Software Engineering C

Week 4 Group Report

**Project Leader:**

Adam Bramley

**Team Members:**

Stephen Allan

Leith Bade

Peter Williamson

Table of Contents

[1. Introduction 4](#_Toc268789884)

[2. Requirements Revision 5](#_Toc268789885)

[2.1 Functional Requirements: 5](#_Toc268789886)

[2.2 Non-Functional Requirements: 5](#_Toc268789887)

[3. Technological Assessment 5](#_Toc268789888)

[3.1 Language 6](#_Toc268789889)

[3.2 Integrated Development Environment (IDE) 6](#_Toc268789890)

[3.3 Architecture 7](#_Toc268789891)

[3.4 Plugin system 8](#_Toc268789892)

[3.5 Testing System 9](#_Toc268789893)

[3.6 Web Framework 9](#_Toc268789894)

[3.7 Web server 10](#_Toc268789895)

[3.8 Database 11](#_Toc268789896)

[3.9 Build System 13](#_Toc268789897)

[4. Project Life Cycle Methodologies 13](#_Toc268789898)

[4.1 Meetings 13](#_Toc268789899)

[4.2 Sprints 13](#_Toc268789900)

[4.3 Spike Tests 13](#_Toc268789901)

[4.4 Milestones 14](#_Toc268789902)

[5. System Architecture 14](#_Toc268789903)

[6. Deployment Strategy 15](#_Toc268789904)

[6.1 Deploying a JSP Application 15](#_Toc268789905)

[6.2 The Database 16](#_Toc268789906)

[7. Risk Management 16](#_Toc268789907)

[8. Issue Tracking Policy 17](#_Toc268789908)

[8.1 Issue Tracking Workflow 17](#_Toc268789909)

[9. Testing and Metrics 17](#_Toc268789910)

[10. Appendices 18](#_Toc268789911)

[10.1 Appendix A 18](#_Toc268789912)

[10.2 Appendix B 19](#_Toc268789913)

[10.3 Appendix C 22](#_Toc268789914)

# 1. Introduction

*Please refer to Appendix E for a snapshot of the project plan.*

The goal of this project is to provide the administration staff at Massey University with a program to assist with the submission of academic publications to the Research Administration Management System (RIMS). The program is to capture relevant information on the academic paper from a generic input (document object identifier) and give the user an automatically generated form of information to submit to RIMS.

Our project team consists of 4 members each with different skill sets and experiences to contribute to an overall solid workforce. The following paragraphs outline brief auto-biographies from the team members with relevant information about their specific skills and talents.

Adam has had a lot of experience using Java and using various techniques to develop appealing interfaces, which are clear and easy to use. His experience comes from studies at Massey University where he has done numerous programming papers along side Software Engineering, Human-Computer Interaction, and Web Development. This knowledge gives him the ability to adapt to new technologies and work well in a team.

Peter is competent at using Java and his experience comes solely from doing Massey University papers. ­­The papers have given him the some skills in database development, human computer interaction, web development and designing of systems. He is able to work well by himself or in a team.

While working for Land Information NZ on their help desk Steve operated a freelance web design studio. As a result his web knowledge is good. He is a capable Java programmer but his real skills are in .Net. He has a B.Comm degree, ITIL level 1 certificate and MCAD (Microsoft Certified Application Developer) accreditation.

Leith is very experienced in using Java and many of the technologies encompassed in this project. Leith often researches new technologies in his spare time and keeps up to date with the latest things out. Leith is also an expert on Eclipse and has a good knowledge base on the various plugins that will benefit a project like this.

# 2. Requirements Revision

After speaking with Karen and Craig on the 21st July and multiple group meetings we came out with a refined list of functional and non-functional requirements. These requirements were further refined after receiving feedback from our project manager.

## 2.1 Functional Requirements:

1. The system needs to be able to take a DOI as an input, and output metadata for the article. This is to be done through a series of steps, going through the major databases like CrossRef and Scopus, then proceeding to smaller publisher’s websites. The meta-data output includes: authors, article title, publication year, publisher, abstract, keywords, URL, and possibly page and volume numbers. If we are able to, we will also include author affiliations.
2. If the system cannot extract any meta-data, the raw article is to be produced.
3. The system has to display the meta-data on screen and allow the user to edit the data.
4. Access to a local database is needed. The academic paper details are saved to a local database along with author details; this is so the user can easily retrieve past submissions and author details.
5. The system needs to be able to print the cover sheet containing all the metadata.
6. If the user doesn’t have a DOI, the system needs an advanced search function with the ability to search the local database and online sources for matches when given the author name, article title, and publication date.
7. A further extension of the system which is slightly outside the scope of Karen’s specific requirements is for the system to be able to export the local database, or the metadata of specific papers, to a CSV or XML file which would then be sent straight to RIMS for importing into their database. *This is a requirement that will only be implemented if there is time near the end of the project.*

## 2.2 Non-Functional Requirements:

1. The system is to be run on windows
2. The system is to return local database results in under 3 seconds and web results in under 15 seconds per plugin.
3. The system needs to handle plug-in architecture
4. The system needs to have an automated deployment and maintenance procedure.
5. The system must have support for access to the internet and local database through a proxy.

