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

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

(Signature of student) ________________________________
Summary

The aim of this project is to produce an open-source workflow development framework for the .NET platform, which will seek to provide developers with a toolkit to construct a bespoke workflow system for their business scenario.

This project explores the workflow domain, software development methodologies and the different technical approaches available for implementing the various components. Such technical analysis will include discussions on, but not limited to; distributed computing (Web Services & CORBA), Object-Relational Mapping, ADO.NET, ASP.NET and scheduling.

The system is then evaluated and conclusions regarding the success of the project are drawn, to include feedback from a .NET professional with workflow development experience.
Acknowledgements

Firstly, I would like to thank Natasha Shakhlevich and Nick Efford for their valuable feedback throughout the project.

I would also like to thank Gordon Eddie for kindly agreeing to review the final product – I look forward to working (all the hours god sends) with you next year.

A big thank-you must also go out to those who proof read this report; Lois, Tony Smith and Charles Donald.

Finally, I would like to thank Jack Bauer for keeping me entertained throughout the project. Tony, Edgar, President Palmer & Michelle – you will be dearly missed.
# Table of Contents

1. **Introduction** .................................................................................................................. 1  
   1.1 Problem Definition ........................................................................................................... 1  
   1.2 Project Aim & Objectives ............................................................................................... 1  
   1.3 Minimum Requirements ................................................................................................. 2  
   1.4 Project Deliverables ........................................................................................................ 2  
   1.5 Relevance to Degree ....................................................................................................... 2  
2. **Project Management** ....................................................................................................... 3  
   2.1 Introduction .................................................................................................................... 3  
   2.2 System Design Methodologies ....................................................................................... 3  
   2.3 Project Schedule ............................................................................................................ 6  
3. **Workflow Management Systems** .................................................................................... 7  
   3.1 Introduction .................................................................................................................... 7  
   3.2 What is Workflow? ........................................................................................................... 7  
   3.3 Current Workflow Management Systems ...................................................................... 8  
   3.4 Conclusions .................................................................................................................... 9  
4. **Requirements Specification** ............................................................................................ 12  
   4.1 Introduction ................................................................................................................... 12  
   4.2 Component Specifications ............................................................................................. 12  
5. **Technical Analysis** ......................................................................................................... 13  
   5.1 Introduction .................................................................................................................... 13  
   5.2 About Microsoft .NET .................................................................................................. 13  
   5.3 Database Selection ......................................................................................................... 14  
   5.4 Data Access Layer ......................................................................................................... 15  
   5.5 Work Request Submission: Platform independent method calls ................................. 19  
6. **System Design** ............................................................................................................... 20  
   6.1 Introduction .................................................................................................................... 20  
   6.2 Data Access Layer ......................................................................................................... 20  
   6.3 Work Request Submission ............................................................................................. 23  
   6.4 Workflow Engine .......................................................................................................... 23  
   6.5 Workflow Activity Block ............................................................................................... 25  
   6.6 Web-based User Interface .............................................................................................. 26  
   6.7 Fitting it together: The “Big Picture” .......................................................................... 27  
   6.8 Work Item Scheduler ..................................................................................................... 27  
7. **Implementation** ............................................................................................................... 28  
   7.1 Introduction .................................................................................................................... 28  
   7.2 General Deployment ..................................................................................................... 28  
   7.3 Data Access Layer: The ORM ...................................................................................... 28  
   7.4 Work Request Submission (Web Service) .................................................................. 32  
   7.5 Workflow Engine .......................................................................................................... 33  
   7.6 Activity Block ............................................................................................................... 35  
   7.7 Web-based User Interface ............................................................................................ 35  
   7.8 Work Item Scheduler .................................................................................................... 39  
8. **Testing** ........................................................................................................................... 41  
   8.1 Introduction .................................................................................................................... 41  
   8.2 Unit Testing ................................................................................................................... 41  
   8.2 Integration Testing ......................................................................................................... 41
# Table of Contents

8.3 End-to-End Testing .................................................................................................................. 41

## 9. Evaluation ............................................................................................................................. 43

9.1 Introduction ............................................................................................................................. 43
9.2 Aims, Objective and Minimum Requirements ....................................................................... 43
9.3 Project Enhancements .............................................................................................................. 44
9.4 Evaluation of Project Approach .............................................................................................. 45
9.5 Evaluation of Background Research ....................................................................................... 46
9.6 Evaluation of Design ................................................................................................................ 46
9.7 Evaluation of Implementation .................................................................................................. 47
9.8 Possible Future Enhancements ............................................................................................... 48
9.9 Evaluation from .NET Workflow Professionals ..................................................................... 49

References ..................................................................................................................................... 50

Appendix A – User Reflection ....................................................................................................... 53

Appendix B – Project Schedule .................................................................................................... 55

- Initial (Pre-research) Gantt Chart ............................................................................................. 55
- Predicted Progress ...................................................................................................................... 56
- Actual Progress .......................................................................................................................... 57

Appendix C – Design ...................................................................................................................... 58

- Entity-Relationship Diagram ..................................................................................................... 60
- ORM Classes .............................................................................................................................. 61
- WorkFlow Engine ....................................................................................................................... 64
- Work Item Scheduler .................................................................................................................. 66
- Web-based GUI ........................................................................................................................... 67
- Entity Forward Engineering Tool ............................................................................................... 68
- Code Generation ......................................................................................................................... 69
- Mapping Generation ................................................................................................................... 70

Appendix D – Implementation ....................................................................................................... 71

- Entity Forward Engineering Tool ............................................................................................... 71
- Work Request Submission - WSDL ............................................................................................. 72

Appendix E – Evaluation ................................................................................................................ 74

- Developer Feedback ................................................................................................................... 74
1. Introduction

1.1 Problem Definition

The problem was first observed during an industrial placement whilst engaged on a large .NET workflow development project for a major financial services provider.

The project’s design & build phases began completely from scratch; with a large part of the initial build effort concentrated solely on developing various fundamental workflow mechanisms. These mechanisms represented the common basic components required for a workflow system to actually function, for example; how a request for work is inputted, how a workflow is selected & processed, how the system should communicate with persistent storage, and so on. This scenario embodied a key problem; creating fundamental elements of workflow functionality from scratch each time removes initial focus and development effort away from the business use cases, which is inefficient for both parties as core workflow functionality is perfectly foreseeable and thus could be pre-prepared.

1.2 Project Aim & Objectives

The overall aim of the project is to produce an open-source workflow development framework for the .NET platform, which will seek to provide developers with a toolkit to construct a bespoke workflow system for their business scenario.

Gamma et al define a framework as “a set of cooperating classes that make up a reusable design for a specific class of software” [28]. In this context, the framework will aim to capture the design decisions that are common to the workflow domain, thereby pre-defining a series of extendable components to enable a business developer to concentrate on the specifics of their application. As such, the objective is to produce the following components:

- **Data Access Layer (DAL)** – To allow customisable communication with a database in a productive, simplified manner.
- **Work Request Submission/Creation** – To ensure “requests” for work can be entered in a platform independent way.
- **Workflow Engine** – A configurable “controller”, that manages the behaviour of a workflow.
- **Workflow Activity Block** – A base component allowing new workflow activities to be specified in a common manner, and controlled by the Workflow Engine.
- **Web-based user interface** – A skeleton UI framework which ties together each of the above components into a working graphical system.
1.3 Minimum Requirements

The minimum requirements that must be achieved in order to meet the aims of the project are:

- To produce a series of interlinked base-components that can be extended by business developers to produce a bespoke workflow system.
- To demonstrate and test the usage of these components by utilising the framework to create a series of sample workflow transactions.

