

UNIVERSITY OF HERTFORDSHIRE

Faculty of Engineering and Information Sciences

Modular MSc Honours in Computer Science (Software Engineering)

7WCM0031 Software Engineering MSc Project (Online)

Final Project Report

January 2016

***Development of a distributed data and document management system
for 'MSc Properties'***

Mr D L Edwards

SRN: 07156987

Supervised by: Thiago Matos

Abstract

With the increasing number of people having to move out of their local area, due to the current unemployment rates and recent legislation changes, resulting in local councils being able to house homeless families outside of the local borough [2] in private rented accommodation, [3] and benefit caps [2] meaning households are unable to afford their local area rental rates.

This means that estate agents such as the fictional estate agent 'MSc Properties' require a secure and fast means to store and share data and documents with all of their different stores that could be 100's of miles apart, and therefore require a system that will enable each store to access data and documents on households who require to move from their local area.

For me to alleviate the issue of data and document storing and access for 'MSc Properties', I decided to incorporate a number of design, implementation and testing techniques, such as UML diagrams, Remote Method Invocation, Object Relational Mapping, Unit Testing, System Testing and more, which is described in more detail during this report, to produce a distributed system that will achieve the requirements of 'MSc Properties'.

From the testing results outlined in the report, the project successfully managed to implement a distributed system that manages 'MSc Properties' data and documents, allowing for any user at any store location, who is logged in to the system to retrieve any data or documents, depending on whether or not they have the right security levels.

Also 'MSc Properties' data and documents is successfully stored to a MySQL database, to enable the system to be robust and protect data and documents against system crashing or downtime.

Furthermore, I have successfully implemented an observer pattern which enables the home screen of each user to be updated, whenever certain data within the system is created, updated or deleted.

Moreover, I have successfully implemented task scheduling functionality that enables tasks to be automated, reliving 'MSc Properties' employees of tasks they would have had to manually complete.

Finally, I believe the project was a success as I successfully completed majority of the project aim and objectives, and although there were a few changes I would have made such as implementing a task scheduling framework and a working graphical user interface, or writing a more accurate project plan at the beginning of the project, the project was overall a success.

Acknowledgements

I would like to thank:

Firstly, my project supervisor Thiago Matos Pinto who has been there to guide and assist me from the start of the project, through to the final submission date of the project.

Secondly, all the lecturers that have taught me over course of both my BSc and MSc Software Engineering courses, each providing me with knowledge and skills that has enabled me to produce this dissertation.

Finally, my family and close friends who have been with me through my ups and downs at University, providing me with love, support and encouragement.

Table of Contents

Abstract	1
Acknowledgements	2
Table of Contents	3
1. Introduction	6
Introduction to the project.....	6
1.1.1. Report Structure	6
1.2. Project Aim	7
1.3. Project Objectives	7
1.3.1. Core Objectives.....	7
1.3.2. Advance Objectives.....	8
1.4. Project Background	8
1.5. Literature Review	9
1.5.1. Design Methodologies	9
1.5.1.1. Software Development Approach.....	9
1.5.1.2. Modelling Software Behaviour	11
1.5.1.3. Design Patterns.....	12
1.5.2. Development Methodologies.....	13
1.5.2.1. Networking.....	13
1.5.2.2. Document Management	14
1.5.2.3. Task Scheduling.....	15
1.5.2.4. Database Management Systems.....	16
1.5.2.5. Web Server	17
1.5.2.6. Graphical User Interface.....	18
1.5.3. Testing Methodologies.....	18
1.5.3.1. Unit Testing	18
1.5.3.2. System Testing	19
1.6. Project Plan	20
1.7. Relevance to target award	21
1.8. Required Resources & Skills	21

1.9. Ethics Approval	22
2. Design	23
2.1. Introduction	23
2.2. Software Lifecycle	23
2.3. Modelling System Behaviour	24
2.3.1. Use Case Diagrams	24
2.3.2. Class Diagram	24
2.3.3. Enhanced Entity Relationship Diagram	24
2.3.4. Sequence Diagrams	25
2.3.5. Storyboard	25
3. Implementation	25
3.1. Introduction	25
3.2. Iteration Cycle 1	25
3.2.1. Design Patterns	26
3.2.1.1. Iterator Pattern	26
3.2.1.2. Inheritance	26
3.2.1.3. Object Composition	27
3.2.1.4. Singleton Pattern	27
3.2.2. Graphical User Interface	27
3.3. Iteration Cycle 2	28
3.4. Iteration Cycle 3	29
3.4.1. Push vs Pull	30
3.4.2. Local Reference vs Remote Reference	30
3.5. Iteration Cycle 4	31
3.6. Iteration Cycle 5	31
3.7. Iteration Cycle 6	31
3.8. Testing	32
3.8.1. Unit Testing	32
3.8.2. Integration Testing	33
3.8.3. System Testing	34
4. Conclusion	34

5. Evaluation.....	36
Bibliography	38

1. Introduction

1.1. Introduction to the project

For my MSc Computer Science Dissertation (Software Engineering), I decided to solve the problem of data and document management through the implementation of a distributed system for a fictional estate agent called 'MSc Properties'.

The reason for me undertaking this project is that there is an ever increasing demand for private rented accommodation due to changes in the social housing market, social benefits offered, and unemployment rates, resulting in less social housing being available and reduced benefits for people on a low income, who would normally need to make use of social housing and/or benefits offered by the government, meaning families are unable to afford the private rents of their current area they live in and need to move to a different area they can afford to privately rent.

1.1.1. Report Structure

During this report I will document the aim and objectives of this project and explain the problem background in more detail, looking at the different factors that contribute to the problem identified.

I will then document the research and literature review I carried out in order for me to identify the different software engineering techniques and technologies available to solve the problem.

I will then go on to document the design of the distributed system, outlining the different software engineering techniques chosen, and why these were chosen. I will then explain how these were used to enable me to structure and manage the project, and just as importantly allow me to produce diagrams that virtualized the structure, behaviour and interaction of the distributed system.

Furthermore, I will document the implementation and testing of the distributed system, explaining which techniques and technologies I chose, and why they were the best solution for this project, given the available resources. I will then go on to provide an analysis of the test results to draw some conclusions on the validity of the software produced.

Lastly I will document my conclusion and evaluation of the project as a whole, stating whether or not I have successfully achieved the project aim and objectives. I will then outline what went well, what didn't go so well, and what I would do different if I was to do the project again.

1.2. Project Aim

The aim of this project is to tackle the issues of data and document sharing across the Internet by developing a distributed data and document management system for a fictional estate agent called 'MSc Properties'.

The distributed system should allow 'MSc Properties' to share business data and documents across the Internet, whilst providing data security and integrity. 'MSc Properties' requires the distributed system to be maintainable, dependable and usable, which means I will explore the different techniques that support program specification, design, validation and evolution of software.

1.3. Project Objectives

1.3.1. Core Objectives

Analyse 'MSc Properties' current business processes by week 5.

Complete literature searches and review of existing data management systems, identifying the software engineering models, methodologies, tools and metrics used in the development process by week 12.

Set out functional and non-functional requirements for the development within the requirements document by week 9.

Ensure required resources are available for the entire project by week 9.

Carry out risk assessment by week 10.

Set out the distribution mechanism I am going to employ for the distributed system by week 13.

Develop a suitable data management system model that meets the requirements defined by week 15.

Write test scripts to test the implementation of the system outlined in the development model by week 15.

Develop a suitable database to handle the business data and import dummy data into the database by week 16.

Develop a suitable application to handle the business processes and connect to the database to store the business data by week 24.

Develop a suitable search facility so users can search for information stored in the database, and should be implemented by week 24.

Develop reporting functionality so certain users can report on business performance indicators by week 24.

Develop a log in facility for users, allowing for restricted access, and to prevent unauthorised access and should be implemented by week 26.

Test the system using the test scripts created, ensuring the test results are above the acceptable failure rate defined in the requirements by week 33.

Develop and test a user manual by week 31.

Evaluate the project in a report to detail the entire development and outline what went well and what could have been done better by week 34.

1.3.2. Advanced Objectives

Develop a website to advertise services offered to potential customers/suppliers. Customers/Suppliers will be able to register and submit a service request through the website and should be implemented by week 26.

Develop document management facility that allows for documents to be stored electronically, and should be implemented by week 26.

Develop a home screen which provides a live feed of the tenancies and leases due to expire by week 26.

Develop a reset password facility, so users are able to reset their password if they have forgotten it allowing users to establish access to the system. This should be implemented by week 26.

1.4. Project Background

‘MSc Properties’ is a fictional estate agent with a number of sites nationwide across England. Due to the current unemployment rates, and recent legislation changes resulting in local councils being able to house homeless families outside of the local borough [3] and benefit caps [2] meaning families have to move out of their local borough due to not being able to afford local rents [3], ‘MSc Properties’ require the need to be able to transfer customers between sites, meaning the transfer of data and documents across sites that could be 100’s of miles apart.

‘MSc Properties’ require me to develop a distributed system to create and manage their property portfolio, as well as creating and managing customer/supplier/employee accounts. They require this so that data can be stored on a server or locally and all the officers of different ‘MSc Properties’ sites will be able to access this data. The system will have a login facility to provide restricted access for users, and will also allow managers of ‘MSc Properties’ stores to manage their employee accounts. The system will also allow ‘MSc Properties’ managers to report on business data.

‘MSc Properties’ have allocated an office manager to the project to assist with analysis of

current business processes, map system requirements, and communicate back to 'MSc Properties' with the work that is occurring in project meetings, and relaying any project document back to the business for sign off. From the project meetings a business analysis and system requirements document was produced which is documented in Appendices V.

1.5. Literature Review

As previously outlined, I will now document the literature reviews I undertook to allow me to understand the different options available to me to enable me to successfully achieve the project aim and objectives.

1.5.1. Design Methodologies

The problem I am trying to solve by undertaking this project, requires a piece of software to be developed for me to successfully achieve the project objectives surrounding data and document sharing.