These requirements have also been signed off by Karen on 5th August. These requirements were revised again with Karen on 8th September, and thought was given to the feedback received from the project managers. We have decided to, for the mean time, not worry about scrapping the page for the author’s affiliations. This is because it seems like a very difficult task with the thousands of different styled pages for the various publications. There are various ways the affiliation is displayed: Sometimes it is shown after each author in simple text, other times it is referenced by numerous characters in a superscript next to the author’s name. Instead, we will store the affiliation of the authors in the database, so that once it is entered; it can automatically be retrieved the next time the author is referenced. The user will also be able to edit the affiliation in case authors change university for example.

Another feature we have removed is the verification email. After reassessing the process that Karen goes through to send the emails, it seems to be easier for her to do it herself since she doesn’t email the author after every submission; it seems to be on a basis of a certain number of submissions of that particular author.

# 3. Technological Assessment

Throughout the project, we have done extensive technological assessment. This report will cover the technologies we have chosen to go with and reasons behind why we chose them.

## 3.1 Language

We have chosen to use Java as our main programming language. All of our team members have experience using Java through study at Massey, and some have extra experience through work and other extracurricular activities. Java is easy to use, and there is plenty of technical expertise available via lecturers at Massey. Java has a great online community which provides extensive support for users and there are plenty of resources available at the library if the online help doesn’t suffice. Furthermore, Java is widely used and freely available and fits in with our object-oriented approach to this project. Java is also easily testable through JUnit and easily maintainable; these tie in with the great integrated development environment (Eclipse). There is also a good plugin system available for Java.

We also considered C#, Python, Ruby, and PHP.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Taught at Massey** | **Group experience** | **Easy to use** | **Technical expertise available** | **Resources at library** |
| **Java** | Y | Y (4) | Y | Y | Y |
| C# | Y | Y (2) | Y | N | Y |
| Python | N | Y (1) | N | N | Y |
| Ruby | N | N |  | N | Y |
| PHP | N | Y (1) | N | N | Y |
|  | **Good support** | **Reliable** | **Widely used** | **Portable** | **Freely available** |
| **Java** | Y | Y | Y | Y | Y |
| C# | Y | Y | Y | N | Y |
| Python | Y | Y | Y | Y | Y |
| Ruby | Y | Y | Y | Y | Y |
| PHP | Y | Y | Y | Y | Y |
|  | **OOP** | **Testable** | **Maintainable** | **Good IDE** | **Plugins** |
| **Java** | Y | Y | Y | Y | Y |
| C# | Y | Y | Y | Y | N |
| Python | N | Y | Y | N | N |
| Ruby | Y | Y | N | N | N |
| PHP | N | N | N | N | N |

## 3.2 Integrated Development Environment (IDE)

We have chosen to use Eclipse JEE Helios as our primary IDE for the project. Again, all of our team members have a lot of experience using Eclipse through teachings at Massey and use at home. It is also easy to use and there is plenty of support available for it through Massey, the internet, and written resources. Eclipse is also very reliable and widely used in the community. It has a very wide range of plugins that can be easily installed and extend to almost any technology. There are also good features in Eclipse for testing, refactoring, and building our system. We will be using plugins such as Subclipse, which allows us to access our repository through Eclipse, and utilising Eclipse’s features that support Tomcat, SQL, and JSP. The version of Eclipse we are using supports Java 6, which is needed to use the plugin architecture we have chosen (ServiceLoader).