1.3.1 Possible Enhancements Above & Beyond Minimum Requirements

The following list represents a series of enhancements that could be made to further extend the functionality of the framework, if time allows.

- **Work Item Scheduler** – To allow for the automatic execution of multiple “Work Items” at a specified time.
- **Distributable Components** - To allow each of the components (as mentioned in Section 1.2) to be deployed across multiple machine boundaries with communication occurring via remotable interfaces.
- **Code Generation/Forward Engineering** – This technique allows code to be automatically generated from (mostly) visual models, thereby removing time intensive and laborious tasks from the developer’s workload.

1.4 Project Deliverables

The framework deliverable will consist of:

- A software design
- A database design
- A .NET application (the framework components)
- A database back-end

1.5 Relevance to Degree

This project will draw upon skills gained from a wide range of taught modules within the School of Computing. Topics from Information Systems (IN11) will allow for the formation of a suitable methodology and project plan. Techniques from object-orientated programming (SE21), Internet Systems Technologies (SY23) and Distributed Systems (SY33) will be used for the implementation of the web-based solution. With knowledge acquired from Database Principles & Practice (DB21) being used to aid in the design and implementation of the SQL database back-end.
2. Project Management

2.1 Introduction

Dictionary definitions of the term “project” place a clear emphasis on it being a “planned” activity [1] [2] - that is to say, something you think carefully about before you do it. This chapter will seek to determine how best to carry out the project by discussing project management, methodologies and schedules.

As an effective plan must be based upon some idea of a method of work [3], the following section will discuss the various software development methodologies available.

2.2 System Design Methodologies

Avison & Fitzgerald define a system methodology as “A recommended collection of phases, procedures, rules, techniques, tools, documentation, management and training used to develop a system”. [4]

2.2.1 Why use a methodology?

There are many justifications for using a methodology to guide development, according to Fitzgerald the key reasons are [5]:

- The subdivision of a complex process into manageable tasks.
- Facilitation of project management and control.
- Purposeful framework for applying techniques.
- Economics – the division of labour to carry out specific tasks of their speciality.
- Standardisation – developers are interchangeable.

However, for many organisations the use of development methodologies has not always yielded positive results, with real world performance actually causing many developers to discard them [4]. As such, several development projects have rejected the use of methodologies all together, adopting for a more flexible ad-hoc approach. This is particularly the case in web-based projects which are typically developed in a “trial and error fashion, relying on the skills and experience of the developers” to complete the application successfully [4].

The key point of consideration for this project is the author’s lack of experience in the field of .NET applications – only 1 year – accordingly, a methodology will be used to help guide the development process in the right direction.
There are a diverse range of methodologies available, therefore it is necessary to explore several of the options to find a suitable fit. The following will be examined in greater detail in order to establish their potential application within this project: Waterfall, The Rational Unified Process (RUP) and Structured Systems Analysis and Design Methodology (SSADM).

2.2.2 The Waterfall Model

The Waterfall model “subdivides the development process into formal stages where the output of one stage forms the input for the next” [5]. The idea is to progress from one stage to the next in a purely sequential manner, achieving end user sign-off at the end of each section. This aspect has obvious advantages as it ensures a developer can only move on when the previous stage is to the end users liking. This could potentially lead to more economical project management – a key consideration as it is estimated that requirements errors cost 200 times more to fix if not discovered before the implementation phase [5].

The clear separation of distinct tasks (due to frozen deliverables) would also allow for easier progress measurements, particularly when using milestones as a key progress indicator. This certainly has positive time management implications.

Whilst the orderly, systematic and stepwise approach to problem solving is appealing, a major concern is that of inflexibility. The life cycle assumes that requirements can defined in reasonable detail and articulated accordingly before the design phase begins – it therefore assumes that requirements are fixed thus disallowing modifications as the project unfolds [5]. Frequent and incremental builds are also not possible under this approach [6], this is of particular concern as such techniques are often required to actually put design concepts into practice, thus avoiding mammoth problems come deployment.

Where requirements can be articulated clearly and the developers have in-depth experience of the particular application domain there is a definite argument for adopting the Waterfall approach. However, the model places numerous restraints on flexibility and indeed the basic ability of one phase to learn (and adapt) from the previous one. As the requirements for this project are loosely defined, an increasing understanding of the problem through successive refinements and incremental growth is likely to play a vital role in producing an effective solution – as such the Waterfall model is not suitable.
2.2.3 The Rational Unified Process (RUP)

The Rational Unified Process is an iterative lifecycle approach that is particularly suited to UML. The process’s activities emphasize the “creation and maintenance of models rather than paper documents” [7] – the idea being to provide rich visual representations of system.

The key emphasis on modelling allows developers to “visualise, specify, construct and document the structure of a system’s architecture in a simplified way” [8], which fully describe the solution. This greatly improves a developer’s ability to manage and understand software complexity [8].

In particular, the process stresses the importance of developing software in an iterative fashion [9]. By recognising that it is not possible to define the problem and build the solution in a single step, iterative and continuous improvement pulls risk back in time, ensuring mistakes are remedied during the earlier phases. This also allows projects to evolve, as each iteration will inform and shape the other.

2.2.4 Structured Systems Analysis and Design Methodology (SSADM)

Structured Systems Analysis and Design Methodology (SSADM) is a methodology that focuses on the earlier stages of a software development – analysis and design. As such, systems closely match users’ needs because these are made clear from the outset during the initial stages.

The 3 most important techniques that are used in SSADM are [10]: **Logical Data Modelling** - the process of identifying, modelling and documenting the data requirements of the system being designed. **Data Flow Modelling** - the process of identifying, modelling and documenting how data moves around an information system. **Entity behaviour modelling** - the process of identifying, modelling and documenting the events that affect each entity and the sequence in which these events occur.

SSADM does not, however, cover the construction, testing or implementation of software. It also builds each step upon the work that was prescribed in the previous step – with no deviations allowed. Due to its rigid structure and ignorance towards the latter project stages SSADM is not suitable.

2.2.5 Justification of Choice

Following review of the above methodologies a decision has been made to adopt the **Rational Unified Process (RUP)**, primarily on the following grounds:
Its iterative approach allows for the identification of risks early on in the project, therefore avoiding any last minute struggles [8]. Specifically, this will allow each key stage of design & implementation to go through two iterations (See Section 2.3).

In Section 1.2, the first minimum requirement was separated into five measurable components. This approach neatly fits with RUP’s emphasis on Component-based development (CBD). As each stage produces an output capable of being measured, tested and evaluated the quality of the software can be verified allowing for the requisite adjustments to be made. CBD also fits well when developing frameworks as it enables further reuse and customisation [8].

The use of UML (Unified Modelling Language) modelling will completely describe the system from a consistent and simplified perspective. The author is personally familiar with UML having used it on several university modules and in the real world as a lead package designer producing technical designs for an offshore (Indian) build team. The other important consideration is that UML is a widely-accepted standard; as such the framework design presented in this project should be easily understood by readers (and of course any real-world developers using the product).

2.3 Project Schedule

All three Gantt charts indicating; initial (pre-research), predicted (post research) and actual progress can be found within Appendix B. The predicted & actual plans follow the iterative doctrine of the chosen methodology (RUP), consequently, the design and implementation stages have been broken down into measurable components and will go through two iterations each.