But as well as trying to solve the business problem surrounding data and document sharing, there is also the problem of producing software that is maintainable, dependable and usable, so it will require me to produce high quality software. This means I am required to undertake literature reviews into different design decisions I will need to make when designing the system, and also require the project to go through a structured development process to give the project the highest possibility of successfully achieving the project aim and objectives.

1.5.1.1. Software Development Approach

The first design principle I am going to discuss is the software development approach and the piece of literature I am going to review is the article called "*Software Quality & Agile Methods*" written by M. Huo, J. Verner, L Zhu and M.A. Babar [5]. This article looks at the quality of software produced when comparing the Waterfall Model and Agile methods, and specifically how agile methods can achieve high quality software even if the process is not linear and a complete requirements specification has not been developed prior to the design and implementation stage of the development.

The article then goes on to conclude that "agile methods do have practices that have Quality Assurance abilities, some of them are inside the development phase and some others can be separated out as supporting practices. The frequency with which these agile Quality Assurance practices occur is higher than in a waterfall development and lastly, agile Quality Assurance practices are available in very early process stages due to the agile process characteristics" [5]. The below diagrams show the different methods and Quality Assurance techniques undertaken within the Waterfall Model and Agile Methods.

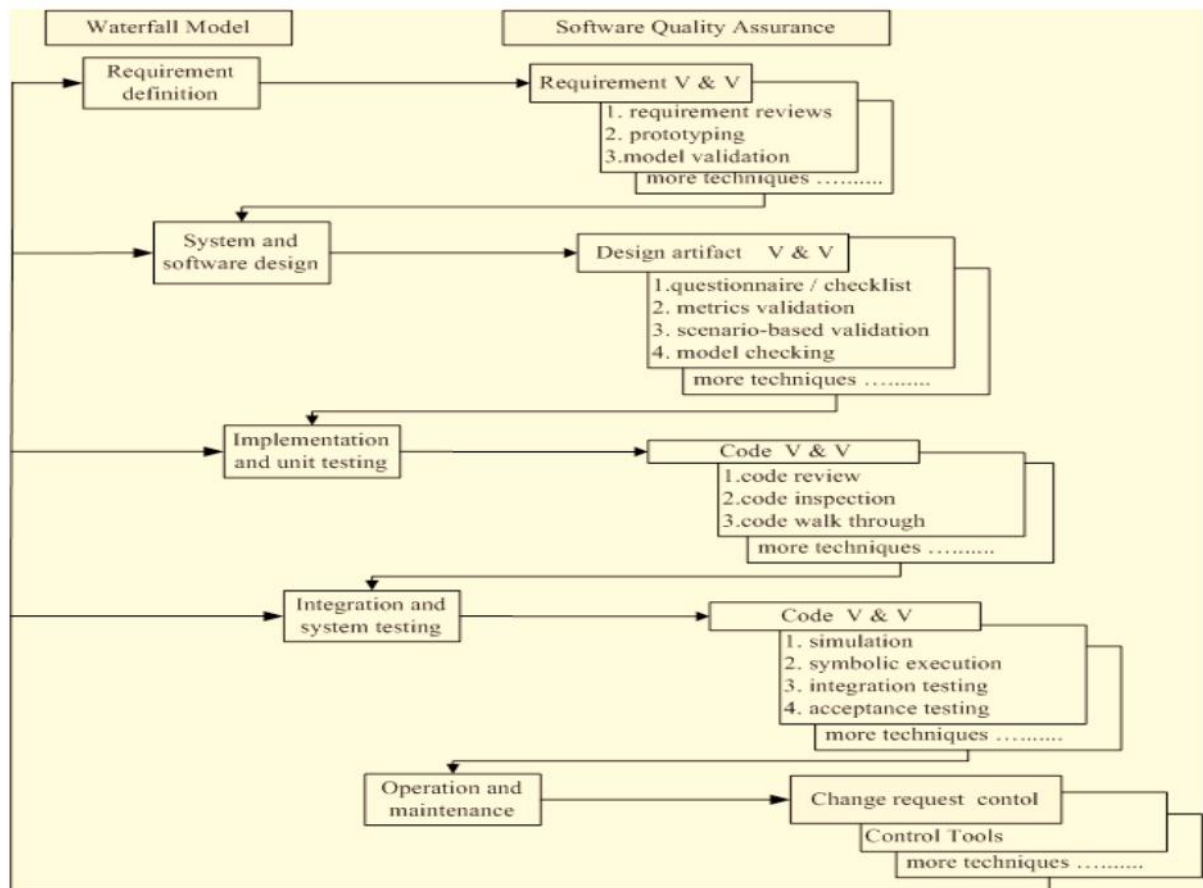


Fig. 1. Waterfall Process Model with Quality Assurance Techniques [5].

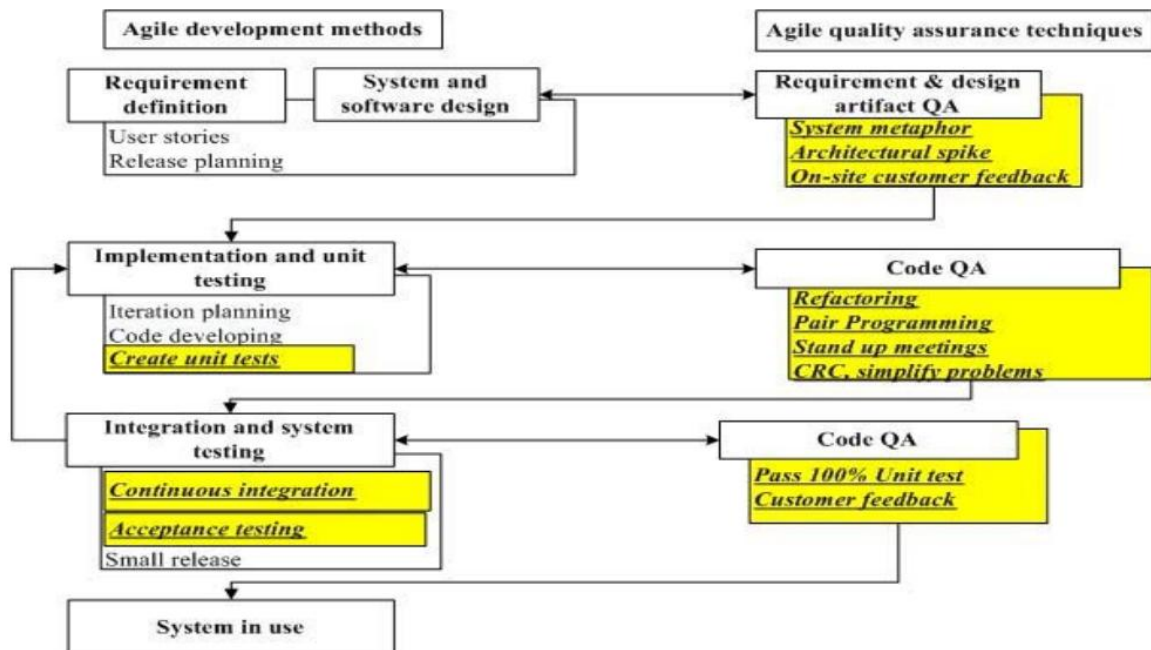


Fig. 2. Agile Development Methods with Quality Assurance Techniques [5].

1.5.1.2. Modelling System Behaviour

The piece of literature I am going to review for the modelling of system behaviour is the article called *“Designing Concurrent, Distributed, and Real-Time Applications with UML”* written by H. Goma [6]. This article looks at two areas, the software design method called Concurrent Object Modelling and Architectural Design Method (COMET), which is an example of a Unified Modelling Language (UML) based method, and the different modelling required for concurrent, distributed and real time applications using UML.

The article explains that “In the requirements model, the system is considered as a black box and the use case is developed... In the analysis model, the emphasis is on understanding the problem, hence the emphasis is on identifying the problem domain objects and the information passed between them... In the design model, the solution domain is considered, so the analysis model is mapped to a concurrent model” [6]. These different models are what I will have to consider when going through the software development process for this project.

Below is a list of different techniques to model system behaviour which I have come across during the research for this project:

- Use Case Diagrams – A representation of a user’s interaction with the system, showing the relationship between the user and the use cases they are involved in [16].
- Data Flow Diagrams – A graphical representation of the “flow” of data through an information system [17].
- Class Diagram - A static structure diagram that describes the structure of a system showing the system classes, their attributes, methods, and the relationships amongst objects [18].
- Entity Relationship Diagram – A data model for describing the data or information aspects of business domain or its process requirements, in an abstract way that lends itself to ultimately being implemented in a database such as relational database [19].
- Class Responsibility Collaborator (CRC) Models – A brainstorming tool used in the design of object-oriented software, documenting the dynamics of object interaction and collaboration [20].
- Sequence Diagrams – An interaction diagram that shows how processes operate with one another and in what order, showing object interaction arranged in time sequence [21].
- Storyboards – A graphic organizer in the form of illustrations or images displayed in sequence for the purpose of pre-visualising a motion picture, animation, motion graphic or interactive media sequence [22].

1.5.1.3. Design Patterns

The last design principle I am going to discuss is design patterns, and most importantly the design patterns I can employ within the software I am going to produce. Again as with modelling system behaviour previously, this project centres on software development, and part of the aim is to produce software that is maintainable, dependable and usable, so it will require me to produce high quality software, and by implementing design patterns, it will allow me to produce software with high cohesion, low coupling, encapsulation, and other metrics of software development which indicate high quality software.

Ian Sommerville explains design patterns as a description of accumulated wisdom and experience, a well-tried solution to a common problem, and the Hillside Group puts it as “Patterns and Pattern Languages are ways to describe best practices, good designs, and capture experience in a way that it is possible for others to reuse this experience” [8].

