been constructed, testing will be undertaken to ensure its stability and functionality.

4.1 Business Modelling

On November 25th 2006, the author conducted a meeting with the managing director of Driving Force UK, to thoroughly understand the processes involved with the current system in place for booking vehicles, and the flow of data around the company. It was understood that the planning and execution of event bookings follows a very similar pattern for all events, which can be summarised as shown below:

1. Find and contract drivers and vehicles available for the duration of the event
2. Contact coordinators at client company to ascertain and store transportation requirements, and classify each booking as ‘allocated’, ‘ad-hoc’ or ‘airport transfer’
3. Approximately 1-2 weeks prior to the date of the first booking, begin preliminary assigning of drivers to jobs, and ensure that drivers are not ‘double booked’ and there is sufficient transport time between bookings.
4. Once transport requirements are finalised by client, and all initial bookings have been firmly allocated drivers, inform the drivers of their job details, and inform client of which drivers will be completing their bookings.
5. Ensure that any changes to bookings are relayed to drivers, and finish driver assignment to jobs, whilst informing all relevant parties of appropriate booking details.

6. On completion of the event, ensure all ad-hoc changes to bookings have been properly recorded, so that the final invoicing will be accurate.

As described in section 2.1, event data is currently stored in excel spreadsheets, with individual files being used for storing driver details and each client’s booking requirements. Any new software solution which replaces this current system must primarily support the storage of all event and booking details, and permit the assignment of drivers to bookings already stored. Of course all data stored within the system must be editable, so that updates to journey details can be made if necessary.

Once the driver assignments have been made, to improve over the current system of contacting clients and drivers to inform them of appropriate booking details, clients should be able to see which driver is allocated to each journey, and drivers should be able to get a clear list of all jobs they have been hard assigned to. All of this must be done with data confidentiality being considered, since clients and drivers should only be able to view details on bookings which apply directly to them.

4.2 Requirements Analysis

Following on from the business modelling discussed above, it is crucial to formalise the requirements for the software solution, so that an effective and appropriate design can be written and implemented in Wicket.

Initially the system must only allow users access on entry of a valid email address and password. Once logged in, a user should be able to create an event by entering the event details. Once an event is registered with the system, a user should be able to create bookings for that event by entering all relevant journey details. Thus the database must at a minimum, have tables to store details of users, events and bookings.

As discussed in the business modelling above, data confidentiality is very important in this system, so to facilitate this, users will be assigned to pre-defined groups. In this phase, for ease of implementation, the user hierarchy detailed in section 2.1 will not be created in its entirety; only three groups will exist: company management, client, and driver.
Clearly a company management user should have access to all events and bookings, and be permitted to create and edit user accounts as required. They should also be able to assign drivers to bookings, and specify if the assignment is soft or hard, depending on if the driver will definitely be completing the job. Thus company management users should have full system access.

A client user should be prevented from viewing or editing any user data, but still have access to view events and bookings. Client users should be registered to events they require transportation for, and only details of these events should be viewable. Clients should also be able to create new bookings, and only be able to view details of bookings made by themselves, and furthermore clients should not have access to any kind of driver assignment functionality.

A driver user should also be prevented from viewing or editing any user data. They should only be able to view events which they are registered drivers for, and should be able to view a list of all jobs for which they have been hard assigned to.

These requirements for the system form the basis for feature implementation in this phase of the project, and are shown diagrammatically in the Use Case diagram in Appendix D. The steps in planning and execution of event bookings, as described in section 4.1 above have been modelled for an online booking system in the Activity diagram shown in Appendix E. As such, under the ‘must-have, should-have, could-have, would-have’ (MuSCoW) method for requirement analysis, these features are all ‘must have’ requirements.

4.3 Design

For the design of the system, considerations were split into front-end and back-end design, which was an appropriate method to follow since Wicket allows for the separation of the presentation layer from the application logic layers. As we have stated in the requirement analysis, user login is necessary, and so it was decided that the Qwicket project [9], found while researching in section 3.1.2, will be used as a basis for the system. This provides a login interface for the system, session management code necessary for maintaining state, and it also handles database transactions within the application, through use of Spring [20] and Hibernate [21].
4.3.1 Front-end Design

For the front-end design of the system, a final end view of the system was considered appropriate to design rather than a fully technical structure and layout. This involved designing the page layout to be used in the system, and structure of input forms for getting relevant details. As described in section 4.2 various users of the system have differing levels of access, however company management have access to all features. This means by designing the page layout for management, a core system structure can be established, and feature access can then be restricted for lower level users.

Therefore the system should include a means to view users, events and bookings stored within the system which the current user is permitted to view. For this, it was decided to utilise the Wicket component ‘DataTable’ [22] for displaying the relevant data, which includes features for automatic sorting by attribute, and automatic pagination of data.

The system should also include forms for creating new objects of type User, Event and Booking. For these forms, standard text entry will be accomplished with use of the Wicket components ‘TextField’ and ‘TextArea’. Date entry fields will utilise the ‘DatePicker’ [23] component, which provides a popup JavaScript calendar for assisting date entry. Finally entry fields where the user must select from a discreet set of data will utilise the Wicket component ‘DropDownChoice’. These form components will be utilised to take input for each of the attributes shown in the ER schema shown below.

4.3.2 Back-end Design

One of the prerequisites for using Qwicket is access to an SQL server compatible with Hibernate, so as was stated in section 3.1.3 above, an Oracle database is required for the prototype to ensure compatibility with Driving Force UK’s existing database server. After visiting the Oracle website a free edition of Oracle’s SQL server was found, known as Oracle Database 10g Express Edition [24], which will be used in the implementation of the prototype.