The project has been split to form seven key milestone deliverables:

- **M1** – Marks the completion of Requirements Modelling.
- **M2** – Marks the completion (both iterations) Data Access Layer Design & Implementation.
- **M3** – Marks the completion (both iterations) of Work Request Input Design & Implementation.
- **M4** – Marks the completion (both iterations) of Workflow Engine + Workflow Activity Block Design & Implementation.
- **M5** – Marks the completion of (both iterations) GUI Design & Implementation.  
  *Upon the completion of M5, the first minimum requirement will have been met.*
- **M6** – Marks the completion of testing (i.e. producing a series of sample transactions).  
  *Upon the completion of M6, the second minimum requirement will have been met.*
- **M7** - Marks the completion of the final report.
3. Workflow Management Systems

3.1 Introduction

This chapter will provide an introduction to workflow, then go on to contrast the Java & .NET workflow landscapes and end with a critical discussion of the current workflow products available.

3.2 What is Workflow?

In its raw form a workflow is essentially a business process, which in turn can be defined as “a set of one or more linked activities that collectively realise a business objective” [11]. An example would be the process of granting a customer loan.

The make-up of each workflow is specified within a process definition, which is used to “analyse, model, describe and document a business process” [12]. This, in effect, states how business activities string together to complete a business process. Process definitions are then interpreted and executed by a workflow engine, which is defined as “a software service or engine that provides the run time execution environment for a workflow instance”. This project will refer to workflow instances as Work Item Transactions – that is, a specific unit of work of a particular type (i.e. a Work Item Transaction Type) [12].

The Workflow Management Coalition sums up the process thus; a work item is created, is processed and changed in stages at a number of processing points to meet business goals.

The above statement can be better viewed in context by applying it to the real-life business scenario of granting a customer loan.

3.2.2 A Working Example: Granting a customer loan

An application is made using the loan company website, where all the relevant information is entered electronically. The request is then submitted to the company’s workflow management system (WFMS) where it becomes an unprocessed Work Item Transaction, of type “Loan Request”.

Most workflows comprise a number of logical steps, each of which is known as an activity. An activity can either involve manual interaction with a clerk (a so-called workflow participant), or can be executed automatically using machine resources [13]. The Loan Request workflow is likely to contain a mixture of both activity types, and as such will be assigned by the WFMS to a clerk (workflow participant) with the appropriate skills. The system will process each activity according to a pre-defined order (as per process definition); the first activity might automatically send the customer an e-
mail with a reference number, second, customer details may be displayed to the clerk so that a credit check can be performed, and so on. Some activities may require the participation of a user with higher authority, which might be a supervisor to finalise the loan payment. A WfMS must therefore also ensure that a flow can be paused and channelled to one or more different users where it can be resumed. This process of transference is known as a “Hand Off”.

It should be noted that this workflow type differs from a workflow consisting entirely of automatic activities requiring no user participation, which will either be executed instantly, or in batches according to a given schedule (this has been identified as a potential project enhancement).

### 3.3 Current Workflow Management Systems

Before delving into the world of .NET workflow management, it is useful to briefly discuss the general WfMS landscape from a multi-platform perspective.

Perhaps the best comparator is the highly active Java workflow domain; with at least 19 large commercial vendors [14] and 22 active open source projects [15]. The Java open source community has produced a myriad of varying solutions pitched at a large variety of end-user needs. For example; some allow non-technical users to express workflows in a basic scripting languages [16] or graphically through flow charts [17] which are then interpreted at runtime. Others provide a low-level workflow backbone which is then extended by developers [18] (much like this project), one approach even focuses on executing complex scientific workflows using grid-based approaches to distributed computing [19], and the list goes on.

In stark contrast, the .NET workflow domain has just four, purely commercial products. Only two will be discussed in detail below as they represent the two differing approaches taken by vendors – three of the four all share remarkably similar functional traits, and fall within the category of “pre-packaged solutions”.

#### 3.3.1 Pre-packaged solutions: Skelta Workflow.NET

Skelta Workflow.NET is an out-of-the-box workflow engine, meaning that it is used to provide workflow functionality to existing applications. This integration is provided by a series of ready-made ASP.NET web controls, which the developer is expected to “drop” into their application. It is aimed primarily at medium-large scale businesses and places most of its emphasis on non-technical users being able to graphically design workflows (using their ASP.NET web controls), which are then interpreted at run time.
Skelta Workflow.NET does an excellent job of providing a set of ready made features for (primarily) non-technical users. However, although an API is available for developers they can only bolt-on custom code, not modify what’s already there. Whilst this approach may be appealing for businesses with very predictable workflow requirements (generally simple document processing & authorisation), the product itself is too rigid for it to be suitable for use within wider workflow scenarios. The comprehensive nature of its offering is its downfall.

The package also includes a series of added features to provide support for; load balancing, server clustering, Work Item tracking, reporting and Microsoft Active Directory and SharePoint services etc.

3.3.2 Microsoft Windows Workflow Foundation

Microsoft Windows Workflow Foundation is one of the new offerings under the Microsoft WinFX product umbrella. WinFX represents a collection of new managed (i.e. .NET) APIs for up-and-coming Windows platforms, such as Windows Vista (Longhorn). Windows Workflow Foundation brings to the table a programming model, engine and tools for the development and execution of workflow-based applications on the Windows platform.

Microsoft should be commended for producing a product capable of satisfying both technical and non-technical users; by providing developers with a rich API for customisation and business analysts with the functionality allowing them to graphically design workflows. This approach should allow each user to stick to their relative area of expertise; the developer solving technical problems and the business analyst designing workflows representing the business processes they know so well.

Whether coded, or designed visually, Windows Workflow foundation can embed a run-time workflow engine in any windows application to interpret a given set of defined workflows. As expected, flows consist of one or more linked activities, however activities within Workflow Foundation represent programmatic steps (i.e. a “for each” loop) rather than identifiable business activities (i.e. Retrieve customer details). Although, looking into the future, this approach could lend itself rather well to integration with web services; as workflows could communicate with UDDI (Universal Description, Discovery, and Integration) registry’s to retrieve a wealth of business activities on demand – unfortunately, there is hitherto no mention of this in the Microsoft literature.

3.4 Conclusions

The research has highlighted a clear difference between the Java and .NET workflow landscapes. The .NET domain has a limited offering of purely commercial products, whereas the solutions available in Java are both numerous and functionally diverse. Although this is probably more an indication of
Java’s relative maturity (.NET was only formally released in 2002) rather than any major technological advantage.

The pre-packaged solutions appear to be good out-of-the-box packages, with the completeness of their offerings making them ideal for large projects with predictable, rigid requirements. However, they fail to accommodate the key issue of flexibility, in that only a limited API is exposed to developers in which to integrate custom code to meet their business needs. Instead, great emphasis is placed on the bundled graphical process designers, which allow non-technical business analysts to create workflows visually. Even within the Java domain, most vendors have gone overboard on this. In doing so they have “failed to provide good enough mechanisms to integrate code into a workflow definition in cases where it may be appropriate” [14]. Incidentally, this is the key reason why a ready-made solution wasn’t adopted on the project mentioned in Chapter 1 – they simply aren’t flexible enough.

Microsoft’s Workflow Foundation has attempted to re-inject a degree of flexibility into the area, by placing an equal emphasis on both custom code and graphical process designers. Whilst this is a positive step, Windows Workflow Foundation fails to address the following issues:

The author’s main point of contention relates to how Workflow Foundation defines its activities. As activities represent programmatic steps rather than identifiable, encapsulated business actions, business analysts are still forced to graphically design workflows as if they were coding them. As such, workflow processes are effectively a graphical representation of how the process would be coded rather than a series of identifiable business activity stages – which is, of course, how the business analyst will generally be visualising the process. To illustrate this point Figure 3.4.1 contains a Simulated Windows Workflow Foundation process definition extract for a simplified “Grant Customer Loan” example.