Below are some design patterns I have come across during my research for this project:

- Observer pattern – A software design pattern in which an object, called the subject, maintains a list of dependents, called observers, and notifies them automatically of any state changes, usually by calling one of their methods [23].
- Singleton pattern – A software design pattern that restricts the instantiation of a class to one object [24].
- Strategy pattern – A software design pattern that enables an algorithm’s behaviour to be selected at runtime [25].
- Creational pattern – A software design pattern that deals with object creation mechanisms, trying to create objects in a manner suitable to the situation [26].
- Iterator pattern – A software design pattern in which an iterator is used to traverse a container and access the container’s elements [27].
- Composite pattern – A partitioning software design pattern, which describes that a group of objects is to be treated in the same way as a single instance of an object, allowing clients to treat individual objects and compositions uniformly [28].
- Inheritance – Is when an object or class is based on another object or class, using the implementation to maintain the same behaviour, and is a mechanism for code reuse and in programming languages that support inheritance, produce an “is a” relationship between sub classes and its parent class. [29].
- Object Composition – Is a way to combine simple objects or data types into more complex ones, and are a critical building block of many data structures. Composition can be regarded as a relationship between types: an object of a composite type “has an” object of a simpler type [30].
- Object Relational Mapping – Is a programming technique for converting data between incompatible type systems in object-oriented programming languages, and

in effect creates a “virtual object database” that can be used within the programming language [31].

1.5.2. Development Methodologies

I am now going to discuss the different development methodologies I uncovered during the course of the project, whilst undertaking the literature review.

1.5.2.1. Networking

One of the major problems I am trying to solve by undertaking this project, is to be able to produce a system that allows ‘MSc Properties’ to share data and documents across the Internet, whilst ensuring that the validity of the data and documents being shared are maintained. For me to do this, it has meant that I have had to undertake a literature review into the different networking technologies that will enable me to successfully achieve the project aim and objectives related to networking.

The first piece of literature I am going to review for Networking is the article called “*Implementing Remote Procedure Calls*” written by A.D. Birrell and B.J Nelson [8]. This article talks about the options that face the designer implementing remote procedural call (RPC) functionality and the considerations that need to be made when making decisions on this type of system.

The article states “when making a remote call, five pieces of program are involved: the *user*, the *user-stub*, the RPC communications package (the RPCRuntime), the *server-stub*, and the *server*... When the user wishes to make a remote call, it actually makes a perfectly normal local call, which invokes a corresponding procedure in the user-stub. The user-stub is responsible for placing a specification of the target procedure and the arguments into one or more packets and asking the RPCRuntime to transmit these reliably to the callee machine. On receipt of these packets, the RPCRuntime in the callee machine passes them to the server-stub. The server-stub unpacks them and again makes a perfectly normal local call, which invokes the appropriate procedure in the server. Meanwhile, the calling process in the caller machine issues pended awaiting a result packet. When calling the server completes, it returns to the server stub and the results are passed back to the suspended process in the caller machine. There they are unpacked and the user-stub returns them to the user” [8]. This process is represented in the below figure.

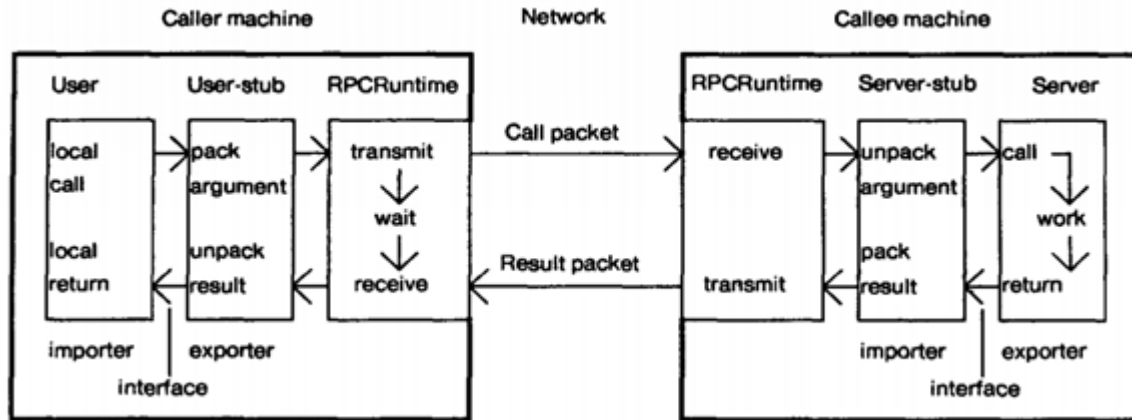


Fig. 3. The components of the system, and their interactions for a simple remote call [8].

The Remote Procedural Call functionality highlights the type of technology I could implement to successfully achieve the project aim and objectives of data sharing amongst 'MSc Properties' hosts that are at different locations.

The second piece of literature I am going to review for Networking is the article called "*Push vs. Pull in Web-based Network Management*" written by J.P. Martin-Flatin [9]. This article talks about two models of network management, which are "The Pull Model" and "The Push Model", which represent two well-known approaches to exchanging data between two hosts with a distance between them, which is one of the major problems I am trying to solve during this project.

The article states "The pull model is based on the request/response paradigm, the client sends a request to the server, then the server answers, either synchronously or asynchronously. This is functionally equivalent to the client "pulling" the data off the server. In this approach, the data transfer is always initiated by the client, i.e. the manager. The push model, conversely, is based on the publish/subscribe/distribute paradigm. In this model, agents first advertise what Management Information Bases they support, and what Simple Network Management Protocol notifications they can generate; the administrator then subscribes the manager (the Network Management Station) to the data he/she is interested in, specifies how often the manager should receive this data, and disconnects. Later on, each agent individually takes the initiative to "push" data to the manager, either on a regular basis via a scheduler (e.g., for network monitoring) or asynchronously" [9]. The article then goes on to state that "the pull model, well suited to ad hoc management, and the push model, well adapted to regular management" [9].

1.5.2.2. Document Management

The next development area I am going to discuss is document management, as the business deals with numerous documents that need to be stored, and made available for access by any host at different locations.

The piece of literature I am going to review for Document Management is the article called *“Electronic Document Management: Challenges and Opportunities for Information Systems”* written by R.H. Sprague, Jr [10]. This article talks about a number of benefits gained from implementing a document management system, such as improving the publication process, supporting organisational processes and communication amongst people and groups, improving access to external information, creating and maintaining documents, maintaining corporate records and lastly promoting training education.

The article states “Documents are stored electronically, shipped over a network and printed when and where they are needed, resulting in reduction in obsolescence, elimination of warehouse costs, and reduction or elimination of delivery time” [10]. The article later goes on to explain “The benefits of Electronic Document Management for these applications are, quicker access to the documents, more efficiency in the search process, simultaneous access by several people to the most current version of the document, and reduced cost of printing and distributing documents” [10].

Below are some document management frameworks I have come across during my research for this project:

- Apache JackRabbit – Is a content repository which implements the Content Repository for Java (JCR), with support for structured and unstructured content full text search, versioning, transactions, observation and more [32].
- Modeshape – Is a distributed, hierarchical, transactional, and consistent data store with support for queries, full-text search, events, versioning, references, and flexible and dynamic schemas, which implements the Content Repository for Java (JCR) [33].

1.5.2.3. Task Scheduling

The next development area I am going to discuss is task scheduling, and in particular task scheduling in real time systems, this is because the distributed system that will be developed for ‘MSc Properties’ during this project will be a real time system, dealing with the processing of scheduled tasks.

The piece of literature I am going to review is the article called *“Application of Real-Time Monitoring to Scheduling Tasks with Random Execution Times”* written by D. Haban and K.G. Shin [11]. This article talks about the calculation of execution time for posteriori tasks (calculation of execution time requires experience with the given task) scheduled within real time systems, and that the worst-case execution time is usually used to ensure that enough time has been allocated for the task to be completed, and discusses the drawbacks with this sort of approach and alternatives to this method.

The article states “real-time tasks are usually scheduled based on their worst-case execution time, and since the worst-case execution time can be several orders of magnitude larger than the true execution time, scheduling tasks based on the worst-case execution time can lead to severe underutilization of CPU cycles and/or incorrect decision on the schedulability

of tasks i.e., some tasks are declared to be un-schedulable even if they can be completed in time” [11].

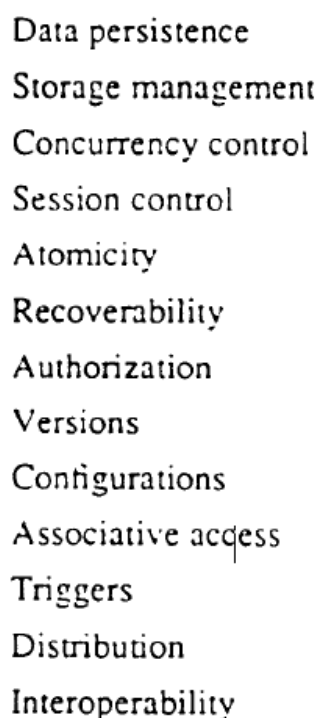
Below are some task scheduling frameworks I have come across during my research for this project:

- Quartz project –
- Haban & Shin project

1.5.2.4. Database Management Systems

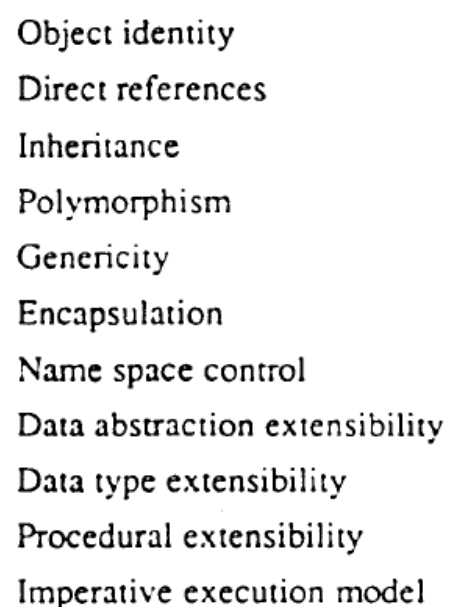
The next development area I am going to discuss is database management systems, and in particular data models based on object-orientated concepts. This is because I have decided to develop a distributed system using object-orientated concepts, which means I will need to adopt a database management model based on these object-orientated concepts.