For the back-end design of the system, an ER model was created to detail the attributes necessary to store for each data type, along with the relationships between the types of objects. As discussed above in section 4.2, there are 3 database tables that need implementing: user, event, and booking. The ER schema is therefore as such:
USER (userID, email, firstName, lastName, password, userLevel)
EVENT (eventID, eventName, eventAddress, startDate, endDate)
BOOKING (bookingID, bookingType, leadPassenger, contactNumber, carType, pickupDate, pickupTime, pickupLocation, dropoffDate, dropoffTime, dropoffLocation, pass, notes, eventID, userID, assignType, driverID)

Since we have stated that clients and drivers must be registered to events for booking purposes, and clearly multiple users can be registered to a single event, we also need a many-to-many join table between users and events as such:

EVENT_USER (userID, eventID)

The Qwicket application provides a ‘User’ entity object already for the purposes of logging in, however this needs to be extended to include the ‘userLevel’ attribute, to allow the assigning of users to groups. Another slight modification necessary to the Qwicket application was to remove the ‘Create a User’ link from the login page. This was because for the purposes of this software solution, a potential user should only be allowed access to the system once someone from company management has created an account for them.

4.4 Implementation

To describe the processes involved in the implementation of the system, the task will be broken down into the initial steps necessary to utilise the Qwicket project, and the main features of the system, represented here by the minimum requirements. Much of the component implementation followed a 3-step process since several of Wicket’s reusable components are used throughout the system.

The 3-step ‘analyse, design and implement’ process followed during the implementation involved:

1. Understanding the example code used to utilise the component;
2. Designing and considering changes necessary to adapt the component to the required use, including adding of variables and methods needed;
3. Implementing the changes to the code, and resolving all compilation and execution issues with support from Wicket’s detailed error system.
4.4.1 Installation of Database Server

The installer for Oracle Database 10g Express Edition was downloaded from Oracle’s website [24], and run. The default port for the http interface to the Oracle server is set as 8080, however as we want the Tomcat server to run on this port, it was necessary to change this to prevent the servers conflicting. By following the instructions on [25] the Oracle server’s http interface port was modified to 8090, to allow direct access to the database through a web browser which aids in development.

4.4.2 Initialisation of Qwicket Project

The initial setup for the project involved building the Qwicket project from the website [9] with the project name ‘EventBookings’. Once the archive was obtained, it was imported into a new Dynamic Web Project within Eclipse with the same name. All jar files necessary were added from the download links from the Qwicket website, and the Tomcat server was integrated with Eclipse as described in section 3.2 above.

Once the Qwicket project was imported into Eclipse, the Oracle server was accessed through the http interface using the URL http://localhost:8090/apex/. Through this interface, a new database user was created, and the ‘application.properties’ file within the project was updated with the appropriate database connection information, so that Hibernate could connect properly to the server. Once the database server connection was setup, the project was configured to run on the Tomcat server through Eclipse. Next the web server was started, and the project website was visited through Internet Explorer with the URL http://localhost:8080/EventBookings/, to ensure the project had been initialised correctly.

4.4.3 Displaying Event Data

One of the primary features of the system is the displaying of data currently stored in the database. As described in the design section above, the Wicket ‘DataTable’ object will be used to output appropriate data retrieved from the database. After investigating the example code from the Wicket website, it was determined that 7 files were required to fully utilise an automatically sortable and paging ‘DataTable’ object. These files are all viewable in the supplied source code, and specifically for the user data table, were required as such:
1. UsersPage.java – This creates a ‘DefaultDataTable’ object, and passes it the appropriate column definitions, and a new ‘SortableUserDataProvider’ object to provide the data to be displayed.

2. UsersPage.html – This provides the template for the page to be displayed to the user.

3. UsersPage$ActionPanel.html – This provides the template code for the action panel which provides links to perform operations on each table item.

4. SortableUserDataProvider.java – This provides the data for the table by utilising the ‘DatabaseLocator’ file to return data from the ‘UsersDatabase’ file.

5. DatabaseLocator.java – This provides a central reference to a unique copy of the ‘UsersDatabase’ object.

6. DetachableUserModel.java – This provides a detachable model for ‘User’ objects, so that when the ‘User’ objects are retrieved from the database, only a reference to undisplayed objects in kept in memory.

7. UsersDatabase.java – This retrieves the list of users from the database, through a helper ‘UserService’ object, and provides mechanisms for comparing ‘User’ objects for sorting, which the ‘DefaultDataTable’ object can use to automatically sort the data as requested by the user.

The same pattern of files was replicated for the output necessary for the event and booking tables, however the ‘DatabaseLocator’ object was reused between tables for providing a unique reference to the appropriate database accessing object. It may have been possible to consolidate these into a smaller number of files, however as this was the exact pattern used in the Wicket examples, it was assumed that this set was the optimal number required.

4.4.4 Storage of Event Data – Requirement 2

To store event data, it was necessary to implement the two extra tables ‘EVENT’ and ‘BOOKING’ as described in the design section above. It was observed that Qwicket uses a ‘User.java’ file to store ‘User’ objects in the system, and this file contains Hibernate annotations which map the object’s attributes to fields in the database table. So initially, two copies were made of the ‘User.java’ file, and renamed to ‘Event.java’ and ‘Booking.java’ respectively. Basic ‘String’ variables for each of the attributes in the ER schema above were created in each class, and appropriate getters and setters for each variable generated by Eclipse.
After following the guidelines from [26], the following code was used to implement the many-to-
many table join required in the design specification:

**User.java:**

```java
@ManyToMany
@JoinTable(name="EVENT_USER")
private Set<Event> events = new HashSet<Event>();
```

**Event.java:**

```java
@ManyToMany
@JoinTable(name="EVENT_USER")
private Set<User> users = new HashSet<User>();
```

Both variables also had appropriate getters and setters defined to retrieve the required ‘Set’ objects.

In addition to this, the ‘Booking.java’ file had three ‘ManyToOne’ joins defined to store the eventID,
userID, and driverID for each booking made.

### 4.4.5 Restriction of System Access – Requirement 3

As already described, the system will be used by three main groups of users: company management,
clients, and drivers. To restrict access to the various sections of the system, it was necessary to
implement a menu system which could adapt, dependent on the level of the currently logged on user.
To accomplish this, it was realised that the code necessary for displaying all of the menu options
needed to be coded into the HTML template file, and that the Java class could be used to control
which menu options were available at runtime.