We also considered NetBeans, Visual Studio, SharpDevelop, and MonoDevelop.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **Taught at Massey** | **Group experience** | **Easy to use** | **Resources at library** | **Good support** |
| **Eclipse** | Y | Y (4) | Y | Y | Y |
| NetBeans | Y | Y (3) | N | Y | Y |
| Visual Studio | Y | Y (2) | Y | Y | Y |
| SharpDevelop | N | N | N | N | N |
| MonoDevelop | N | N | N | N | N |
|  | **Reliable** | **Widely used** | **Portable** | **Freely available** | **Source control** |
| **Eclipse** | Y | Y | Y | Y | Y |
| NetBeans | Y | Y | Y | Y | Y |
| Visual Studio | Y | Y | N | Y | Y |
| SharpDevelop | Y | N | Y | Y | Y |
| MonoDevelop | N | N | Y | Y | Y |
|  | **Refactoring** | **Testing** | **Language** |  |  |
| **Eclipse** | Y | Y | Java |  |  |
| NetBeans | Y | Y | Java |  |  |
| Visual Studio | Y | Y | C# |  |  |
| SharpDevelop | Y | Y | C# |  |  |
| MonoDevelop | Y | Y | C# |  |  |

## 3.3 Architecture

We have chosen to structure our software in a Web app fashion. All of our team members have good experience with technologies surrounding web apps through Massey and other work. Using Web app architecture allows us to have a very straight forward deployment strategy which is easy to update, install, and maintain. If the project were to grow in popularity, our architecture will allow it with its scalability. It is also a widely used architecture with a huge amount of support available through various sources. It is a lightweight approach and freely available, and also supports our SQL database.

We also considered a Desktop application, and a Client/Server application.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to maintain** | **Easy to update** | **Easy to install** |
| Desktop App | Y (4) | Y | N | Y |
| Client/Server | Y (1) | N | N | N |
| **Web App** | Y (3) | Y | Y | N |
|  | **Scalable** | **Only need 1 computer** | **Need network connection** |  |
| Desktop App | N | Y | N |  |
| Client/Server | Y | N | Y |  |
| **Web App** | Y | Y | Y |  |

## 3.4 Plugin system

We have chosen to use a plugin system called ServiceLoader (SL). While our group doesn’t have much experience with SL, there isn’t a plugin system that we are all familiar with so it was a good option to go with due to its ease of use. There is ample support available online for it and it is widely used. It’s reliable due to its simplicity and the fact that it’s type safe. SL is also freely available and has a well defined interface.

We also considered Guice, OSGi, Eclipse Equinox, Java Plugin Framework, and Spring Framework.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to use** | **Technical expertise available** | **Resources at library** |
| Guice | N | Y | N | N |
| OSGI | N | N | Y | Y |
| Eclipse Equinox | N | N | N | Y |
| Java Plugin Framework | N | N | N | N |
| **ServiceLoader** | N | Y | N | N |
| Spring Framework | N | N | N | N |
|  | **Good support** | **Reliable** | **Widely used** | **Portable** |
| Guice | N | Y | N | Y |
| OSGI | Y | Y | Y | Y |
| Eclipse Equinox | Y | Y | Y | Y |
| Java Plugin Framework | Y | Y | N | Y |
| **ServiceLoader** | Y | Y | Y | Y |
| Spring Framework | Y | Y | Y | Y |
|  | **Freely available** | **Runtime** | **Type safe** | **Interface** |
| Guice | Y | Y | Y | Y |
| OSGI | Y | Y | Y | Y |
| Eclipse Equinox | Y | Y | Y | Y |
| Java Plugin Framework | Y | Y | Y | Y |
| **ServiceLoader** | Y | Y | Y | Y |
| Spring Framework | Y | Y | Y | Y |

## 3.5 Testing System

For testing our system we will be mainly using JUnit 3, but could also extend testing to include HTTPUnit, DBUnit, and Apache Cactus for testing JSP pages, the database, and the server respectively. All of our team members have experience through Massey and other work using JUnit. It’s also very easy to use and there is plenty of support easily available through Massey, online, and other resources. JUnit is widely used in the community and easily portable. JUnit also follows our open-source approach. JUnit’s automated system allows for fast testing that can be done often, this fits into our agile methodology.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to use** | **Technical expertise available** | **Resources at library** |
| **Junit 3** | Y (4) | Y | Y | Y |
| Nunit | N | Y | N | N |
|  | **Good support** | **Reliable** | **Widely used** | **Portable** |
| **Junit 3** | Y | Y | Y | Y |
| Nunit | Y | Y | Y | N |
|  | **Freely available** | **Automated** | **Fast** |  |
| **Junit 3** | Y | Y | Y |  |
| Nunit | Y | Y | Y |  |

## 3.6 Web Framework

We have chosen to use Java Service Pages and servlets as our web framework. Most of our team members have experience through Massey and other work with this framework. Even though not everyone is totally familiar with this framework, there is still an underlying knowledge of how it works, and there is a lot of support through Massey and the internet on it. JSP is fast, portable, and reliable. It also supports our other technologies and fits in to our Model View Controller system.