The piece of literature I am going to review is the article called *“An Introduction to Object-Oriented Database and Database Systems”* written by M.L. Horowitz [12]. This article looks at the possibilities of combining most of the desirable features of database systems with desirable features of the object orientated model of computation and the below diagram outlines these features.



Data persistence
Storage management
Concurrency control
Session control
Atomicity
Recoverability
Authorization
Versions
Configurations
Associative access
Triggers
Distribution
Interoperability

Fig. 4. Features provided by database oriented systems [12].



Object identity
Direct references
Inheritance
Polymorphism
Genericity
Encapsulation
Name space control
Data abstraction extensibility
Data type extensibility
Procedural extensibility
Imperative execution model

Fig. 5. Features provided by object-Languages [12].

The article later goes on to explain the issues that can arise when combining the database and object-oriented model concepts, by stating that “First integration should occur without

impedance mismatch. In particular, language support for object-oriented database services should be orthogonal and transparent. Second, integration should not lose any advantages of existing data models. For instance, object-oriented programming does not support data independence inherently, so features such as relationship support and query joins should be provided. Finally, integration presents an opportunity for introducing new desirable features” [12].

Below are some database management systems I have come across during my research for this project:

- Microsoft Access – Is a Database Management System (DBMS) from Microsoft that combines the relational Microsoft Jet Database Engine with graphical user interface and software development tools [34].
- MySQL – Is an open-source relational database management system (RDBMS) and the most widely used open-source client-server model RDBMS [35].

1.5.2.5. Web Server

The next development area I am going to discuss is web servers, and in particular the different web server software that can be implemented to receive and manage the Hypertext Transfer Protocol requests sent by a user of the website that will be developed to advertise ‘MSc Properties’ services to potential customers, and also manage communication between the web server and the database management system discussed previously.

The piece of literature I am going to review is the article called *“Specification and Implementation of Dynamic Web Site Benchmarks”* written by C. Amza, A. Chanda, A.L. Cox, S. Elnikety, R. Gil, K. Rajamani and W. Zwaenepoel [13]. This article looks at the movement from web content being static HTML or image files, to web content becoming dynamic through the combination of a front end web server (web browser such as Internet Explorer), an application server (software such as Apache, along with server side scripting in PHP and SQL), and a back-end database (software such as Microsoft Access) and in particular identifies benchmarks for dynamic web sites by comparing 3 different dynamic web sites, looking at the bottleneck characterizations for these web sites.

This combination of technology would allow me to develop a dynamic website that can advertise ‘MSc Properties’ services to potential customers, and be updated by staff when they perform actions in the system which result in updates to the website, for example, a property being rented to a customer meaning the property is no longer available to rent and therefore should no longer be advertised on the website as available to rent.

Below is some web server software I have come across during my research for this project:

- Apache HTTP Server – Is the worlds most used web server software, and played a key role in the growth of the World Wide Web quickly becoming the most dominant HTTP server [36].

- Nginx – Is a web server with a strong focus on high concurrency, performance, and low memory usage [37].
- Cherokee – Is an open source cross-platform web server that runs on Linux, BSD, variants, Solaris, Mac OS X, and Microsoft Windows [37].

1.5.2.6. Graphical User Interface

The next development area I am going to discuss is graphical user interface (GUI), and in particular the different frameworks that can be adopted with my chosen programming language to develop an interface for the ‘MSc Properties’ staff to interact with the system I am going to produce during this project.

Below are some Java GUI frameworks I have come across during my research for this project:

- Abstract Window Toolkit (AWT) – Is Java’s original platform-dependent windowing, graphics and user-interface widget toolkit, and is part of the Java Foundation Classes (JFC) [38].
- Swing – Is a GUI widget toolkit for Java, was developed to provide a more sophisticated set of GUI components than the earlier AWT, and is also part of the JFC [39].

1.5.3. Testing Methodologies

As previously explained, this project will centre around developing a system to tackle the major problem of data and document management and sharing, and whilst meeting these functionality requirements of ‘MSc Properties’, the system also needs to be maintainable, dependable and usable. This means that I will need to carry out testing to ensure that the system I have developed for ‘MSc Properties’ is compliant with the specified requirements and that the system has no faults or errors at runtime of the system, or the system is at an acceptable level for ‘MSc Properties’.

1.5.3.1. Unit Testing

The first testing principle I am going to discuss is the unit testing approach to testing, and will be fundamental to ensuring each of the elements of the system work independently of each other (where elements are not coupled).

The piece of literature I am going to review for unit testing is the article called “*A Simple and Practical Approach to Unit Testing: The JML and JUnit Way*” written by Y. Cheon and G.T. Leavens [14]. This article looks at ways programmers can reduce the writing of labour-intensive code for unit testing, by writing formal specifications (for example, pre and post-conditions of methods).

The article goes on to explain that “writing formal specifications instead of test code makes

the programmer's task easier, because specifications are more concise and abstract than the equivalent test code, hence more readable and maintainable. Furthermore, by using specifications in testing, specification errors are quickly discovered, so the specifications are more likely to provide useful documentation and inputs to other tools" [14].

Below is some unit testing framework I have come across during my research for this project:

- JUnit Testing – Is a unit testing framework for the Java programming language, and is linked as a JAR at compile time [42].
- Java Modelling Language (JML) – Is a specification language for Java programs using Hoare style pre and post conditions and invariants, that follows the design by contract paradigm. Specifications are written in Java annotation comments to the source files [43].

1.5.3.2. System Testing

The next testing principle I am going to discuss is the system testing approach to testing, and will be fundamental to ensuring each of the elements of the system work together as they should.

The piece of literature I am going to review for system testing is the article called "*A UML-Based approach to System Testing*" written by L. Briand and Y. Labiche [15]. This article looks at system test cases being derived from the analysis stage documents such as use case diagrams and sequence diagrams.

The article goes on to explain that "Deriving test requirements from early artefacts produced at the end of the analysis development stage, namely use case diagram, use case description, interaction diagram associated with each use case (sequence or collaboration), and class diagram (composed of application domain classes and their contracts). This early use of analysis artefacts is very important as it helps devising a system test plan, size the system test task, and plan appropriate resources early in the life cycle. Once the low level design is complete, when detailed information is available regarding both application domain and solution domain classes, then test requirements can be used to derive test cases, test oracles and test drivers" [15].

Below are some testing technologies I have come across during my research for this project that will assist in bug tracking:

- Bugzilla
- The Bug Genie

1.6. Project Plan

As this is a large project, it is very important that I planned, monitored and managed the project smoothly from start to finish. I have used a Gantt chart, which provides a graphical illustration of the schedule of the project, broken down by project objectives, with completion dates for each objective, which has helped me track the activities in the project and make changes to work being carried out if necessary. This tool has been used to manage my time and allow me to stay on schedule as best as possible, as there was a lot of tasks that needed to be completed in a limited time frame.

This process of project planning is outlined in a software management article [5], where the article identifies “a recent update of the Chaos Report from the Standish Group, outlines a recipe for success that includes 10 items. The first three items are executive support, user involvement, and experienced project management” [5]. So project management is one of the 3 key factors to successful projects, and means I will need to ensure this project is correctly managed, so I can successfully achieve the project aim and objectives.

My project Gantt chart is documented under Appendices U.

I am now going to explain how the project objectives have been successfully completed by the project deadline date of 11 Jan 2015.

1. I wrote a project document outlining the details of the project, defining project objectives, scope, risks and approaches. I have constantly referred to this document to ensure the project progresses in the correct direction (Mid-Project Report).
2. I wrote a work plan outlining the project objectives, with deadlines for each objective (Gantt Chart).
3. I defined relevant resources for the project, outlining decisions made on technology, equipment and software applications to use, ensuring that I have tested equipment and software applications, and am competent with the use of the selected technologies, prior to the start of the development (required skills and resources).
4. I kept an eye on the project plan ensuring that objectives do not overrun past their completion date (where possible).
5. I stayed vigilant and alert to early warning signs of problems occurring in the project that may have resulted in the project being delayed and not meeting project deadlines.
6. I safeguarded against my project creeping outside of scope, so as new requirements were introduced during the development process, I had to ensure these are all still within available resources and overall aim and objectives of the project.
7. I managed risks as the project progressed, and as new risks were discovered, I had to evaluate them to ensure they do not cause a major problem to the project (Highlight Report).

8. I tried to keep my project supervisor informed of any major problems occurring during the project, and did at times, seek advice where necessary, to resolve major problems as early as possible.

1.7. Relevance to target award

Software Engineering is defined by Ian Sommerville as an engineering discipline concerned with all aspects of software production (specification, development, validation and evolution), and goes on to say it is concerned with the practicalities of developing and delivering useful software [1].

My project aim is to develop a distributed data and document management system, and to do this I had to explore the different software engineering techniques and decide which are best suited to tackling the software engineering task, and then develop and implement a piece of software that successfully meets the aim and objectives of the project.

This means the work I carried out during this project fits in with my target award MSc Computer Science (Software Engineering), because I applied software engineering models I have studied during my course such as agile to my software development. I also applied software engineering methodology I have studied during my course such as Inheritance and Encapsulation to my software development. I also applied the software engineering tools I have studied during my course such as unified modelling language (UML) to my software development. Lastly applied metrics such as cohesion, coupling, bugs etc. to my software development. By me exploring and applying these different software engineering techniques it allowed me to deliver useful software to 'MSc Properties' which in essence is Software Engineering.