After searching the web, a suitable JavaScript based menu system was found [27], which rendered the
menu based on the content of a table in the HTML file where it was used. By enclosing each menu
section within Wicket fragment tags in the HTML file, and associating each with a ‘Fragment’ object
in the Java class, it was possible to selectively choose which fragments were included in the final
rendered file, based on the level of the user currently logged in. This allowed the user control sections
to be restricted to only management users, and prevented drivers from adding or changing any data
stored within the system.
The next important step for preventing confidential or unnecessary data from reaching clients or drivers was the restriction of output displayed in data tables. As has been described in section 4.4.3, several files were required to display a data table; to restrict the output of the bookings table, it was necessary to modify the ‘BookingsDatabase.java’ file to check the ‘userLevel’ attribute of the currently logged on user, and call an appropriate method of the ‘BookingService’ object. For management all bookings are displayed, for clients only bookings made in their name are displayed, and for drivers only bookings where they are the hard allocated driver are displayed.

To restrict the output of the event table to only show events that the current user is registered to, it was necessary to modify the ‘EventsDatabase.java’ file. It was altered to pass the ID of the currently logged on user to the ‘getEvents()’ method of the ‘EventService’ object, unless the user was from company management, whereby the ‘getAllEvents()’ method was called instead. The ‘getEvents()’ method will return the current visible events for the user ID given by calling the ‘NamedQuery’ defined within the ‘Event’ entity object. This query is written in HQL, and forms an inner join to the ‘Event’ table, to only return registered events. The HQL query string used was:

’vevent e from Event e inner join e.users as users with users.id=?’

4.4.6 Assigning Drivers to Jobs – Requirement 4

The final step required to complete the minimum requirements is the assignment of drivers to jobs. This was again a multi-step process, which firstly required the ‘add booking’ screen to be extended to allow the assignment of a driver, and the choice of hard or soft allocation. The next step was to modify the bookings table to restrict the client to only view the assigned driver when they were hard allocated to the job. The final step was to only show the driver details of the booking when they were hard allocated to the job.

By the nature of the business model, only a company management user should be able to assign drivers to bookings stored in the database. To accomplish this, the ‘add booking’ screen needed to be modified to include two dropdown boxes to specify the type of assignment, and the driver to be assigned. The list of drivers was retrieved using an ‘AJAX form component update’ object linked to the event name dropdown box, which retrieves all drivers from the EVENT_USER table who are registered to the chosen event. This restricts the management user to only selecting drivers for a booking who are contracted to work for that event. As the same booking input screen is used for clients as well as managers, it was necessary to hide these options from a client when logged in. This was accomplished by applying a restrictive cascading style sheet (CSS) style to the HTML template if
the currently logged in user was not a management user, which set the ‘display’ CSS property of the dropdown boxes to none.

Next the ‘BookingsPage.java’ file was modified to make the bookings table only display the ‘Assign Type’ attribute if a management user is logged in, and to display the ‘Assigned Driver’ attribute only if a manager or client is logged in. The client should only see the name of the assigned driver however, if the data in the ‘assignType’ attribute is set to ‘Hard Allocated’.

The final step of only showing drivers jobs which they have been hard allocated to was accomplished by modifying the ‘BookingsDatabase.java’ file to call a specific method when the currently logged in user is a driver. This method calls the ‘NamedQuery’ defined in the ‘Booking’ entity object, which only retrieves bookings where the driverID is equal to the value passed to the method, and the ‘assignType’ attribute is ‘Hard Allocated’. The HQL query string used was:

from Booking b where driverID=? and assignType='Hard Allocated'

4.5 Testing

Testing on the system was conducted in two ways:

- The system interface was used to populate the database with some test data from Driving Force UK’s event archives, for the testing stage of the software engineering process;
- Driving Force UK were contacted again to request feedback on the system, which had been previously mentioned to them in the requirements meeting.

4.5.1 Database Population Testing

To thoroughly test the system’s capability for storing event data, the system was used to input portions of previous event data from the 2006 Farnborough Airshow, kindly supplied by Driving Force UK in the initial requirements meeting for this purpose. One management account, three client accounts and three test driver accounts were created, and an event was created to store the bookings against. Using the client accounts, several bookings from the Airshow were created in the system, and once all relevant data entered, the management account was used to assign drivers to the bookings. Once complete, the driver accounts were checked to ensure each displayed the correct data, and the client accounts were checked to ensure they could view the driver assigned to each booking made.
The test was a complete success: all data was successfully entered into the system, and after logging in with the client and driver accounts, only the relevant data was displayed to each user.

4.5.2 Introduction to User Feedback

Once the implementation of the minimum requirements was completed, the MD’s participation in testing, and feedback from real clients and real drivers, as well as company management, was very useful in evaluating the prototype. So the database was cleared of all data, and following a phone conversation with the MD, one management account was setup for himself, and brief instructions were given to initiate testing of the system.

Firstly the MD was asked to login with the supplied credentials, and create a secondary management account for another coordinator within his company. He was also asked to contact a client and driver with good company relations, and create accounts for them in the system. Once these user accounts were created, the MD was asked to create several events in the system, and assign the client to one or more of them, to permit them to make bookings. He was then to instruct the client to make several example bookings, and report back once complete. Once the bookings were made, the MD instructed the other company coordinator to edit the bookings, and hard and soft assign the driver to some of the jobs, and then contact the driver to ask him to login and use the system to retrieve job information, with the user account created for him previously.