We also considered ASP, Zend Framework, Djano, EPB, and Ruby on Rails.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Taught at Massey** | **Group experience** | **Technical expertise available** | **Resources at library** |
| ASP | N | Y (1) | N | Y |
| Zend Framework | N | Y (1) | N | Y |
| Djano | N | N | N | N |
| **JSP/Servlets** | Y | Y (3) | Y | Y |
| EPB | Y | Y (3) | Y | Y |
| Ruby on Rails | N | N | N | Y |
|  | **Good support** | **Reliable** | **Widely used** | **Portable** |
| ASP | Y | Y | Y | N |
| Zend Framework | Y | Y | Y | Y |
| Djano | Y | Y | Y | Y |
| **JSP/Servlets** | Y | Y | Y | Y |
| EPB | Y | Y | Y | Y |
| Ruby on Rails | Y | Y | Y | Y |
|  | **Freely available** | **Fast** | **MVC** | **HTML** |
| ASP | Y | Y | Y | Y |
| Zend Framework | Y | Y | N | Y |
| Djano | Y | Y | Y | Y |
| **JSP/Servlets** | Y | Y | Y | Y |
| EPB | Y | Y | Y | Y |
| Ruby on Rails | Y | Y | Y | Y |
|  | **Other file types** |  |  |  |
| ASP | Y |  |  |  |
| Zend Framework | Y |  |  |  |
| Djano | Y |  |  |  |
| **JSP/Servlets** | Y |  |  |  |
| EPB | Y |  |  |  |
| Ruby on Rails | Y |  |  |  |

## 3.7 Web server

We have chosen to use Tomcat as our web server. All of our team members have experience with Tomcat through papers taught at Massey. The fact that Tomcat is so easy to deploy is one of the main reasons we chose to use it. It is easy to install, update, and maintain. There is a wide range of support available online and through Massey for Tomcat. Tomcat is lightweight, portable, and fast; Tomcat also supports all of our technologies and is dynamic.

We also considered Apache, IIS, and Jetty.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to maintain** | **Easy to update** | **Easy to install** |
| Apache | Y (2) | N | N | N |
| IIS | Y (2) | N | N | N |
| **Tomcat** | Y (4) | Y | Y | Y |
| Jetty | N | Y | Y | Y |
|  | **Easy to use** | **Technical expertise available** | **Resources at library** | **Good support** |
| Apache | Y | N | Y | Y |
| IIS | N | N | Y | Y |
| **Tomcat** | N | Y | Y | Y |
| Jetty | Y | N | N | Y |
|  | **Reliable** | **Widely used** | **Portable** | **Freely available** |
| Apache | Y | Y | Y | Y |
| IIS | Y | Y | N | Y |
| **Tomcat** | Y | Y | Y | Y |
| Jetty | Y | Y | Y | Y |
|  | **Fast** | **Lightweight** | **Doesn't need a server** | **Dynamic** |
| Apache | Y | N | Y | Y |
| IIS | Y | N | Y | Y |
| **Tomcat** | Y | Y | Y | Y |
| Jetty | Y | Y | Y | Y |
|  | **HTML** | **Other file types** |  |  |
| Apache | Y | Y |  |  |
| IIS | Y | Y |  |  |
| **Tomcat** | Y | Y |  |  |
| Jetty | Y | Y |  |  |

## 3.8 Database

We chose to go with MySQL for our Database. Most of the team members have experience with MySQL through Massey. MySQL is easy to use, install, update, and maintain. There is a lot of support available for MySQL through Massey and other resources online. MySQL is fast, reliable, portable and freely available. It also supports SQL and has a standard interface.

We also considered Microsoft SQL Server, Oracle, Microsoft Access, PostgreSQL, and Sybase.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to maintain** | **Easy to update** | **Easy to install** |
| Microsoft SQL Server | Y (2) | Y | Y | Y |
| Oracle | Y (3) | N | N | N |
| **MySQL** | Y (2) | Y | Y | Y |
| Microsoft Access | Y (1) | Y | N | Y |
| PostgreSQL | N | Y | Y | Y |
| Sybase | N | N | N | N |
|  | **Easy to use** | **Technical expertise available** | **Resources at library** | **Good support** |
| Microsoft SQL Server | Y | N | Y | Y |
| Oracle | N | Y | Y | Y |
| **MySQL** | Y | Y | Y | Y |
| Microsoft Access | Y | N | Y | N |
| PostgreSQL | Y | N | N | Y |
| Sybase | N | N | N | Y |
|  | **Reliable** | **Widely used** | **Portable** | **Freely available** |
| Microsoft SQL Server | Y | Y | N | Y |
| Oracle | Y | Y | Y | N |
| **MySQL** | Y | Y | Y | Y |
| Microsoft Access | N | Y | N | N |
| PostgreSQL | Y | Y | Y | Y |
| Sybase | Y | Y | Y | N |
|  | **Fast** | **Lightweight** | **Doesn't need a server** | **SQL** |
| Microsoft SQL Server | Y | N | Y | Y |
| Oracle | Y | N | N | Y |
| **MySQL** | Y | Y | Y | Y |
| Microsoft Access | Y | N | Y | Y |
| PostgreSQL | Y | Y | Y | Y |
| Sybase | Y | N | N | Y |
|  | **Standard interface** |  |  |  |
| Microsoft SQL Server | Y |  |  |  |
| Oracle | Y |  |  |  |
| **MySQL** | Y |  |  |  |
| Microsoft Access | Y |  |  |  |
| PostgreSQL | Y |  |  |  |
| Sybase | Y |  |  |  |