1.8. Required Resources and Skills

1.8.1. Hardware

- Operating System – Windows, Solaris, Linux or OS X;
- Processor – Intel® Core™ i5-4288U CPU @ 2.60GHz (or similar);
- Memory – 8.00 GB (or similar)

1.8.2. Software

- Platform – Windows XP or higher (or similar);
- A JDK for Java 5 or later
- An Integrated Development Environment (NetBeans or similar)
- A concurrent version system (Git or similar)
- A bug tracking and testing tool (Bugzilla or similar)
- A web server (Apache or similar)

1.8.3. Access

I will require access to the following:

- MySQL database

1.8.4. Skills

- Research skills
- Project management skills
- Report writing skills
- Ability to use Unified Modelling Language to model the distributed system
- Ability to write code in Java, HTML, Java Script, PHP and SQL.
- Ability to implement design patterns such as Observer
- Ability to use frameworks and API's such as Spring and JRC respectively

I met these project resource and skill requirements, by ensuring I had the required hardware in place before development work began, I then downloaded the majority of the required software resources and of the ones I did download, I tested these to ensure they work appropriately. Once I carried out a literature review of the required skills, methods and methodologies I could employ to meet the project aims and objectives, I then undertook exercises to ensure that I have understood these methods and methodologies before development work began and if any problems arose during the development I attempted to seek assistance from my project supervisor to overcome these issues.

1.9. Ethics Approval

Ethics Approval is when a committee of University of Hertfordshire staff approve “any student undertaking a study involving the use of human participants which is undertaken as part of a programme of work for which the University of Hertfordshire is responsible for” [4].

My project will not require ethics approval because I did not undertake research that involved collecting data from human participants, and although my system does store business data which includes personal information, I used dummy information which replicates the personal information throughout the development.

2. Design

2.1. Introduction

I am now going to discuss the design decisions I made during this project, and how I came to make these decisions.

2.2. Software Lifecycle

Considering the project aim and objectives, and also the project background, I believe an agile method is the best software process model to choose from, this is because in a fast moving business environment, software needs to be ready and available as quick as possible, and as original software requirements can quickly become out of date, it makes software developed useless very quick.

Also due to the type of software I was developing, the software process model chosen needed to provide rapid development and delivery of software, and with the conventional plan driven software process models, it can be difficult to do this because of the amount of documentation that needs to be created and signed off, and the lack of interleaving development stages makes it difficult to cope with quickly evolving requirements.

Furthermore, as there will be an office manager from 'MSc Properties' working with the project assisting the development, it would make sense in being able to deliver software quickly to allow for this to be evaluated and confirm the project is moving in the correct direction at each iteration of the agile lifecycle, as it can be difficult to gather exact system requirements from clients without going over the development process and having something to evaluate and add to or remove from to develop a system that successfully meets the project aim and objectives.

The agile method I have chosen is the Agile Scrum methodology, and although I decided to use this agile method, as explained previously in the project background a business analysis and system requirements exercise was carried out with an officer from 'MSc Properties', which allowed me to have a fairly strong idea of what functionality 'MSc Properties' required from the system to be developed, although these were likely to evolve and did.

The information gathered put me in a position where I was able to carry out a fair amount of design work for the system prior to the first development iteration. I then used the divide and conquer technique which allowed me to break the development into smaller pieces, and accomplish one or a number of the smaller problems with each iteration of the development process.

The breaking down of the development into smaller pieces enabled me to tackle each smaller task on its own and then combine the solutions to the smaller problems to provide a solution to the original problem, which meant providing a solution to the original problem was easier and more manageable.

Finally, as you can see from Fig 2, from "Software Quality & Agile Methods" article, the figure outlines the different quality assurance techniques, which will enable me to ensure

that the software I produce for 'MSc Properties' is of a good quality, due to the quality assurance practices that occur during the agile software lifecycle, which will assist me to meet one aspect of the aim, which is to produce dependable software.

2.3. Modelling System Behaviour

I used a number of design techniques to assist in the modelling of the system behaviour and am now going to explain what design techniques I used and why I decided to use these for the development.

However due to the size of the project and due to the level of resources available, I decided that for me to successfully achieve as much of the project aim and objectives, I would not be able to develop diagrams to model the entire system behaviour, and instead I selected a sub set of the system functionality to model.

2.3.1. Use Case Diagrams

For this development I decided to develop a number of use case diagrams which have been documented under Appendices A. By developing use case diagrams, this enabled me to identify the relationships between actors (roles within the system, for example a user of the 'MSc Properties' system) and use cases (functions within the system, for example creating a property), which allowed me to identify what actors was involved with which use cases.

The use case diagrams developed during the project were not only used to model the system behaviour, but I also used them to create test scripts for system testing, as the use case diagrams outlined the different functions that should occur within the system, and therefore can be used to carry out the black-box system testing, which will be explained in further detail in the implementation section of this report.

2.3.2. Class Diagram

For this development I decided to develop a class diagram which have been documented under Appendices B. By developing a class diagram, it enabled me to visualise the structure of the system I intend to develop, and allows me to document the variables and methods of each class, and how classes are related to each other, for example composition, multiplicity, inheritance, etc.

2.3.3. Enhanced Entity Relationship Diagram

For this development I decided to develop an enhanced entity relationship diagram (ERD), which have been documented under Appendices C. An ERD is very similar to a class diagram, but instead of visualising the structure of the system, it visualises the structure of

the database to be developed, and allows me to document the tables, columns and relationships between tables, for example one-to-one, one-to-many, many-to-one and many-to-many.

2.3.4. Sequence Diagrams

For this development I decided to develop sequence diagrams, which have been documented under Appendices D. By developing sequence diagrams, it enabled me to visualise the interaction between classes/objects within the system to be developed. Furthermore, the sequence diagrams also allowed me to produce test scripts along with the use case descriptions, for black box testing which will be explained in further detail in the implementation section of this report.

2.3.5. Storyboard

For this development I decided to develop a storyboard for the graphical user interface (GUI), which has been documented under Appendices E. The storyboard outlines the different screens I will be attempting to develop during this project and show how each of these flow on to one another through user interaction with the system.

The storyboard will also provide an annotation of how I plan to develop the GUI with regards to different layout managers, components, font, colours, sizes etc. to be used.

3. Implementation

3.1. Introduction

I am now going to discuss the implementation decisions I made during this project to produce a distributed system from the design model produced in the previous section, and in doing so, successfully meet the project aim and objectives.

I decided the best way to go through the implementation decisions I made, was to take you through each of the iterations of the project development, as I have decided to adopt the Agile Scrum methodology.

And although with each iteration I carried out unit, integration, and system testing, I have decided to leave the explanation of testing to its own section, where I will then why certain decisions were made, and what the outcomes of the testing was.

3.2. Iteration Cycle 1

As this project involved developing a distributed system, for the first iteration I decided to implement the basic functionality of the system as a desktop version, which involved

adopting a number of the design patterns I identified during my research for this project.

Although I implemented a desktop version of the system prior to implementing the networking functionality, I knew this was going to be a distributed system so I decided to split the desktop version of the system into two packages. The two packages are a common package which would include any classes and interfaces that will be adopted by both the server package and the client package (once I implement networking functionality), and the server package which will include the model and controller of the model, for the system I am to implement.

3.2.1. Design Patterns

As explained previously, I decided to adopt a number of design patterns to assist in producing software that is maintainable, dependable and usable. I am now going to discuss the design patterns I adopted during cycle 1 and why these were relevant to this project.

3.2.1.1. Iterator Pattern

During this project, the iterator pattern was one of the most frequent design patterns adopted, this is because as I was producing a system that will hold a lot of data, and one of the functions of the system will be to search the data using different search criteria, it would be crucial to be able to traverse over lists of data to extract data that matches the search criteria.

As you can see from Appendices F, I used a variety of Iterator patterns such as the while loop and the enhanced loop, and each provided its own benefits.

The while loop allows the programmer to loop over a set of statements as long as the condition within the while loop condition is true, which gives the programmer flexibility to supply any Boolean condition to determine if the while loop statement should be executed.

The enhanced for loop is similar to the standard for loop, however is less flexible, and should only be used when the programmer needs to loop over all of the elements within a list, and don't need to know the index of the object being retrieved [43].

3.2.1.2. Inheritance

As you can see from the class diagram in Appendices C, I have adopted the design pattern inheritance, this is to re-use code and make use of an important programming technique called polymorphism. Inheritance is achieved through the use of parent (super classes) and child classes (sub classes), where the child class has a "is a" relationship, to the parent class, and the child class extends the parent class.

By implementing inheritance, it also makes the system easier to evolve, for example if 'MSc Properties' was to expand and create a new Agreement, the system will be able to be amended to add an additional AgreementImpl subclass which can then make use of any

methods that use polymorphism without any changes to the existing code, an example of a method which makes use of polymorphism is in Appendices G.

3.2.1.3. Object Composition

As you can see from the class diagram in Appendices C, I have adopted the design pattern object composition, which similarly to inheritance allows for the reuse of code and polymorphism.

However, the difference with object composition to inheritance is that, a class, say class A, will contain a variable of another class, say class B, that implements functionality that is desired by class A, meaning that class A re-uses the code of class B, by implementing a “has a” relationship between class A and class B.

3.2.1.4. Singleton Pattern

As you can see from Appendices H, I have adopted the design pattern singleton pattern, this is to ensure that only one instantiation of an object can occur, to ensure that only 1 object is used throughout the system in all instances when this object is needed. I have documented an example of the implementation of the singleton pattern within Appendices H, however I have used the singleton pattern a few times within the development.

3.2.2. Graphical User Interface

Unfortunately, during the development process, I had some issues with building the graphical user interface (GUI), and although I could have decided to use a GUI builder that is provided by most integrated development environments (IDE) such as Netbeans, I wanted to build the GUI from scratch, writing all the code myself.