It was requested of the MD to provide feedback of his experiences using the system, and also request some feedback from his coordinator, and the client and driver asked to participate in the testing. The following sections detail the feedback received from each user of the system, and some suggestions from the author for resolving issues experienced.
4.5.3 Feedback from MD

The MD firstly praised the system’s intuitive and simple login system, and commented that the menu system was easy to navigate. After he had experimented with creating and manipulating user accounts, he commented that each user should also include a telephone number field for quick retrieval of contact information. He also noted that the ‘edit’ and ‘delete’ links for each user were very close together, and that as there was no delete confirmation box, it would be very easy to accidentally delete a user from the system when attempting to edit their details. These issues are easily resolved with the addition of a ‘phoneNumber’ field to the ‘User’ table, and appropriate textboxes on the ‘add user’ screen. The accidental deletion issue can be resolved with the implementation of a popup confirmation box when clicking the delete link, using the Wicket object ‘SimpleAttributeModifier’, as is described in the framework documentation in [28].

The next comment that the MD made was after creating events in the system. He stated that when entering event details, it should be possible to appoint someone from within the host company as the lead contact regarding an event, and there should be extra fields for entering notes regarding the event. Both of these event attributes can easily be added to the database schema, and corresponding text fields created within the ‘add event’ screen.

4.5.4 Feedback from Company Coordinator

The company coordinator chosen for providing feedback on the system in the allocation of drivers to bookings, firstly commented that occasionally multiple drivers are required for a single booking, if the total number of passengers making that journey exceeds the capacity of the car type required. This enhancement to the system is somewhat complex, and the implementation of it will be discussed further in phase 3.

The second comment made by the coordinator was that for each booking made, the client should be asked to tick an ‘accept terms and conditions’ checkbox, complete with a link to popup the relevant terms and conditions. This is very important for legal reasons, as for example a late booking cancellation may incur a charge. The implementation of this feature requires only the simple adding of a Wicket ‘CheckBox’ component to the ‘add booking’ screen, which must be ticked to add the booking.
4.5.5 Feedback from Client

The client asked to participate in testing was satisfied with the system functionality itself, and found the ‘add booking’ screen to be comprehensive in the details requested. The main limitation she found in the system which was observed was the fact that after completing the input of bookings into the system, it was required to contact the Driving Force UK again to confirm that the bookings had been made. It was commented that in a real life situation, many bookings would often be made and that manual notifications would become very time consuming, so some form of an automated alert system for the company coordinators would greatly assist the booking procedures.

It was concluded by the author that it was possible to implement this feature in one of two ways. The first would be on completion of a new booking, or a change to an existing one, an email would be sent to the appropriate coordinator to inform them of this fact, however this could generate many emails to sort through. The second, and potentially better method, would be to inform the appropriate coordinator with new bookings or changes to existing bookings upon login to the system, by some form of a ‘new changes’ list on the home screen. These feature enhancements and some possible implementation techniques will be further discussed in phase 3.

4.5.6 Feedback from Driver

The driver asked to participate in user testing first commented that the system seemed very user-friendly, even to an inexperienced computer user such as himself. He stated that the functionality of the buttons on offer was clear, and it took very little time to become familiar with the features. The main drawback he found in the system however was in the presentation of the jobs assigned to him. He commented that the tabular list of data was not easy to peruse, and that when he attempted to print the list, the formatting meant that individual job details were split across pages. He stated that once assigned to a job, it would be extremely helpful to simply print off the details of the bookings, in an easily readable fashion, so that he could take the printout with him in his car.
To accomplish this within the system, there are three choices of implementation which could be performed. These are redesigning the data table to fit on a single printed page, or creating a new screen in the system to summarise each booking in a clear way. And finally it would be useful to download the data stored in the table into a file format (such as CSV), which can be imported into another program such as Microsoft Excel, and formatted to fit on a single page with clear formatting and highlighting. The first two options are fairly trivial exercises in modifying the current implementation, however the data outputting implementation is more of a challenge, which will be discussed further in phase 3.
Chapter 5 – Phase 3

In this phase of the project, enhancements to the prototype created in phase 2 will be assessed and designed, and the compatibility of the prototype with another database server will be tested. The purpose of this phase is to help consider the complexity of continuing the software development process in Wicket, by assessing the enhancements possible to the prototype created in phase 2.

5.1 Additional Requirements Analysis

To improve the prototype from phase 2 for event data management, several enhancements described in the prototype testing (see section 4.5 for details) could be implemented, and are listed below:

1. Delete confirmation;
2. Event contact details and notes;
3. ‘Accept Terms and Conditions’ checkbox on new bookings;
4. Multiple drivers assigned to an individual booking;
5. New booking alert system for management;
6. Printing and exporting of bookings.

These features all improve the suitability of the system for Driving Force UK’s requirements. The justification and reasoning behind each of these features has already been described in phase 2 testing however, so will not be mentioned again in this requirements analysis.

As well as these system enhancements, several other useful features were previously mentioned in the mid-project report, as part of the ‘should have’ and ‘could have’ functionality. The reasoning for each of these features will be described below, and designs for the implementation of each will be shown in section 5.2. A selection of these which will be analysed here are:

7. Listing of individual driver’s job details for management to print;
8. Email driver with job details;
9. Providing driver with map views.
One further consideration was the interoperability of the prototype with varying database servers. The prototype from phase 2 was designed to work with an Oracle 10g Express Edition database, however in reality, the host company for this booking system may use any type of server. Therefore in this phase, another database system will be linked to the prototype, to ensure the systems cross-compatibility.

5.1.1 Listing of a Driver’s Jobs for Management

Throughout phase 2 of the project, we have made the general assumption that all drivers who are contracted to work on an event have access to a computer with the internet, and are comfortable using online systems. However as was mentioned in section 2.1, this assumption does not hold in the real event transportation industry, as drivers come from a large range of backgrounds and age groups. To accommodate this limitation in the prototype already created, it should be possible for a management user to retrieve a list of jobs specific to an individual driver. The list shown should be printable, so that as required the printout can be sent or faxed to a driver without access to the system themselves.