To fully comprehend this issue it is necessary to look through the eye of the organisation, more specifically, to take a look at how businesses model. Following the widespread adoption of the Unified Modelling Language (UML) most functional documents outlining processes avoid technical terminology, preferring instead to use the language of the end user or domain expert [20]. Taking this into consideration it would be more useful to regard activities as identifiable business stages rather than programmatic steps (Figure 3.4.2). This project seeks to do just that.
This second approach is likely to offer a cleaner high-level fit between the process design and the use case, especially if the various flows within a use case are written to represent specific business activities. Indeed, this would also allow for the clear association of UML Actors (workflow participants) to distinct activities within use case diagrams – which are central when modelling the behaviour of a system [7].

Secondly, for those activities requiring participant input, Windows Workflow Foundation fails to specify how a graphical user interface relates to the process. This is not surprising, as a GUI is likely to span across multiple programmatic steps, and therefore activities, within a single process. Again, this problem could be overcome by viewing activities as identifiable business stages – which pairs up business logic with its respective GUI.

Finally, the documentation does not mention how custom business objects (i.e. tables holding customer addresses etc) are created or persisted to a physical medium and how work requests are submitted is not made clear, nor whether this can be done in a platform independent manner.
4. Requirements Specification

4.1 Introduction

This section will draw upon the project aim, minimum requirements and background research to outline a series of concise, measurable project requirements that can serve as evaluation criteria when assessing the success of the project.

4.2 Component Specifications

4.2.1 Data Access Layer (DAL)

The DAL will be required to provide persistence to a given configurable database. An appropriate API must be exposed to allow for the creation, retrieval, update and deletion (CRUD) of information. More specifically, the DAL must be constructed in such a way as to fully accommodate true workflow functionality; which includes the storage of partially complete transactions (including all data entered) and allow new business-related data to be created and accessed with ease from within activities.

4.2.2 Work Request Submission/Creation

This component must allow work requests to be entered into the WfMS in a platform independent manner by accepting each request, converting it to a Work Item Transaction and then persisting that transaction to physical storage ready to be picked up by a workflow participant. A mechanism is also required to allow any information related to the request be entered into the WfMS along with its parent transaction (for example, customer information).

4.2.3 Workflow Engine

This component must be a configurable controller, capable of managing the behaviour of each transaction executed. Specifically it must:

- Interpret the process definition
- Control the execution process of each transaction, including – activation, automatic activity navigation and finally, termination.

4.2.4 Workflow Activity Block

This must act as a base component; as an identifiable business stage along with its’ respective GUI, in a common manner that is directly controllable by the Workflow Engine.
4.2.5 Web-based user Interface

This must comprise a series of components which tie together each of the above components into a working system. Specifically, it must include the following:

- A component capable of logging a workflow participant onto the system.
- A component that, for a given participant, can display all relevant unprocessed transactions and provide a mechanism for communication with the engine to activate the selected transaction.
- A base component allowing the GUI constituent of an activity to communicate with its parent activity and the workflow engine.

5. Technical Analysis

5.1 Introduction

This section provides an overview of the Microsoft .NET framework, then goes on to investigate the various options available for implementing a data access layer (DAL) and distributed approaches for inputting work requests.

5.2 About Microsoft .NET

The .NET Framework is a software development platform created by Microsoft which aims to provide the following [21]:

- A managed run-time environment that offers automatic garbage collection, security versioning and, above all, interoperability between programs written in different languages. This is provided by the framework’s Common Language Runtime (CLR) which provides an execution environment based upon a virtual machine (similar to Java), with its own instruction set (CIL – Common Intermediate Language) into which programs written in any .NET language are translated. CIL programs are then compiled, just in time (JIT), into the code of the target machine – this JIT compilation ensures programs are as efficient as possible, as compilation takes place on demand (see Figure 5.2.1 for illustration).

![Figure 5.2.1: Execution model for the .NET architecture](image-url)
• An **object-orientated class library**, providing functionality for graphical user interfaces (Windows forms), web interfaces (Web forms), data connectivity (ADO.NET), collections, threads and so on. In many cases .NET is replacing the current windows API.

Of the various programming languages supported by .NET, the **C# language** has been chosen for this project, primarily because the author already has some experience with it, and its application to solving workflow problems. The C# language follows syntax very similar to that of Java and as such should be easily readable by readers with little C# exposure.

Finally, it is worth noting that this project will be developed using .NET Framework version 1.1.

### 5.3 Database Selection

Elmasri and Navathe define a database simply as, “a collection of related data” [22]. Workflow Management Systems generally employ some form of persistent storage for two reasons. Firstly, as a mechanism is required for the storage of Work Item Transactions and any related saved information - it would dangerous to store such data in volatile memory in case of system failure, plus this would no doubt represent an inefficient use of system resources. Secondly, workflows will often make modifications to business-related data (such as updating bank account details); accordingly the WfMS will need to have knowledge of the business’s storage medium in order to ensure the results of a workflow can be reflected within the business-related tables. It has been decided that a database will offer the best, most efficient storage medium for this purpose. As such, the following sections will analyse three different database systems in order to establish which one is most suitable.

#### 5.3.1 Microsoft Access

Microsoft Access is a database system usually shipped along with the Microsoft Office Suite, which has the distinct advantage that many businesses will already have it on at least one system and therefore will not need to incur any extra costs. However, as explained by Lavin [23], “[Microsoft] Access cannot effectively manage multiple concurrent accesses to its underlying database as it is not a true client/server RDBMS”. A WfMS is likely to place large demands on its database in terms of inputting, retrieving and restoring transactions, as such Access can be immediately ruled out as a suitable candidate.

#### 5.3.2 My SQL

MySQL is a cross-platform Database Management System (DBMS) which is made available free of charge under the GNU General Public License (GPL), therefore users incur no added costs.
An important point to consider is that a Workflow Management system will often represent only a single element of a business's total IT function. Accordingly, it will be expected in many cases to communicate with the firm's core database, as this is likely to already contain key data applicable to workflows (such as customer details etc). Taking this into consideration, the database chosen must be able to satisfy business needs beyond the scope of just workflow and this is where MySQL falls short. As Chigrik explains in [24], the SQL dialect used in MySQL (named “MySQL Dialect”) does not support features such as: triggers, stored procedures, views, cursors, foreign keys and so on. When considering the wider, non-workflow related applications of a database it is impossible to foresee whether or not a business will actually require said functionality at some point. As such, MySQL will be ruled out in favour of a more comprehensive database management system.

It should be noted, however, that version MySQL 5.0.19 released in March this year (2006) does address many of the issues mentioned, but has unfortunately come too late for use within this project.

### 5.3.3 Microsoft SQL Server

Microsoft SQL Server is the third most popular relational database management system (RDBMS) in the world [25]. In contrast to many of its competitors, SQL Server runs solely on Microsoft Windows based operating systems, although, the effects of this within this project are lessened as .NET was designed primarily for Windows (though, other projects such as Mono are in existence). Unlike MySQL, by employing the powerful Transact-SQL (T-SQL) dialect, MS SQL Server is able to offer comprehensive support for triggers, stored procedures, views, cursors, foreign keys etc.

Taking its relative popularity and wealth of functionality into consideration, Microsoft SQL Server has been chosen as the database for this project.

### 5.4 Data Access Layer

In [26], Wikipedia.com defines a Data Access Layer (DAL) as “a module of computer programming logic that provides simplified access to data stored in persistent storage of some kind”. Placed in context; this module will be used by the various workflow components to access and manipulate the data within the database without requiring a direct coupling or explicit knowledge of it.