As you can see from Appendices J, I was attempting to use the grid bag layout manager for data entry forms (Person creation, Application creation, etc), this is because this layout manager provides flexibility to the way in which components such as buttons, labels etc, can be laid out on screen, as they are laid out in cells, with columns being any width, and rows being any height, providing greater flexibility for layout. Within Appendices J, I have documented how I attempted to implement GridBag Layout manager.

Another concept I was adopting through the system, was the model-view-controller (MVC) architectural pattern, which I have explained was being adopted, with the client package being the view, the Server class being the controller, and the rest of the server package classes being the model.

However, I also adopt MVC within the GUI, with the ClientImpl being the model for the system, and a main frame GUI such as the HomeForm being the controller, and then panels of components, that have been added to the main frame are the view such as a JTable or JTree.

The controller (main frame) would then deal with updating the view (panels of GUI components), as and when something happens in the model (ClientImpl class). The use of MVC within the GUI reduces the level of coupling between the model and the view, as the controller manages communication between the two.

The adoption of the Model-View-Controller, allows me to develop good software for 'MSc Properties' as it ensures that the model and the view are not coupled, meaning that the two components work independently of each other, which again allows me to meet aspects of the project aims as it ensures that the software is maintainable, and open to evolution due to the low coupling between components.

I have documented the main screens of the GUI I was attempting to develop for this project in Appendices E, along with annotations on what components and layout managers I was attempting to use to develop the GUI.

3.3. Iteration Cycle 2

For the second iteration of this project, I decided to implement the object relational mapping design pattern to store and manage the object-oriented objects in the database class of the system ('MSc Properties' business data), into the MySQL database I am going to create.

I need to make use of an object relational mapping design pattern because as you can see from the class diagram in Appendices B, the data stored in the Database class is non-scalar values, however as you can see from the enhanced entity-relationship diagram in Appendices C, the data stored in the MySQL database is scalar values and therefore the non-scalar values need a way to be broken down and stored in the MySQL database.

For the project, I had to create the database that was designed in the enhanced entity-relationship diagram using MySQL Workbench, and set up the settings for the database, such as primary keys, foreign keys, column data types and enforce referential integrity, which will ensure that any data stored in the database has to meet the rules set out by the relationships, meaning a foreign key value of a table, has to be present in the related table as a primary key.

Once the database was created I then had to use an application program interface (API) to allow the system to interact with the MySQL database. I need to use an API for the application and MySQL database because the two parts don't know how to communicate with each other, and the API sets out how communication between these two take place.

The API I will use for this project is Java Database Connectivity (JDBC) and is part of the Java Standard Edition platform. Once I had created the MySQL database, I then had to implement a number of JDBC techniques to enable me to successfully achieve the project aim and objects associated with the MySQL database, and these are listed below and documented in Appendices K.

- Connecting to the database
- Loading System data at start up (Read)
- Create, Update and Delete Data.

As you can see from Appendices K, I use a number of methods to allow my Database class to connect to the database, and allow for objects created, updated and deleted within the system, to then be updated within the MySQL database with the usual database create, read, update and delete (CRUD) functions.

Furthermore, as explained in Appendices K, the loading of system data was one of the more difficult tasks, as I had to ensure that no objects were loaded up prior to an object that the loaded object is dependent on was loaded, and also needed to ensure all system elements such as title codes, religion codes was loaded up first.

This is because, loading objects which are dependent on other objects prior to the dependent objects being loaded, can cause issues for the system load, as there is if statements preventing objects from being created if the elements they are trying to be created with, does not already exist in the system.

3.4. Iteration Cycle 3

For the third iteration of this project, I decided to implement the networking functionality, to enable a user of the system to have access to data and documents stored within the system, where the system is not necessarily local to them.

From the research undertaken, I decided to implement remote procedural call functionality, and as I decided to write the project code in Java, making use of the object-oriented concepts, the Java API, Remote Method Invocation (RMI) was the best decision for me as this enables the user in one Java Virtual Machine (JVM), which may or may not be remote to the server, to be able to invoke a method on a server object in another JVM.

Furthermore, by adopting RMI, instead of implementing the sockets, threads, and serialization/marshalling of objects allows the programmer to not worry about these aspects of networking as the RMI framework takes care of this for the programmer.

For me to implement RMI there was a number of steps I needed to take, which are documented within Appendices L, and are listed below:

- Set up Server
- Set up Client
- Push vs Pull

3.4.1. Push vs Pull

As outlined previously in the literature review carried out, there are two concepts for a distributed system to manage data, which outline the way a client and server interact with each other and initiate tasks, these are called the push model and the pull model.

For this project I decided to implement both forms of client server interaction, this is because when a client wants to perform an action, I decided to implement the pull model, this meant that clients of the 'MSc Properties' system will request for the server to perform an action, so the client is initiating a request, and the server is responding to that request, by performing some action and returning information to the client. Therefore, the client is pulling from the Server, and the implementation of the pull model and how this was implemented is documented in Appendices L.

However, I also decided to implement the push model, which meant that the server of the 'MSc Properties' system will initiate an action through a push to the client, and then the client will deal with this as required. The pull model is only used when another client updates an agreement, or rent account, which can then be reflected on all client home screens, and is used in conjunction with the observer pattern, which is discussed later in this report, and documented in Appendices N.

3.4.2. Local Reference vs Remote Reference

With Remote Procedural Call functionality, there are two big concepts with regards to local and remote references. This is because, a client server implementation, can either have the client create an object locally and pass a copy of this to the server for a copy to be stored locally (local reference), or the client can request the creation of an object by the server (remote reference), and a remote reference to the object is then passed back to the client.

For this development, I decided to implement the remote reference version of RMI implementation, this is because as there is only one copy, it is easier to keep track of updates to objects, as the client is able to invoke methods on these remote object references, and then the changes can be seen by any client who has or requests a remote reference of the object.

Whereas with the local reference, an update to a local method, is not seen by the server (nor any other client), and similarly, an update made by the server is not seen by the client, meaning that each update has to be pushed to all clients and the server (if it was a client who made the update), and it then becomes difficult to keep up-to-date records of each local reference (with clients and the server).

3.5. Iteration Cycle 4

To implement the document management functionality, I decided to write my own document management functionality that would deal with storing documents and version control,

instead of using a document management framework such as Apache JackRabbit.

To do this I firstly created a Document class which stored a file and any previous versions of that file. I then had to implement methods both on the Client side and the Server side, that will convert a document into an array of bytes, this allows for the array of bytes to be passed between the client and server and then reconstructed at the other side, for the client or user to either view or save depending on whether a document is being uploaded to the server or downloaded by a client.

The implementation of the document management system, has been documented within Appendices M, and the stages taken is listed below:

- Uploading a document to the server
- Downloading a document from the server
- Version Control

3.6. Iteration Cycle 5

For the next iteration I decided to implement the observable pattern, which will be used to update a client of 'MSc Properties' graphical user interface (GUI) when an agreement or rent account is updated, without the ClientImpl class from the Client package actually knowing about the GUI.

I decided to implement the push model of the observable pattern, instead of the pull model that the client server data exchange is predominantly based on (as explained in iteration cycle 3 – push vs pull), this is because the pull variant can be quite costly, because each observer will invoke the method to pull through the current state even if there have not been any changes, whereas the push variant only updates observers when an update has occurred. This selection was best for this type of implementation because there will not be a frequent number of updates to 'MSc Properties' agreements or rent accounts so it is unlikely the observers will need to be updated all the time.

To implement the observer design pattern, there was a number of steps I had to take, which has been documented in Appendices N.

3.7. Iteration Cycle 6

My final iteration cycle was to automate a number of tasks that have to be carried out by 'MSc Properties' officers. To do this I decided to use the Java class TimerTask, and created a class called TaskGenerator, which extended TimerTask class from the Java util package.

Although this may not have been the best solution to accomplish the task, due to time constraints, I was not able to implement a task scheduling framework such as Quartz, which would have allowed the client to schedule tasks independently, whereas with this

implementation, the scheduling of tasks is hard coded into the Server side classes, and therefore would require amendments to the code, and then recompiling if the code (which would require server downtime), for any new task schedule to be implemented.

By implementing the task scheduling functionality, it means that an 'MSc Properties' employee will not have to manually create rent transactions for each tenancy every month (the start date of the tenancy determines when the rent is charged to the rent account), or the manager will not have to generate monthly reports each month, which will reduce the work load of 'MSc Properties' employees, but still provide the monthly statistics and deal with certain tasks that were previously done manually.

I have documented my implementation of the task scheduling functionality within Appendices O.

3.8. Testing

As explained previously in iteration cycle 1, I was attempting to carry out unit testing, integration testing and system testing during each iteration cycle, but I decided to document the majority of my testing results in my Appendices P, Q & R, with a description within this Testing section of the report, on what decisions was made during the project and why, and what the outcome of the results were.

3.8.1. Unit Testing

For the unit testing, I decided to use JUnit testing, this allowed me to write the tests prior to development, and as I was producing the classes, I could just run the Junit tests to ensure the semantics of the classes was correct, and was working as they should. Also by writing Junit tests, it meant that I was not writing long testing class, and having to amend the test classes with each amendment to a class.

For the unit testing I adopted the white box testing approach, which meant that I coded my test classes with the knowledge of how my classes were programmed, meaning I could take advantage of knowing where the boundaries of if statements were, etc., and adopt a testing policy where I will test at boundaries as well as using incorrect and correct data to test.

As you can see from Appendices P, I have carried out unit testing for each class, by creating test classes for each class, and the test results are documented within Appendices P.

As you can see from Appendices P, each class is able to track who created the object, and who has modified the object, it is also able to add notes to each object or a comment, as well as store the information associated with each object.

For example, for the Tenancy test class I created, the test Tenancy object stores the inherited information from Agreement (its super class) such as agreement reference, agreement name, start date, expected end date, the actual end date, the length of the tenancy, the office the tenancy is associated with, the account reference that is associated with the tenancy, as well as who created the tenancy and what date it was created, and a

list of notes, documents, and modifications (modified by and modified date).