5.1.2 Email Driver with Job Details

As has been described previously, when a driver has been hard assigned to a booking, the job details appear on the system for them to read and carry out. This process however requires the driver to sign in to the system on a regular basis to retrieve this information, since there is no indication to the driver of when new jobs have been allocated to them. Since it is not very practical to instruct each driver to check the system on a frequent basis, the system is required to alert drivers when they have been allocated to a new job. This alert would ideally take the form of an email sent to the driver’s registered email address, informing them of the details of the new booking, and ideally an alert would also be sent upon changes to an existing allocated booking.
5.1.3 Providing Driver with Map Views

It was mentioned in the original requirements gathering meeting with the MD of Driving Force UK, that one problem occasionally experienced during events was that of drivers getting lost, or not being completely familiar with the exact pickup or drop-off location. Some drivers now utilise GPS systems within their car to assist in these problems, however many still do not have this facility. To aid these drivers without a GPS system, a map facility built into the software solution would be of great help, showing the pickup and drop-off locations in detail, and an overview of the journey to be made.

The now popular Google Maps API service [29] has been considered for use in the system for this purpose. However there is a restriction with using this service, as routing data is not available to use through the API. Thus Google Maps is only able to provide visualisation of the pickup and drop-off locations. To include a route map, it is necessary to use a secondary API such as MapQuest [30], which will provide this data. This could be used in conjunction with Google Maps, or all mapping could be done through this API instead.

5.2 Enhancement Design Considerations

Each of the 10 feature enhancements described in the additional requirements analysis above have varying degrees of complexity. The design considerations for features 1, 2 and 3 have already been addressed in the phase 2 testing section, so only the other 7 features will be considered here in the various design subsections below.

5.2.1 Multiple Drivers Assigned to a Single Booking

The prototype created in phase 2 was designed to only allow a single driver to be assigned to each booking, however as discussed in section 4.5.4 occasionally multiple drivers are required for a single journey. As there could be any number of passengers making a journey, the system should be designed in a way whereby an unlimited number of drivers can be assigned to a booking.
To accommodate this change, there needs to be a many-to-many table join created in the database schema between the booking and user tables. The table should take the form:

**BOOKING_USER** (bookingID, driverID, assignType)

which allows each driver assigned to a booking to have their own individual assignment type. The code necessary for this is very similar to that used in the ‘EVENT_USER’ many-to-many join table as shown in section 4.4.4. To facilitate the adding of additional drivers, the booking details screen must also be extended. By default it should contain one set of assignment fields (assignment type and assigned driver), but a new link should be included, which upon clicking inserts another set of assignment fields into the page. This link should be clickable any number of times, to allow as many drivers to be assigned to a booking as necessary. Once the booking is saved, the form data should be parsed, and all assignment details added to the ‘BOOKING_USER’ table.

5.2.2 Booking Alert System for Management

It was commented during user testing in phase 2 that the manual notification of new bookings was not practical in real life situations, as the act of the notification itself would become very time consuming. Thus as described in section 4.5.5, the ideal solution to this problem would be an automatic alert feature built into the system, so that company management can view new bookings and changes to existing bookings upon logging in.

To facilitate this, a form of a logging system is required in the application. Whenever a new booking is created, or an existing booking is amended, the system should write a new entry to a log table. This entry should include the booking affected, the user who made the change, the date and time of the change, and what type of change was made (new or changed booking). This can be accomplished by creating another database table of the form:

**LOG** (logID, bookingID, userID, changeDateTime, changeType).

It is also required that a new attribute is added to the user table, which contains the date and time of the user’s last login. Using this data, it is possible to show a company management user the latest changes, by retrieving log records with a date and time more recent than the last login, and displaying the results to screen on the user’s homepage in some form of a data table.
5.2.3 Printing and Exporting of Bookings

To facilitate the clear printing of booking details for drivers, as described in section 4.5.6, the two methods outlined can be easily implemented by redesigning the data table into a more compact form, or by creating a new output screen available for each booking. To make the exporting of event data possible however, a somewhat more complex design must be implemented.

After searching the Wicket examples on the web, a suitable example was found [31] which contained details of the exact type of data export required here. Essentially a hyperlink on the page is replaced by an ‘ExportLink’ object which takes as a parameter the data from an ‘ArrayList’ object which is used to populate the table. Several types of file can be exported by utilising various data classes, downloadable from the examples page (CSV, Excel, and XML). This implementation fits in very well with the current data table system in use, so this feature should not be difficult to implement.

5.2.4 Listing of a Driver’s Jobs for Management

As has been detailed in section 5.1.1, not all drivers will be able to access the booking system, so an alternate method of informing these drivers of their jobs is required. The simplest way of extending the current system to accommodate this requirement is to provide filtering on the booking table, so that output can be restricted to a specific driver, chosen from a dropdown list. Once only that driver’s jobs are shown on screen, the page can be printed, and sent to the driver in question.

To implement this table filtering, the Wicket phonebook example studied in phase 1 (see section 3.2.2) was observed to use ‘ChoiceFilteredPropertyColumn’ objects in the columns ArrayList instead of the standard ‘PropertyColumn’ objects, to permit filtering of that column. In addition to this, the ‘SortableBookingDataProvider’ class must implement the ‘IFilterStateLocator’ interface to allow the table data to be sorted. These changes in implementation are easily accomplished, so it should be straightforward to include this feature in the prototype.
5.2.5 Email Driver with Job Details

As has been detailed in section 5.1.2, not all drivers can be expected to frequently check the system for updates. Thus an automated email system alerting drivers to new bookings and changes should be implemented to compensate for this. Since email functionality already exists within the system from the Qwicket framework for sending users forgotten passwords, the mailing libraries and protocols are already in place.

To utilise this system to email drivers details of new bookings, the ‘UserServiceImpl’ class must be extended with a new method to send an email. The framework of the ‘emailPassword()’ method can be duplicated and modified to take a ‘Booking’ object as a parameter, and email the assigned driver the relevant details extracted from the object. This method should be called when saving a booking, once a driver has been hard assigned to it. This feature implementation is a relatively simple process, and will utilise only classes and libraries already contained within the system.