#### 5.4.1 Data Access in .NET

In the Microsoft .NET Framework access to a wide variety of data sources is enabled through a group of classes collectively named ADO.NET, which effectively act as the link between an application and a given data source. It is built on top of OLE DB (Object Linking Embedding Database), which provides a number of interfaces and components for accessing structured data, like that in the database. Specifically, ADO.NET provides a number of objects which represent connections,
commands, data adapters and data readers through which you interact with a data source. So, the key questions to address at this stage are to work out exactly how they should be utilised; should workflow components be directly coupled or loosely coupled to the data source? How much knowledge should they have regarding the structure of the data layer? Should developers communicate in SQL, or by using ‘live’ data objects? The section below attempts to tackle these issues.

5.4.2 The Importance of Decoupling

The Model/View/Controller (MVC) triad of classes used in Smalltalk-80 is a classic example demonstrating the importance of decoupling. It shows that isolating functional units from each other as much as possible makes it easier for the application designer to understand and modify each particular unit, “without having to know everything about the other units” [27].

Applying this central precedent to the data access problem would advocate splitting the solution into three logical layers; Application (1) – DAL (2) – Database (3). In this instance, the DAL is acting as a Façade, a software design pattern developed by the so-called “Gang of Four”¹ to provide a “single, simplified interface to a complex sub system” [28]. This concept is illustrated below (Figure 5.4.2.1).

![Diagram of Façade Design Pattern](image)

Figure 5.4.2.1: An illustration of the façade design pattern

5.4.3 Business Logic in SQL

ADO.NET provides the functionality for a .NET application to directly query a given database using Structured Query Language (SQL) – the industry standard. SQL undoubtedly has many strengths, of particular interest here is its ability to query a database using widely understood syntax, allowing the

¹ Erich Gamma, Richard Helm, Ralph Johnson & John Vlissides
filtering and summarisation of large amounts of data with fairly few lines of SQL code. As such, a relatively simple DAL could be implemented which accepted SQL commands from the various workflow components, queried the database and returned the result, at which point the relevant data could be extracted from the DataReader object. However, therein lies a fundamental problem; expecting a developer to explicitly use SQL within their workflow components would involve embedding data domain logic within the application layer, which goes against the basic principles of a layered enterprise application architecture [29]. Accordingly, the speed advantages gained by developing this type of DAL may come at the expense of following good design principles; as explored in the following paragraph.

The above approach can lead to a semantic gap; where programmers are forced to mix object-oriented principles with those present within the relational model to form their final solution [30]. This inherent difference between the two approaches has been labelled the “object-relational impedance mismatch” [31]. The impedance mismatch becomes apparent when you look at the preferred approach to access: With the object paradigm you traverse objects via their relationships whereas with the relational paradigm you join the data rows of tables. This fundamental difference results in a non-ideal combination of object-orientated and relational technologies [31].

Ideally, then, a DAL should attempt to use object-orientated concepts to solve problems inherent in standard data access.

5.4.4 Object-based Approach

One possible solution to the problems outlined in the previous section is to produce a DAL using a technique known as Object-Relational Mapping (ORM). In [32], Wikipedia.com defines Object-Relational Mapping as “a programming technique that links databases to object-oriented language concepts, creating (in effect) a virtual object database”. In other words, it allows developers use Object-Orientated methods to access the DAL, which, in turn, automatically converts the Object-Orientated request into SQL (and vice versa).

With this approach each database table will have a corresponding concrete C# class. These concrete classes contain logic that invisibly connects them to the DAL, and therefore the database. As such, the developer need not worry about formulating SQL queries; instead they request ‘live’ objects, or entities from the DAL. As a simplified example, a retrieval may look something like:

```csharp
DAL.Retrieve(Customer, "Customer Name = Adam")
```

This will return a Customer entity, which the developer can then treat like any other object. For example; assignments are done in the usual manner, using the equals (=) operator – the DAL simply intercepts this assignment and automatically updates the corresponding
database table (by dynamically creating the relevant SQL statement). In addition to alleviating the problem of impedance mismatch, adopting the ORM approach will also provide the following additional advantages/features:

**Lazy Loading** [33]:
Once an initial entity has been retrieved, *lazy loading* (or lazy reading) allows developers to automatically traverse each relationship following normal object-oriented principles. For example, once a customer entity has been retrieved, their post code can be obtained (from the DB) using the following syntax: `Customer.Address.PostCode` - thereby avoiding the need for developers to write additional SQL queries to accommodate further navigation (through extra connections & joins).

**Layer Modification:**
Good ORM practice advocates having an editable mapping file to detail how each entity relates to its corresponding table – which the DAL will use to make its associations. Accordingly, this allows developers to change the underlying data structure without causing a large ripple effect across the system, specifically having to re-write code. Instead, a developer would simply modify a text-based mapping file. This, in effect, provides a degree of database encapsulation.

**Helps Avoid Duplication:**
A key consideration in any (pragmatic) software development project should be following the “*Don’t repeat yourself*” principle [34]. Following an ORM approach will ensure that developers only write the bare minimum amount of code; using lazy loading techniques to retrieve data purely on demand. Unlike other DAL techniques, an ORM avoids the need for programmers to write identical SQL statements on multiple occasions [29].

**5.4.5 Conclusion**
Before coming to any decisions regarding DAL approach it is necessary to take a step back and fully recognise the aims of this project; which is to provide developers with a *flexible development framework*. As such, the DAL should aim to make the development process as painless as possible, by assisting efforts, not hindering them. Needlessly introducing SQL-based data access logic into the application layer is likely to pose such a hindrance, particularly if it serves the obscure the underlying business logic code. Flexibility should always be promoted, by ensuring changes do not result in large volumes of re-work and duplication must be avoided where possible. It is the author’s opinion that this can only be achieved by following sound software engineering principles; the ORM approach offers a superior method of achieving these goals.
5.5 Work Request Submission: Platform independent method calls

5.5.1 Introduction
A core requirement of the framework is to provide a platform independent way of entering work requests onto the system. This ensures clients aren’t tied down to specific architectural constraints and ultimately makes the WfMS more usable. The following section will briefly review two distributed techniques.

5.5.2 CORBA
CORBA, or Common Object Request Broker Architecture, is a standard architecture for distributed object systems. It allows a distributed, heterogeneous collection of objects to interoperate over a network (via the IIOP protocol), which includes the Internet. In other words, it will allow a client to invoke the relevant framework method remotely, independently of programming language or operating system. As client-server interaction is mediated by CORBA’s Object Request Brokers (ORBs), the only system requirement is that the client must be running an implementation of CORBA (with a compatible ORB).

5.5.3 Web Services
In [21], Beer defines web services as “a network of software services that can be created and used independently of operating system, programming language and binary transmission protocol”.

The fundamental difference between CORBA is that web services do not define a new binary protocol (such as IIOP) for communication, instead opting for an XML-based protocol, SOAP (Simple Object Access Protocol). This has the advantage that existing network infrastructure can be used, as SOAP calls are essentially nothing more than text files travelling between a client and a server. Indeed, the only requirement of the platform/language is that it must be able to handle XML. As such, web services arguably offer greater platform independence than CORBA.

5.5.4 Conclusion
The Work Request Submission component is providing a service to its clients, which means communication is one-way; a client will issue the framework a work request with no further communication necessary. As such, it has been decided to use web services which are asynchronous and thereby provide a loose coupling between client and server – this approach would therefore provide a neater fit with the service-orientated requirement. CORBA’s tightly-coupled approach is far better suited to the synchronous transfer of objects and would represent an ‘overkill’ if used in this scenario. Moreover, the requirement of client machines to implement CORBA places additional, unnecessary requirements upon them which should ideally be avoided.
6. System Design

6.1 Introduction

The purpose of this Chapter is to investigate and outline the design of each component.