## 3.9 Build System

We chose to use ANT build scripts as our build system. All of our team members have experience using ANT through Massey. There is a lot of support available for ANT through Massey and other resources online. ANT is also fast and automated with follows our agile methodology.

We also considered MSBuild.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Group experience** | **Easy to use** | **Technical expertise available** | **Resources at library** |
| MSBuild | N | Y | N | N |
| **ANT** | Y (4) | Y | Y | Y |
|  | **Good support** | **Reliable** | **Widely used** | **Portable** |
| MSBuild | Y | Y | Y | N |
| **ANT** | Y | Y | Y | Y |
|  | **Freely available** | **Automated** | **Fast** |  |
| MSBuild | Y | Y | Y |  |
| **ANT** | Y | Y | Y |  |

4. Project Life Cycle Methodologies

For this project the team have decided to use a design methodology mainly based on Scrum.

## 4.1 Meetings

Weekly team meetings will be held each Monday at 11am. During each meeting the members will summarise what they achieved in the previous week, what issues they had and explain what they will be working on next. Following this, each Tuesday at 3pm the team will meet with the project supervisor to update him on the team’s progress.

For the following week’s work the team leader (Adam) will assign the next round of tasks to the team members based either on relevant skills or randomly assignment if no one person is a better choice than another.

## 4.2 Sprints

As the team meets weekly to allocate tasks and detail progress, a weeklong sprint makes sense. Tasks that could take longer than a week to achieve should, where possible, be broken down into smaller sub tasks that can be finished within a sprint.

## 4.3 Spike Tests

Similar to SCRUM, we can spike test any new ideas or technologies for a day or two, no longer, to decide on the merits and either incorporate the technology into the project or discard the test. Given the week long sprints spike tests should ideally be held before the sprint that would use that technology.

## 4.4 Milestones

The project will follow the standard IID process. A working prototype should be the result of completing each milestone – full functionality isn't expected until the final completion date but the individual functions of the application will be built, tested and signed off in an incremental fashion.

Each milestone will focus on specific functions of the project, and can mostly be broken down into incremental plug-in development. That is, once the initial demonstration is finished, the project will follow a feature based development where each week the team will focus on extending the system to cope with more advanced plug-ins. This will focus on the most important, but also easiest to implement, first; then move on to the more advanced forms of gathering meta-data from the more obscure websites.

# 5. System Architecture

*Please refer to Appendix A, Figure 1.*

From the figure, you can see that there are several main features that encompass our system architecture. Beginning at the left, the user (or the internet they are using) is directly interacting with the various servlets. There is one servlets per Java Service Page, which is for each different page that the user sees, that acts as the interface between the user and the service. The servlets’ input is http; this is the input that the system will receive from the forms that the user is filling out. The servlets’ output is html to display results of the system working on the input. The JSPs format the output of the servlets into html.

The servlets interact with a singleton class called the Service. The fact that it is singleton means that there exists only one instance of the class for the lifetime of the server. That allows the class to know the entire state of the application. All the servlets pass incoming form data to the Service, and display the results returned back from the Service, so all the main logic is done in this class. It calls on the other classes like the plug-in system, or the database interface to perform delegated tasks.

The various modules on the right of the diagram handle the different tasks delegated to them by the Service. The first of these modules is the Plug-in Factory. This acts as an interface between the Service and the multiple plugins to the system. The plugins are what process the input from the user and gather the meta-data. There will be a plugin for each of the resources we use to gather the meta-data, for example CrossRef, Scopus, BibTeX, etc. The Plug-in Factory will prioritise these plugins and step through them until sufficient meta-data is gathered.

The Database Interface is the next module interacting with the Service. This module is responsible for handling the queries to the local database. These queries will include saving to the database, searching the database, and exporting information from the database to CSV or XML.

The final module is the Email client. This module will handle requests to email the authors of the papers with the confirmation email. This will be an automated process.