5.2.6 Providing Driver with Map Views

As has been described in section 5.1.3, it would be of great help to drivers to see and print the pickup and drop-off locations on a map, and the whole journey if possible. The Google Maps and MapQuest API’s were considered as suitable services to use for this feature.

The Wicket Stuff library contains a ‘GMap’ object [32] which can be used to insert a Google Map into a Wicket page with ease. Following the ‘GMap’ examples from the same source, it is a simple process to integrate appropriate maps into the prototype system for drivers. However as discussed previously, Google Maps does not provide routing data through the API, so if a journey map is required, the MapQuest API must be used.

The MapQuest API however does not have prewritten support in Wicket, so the user guide from the website [30] was studied to understand the implementation method. To utilise the API, a script must be imported to include the OpenAPI JavaScript library, and the map objects are created within JavaScript using ‘MQMap’ objects. This is possible to accomplish in Wicket, using JavaScript injection into page components, however this procedure has not been employed in the prototype from phase 2, so to utilise the MapQuest API, a relatively large amount of work will be required in contrast to the Google Maps API.
5.2.7 Cross-Compatibility of System with Database Servers

The last consideration considered in section 5.1 was the ability of the system to utilise other database vendor’s servers. Currently with the prototype created in phase 2, the ‘application.properties’ file (part of the Qwicket framework) contains all the database connection specifications necessary for Hibernate to create a session. To use another database system, this data must be changed, so to utilise a MySQL database server, the following connection data must be used instead:

```
hibernate.driver=com.mysql.jdbc.Driver
hibernate.url=jdbc:mysql://localhost/db
hibernate.user=dfuk
hibernate.password=kufd9
hibernate.dialect=org.hibernate.dialect.MySQLDialect
```

Once this data has been entered into the ‘application.properties’ file, the server must be restarted, and if the connection data used is correct, the system should create all tables necessary for the application.

5.3 Enhancement Implementation and Testing

Due to difficulties experienced whilst undertaking the implementation of phase 2, little time was left to implement new ‘should have’ and ‘could have’ features into the prototype system created. Thus the only enhancement which was attempted in the system was the swapping of database systems, as described in section 5.2.7 above.

Once the changes to Hibernate’s connection data were implemented as described, the system was started successfully. Direct access to the MySQL server was necessary through a command prompt, to create a single management user account so that system login could be attempted. This was completed using the SQL query string:

```
"insert into users (email, password, firstName, lastName, userLevel) values ('test@test.com', 'secret', 'Joe', 'Bloggs', 1);"
```

Once logged into the system using these credentials, a new user, event and booking were created. All data entered was checked against the data retrieved from the database through a command prompt, to ensure no discrepancies existed. This testing was successfully completed with no discrepancies, leading to the conclusion that the system should be easily transferable to any database server compatible with Hibernate.
Chapter 6 – Evaluation

6.1 Evaluation Criteria

To fully evaluate the project, several factors must be taken into consideration, including:

- How did the frameworks investigated in phase 1 perform?
- Was the software solution produced in phase 2 appropriate for the project aims?
- Did the project go according to plan, and was the methodology used successful?

These factors for evaluation justify the following criteria, used to quantify the success of the project against the initial goals:

- Suitability of the choice of framework from phase 1;
- Achievement of the minimum requirements;
- System correctness, maintainability, integrity and usability;
- Adherence to the project schedule;
- Effectiveness of the methodology used.

6.2 Evaluation of Frameworks

Much was learnt from the research and investigation into frameworks conducted in phase 1 of the project. The search for relevant written literature on the subject was challenging however as the frameworks being investigated are relatively new and constantly changing. Only one book exists for programming in Wicket [33], and only two books were found to exist with guidance for Tapestry [7; 34].

These books helped in the comprehension of the framework features and components for phase 1, and the Wicket book aided considerably in the implementation of phase 2, for example by detailing the use of ‘models’ which are a critical part of all Wicket pages containing input forms. Without the thorough explanations given in the book, development of the input screens and other features within the system may have taken significantly longer.
Much knowledge on these frameworks however was acquired through the homepages of the relevant framework, or from other independent sites giving tutorials or reviews of them. It was through independent sites such as these [35], that the original 5 choices of frameworks were chosen.

6.2.1 Evaluation of Tapestry

After implementing the Tapestry jumpstart application in phase 1, it was observed that Tapestry works with a templating system whereby static content is defined in an HTML file, dynamic content is generated by a Java class file, and an XML file ties the components of the two files together to render the final HTML page shown to the user with dynamic content. This triple file system gives the developer great control over the content of pages, and even includes other languages such as OGNL [19] for specifying object references.

One of the criterions used for selection of the frameworks in section 3.1 was the availability of examples exhibiting the various features and components at the developer’s disposal. Whilst there were several example applications available with source code [35], there seemed to be a lack of component by component examples, thus much more comprehension of source code would be necessary to extract the relevant code for integrating various types of component into the prototype.

Tapestry does seem a very capable and sophisticated framework, and it seems to be very popular on the web from the abundance of examples and tutorials discovered during research, likely due to its high extensibility and productivity. However as little previous experience existed with these types of framework, it was opted not to use Tapestry in the prototype application written in phase 2. This was because from the conclusions drawn in section 3.4, the complexity of Tapestry led to the judgment that the learning curve in familiarisation with the framework would be too great to complete the minimum requirements in the time available.

6.2.2 Evaluation of Wicket

The Wicket phonebook application (see section 3.2.2) gave a good example of implementing a data table with sortable columns and automatic pagination. After the inspection of the code it was deemed capable enough for the next phase in that it contained the necessary components for the booking system, yet it was simple enough to allow development to continue at a decent pace, and allow the minimum requirements to be met.
For the prototype application written in phase 2, several components were utilised, such as ‘DatePicker’, ‘DataTable’ and ‘DropDownChoice’. Once the source code had been studied from the Wicket library’s example applications, it was not too challenging to implement the components within the prototype. The most time-consuming part of this phase was with linking the components to the persistence layer provided by Spring and Hibernate.