The test Tenancy object also stores its own information, such as an application object, a property object, a tenancy type element, the rent value and charges. And majority of this information is able to be updated through the Tenancy class by invoking its own or inherited methods, and is shown to work within Appendices P along with the other unit results.

3.8.2. Integration Testing

For the integration testing, I decided to adopt the Big Bang testing along with white box testing techniques, instead of top-down, or bottom-up testing, this is because I only carried out integration testing after the final iteration due to time constraints, so although this method requires nearly all of the units to be developed and integrated, this was the case when I undertook integration testing.

So although the normal process of a large agile software development, is to carry out unit testing and then either top-down or bottom up integration testing after each iteration cycle (testing a smaller number of integrated units, either from the highest priority down, or lowest priority up), and then system testing would occur, I decided it would be best for me to carry out integration testing at the final iteration of my project, which meant all of the modules was basically complete, and therefore could use the big bang method.

For the integration testing I decided to test all of the server side classes except for the Server class. I decided to use the Database class as the basis for my integration testing, as the database class dealt with storing all of the system data, and loading all of the system data at system start up. To carry out the integration testing I created a test class which created a database object, and my test class was then a client of the Database class, which meant that I could test a large amount of the server functionality to ensure the system (excluding the server class) worked.

As you can see from Appendices Q, the Database class was able to create all of the individual objects such as People, Applications, Tenancies, etc. and write these to the MySQL database. The Database class is also able to retrieve objects from the MySQL database class and load these up, and update them as necessary, which in turn will be written to the MySQL database. Furthermore, the database class can delete any of these objects that have been created, as long as there has been no modification to the object, such as an update, or an object being created for that object, such as a note object. Also as explained, at system start-up the database class loads up all the system data from database correctly, with all objects being loaded up as they were for the last server session. All of the integration tests are shown within Appendices Q.

3.8.3. System Testing

For the system testing, I decided to use black box testing, meaning that the testing process

would not use knowledge of how the program was coded, but instead use the use case descriptions, and sequence diagrams, along with the system functionality drawn up, to test the system, and the test results for the system testing is documented in Appendices R

Unfortunately I was unable to test the entire system functionality, firstly due to time constraints, but also due to the graphical user interface not being implemented, which meant that the security for users was unable to be implemented, as I was planning to use the boolean variables within the User class, which is assigned to each client when they log in, to determine the users security level, for example if a user had false for update variable, then either the update button would not be visible, or would be greyed out (inactive) meaning that the client would only be able to perform particular functions depending on their security level.

However, even though the GUI was not implemented, meaning I did not fully implement the home screen with the JTables that would display the list of agreements and rent accounts for the user's office, I was able to test the observer pattern functionality by allowing the observer objects to invoke a `System.out.println()` method to print a message to the console of each user, whenever an update occurred, showing that the observer pattern was successfully implemented.

Also from the system testing, I was able to ensure that the server class is able to be passed a file in bytes, which can then be put back together, this file is then stored as a document object, and the file is then saved locally to the server. The file location and some other document details is then saved to the MySQL database through the database class, and again is able to be amended, with previous versions of the document being saved in a list, and being able to be retrieved for viewing the same way as the current document is viewed. The document objects would then be able to be loaded up, the same way in which other system objects are loaded at system start up.

Furthermore, during the system testing, I was able to confirm that the task scheduling worked, and although I have not fully tested the methods that will run at the scheduled intervals, the implementation of the task scheduling actually works and is shown through `System.out.println()` statements, which are invoked at set intervals depending on the initialisation of the TaskGenerator object. Again all of the system testing results are documented in Appendices R.

Finally, there is also a bugs list for the bugs that I identified and resolved during the testing phases, and this is documented under Appendices S.

4. Conclusion

I believe that overall this project was a success, this is because although I did not achieve all of the advanced objectives I set out for the project, and although I believe with further resources I may have been able to achieve the project objectives with higher quality, I did successively achieve a large portion of both the core and the advanced project objectives.

Of the core objectives, I was unable to fully achieve the following, ensuring **all** resources were available (not being able to install Apache to a Windows machine, meaning I was not able to create a web server, nor was I able to install Bugzilla, meaning I had to use Excel to record any bugs within a Bugs List), writing **all** test scripts, carry out **all** unit and system tests, and finally develop a user manual.

Of the advanced objectives, I was unable to develop a website or develop a web server to manage the communication to the MySQL database, which would enable the website to be more dynamic, so when the database is updated with new properties to let, these will then become available to view through the website.

Also although I believe the project was a success, the project plan I defined earlier in the project was not very realistic with regards to time allocation for tasks, and although I believe this was because of the lack of experience I have with developing systems of this size, it resulted in me allocating not enough time for tasks, which then had a knock on effect on other tasks when tasks overran.

However, although my project plan was not realistic I believe I managed the project well through constantly assessing the risks to the project which could result in me not being able to successfully meet the project aim and objectives and was important in me realising early in the development that there was a major risk of not achieving all goals due to a lack of time, and making decisions with regards to the number of design documents produced and functionality I was going to implement to allow me to achieve a large portion of the project aim and objectives.

The final area which I believe I did not do so well in is the development of the graphical user interface, as although I had designed a number of screens, and understood the theory to implement what I was trying to achieve, when I was actually attempting to develop the GUI, the alignment of components on the screen was not correct, and because this is a project that will need to display large amounts of information to the client, I needed to ensure that the components were laid out correctly, to allow for the Client to understand the information, for example labels for field entry boxes, would need to be aligned correctly for an 'MSc Properties' employee to be able to use a creation form correctly. This meant that I got bogged down with alignment issues and spent unnecessary time, trying to implement the GUI. Also by me not implementing the GUI, it meant that I was unable to successfully implement the observer pattern or user security (explained in system testing section), although as explained I was still able to get the observer pattern to work but not in its intended manner.

Another area I believe was a success was the literature review, as this enabled me to get different views of different experts within Computer Science, and which assisted me with design and implementation choices made during the project.

Lastly my choices of methodology overall were good, and I believe may have only been bettered in certain areas. This is because the choice of language I decided to code my system in, Java, was the best choice as it allows compiled code to run on all platforms, meaning that the code is usable across a wide range of devices, and as 'MSc Properties' may

have different computer set ups at different sites, it allows for the system to still work on the different set ups.

The choice of Remote Method Invocation (RMI) as networking functionality, was the best because I was coding in Java and as explained before, RMI allows for objects to be passed as values, and therefore behaviour of an object can be passed across a network from the server to the client, instead of just primitive values such as integers. Also RMI makes it easier for the programmer as the programmer does not need to manage the sockets, threads and serialization of objects, making the development of the system easier.

The choice of software's used for this project were also good, and is highlighted in my choice to use a concurrent version system, as I encountered some issues during the implementation stage which required me to roll back to a previous stage. And if I was just using external back-ups with no version control I would have not been able to roll back to a previous version to resolve the issues encountered.

5. Evaluation

As I outlined in the conclusion, I believe that overall the project was a success, however there is a few decisions I would have made differently if I was to complete the project again.

The first change to the project would be to set out a realistic project plan from the offset, which would result in a more manageable project, as I would hopefully not run over allocated time for tasks, meaning an amendment to documents I wanted to produce and functionality implemented.

The second change I would make to the project is to ensure that I had a Linux operating system available to me, because from research, I have found that this is the best operating system to manage a web server with, and also it would have allowed me to overcome the issues of installing Bugzilla, which in turn would have meant that I had a better bug tracking tool than what I decided to use due to not being able to install Bugzilla on the Windows machine I had available.

The final change to the project would be to amend the design of the project, this is because in the early stages of the development I made a decision to make a number of classes be related to others through object composition, for example the Tenancy class is related to both an application and a property, as a Tenancy has an application, and has a property, which means that the Tenancy is tightly coupled to both of these classes. This means that if the software was to evolve and the application class was to be amended it could result in the Tenancy class also needing to be amended.

The reason for this was because as it is a distributed system, the client will need to make remote calls back to the Server to retrieve objects, whereas because the Tenancy class has both an Application and Property related to it, the information is held within the tenancy and therefore the client just needs to invoke a get method to return a remote object of the Application or Property, without having to make a remote call back to the Server. Whereas

if these were loosely coupled, by only holding a reference such as application ref, or property ref, then the client would need to make a remote call back to the server using the reference to get this object, which I believed would increase network traffic, and workload on the server.

However, since carrying out my unit testing, I have realised that these objects are tightly coupled, and to create a Tenancy, it requires the actual Application object and Property object to be initialised before the Tenancy object can be created. Although one way I have overcome this tight coupling between these dependant classes, is by the use of interfaces, so Tenancy class does not actually interact with a concrete class, but instead the interface, which means that I am able to control what methods the classes are able to invoke, which increase encapsulation of the system, and therefore allows the concrete classes to be able to be amended as long as they still implement the interface of the concrete class.