The Database itself is comprised of 7 tables. These make up all the information we need to store on the local database to make the users’ process of submitting the papers to RIMS as easy as possible. There are 2 tables “Publication” and “Author” linked by a bridge table “Published”. These are many-to-many relationships to allow for multiple publications by multiple authors. Following this, there are 3 tables linked to “Publication”, and these correspond to the 3 different types of publication: Book, Journal, and Conference. The final table is to account for the editors of a Book.

*Please refer to Appendix C for the ER Diagram.*

# 6. Deployment Strategy

The deployment strategy focuses on how to deploy a JSP application but some thought is also given to other deployment strategies should they become relevant (e.g. The same web application could possibly be built to incorporate a web service and client that connects to it which would have a different deployment set up).

## 6.1 Deploying a JSP Application

Any JSP application is going to need a web server to run on. In the Massey environment that could possibly even include a server set up on the client's individual machine, but will most likely require being set up on an existing Massey server. Internal Massey policy will most likely have some impact on the final location and the final installation procedure.

A permanent location on a Massey server allows for easier deployment for other users if the application becomes more popular. An individual machine set up would need to be replicated on each user's computer, taking considerable time and making maintenance very difficult.

Deploying the actual files and directories that make up the application would be a simple matter of deploying as a WAR file (Web Archive) to the server or copying the file set up directly via FTP or similar.

One consideration is the client's desire to have an email function included. This would require an email server to be available to the application and, along with the difficulty of maintenance, is one of the primary reasons an individual set up is best avoided.

## 6.2 The Database

A locally accessed database is required for storing the entries to the RIMS system and supplementary tables such as Massey staff and emails.

Storing the database on a server along with the application has several advantages; namely, that the database can be used by multiple users very easily and is easier to update, backup and maintain. A database on the user's computer lacks these advantages and also has the additional issue of replication of data if other users have their own databases set up. The local database needs to be able to avoid repetition of work which it would be unable to do if several copies of the same database existed.

Relocation of the database after installation can be managed by altering the web.xml file which contains the connection information.

# 7. Risk Management

*Please refer to Appendix B.*

At the beginning of the project, several risks were identified with varying degrees of likelihood and impact on the project.

If the Team Leader puts too much workload on team members for a week this can lead to increased stress and unreasonable expectations, similarly if too little is assigned, then it can put the project behind schedule. Furthermore, if the Team leader leads the group away from the scope of the project then it can have dire consequences that the project will not be valid.

If the Team manager does not give the guidance that is required by the team, it could be potentially harmful towards the project. Misunderstanding Scope is another potential risk and is the most common cause of failure for software projects.

If a group member fails to produce the workload that is assigned to them for the week, it can have an effect on the flow of the project, putting the groups schedule behind and puts increased pressure on next week.

Sickness is another risk on the team. If a member falls ill work may not be completed to the deadline, or a substandard effort may be made. There is also a small risk a member of the team drops out of the paper, this will put all workload done by the group member onto the other members of the group and potentially hinder the quality of work.

There is a risk of feature creep to the project. If features keep being added to the project then it has the impact that we do not get the most important feature working. This ties into another risk of changing requirements, this could essentially mean all the previous work done to date, would be invalid

There is also the risk of team members not having enough experience and skill in specific parts of the project. If we are unable to accomplish something it may delay the schedule, and extra resources may have to be placed in solving the issues.

The solution of a client/server application poses extra risks that must be addressed. These include server connectivity, accessibility, and permissions. The risk of the client having insufficient permissions for access to the server is almost non-existent as there is no security on the application. On the other hand, accessibility and connectivity to the server is crucial for the application to run at all, and will have massive impacts if either is interrupted.

# 8. Issue Tracking Policy

Throughout the sprints when issues arise, they are to be dealt with immediately. These issues can include impediments discovered during the weekly meetings, bugs, or change requests. When dealing with bugs and other high priority issues, they are to be resolved before the feature they are associated with is signed off.

8.1 Issue Tracking Workflow

* Open a new issue through the Google code project page
* Fill out the summary and description of the issue.
* Cc to relevant team members.
* Make sure the issue type and priority are set accordingly.
* Submit the issue.   
    
  *After the issue has been submitted:*
* I (Adam) will assign the issue to a team member based on the workload of the team members at the time.  
    
  *After the issue has been resolved:*
* The two team members that have not assigned or submitted the issue will then sign the issue off.
* Issue page is then updated.

If an issue is open for more than 7 days, the team will have an extra meeting discussing the issue until it is resolved. In most cases, we will be able to resolve the issue with all team members discussing it. If the issue is still unable to be solved then further investigation will take place by a specific team member. If the particular issue is due to an extra feature, then that feature may be removed.