By using the Qwicket application to rapidly start development of the system, most of the ‘boilerplate code’ for using the Spring and Hibernate API’s was already written, such as database schemata building, and Hibernate session management. A generic ‘Create, Read, Update and Delete’ (CRUD) backend exists in Qwicket, in the form of ‘Data Access Object’ (DAO) classes with public methods, which were used to link the stored data to the components implemented. These DAO classes utilised the Wicket-Spring API, an extension to the Wicket framework, which integrates the Spring business layer into the Wicket pages “in a safe, non-intrusive and consistent way” [37].

I believe Wicket was the better choice of framework over Tapestry, since it was relatively easy to implement the various features and components of the prototype, once a suitable example had been found and the source code understood. Whilst Tapestry is capable of implementing the same types of features and components as Wicket, the time necessary for comprehension of the relevant examples would be greater, as was discussed in section 6.2.1 above. Without the added complexity of an XML file per page, I also believe that development was somewhat quicker with Wicket, but without functionality being compromised. Whilst there are inevitably shortcomings in Wicket’s capabilities, no issues were experienced with it in this project, likely due to the relatively low complexity of the prototype produced.

6.3 Evaluation of Software Solution

The criteria for evaluation of the software solution involve checking if the product achieved minimum requirements, and assessing feedback from potential users. In [38] several other metrics are stated which will also be used in measuring the quality of the system produced, which are: correctness, maintainability, integrity and usability. These metrics will be addressed below in sections 6.3.1 to 6.3.4.
The software solution produced in phase 2 fulfilled the three minimum requirements relevant to the software solution produced, so by this assessment criterion, the solution can be considered appropriate to the project aim. After real end-user testing, the feedback received regarding the system was highly positive, and the company management stated that the system fulfilled all the data storage requirements necessary for any size of event. Although it was commented that the prototype would not be suitable in a real-world situation until some improvements were made, given more time the features suggested coupled with the other enhancements discussed in phase 3 could be implemented, and the system produced should be deployable at Driving Force UK for use.

The prototype produced did however show the great potential in the system for improving the efficiency of the flow of data, by primarily storing all booking details only once, and secondly by allowing direct access to the data by client and drivers, thus reducing the workload on company management. By handling all data additions, requests and modifications, the system allows the management and coordinators at Driving Force UK more time to deal with other issues concerning the event, such as finding and contracting additional drivers, and familiarisation with the area local to the event.

6.3.1 System Correctness

This metric is concerned with how accurately the system performs the required functionality. In [38] it is stated that “The most common measure for correctness is defects per KLOC (thousand lines of code), where a defect is defined as a verified lack of conformance to requirements”. Since the system has not undergone versioned release, the only defects found were occasional small semantic programming errors, which were resolved during the course of the prototype coding. Thus as was stated above, the system has achieved the minimum requirements laid down in section 1.5, so although it is not possible to classify the correctness in a numerical sense since there is no verified lack of conformance to requirements, it is fair to say that the system has a high degree of correctness.
6.3.2 System Maintainability

This metric is concerned with the ease of maintenance of the system, such as repairing defects, implementing new features, and altering the operating environment. A common metric used is mean-time-to-change (MTTC), which measures the time taken from the analysis of a change, to the modification being distributed to all users. Since only prototypes have been made for the system however, this metric cannot be measured in this case. Phase 3 discusses some of these maintenance issues in depth though, to assess the challenges of altering the prototype created in phase 2.

As stated in section 6.3.1 above, the defects observed in the prototype were quickly and easily repaired during the implementation stage, once the cause of the problem was understood. The implementation of new features should not pose an issue for the author due to the abundance of examples available for Wicket. As has been described in sections 5.2.3, 5.2.4 and 5.2.5, the enhancement of the prototype with these new features should be a simple task, since the existing implementation methods facilitate the inclusion of the relevant Wicket examples. The time to implement some new features and components however may be substantial such as in sections 5.2.1, 5.2.2 and 5.2.6, but by following the ‘analyse, design and modify’ method of implementing components used in phase 2, it should be possible to incorporate the features in a reasonable timescale by the original author of the phase 2 prototype.

Finally, altering the operating environment is an easy maintenance task with Wicket, as environment definitions are generally stored in external files. An example of this was performed in section 5.2.7, by altering the ‘application.properties’ file to access a database server from a different manufacturer. The switch from using an Oracle server to a MySQL server was quick and simple, and the change did not affect application performance or any features. Thus as the system has been constructed using the Wicket framework integrated with Spring and Hibernate, alteration of the system environment is an easy and quick procedure.

6.3.3 System Integrity

This attribute of the system measures the ability with which it can withstand an attack meant to disrupt normal operation. Since it was not one of the original requirements, nor an aim of the project to create a secure system, this metric is not of great importance, however it is still relevant to any web application. The Wicket framework takes care of maintaining state automatically, so to quote [39]:

- 43 -
“Wicket is secure by default. URLs do not expose sensitive information and all component paths are session-relative. Explicit steps must be taken to share information between sessions.”

There also exists an additional Wicket module, known as wicket-auth [37], which can be used for increasing the integrity of the system, through use of per-component authorisation based on roles. This module has not been used in the prototype written in this project, as high security was not one of the original aims set out. Nonetheless if further system development was undertaken to create a fully functioning application for release, this module would be studied in order to integrate it with the application being developed.

6.3.4 System Usability

The usability of the system is a hard metric for the developer of a system to assess, since through designing and implementing the system, a thorough and comprehensive understanding of the functionality is obtained. Therefore only through real user testing can an evaluation of the usability be made, which is what was conducted in the testing section of phase 2 of the project.