Bibliography

References

1. Sommerville, I. (2011). Introduction. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P1-26.
2. Citizens Advice. (2015). *The benefit cap - what you need to know*. Available: <https://www.citizensadvice.org.uk/benefits/the-benefit-cap/the-benefit-cap-what-you-need-to-know/>. Last accessed 20th Jun 2015.
3. Enfield Council. (2013). *Enfield's Homelessness Strategy 2013-2018*. Available: http://www.enfield.gov.uk/download/downloads/id/8004/enfields_homelessness_strategy_2013-2018. Last accessed 20th Jun 2015.
4. Hunt, B. (2015). *UH Ethics Approval*. Available: <http://www.studynet2.herts.ac.uk/ptl/common/ethics.nsf/Homepage?ReadForm>. Last accessed 19th Jul 2015.
5. Cockburn, A and Highsmith. (2001). Agile Software Development: The People Factor. *Computer*. 34 (1), p131-133.
6. Huo, M, Verner, J, Zhu, L and Babar. (2004). Software quality and agile methods. *Computer Software and Applications Conference*. 1 (1), p520 - 525.
7. H, Gomaa. (2001). ICSE 01 Proceedings of the 23rd International Conference on Software Engineering. *Designing Concurrent, Distributed, and Real-Time Applications with UML*. 1 (1), p737 - 738.
8. Sommerville, I. (2011). Software processes. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P27-55.
9. A.D, Birrell and B.J, Nelson. (1984). ACM Transactions on Computer Systems. *Implementing remote procedure calls*. 2 (1), p39 - 59.
10. J.P, Martin-Flatin. (1999). Push vs. pull in Web-based network management. *Integrated Network Management, 1999. Distributed Management for the Networked Millennium. Proceedings of the Sixth IFIP/IEEE International Symposium on*. 1 (1), p3 - 18.
11. D, Habam and K.G, Shin. (2002). Application of real-time monitoring to scheduling tasks with random execution times. *Software Engineering, IEEE Transactions on*. 16 (1), p1374 - 1389.
12. M.L, Horowitz. (1991). An Introduction to Object-Oriented Database and Database Systems. *Object-Oriented Database and Database Systems*. 1 (1), p1 - 86.
13. C, Amza, A, Chanda, S, Elnijety, R, Gil, K, Rajamani, W, Zwaenepoel, E, Cecchet and J, Marguerite. (2002). Specification and implementation of dynamic Web site benchmarks. *Workload Characterization, 2002. WWC-5. 2002 IEEE International Workshop on*. 1 (1), p3 - 13.
14. Y, Cheon and G.T, Leavens. (2002). A Simple and Practical Approach to Unit Testing: The JML and JUnit Way. *ECOOP 2002 — Object-Oriented Programming*. 2374 (1), p231 - 255.
15. L, Briand and Y, Labiche. (2002). A UML-Based approach to System Testing. *Software and Systems Modeling*. 1 (1), p10 - 42.
16. Wikipedia. (2012). *Use Case Diagram*. Available: https://en.wikipedia.org/wiki/Use_Case_Diagram. Last accessed 10th Oct 2015.
17. Wikipedia. (2004). *Data flow diagram*. Available: https://en.wikipedia.org/wiki/Data_flow_diagram. Last accessed 10th Oct 2015.
18. Wikipedia. (2005). *Class Diagram*. Available: https://en.wikipedia.org/wiki/Class_diagram. Last accessed 16th Aug 2015.

19. Wikipedia. (2004). *Entity Relationship Model*. Available: https://en.wikipedia.org/wiki/Entity%E2%80%93relationship_model. Last accessed 23rd Oct 2015.
20. Wikipedia. (2004). *Class-responsibility-collaboration card*. Available: https://en.wikipedia.org/wiki/Class-responsibility-collaboration_card. Last accessed 11th Aug 2015.
21. Wikipedia. (2005). *Sequence Diagram*. Available: https://en.wikipedia.org/wiki/Sequence_diagram. Last accessed 10th Oct 2015.
22. Wikipedia. (2002). *Storyboard*. Available: <https://en.wikipedia.org/wiki/Storyboard>. Last accessed 10th Oct 2015.
23. Wikipedia. (2005). *Observer pattern*. Available: https://en.wikipedia.org/wiki/Observer_pattern. Last accessed 30th Oct 2015.
24. Wikipedia. (2002). *Singleton pattern*. Available: https://en.wikipedia.org/wiki/Singleton_pattern. Last accessed 9th Oct 2015.
25. Wikipedia. (2003). *Strategy pattern*. Available: https://en.wikipedia.org/wiki/Strategy_pattern. Last accessed 10th Aug 2015.
26. Wikipedia. (2004). *Creational pattern*. Available: https://en.wikipedia.org/wiki/Creational_pattern. Last accessed 10th Aug 2015.
27. Wikipedia. (2003). *Iterator pattern*. Available: https://en.wikipedia.org/wiki/Iterator_pattern. Last accessed 10th Aug 2015.
28. Wikipedia. (2003). *Composite pattern*. Available: https://en.wikipedia.org/wiki/Composite_pattern. Last accessed 10th Aug 2015.
29. Wikipedia. (2001). *Inheritance (object-oriented programming)*. Available: [https://en.wikipedia.org/wiki/Inheritance_\(object-oriented_programming\)](https://en.wikipedia.org/wiki/Inheritance_(object-oriented_programming)). Last accessed 10th Aug 2015.
30. Wikipedia. (2005). *Object composition*. Available: https://en.wikipedia.org/wiki/Object_composition. Last accessed 10th Aug 2015.
31. Wikipedia. (2002). *Object-relational mapping*. Available: https://en.wikipedia.org/wiki/Object-relational_mapping. Last accessed 31st Aug 2015.
32. The Apache Software Foundation. (2006). *Apache JackRabbit*. Available: <https://jackrabbit.apache.org/jcr/index.html>. Last accessed 11th Aug 2015.
33. JBossDeveloper. (2010). *Modeshape*. Available: <http://modeshape.jboss.org/>. Last accessed 11th Aug 2015.
34. Wikipedia. (2002). *Microsoft Access*. Available: https://en.wikipedia.org/wiki/Microsoft_Access. Last accessed 11th Aug 2015.
35. Wikipedia. (2001). *MySQL*. Available: <https://en.wikipedia.org/wiki/MySQL>. Last accessed 31st Aug 2015.
36. Wikipedia. (2001). *Apache HTTP Server*. Available: https://en.wikipedia.org/wiki/Apache_HTTP_Server. Last accessed 11th Aug 2015.
37. Wikipedia. (2007). *Nginx*. Available: <https://en.wikipedia.org/wiki/Nginx>. Last accessed 11th Aug 2015.
38. Wikipedia. (2010). *Cherokee (web server)*. Available: [https://en.wikipedia.org/wiki/Cherokee_\(web_server\)](https://en.wikipedia.org/wiki/Cherokee_(web_server)). Last accessed 11th Aug 2015.
39. Wikipedia. (2002). *Abstract Window Toolkit*. Available: https://en.wikipedia.org/wiki/Abstract_Window_Toolkit. Last accessed 11th Aug 2015.
40. Wikipedia. (2003). *Swing (Java)*. Available: [https://en.wikipedia.org/wiki/Swing_\(Java\)](https://en.wikipedia.org/wiki/Swing_(Java)). Last accessed 11th Aug 2015.

41. D, Matuszek. (2009). *Enhanced for Loop*. Available: <https://www.cis.upenn.edu/~matuszek/General/JavaSyntax/enhanced-for-loops.html>. Last accessed 27th Nov 2015.
42. Wikipedia. (2001). *JUnit*. Available: <https://en.wikipedia.org/wiki/JUnit>. Last accessed 10th Dec 2015.
43. Wikipedia. (2005). *Java Modeling Language*. Available: https://en.wikipedia.org/wiki/Java_Modeling_Language. Last accessed 10th Dec 2015.

Further Reading

1. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Characterization of Distributed Systems. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P17-52.
2. Sommerville, I. (2011). Distributed software engineering. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P479-507.
3. Cockburn, A. and Highsmith, J. and Bohem, B. (2001). Agile Software Development: The Business of Innovation. *Computer*. 1 (1), p131-133
4. Sommerville, I. (2011). Requirements engineering. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P82-117.
5. Sommerville, I. (2011). System modeling. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P118-146.
6. Connolly, T. and Begg, C. (2005). Normalization. In: McGettrick, A. *Database Systems A Practical Approach to Design, Implementation and Management*. 4th ed. United States of America: Pearson. P387-414.
7. Connolly, T. and Begg, C. (2005). Entity-Relationship Modeling. In: McGettrick, A. *Database Systems A Practical Approach to Design, Implementation and Management*. 4th ed. United States of America: Pearson. P387-414.
8. Sommerville, I. (2011). Design and implementation. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P176-204.
9. Reges, S. and Stepp, M. (2011). Graphical User Interface. In: Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Building Java Programs A Back to Basics Approach*. 2nd ed. Boston: Pearson. P846-909.
10. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Remote Invocation. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P201-246.
11. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Distributed Objects and Components. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P351-396.
12. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Transactions and Concurrency Control. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P691-742.
13. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Distributed Transactions. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and

- Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P743-780.
14. Connolly, T. and Begg, C. (2005). SQL: Data Manipulation. In: McGettrick, A. *Database Systems A Practical Approach to Design, Implementation and Management*. 4th ed. United States of America: Pearson. P112-156.
 15. Connolly, T. and Begg, C. (2005). Security. In: McGettrick, A. *Database Systems A Practical Approach to Design, Implementation and Management*. 4th ed. United States of America: Pearson. P541-571.
 16. Connolly, T. and Begg, C. (2005). Transaction Management. In: McGettrick, A. *Database Systems A Practical Approach to Design, Implementation and Management*. 4th ed. United States of America: Pearson. P572-629.
 17. Coulouris, G. and Dollimore, J. and Kindberg, T. and Blair, G. (2012). Security. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Distributed Systems Concepts and Design*. 5th ed. United States of America: Pearson. P479-536.
 18. Sommerville, I. (2011). Software testing. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P205-233.
 19. Sommerville, I. (2011). Project management. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P593-617.
 20. Sommerville, I. (2011). Project planning. In: Horton, M. and Hirsch, M. and Goldstein, M. and Bell, C. and Holcomb, J. *Software Engineering*. 9th ed. Boston: Pearson. P618-650.
 21. Waldo, J. (1998). Remote procedure calls and Java Remote Method Invocation. *Concurrency, IEEE*. 6 (3), P5-7.
 22. Guan, H. and Ip, H. and Zhang, Y. (1998). Java-based approaches for accessing databases on the Internet and a JDBC-ODBC implementation. *Computing & Control Engineering Journal*. 9 (2), P71-78.
 23. Wikipedia. (2004). *Data Flow Diagram*. Available: 17.
https://en.wikipedia.org/wiki/Data_flow_diagram. Last accessed 16th Aug 2015.