# 9. Testing and Metrics

For testing our system and generating metrics we will utilise as many automated systems as possible. For testing we have decided to use Junit 3. This allows us to frequently run tests on our code to test for any new bugs that may have been introduced from freshly written code or refactoring. As part of our testing policy, each time a new issue is submitted a new test case will be added to account for the newly introduced bug. This will ensure the system is safe from the bug and allow us to change our code to suit it.

To automate metrics, we will be using EMMA and PMD to gather information on test coverage and any possible bugs or redundant code. This will give us a good range of metrics to ensure our code is up to standard.

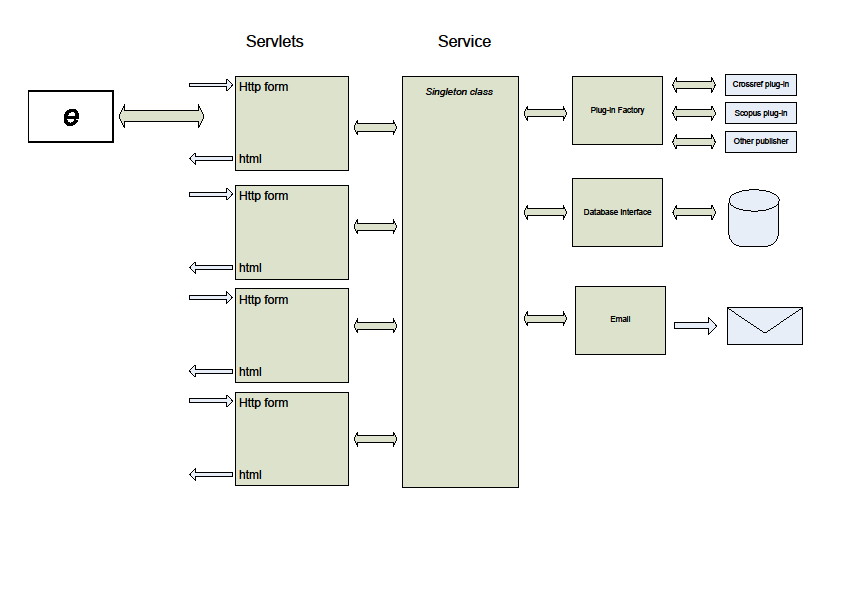
We have also included a metric for the team’s participation. This is based on the percentage of commits to the repository each member has done. Keep in mind even though a percentage may be higher, different tasks require more or less work and in turn require more or less commits.

Total commits from 4/08/10 to 10/09/10:

Adam: 40% Leith: 26% Peter: 24% Steve: 10%

# 10. Appendices

## 10.1 Appendix A



*Figure 1: High level system architecture.*

## 10.2 Appendix B

|  |  |
| --- | --- |
| **Risk:** | Team leader assigning workload improperly |
| **Likelihood of Occurrence:** | High |
| **Potential Impact:** | High.If the Team Leader puts too much workload on for a week this can lead to increased stress and unreasonable expectations, similarly if too little is assigned, then it can put the project behind schedule. |
| **Mitigation plan:** | Communication to the leader in the group meetings of what members are likely to accomplish and that they feel is the correct amount of work. |

|  |  |
| --- | --- |
| **Risk:** | Team leader leading group astray |
| **Likelihood of Occurrence:** | Medium |
| **Potential Impact:** | High.If the Team leader leads the group away from the scope of the project then it can have dire consequences that the project will not be valid. |
| **Mitigation plan:** | Communication to the leader and between members in the group to ensure that the team stays on the target and does not get sidetracked. |

|  |  |
| --- | --- |
| **Risk:** | Team manager |
| **Likelihood of Occurrence:** | Medium |
| **Potential Impact:** | Medium.If the Team manager does not give the guidance that is required by the team, it could be potentially harmful towards the project |
| **Mitigation plan:** | There is no way to prevent this from happening, although the weekly meetings should keep this risk under control |

|  |  |
| --- | --- |
| **Risk:** | Misunderstanding Scope |
| **Likelihood of Occurrence:** | High |
| **Potential Impact:** | High.Misunderstanding Scope is the most common cause of failure for software projects. |
| **Mitigation plan:** | Constantly talking to the client and informing them of progress, checking what has been completed meets their requirements. |

|  |  |
| --- | --- |
| **Risk:** | Group member doesn’t complete a weekly workload |
| **Likelihood of Occurrence:** | High |
| **Potential Impact:** | High.If a group member fails to produce the workload that is assigned to them it can have an effect on the flow of the weeks work by putting the groups schedule behind and puts increased pressure on next week. |
| **Mitigation plan:** | Regular meetings to ensure that the workload is on schedule and that if a member is unlikely to complete the task, steps can be taken to help the burden or alter the plan. Communication from the member in question is important. |