From the feedback received, it seems that the system functionality was easily understood, thanks to large self-descriptive buttons, and an easily navigated menu system. From direct user observation when asking another unrelated individual to use the system, the user quickly became familiar with the system, and in a short period of time became comfortable with all the features available. Thus the system can be evaluated as having a high usability, since all functionality is clearly available to the user, and simple to operate.

6.4 Evaluation of Project

The project achieved all of the minimum requirements set out in section 1.5, so the project was successful in this consideration. All of the objectives listed in section 1.2 have also been completed, which leads to the conclusion that the project aim has been accomplished. A firm understanding of a range of web development frameworks has been gained from the research conducted in phase 1 of the project, and a thorough appreciation of the use of Wicket has been acquired through its use in phases 2 and 3. A standard system development process used throughout phase 2, in this case the waterfall model, has been successfully followed in the Wicket framework, and enhancement considerations and
designs detailed in phase 3, shows the ability of Wicket for continuing with an evolutionary prototype approach.

As written in the mid-project report, the original project objectives included deployment, which was initially expected to take place when the prototype was complete. It was previously assumed, based on incorrect information being given, that Driving Force UK’s servers already hosted an Oracle database server, which was to be used to store company data on the already configured host. During a site visit as part of the research conducted in phase 2 of the project, a check was made on the servers and it was found that no Oracle database server existed. It was determined that the installation and setup of an Oracle database on the company’s servers would be outside of the scope of this project, so the decision was made to not undertake deployment, as the time spent on installing and configuring a database server as well as the web servlet environment would be detrimental to the project as a whole. It is the opinion of the author that the project aim has not suffered as a result of this, since deployment of the web servlet is unrelated to application development using the Wicket framework.

The methodology followed in this report of breaking the project down into three iterations has seemed to work well given this situation. Background framework testing was completed in phase 1; a functioning prototype for Driving Force UK’s requirements was produced in phase 2; enhancements to the prototype were considered in phase 3, with one design consideration being implemented. By following the MuSCoW method of requirement analysis, risk was successfully managed, and the implementation of features into the software solution was split apart into the relevant phases. Due to some difficulties experienced during the comprehension of the use of the Wicket framework, such as the linking of components with Spring and Hibernate, the implementation of the prototype in phase 2 took longer to complete than was initially expected. These time constraints led to only the ‘must-have’ features being implemented in phase 2 of the project, however the detailed analysis of the unimplemented features in phase 3 would allow a developer with more time to implement the additional features and create a finalised product. By breaking the project into phases, the time constraints led to a workable prototype still being produced, allowing feedback to be obtained. This is far preferable than producing a semi-complete application, with several features left partially implemented, and unable to be tested properly.
Chapter 7 – References


Chapter 8 - Appendices

8.1 Appendix A – Project Reflection

I feel this project achieved my personal objectives of familiarisation with the latest generation of web development frameworks, and in particular I have gained a deep understanding of the Wicket framework.

The hurdles I discovered along the way prevented me from fully completing my initial objectives of reaching full system testing and deployment, however as this was somewhat of a superfluous requirement, since these phases of the software engineering process are not specific to the web development framework used, I do not feel that my project has deteriorated because of it. These objectives were there to give the application a finished appearance, however in reality a ‘finished’ system can take many months to complete, with ongoing user-testing and code refinement, far outside the scope of this project. I am very satisfied with the iterative methodology which I followed in the project, which still allowed me to reach the user testing phase, even though the hurdles I encountered were potentially damaging to the project’s goals.

I learnt much from my experiences with the host company, Driving Force UK, since I was told they had an Oracle database server running, and so based some decisions on this incorrect information. This has made me realise that I should not assume that information from inexperienced computer users is necessarily true, and that I should have investigated the situation myself thoroughly at an earlier stage of the project.

One further issue I experienced during the project was that I did not manage to implement as many of the system enhancements in phase 3 as I had planned to earlier in the project, for two primary reasons. The programming of the phase 2 prototype took longer than expected, as it was assumed based on the Wicket website’s blurb that programming in Wicket is a very simple process, however with no previous experience in the framework, it took more time than predicted to get to grips with the programming style. Also the original timescale assumed much work would be done on the project at the end of December and throughout January, however due to the need to do exam revision, the project was somewhat neglected during this time. In hindsight, my goals were somewhat unrealistic, and I should have aimed to complete fewer tasks, which if completed quickly could have been expanded upon later in the project, rather than reducing the project goals at a later date.
8.2 Appendix B – Jakob Nielsen’s Ten Usability Heuristics

Jakob Nielsen’s Ten Usability Heuristics [5]

1. **Visibility of system status**
   The system should always keep users informed about what is going on, through appropriate feedback within reasonable time.

2. **Match between system and the real world**
   The system should speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.

3. **User control and freedom**
   Users often choose system functions by mistake and will need a clearly marked "emergency exit" to leave the unwanted state without having to go through an extended dialogue. Support undo and redo.

4. **Consistency and standards**
   Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.

5. **Error prevention**
   Even better than good error messages is a careful design which prevents a problem from occurring in the first place. Either eliminate error-prone conditions or check for them and present users with a confirmation option before they commit to the action.

6. **Recognition rather than recall**
   Minimize the user's memory load by making objects, actions, and options visible. The user should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate.

7. **Flexibility and efficiency of use**
   Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user such that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.

8. **Aesthetic and minimalist design**
   Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.
9. **Help users recognize, diagnose, and recover from errors**

   Error messages should be expressed in plain language (no codes), precisely indicate the problem, and constructively suggest a solution.

10. **Help and documentation**

    Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Any such information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.
8.3 Appendix C – Phonebook Application

Below the file structure used within the phonebook application’s persistence system (see section 3.2.2 for details) has been modelled, to gain an understanding of how the various files interact.

Figure 1: Phonebook application persistence file structure
8.4 Appendix D – Use Case Diagram for Phase 2

Figure 2: Use case diagram for minimum requirements of booking system

- 53 -
8.5 Appendix E – Activity Diagram for Phase 2

Figure 3: Activity diagram for planning and execution of event bookings