6.2 Data Access Layer

As discussed in Chapter 5, the data access layer will be designed by following Object-Relational Mappings techniques. Three aspects of the DAL will be discussed here:

6.2.1 Relational Database Design

Whilst the majority development effort for this framework is concentrated at the application level, there still remains the central need to store workflow-related information in a database. For example, workflow participant login credentials & skills, Work Item Transactions & their corresponding Notes etc. It is expected that this information will be stored along side business-specific data, as the workflow framework will be used to complement existing business operations. Naturally, the following design will only concern workflow related tables, it will be up to the business developer to ensure their business-specific tables are ORM compatible, normalised etc. Whilst Appendix C contains a full description of each workflow-related table and its corresponding columns, three columns are worth noting now:

- **WorkItemTransactionType** represents a single business process (i.e. “Grant Loan”).
- **WorkItemTransaction** represents a single instance of a business process; as such one type can have many transactions.
- **WorkStreamUser** represents the workflow participant – each of which has a user level that stipulates which transaction types they are able to perform.

![Database schema](image)

*Figure 6.2.1.1: Database schema*
Appendix C models the database requirements using an Entity-Relationship diagram. This E-R diagram models the relations between the various entities which is then used to formulate a relational database schema, as illustrated in Figure 6.2.1.1.

In [22], Elmasri and Navathe explain that database schemas must adhere to two core integrity constraints. *Entity integrity* states that no primary key can be NULL; this will be achieved by assigning NOT NULL to each primary key column. *Referential integrity* advocates that all foreign keys must map to a corresponding key in a related table; this will be handled automatically by the DAL.

**Normalisation**

The normalisation process, as first proposed by Codd, takes a relational schema through a series of tests to attest whether it satisfies a certain normal form [22] – this ensures that tables do not contain any redundant data. According to Elmasri and Navathe [22], schemas should be designed such that they have no partial or transitive dependencies; in other words, must adhere to third normal form (3NF). That is to say, the removal of all conditions where if A, B, C are attributes of a relation R and A → B, B → C then A → C, and if A and B are attributes of a table where there is some attribute that can be removed from A and yet the dependency still holds, respectively. Accordingly, the schema presented in Figure 6.2.1.1 has been normalised to third normal form.

**6.2.2 ORM Design**

Architecturally, the ORM will be sandwiched in between the entities and the relational database. It will have two main points of contact with the application (business logic) layer.

Firstly, with the application itself; accepting explicit commands relating to the creation, retrieval or deletion of entities from persistent storage. In the case of creating and retrieving entities an adapted version of the design pattern Abstract Factory [28] will be employed. The adaptation relates to the fact that it will not be possible to subclass the main factory object, in this respect the term *abstract* only really applies to the base entity type (*EntityBase*) of which all concrete entities will derive. An interface named *EntityFactory* will be created to include operations that creates new instances of a particular entity type (i.e. a customer) and retrieve given entity instances based upon criteria specified by the developer (i.e. a customer with a particular ID and/or name), as illustrated (Figure 6.2.2.1).

![Diagram](image.png)

*Figure 6.2.2.1:* An illustration, in context detailing Abstract Factory usage
The Abstract Factory approach has the benefit of isolating developers from the complex implementation of the ORM by encapsulating the responsibility of database communication and object – SQL conversion. It also promotes consistency within the application as data access is done in a common manner throughout; this also allows modifications to be made inside the ORM without causing large amounts of re-work within the application (as developers communicate with an interface).

Many Abstract Factories are often implemented as a singleton; however in this scenario a new instance of EntityFactory will be created for each database transaction, that is to say a logical group of related operations performed on one or more given entities. This way, each instance can keep a record of the changes made to groups of entities, ensuring the final DB commit conforms to standard ACID (Atomicity, Consistency, Isolation, and Durability) principles – committing the related entity changes together. The only exception to this is where a developer issues a delete request to the ORM, where it will be instantaneously carried out on the relational database.

The second and main point of contact will be with the entities themselves. The steps outlined above are effectively the method by which an entity is initialised. Once a ‘live’ entity is returned to the application all data access to it or its relations is done via the entity itself. Entities will be returned from the database with each local (value-type) property populated with its respective value, however, references to related entities will remain null until a traversal is explicitly required – where the referenced entity will be automatically lazy loaded (Figure 6.2.2.2). Modifications to the state of a given entity will result in a local update (as per normal), then go on to fire an event which informs the ORM of the change – these internal actions will go unnoticed by the developer as discussed in the following section.

A complete class structure of the ORM can be found in Appendix C.

---

2 A type used to indicate an individual object. A singleton has only one instance.
6.2.3 Entity Design

As entities are effectively object-orientated representations of their table-based counterparts they will contain a number of C# properties relating to their corresponding relational columns in the database. The mapping between both names and types will be maintained by the parent ORM, as discussed in Section 7.2.1. Each entity will derive from an entity base class, which will provide the common mechanisms to inform the ORM of any state changes.

6.3 Work Request Submission

The Work Request Submission component will be designed as an ASP.NET web service, using Microsoft Internet Information Services (IIS) as a web container and placed within a virtual directory where it can be invoked by outside callers. It will require two parameters; the transaction type name (i.e. “Grant Loan”) and an XML payload, which by default allows callers to specify one or more notes to place against the transaction. The second parameter has been deliberately designed to take an XML payload (rather than an array of strings, for example) to allow for easy customisation by developers – this way, during future development any information (not just transactional notes) can be passed to the service providing it is told what to look for.

The web service will function by validating that the business is able to perform the transaction specified, create the relevant WorkItemTransaction and Note entities, and then use the ORM to persist the new WorkItemTransaction and its Notes to the DB – where it can be picked up by a workflow participant. A simple acknowledgement string will be returned to the caller, either containing an error message or, if successful, the primary key of the WorkItemTransaction instance and the time it was created. This process is illustrated graphically below (Figure 6.3.1).

6.4 Workflow Engine

The workflow engine design will centre on a Base Transaction Controller class. Conceptually, a Transaction Controller is the coordinator of a business process; responsible for reading the process definition, loading each activity and then performing a final database commit (effectively ‘saving’ the outcome of the transaction).
Accordingly, the base transaction controller will form the basis of how custom business transactions are implemented by developers. This means that this base class must be easily extendable without requiring any internal modification, or as Bertrand Meyer would put it; “open for extension but closed for modification” [35]. This statement is summed up in Meyer’s Open-Closed Principle (OCP), which advocates designing modules that never change – instead, changing requirements (i.e. new transaction types) are accommodated by extending the behaviour of existing classes. According to Martin [35], abstraction is the key to achieving this; as such, the base transaction controller will be designed as an abstract (polymorphic) class which developers extend to derive their own business transactions.

As well as providing a contract that must be honoured by each derived class, the base transaction controller will automatically handle the loading of activities (to ensure derived classes are only contain business logic) and provide transactional state management – including the means by which to save the state of a given transaction to a database, and conversely, reload that state from the database (this will accommodate “Hand Off’s”).

Finally, the workflow engine will follow the factory pattern mentioned in 6.2.2, thereby exposing a Transaction Controller Factory to facilitate the instantiation of derived transaction controllers without the need for the caller to depend upon a concrete class. The factory will not contain any inherent knowledge about the derived controllers it creates; instead it will obtain this information by parsing the XML process definition. Using this, the factory can create the relevant controller by transaction type and pass it one or more activity placeholders - which inform the newly-created controller of the activities it must process. Adopting this approach ensures the design follows the rules laid down in the Liskov Substitution Principle (LSP), which advocates that any function using “a reference to a base class as a parameter need not know about each derivative of that base class” [36]. Incidentally, any violation of LSP also violates the Open-Closed principle because it results in a class modification whenever a new derivative of the base class is created. A high level architectural design can be found bellow (Figure 6.4.1) and the full class structure design in appendix C.