|  |  |
| --- | --- |
| **Risk:** | Sickness |
| **Likelihood of Occurrence:** | High |
| **Potential Impact:** | Medium. Work may not be completed to the deadline, or a substandard effort may be made |
| **Mitigation plan:** | Staying at home and not transferring the sickness to other group members is preferable. Communication via electronic means rather than face to face in order to stay in touch |

|  |  |
| --- | --- |
| **Risk:** | Group member resigns from Software C |
| **Likelihood of Occurrence:** | Low |
| **Potential Impact:** | High. This will put all workload done by the group member onto the other members of the group |
| **Mitigation plan:** | Communication between members is essential, and encouragement to stick with the project if a member considers quitting. However it is better they quit, than stay on and not do the workload |

|  |  |
| --- | --- |
| **Risk:** | Loss of Code/Documentation |
| **Likelihood of Occurrence:** | Extremely Low |
| **Potential Impact:** | High. All the work that is done will be lost |
| **Mitigation plan:** | The Repository is set up to prevent this from happening, and will ensure that nothing is lost |

|  |  |
| --- | --- |
| **Risk:** | Feature Creep |
| **Likelihood of Occurrence:** | Medium |
| **Potential Impact:** | Medium. If features keep being added to the project then it has the impact that we do not get the most important feature working |
| **Mitigation plan:** | By prioritising and evaluating what features are important and those that would be nice but are not necessary |

|  |  |
| --- | --- |
| **Risk:** | Changing Requirements |
| **Likelihood of Occurrence:** | Medium |
| **Potential Impact:** | High. The Changing of requirements could essentially mean all the previous work done to date, would be invalid |
| **Mitigation plan:** | Constant communication with the client in order to catch any requirements that change early, so the project schedule suffers as little as possible |

|  |  |
| --- | --- |
| **Risk:** | Running out of time |
| **Likelihood of Occurrence:** | High |
| **Potential Impact:** | High. 14 weeks is all the time we get, everything needs to be up and running up then |
| **Mitigation plan:** | Proper planning and communication to prevent any bottle necks from occurring. Prioritising requirements which are essential will produce a working project |

|  |  |
| --- | --- |
| **Risk:** | Technological Issues/Lack of skills |
| **Likelihood of Occurrence:** | Medium |
| **Potential Impact:** | High. If we are unable to accomplish something it may delay the schedule, and extra resources may have to be placed in solving the issues |
| **Mitigation plan:** | A technological review and the team leader knowing the skills of individual group members will try prevent this from happening, though it may be unavoidable |

|  |  |
| --- | --- |
| **Risk:** | Loss of client/server connection or access |
| **Likelihood of Occurrence:** | Low |
| **Potential Impact:** | High. If the client loses access or connection to the server the application will not be able to function at all. |
| **Mitigation plan:** | TODO |

## 10.3 Appendix C

**Author**

Author\_ID **(PK)**  
Author\_First\_Name

Author\_Last\_Name  
Author\_Middle\_Name

Type

Department

College

Email

Publication\_Type

0..\*

1..1

Editor\_ID **(PK)**

Publication\_ID (*FK)*

Editor\_First\_Name

Editor\_Last\_Name  
Editor\_Middle\_Name

Is Edited by

Publication\_ID **(PK)***(FK)*  
Volume\_NO

Issue\_NO

Journal\_Title

Article\_Title

Publication\_ID **(PK)***(FK)*  
Abstract\_Title

Conference\_Name

Start\_Date

End\_Date

Location

**Book**

**Conference**

**Editor**

1..1

1..\*

1..\*

1..1

Has

Publishes

Publication\_ID **(PK)***(FK)*

Author\_ID **(PK)***(FK)*

**Published**

d

Publication\_ID **(PK)**

Publication\_Category  
Publication\_Year   
Start\_Page  
End\_Page  
ISSN\_ISBN

Abstract

KeyWords

URL\_Location

Quality\_Assured

**Publication**

Publication\_ID **(PK)***(FK)*  
Place\_Published

Chapter\_Title

Book\_Title

**Journal**

## 10.4 Appendix D

<<includes>>

<<includes>>

<<includes>>

Add RIMS   
Identifier

Academic Researcher

Add Staff   
Information

Email   
Academics

Modify existing   
publication details

Print

Coversheet

<<includes>>

Fills out Coversheet

/Adds to Database

<<includes>>

<includes>>

<<includes>>

<<includes>>

<<includes>>

Gather publication Information (No DOI)

Gather publication Information (DOI)

## 10.5 Appendix E