![Figure 6.4.1: High level architectural representation of Workflow Engine](image-url)
6.5 Workflow Activity Block

As mentioned in Chapter 3, a workflow activity will be considered as an isolated, identifiable stage in business process. Given this, an activity will consist of *three* logical components; a GUI, GUI logic and business logic – hence it will be referred to as an activity block (Figure 6.5.1).

<table>
<thead>
<tr>
<th>GUI Logic</th>
<th>Presentation – i.e. a HTML (ASP) page</th>
</tr>
</thead>
<tbody>
<tr>
<td>GUI</td>
<td>UI Specific Logic – i.e. populating combo boxes.</td>
</tr>
<tr>
<td>Business Logic</td>
<td>Logic for the processing of that activity, i.e. checking for a sufficient balance</td>
</tr>
</tbody>
</table>

**Figure 6.5.1:** Activity Block

### 6.5.1 Graphical User Interface (GUI)

This represents the presentational aspect of the activity allowing the workflow participant to interact with it. This will be designed as a standard ASP page, using HTML tags for the presentational aspects.

### 6.5.2 GUI Logic:

This .NET class will contain any logic required to assist with the presentation of the page, for example; what to do each time a button is clicked or how to populate a drop-down list. Whilst it is feasible to place this logic within the presentational component (above) the author feels that this embedded .NET code will unnecessarily obstruct the view of the surrounding HTML. As such, this logic will be separated from the presentational component in a so called “code-behind” file.

### 6.5.3 Business Logic:

This represents the actual business logic relevant to the given activity, for example; a method to perform business rule checks or to retrieve a customer record using their name and address. Clearly not all business activities require user participation, in these instances an activity block will simply consists solely of this business logic component.

Again, true to the Open-Closed Principle an abstract Base Activity Controller will be created. This will define the common contract that derived activities must follow; above all this ensures that the Base Transaction Controller can always automatically deal with the processing of *any* derived activity, without the developer having to worry about how it is loaded and progresses on to the next one etc. In addition, the Base Activity Controller will expose methods which allow derivatives to use the state mechanism on the Base Transaction Controller – this provides a convenient medium by which related activities can communicate with each other (as ASP pages are stateless, explained in 7.7.3).
6.6 Web-based User Interface

6.6.1 The Model View Controller

The Model View Controller (MVC) triad of classes is a software architecture that separates an application's data model, user interface, and control logic into three distinct components, so that modifications to one component can be made with minimal impact to the others [37]. This “pattern” (MVC is actually an aggregation of multiple design patterns) has already been employed in the design of the Activity Block; with the GUI representing the View, GUI Logic representing the Controller and business logic representing the Model. Effectively, the MVC paradigm introduces the controller object in between the view (the GUI) and the model (activities, transactions etc) to communicate between the other two objects. The following section (6.6.2) concerns the design of this Controller.

6.6.2 Activity GUI Logic Design: The “Page Base” Class

As previously mentioned, ASP.NET promotes the use of code-behind files to encapsulate activity GUI Logic, this in effect, is an attempt to follow the principles of separation laid down by the MVC. However, an addition will be required if the controller is to allow the dynamic loading of views. Specifically, the controller must dynamically load the view relating to each activity when its requested – this avoids any unnecessary hard coding.

Whenever ASP pages are requested, ASP.NET will compile the .aspx file (which in this case is the view) and any related files (code-behind etc) into a series of classes which output the HTML (and some Java Script) to the browser. These classes inherit from the .NET framework class System.UI.Page (Figure 6.6.2.1); this architecture allows developers to further sub-class the Page object to implement the required controller. Specifically, a PageBase class will be created to contain the functionality needed to facilitate this automatic navigation. It will also include common methods that validate workflow participants, allow the developer raise and display errors etc. The final picture will look something like this; the view class will inherit from the code-behind file which will in turn then inherit from the PageBase class, as depicted in Figure 6.6.2.2.

![Figure 6.6.2.1: Inheritance hierarchy of Page classes.](image1)

![Figure 6.6.2.2: Modified hierarchy, including additional Page Base class](image2)
The full design can be found in Appendix C.

6.7 Fitting it together: The “Big Picture”

The following diagram presents a “big picture” view of the system, illustrating by example how each component will fit together to form the final solution.

![Diagram of system components with process flow]

**Figure 6.7.1:** Complete end-to-end view of the solution

6.8 Work Item Scheduler

The Framework design hitherto has only been concerned with the execution of those transactions requiring some form of user participation. As such, this component is being introduced as a project extension and seeks to provide a mechanism for the execution of those transactions comprised totally of non-GUI activities.

The design allows a developer/business to specify a list of transaction types that they want to schedule, along with their respective start (initial run) and interval times (duration thereafter). Transaction types will be grouped by this scheduled time, allowing different types to be executed at different times. These will be specified in a human readable XML configuration file.

The design will follow the principles laid down by the observer pattern [28], whereby the parsed XML will result in the creation of various TaskList instances (‘observers’) for each time span (which can contain many transaction types) and register it with a scheduler (the ‘subject’). When each specified time is reached the relevant TaskList instance receives a call back from the scheduler and then proceeds to retrieve each incomplete transaction of the requisite type(s) from the database. A new
TransactionTask instance will be created for each transaction, which is responsible for executing that transaction, as shown below (Figure 6.8.1). The execution will occur by the normal means of using the relevant transaction controller – as such, no modification is required to the current architecture.

![Diagram](image)

**Figure 6.8.1:** High level design of scheduler

### 7. Implementation

#### 7.1 Introduction

The purpose of this chapter is to describe how each project component was implemented.

#### 7.2 General Deployment

Each of the components below has been deployed as a separate dynamic link library (dll), which contains the component code compiled into the Common Intermediate Language (CIL); allowing them to be plugged into any .NET application. Many of the assembly namespaces contain the name “WorkStream” – this is simply the name given to the overall product to allow sensible and consistent naming throughout the project.

#### 7.3 Data Access Layer: The ORM

To use the functionality exposed by the ORM a developer must include the namespaces WorkStream.ORM and WorkStream.Entities in their assembly. In addition, they must specify the database connection settings in the respective App.Config file thus:

```xml
<appSettings>
  <add key="DataSource" value="127.0.0.1"/>
  <add key="Catalog" value="fyp"/>
  <add key="DBUser" value="SqlUser"/>
  <add key="DBPassword" value="SqlUser"/>
</appSettings>
```

**Figure 7.3.1:** ORM Configuration
7.3.1 Entities:

Each entity is derived from the base class `EntityBase`; a key feature of which being the `UpdateCache()` method – which automatically notifies the ORM of any changes made to the state of a given derivative. As mentioned in Chapter 6, entities are instantiated by the `EntityFactory` which populates each local (value typed) property with its respective value from the database. Accordingly, each entity must include a constructor containing each of its’ value-typed properties, as illustrated in the `Party` class below (Figure 7.3.1.1):

```csharp
public Party(String FirstName, String Surname, Int32 PartyID, EntityFactory DB)
{
    this.firstName = FirstName;
    this.surname  = Surname;
    this.partyID = PartyID;  //Read only PK (no set accessor)
    base.myDB = DB;  //Set reference to relevant EntityFactory
}
```

**Figure 7.3.1.1: Entity Constructor**

Here, the final line informs the entity of its corresponding `EntityFactory` instance so that the ORM can group entities by transaction, thus maintaining ACID.

Value-typed properties are then simply retrieved and updated using standard object-orientated dot notation; such as `this.property` to retrieve and `this.property = value` to update. The code sample below shows the make-up of a standard entity property, including the `UpdateCache()` call to inform the ORM of the change (the base class uses reflection to obtain the new value):

```csharp
public String Surname
{
    get
    {
        return surname;  //Return the surname
    }
    set
    {
        this.surname = value;  //Set the value locally
        base.UpdateCache("Surname", this);  //Inform the ORM of change.
    }
}
```

**Figure 7.3.1.2: Value-type property**

Reference types (i.e. relationships to other entities) follow a similar pattern (Figure 7.3.1.3), with the key difference being that they are returned from the `EntityFactory` as NULL – references are only retrieved (lazy loaded) when explicitly requested: i.e. `Party.Address.MyProperty`
```csharp
get
{
    if (address == null) // If null, lazy load instance
        address = myDB.RetrieveReference(this, typeof(Address)) as Address;
    return address;
}
set
{
    this.address = value;
    base.UpdateCache("Address", this);
}
```

Figure 7.3.1.3: Reference type property

The process of manually coding each entity is reparative and moderately time-consuming for any developer, and a task which should be totally unnecessary given the structure of each entity is highly defined and ultimately predictable. As such, a **project extension** was introduced in the form of a *graphical forward engineering tool* facilitating the automatic creation of entities (Screenshot in Appendix D). The tool produced allows a developer to graphically specify each entity/property and then automatically generate the full source code. In addition to saving a considerable amount of (dull) time, this has the added advantage of always producing consistently correct code, and allows developers to load and make modifications to existing entity assemblies.

In order for entity properties to be correctly mapped onto their corresponding database columns developers must create a mapping file. As indicated in the extract included below (Figure 7.3.1.4), naming modifications can be made in either layer providing this change is acknowledged in the mapping.

```xml
<Type name="Party">
    <Member name="FirstName"> // Property name in entity
        <DBField columnName="FirstNames" /> // Column name in DB
    </Member>
    <Member name="Account">
        <DBField columnName="Accounts" />
    </Member>
</Type>
```

Figure 7.3.1.4: Extract from ORM mapping file

**7.3.2 Entity Factory: How the ORM is used**

Once the requisite references have been added, a developer can easily access the database using relatively few lines of code, as demonstrated below (Figure 7.3.2.1).

```csharp
using (EntityFactory myDB = new EntityFactory(DBAccessMode.ReadWrite))
{
    // Retrieve multiple transactions that satisfy criteria
    Set transactions = myDB.RetrieveCollection(typeof(WorkItemTransaction),
        "WorkItemTransaction.WorkItemTransactionType.UserLevel.LevelRank >= 2"
```
+ “&&” // Of course, OR (||) operators are also supported  
"WorkItemTransaction.WorkItemTransactionStatus.Status = Input");

// Update each creation date to the current date
foreach (WorkItemTransaction WIT in transactions)
{
    WIT.CreationDate = DateTime.Now;
}

// Persist updates to database
myDB.Commit();
} Dispose method on EntityFactory called

Figure 7.3.2.1: ORM Usage

The using statement defines a scope for the lifetime of an object, in this case the ORM connection. By implementing the .NET IDisposable interface the factory’s Dispose() method is automatically called whenever the end of a using statement block is reached. This provides a useful mechanism to explicitly free-up unmanaged resources, like a database connection [38].

Whenever a new EntityFactory is instantiated it is necessary to supply the enumeration DBAccessMode (as seen on first line). This has two possible values; Read & ReadWrite – the latter ensures that all changes are committed at the same time (to maintain ACID); by associating the .NET IDbTransaction object to the underlying ADO.NET DB connection.

When looking at the above code snippet the advantages gained through the adoption of an ORM are plain to see. The code itself is fully object-oriented (even the query retrieval criteria – in dark red above), simple and much shorter than its SQL counterpart. As a case in point; the above example is retrieving a collection (or Set) of transactions that a given user (with the level rank 2) is able to process. It then iterates through each instance updating the CreationDate property to today’s date and finally commits the changes to the database. Figure 7.3.2.2, below, illustrates just the retrieval aspect of this in SQL – which the ORM performs invisibly for the developer.

```
SELECT WorkItemTransaction.WorkItemTransactionID,  
WorkItemTransactionType.WorkItemTransactionTypeID,  
UserLevel.UserLevelID, WorkItemTransactionType.TypeName  
FROM WorkItemTransaction INNER JOIN WorkItemTransactionType  
ON WorkItemTransaction.WorkItemTransactionType =  
WorkItemTransactionType.WorkItemTransactionTypeID  
INNER JOIN UserLevel ON WorkItemTransactionType.UserLevel =  
UserLevel.UserLevelID WHERE UserLevel.LevelRank <= (SELECT LevelRank FROM UserLevel, WorkStreamUser WHERE WorkStreamUser.UserName = 'Fred' AND  
UserLevel.UserLevelID = WorkStreamUser.UserLevel)
```

Figure 7.3.2.2: Retrieval shown in Figure 7.3.2.1 expressed in SQL
7.3.3 Type Mapping

Internally, the ORM uses the standard ADO.NET `DataReader` to retrieve data from the database and the `DataSet` object to perform updates. Both classes operate using SQL Server data types (i.e. BIG INT), or more specifically using .NET data providers which symbolise SQL types. Accordingly, as the ORM uses MS SQL Server each type is represented using its corresponding value from the `SqlDBType` enumeration, i.e. `SqlDBType.VarChar`. The problem is that .NET classes (the entities) are written in C#, and as such require their properties to be .NET Framework types (such as `int`, `string` etc). Consequently, the ORM must automatically perform a mapping between SQL types and .NET Framework types whenever it negotiates between C# objects and DB tables. This has been carried out by implementing a small utility class which, when presented with a DB type (i.e. VarChar), will return the corresponding Framework type (i.e. string) using the mapping list provided by Microsoft in [39].

7.4 Work Request Submission (Web Service)

This was implemented as an .asmx file with a corresponding code-behind class inheriting from `System.Web.Services.WebService`. The XML payload is parsed using a SAX parser (the `XmlTextReader` class), by providing it with a `MemoryStream` instance containing the bytes of the XML payload string. An example XML payload is shown below, along with key elements of the code responsible for parsing it.

```
<Payload>
  <Note title="A title">
    <Text>Some text</Text>
  </Note>
</Payload>
```

```
if (xtr.IsStartElement("Note"))
{
    // Create new Note in DB
    Note n = myDB.New(typeof(Note)) as Note;
    // Assign title
    n.Title = xtr.GetAttribute("title");
    // Assign text
    n.Text = ReadNoteText();
    ...  
    // Extract of ReadNoteText()
    if (xtr.IsStartElement("Text"))
    {
        return xtr.ReadString();
    }
    ...
    // Finally, all data is committed.
    myDB.Commit();
}
```

**Figure 7.4.1:** Illustration of XML parsing

The web service can be called in a multitude of ways. A common approach adopted by both Java and .NET is to generate proxies [21]; which allow developers to treat the web service as a local object. This is done by generating a proxy object from the WSDL (Web Service Definition Language) which describes the structure of the service – this WSDL can be found in Appendix D. The proxy object can be created automatically using a series of automated tools such as WSDL2JAVA [40] in Java or
wsld.exe [41] in .NET. The code snippet below (Figure 7.4.2) demonstrates the ease of making web service calls using proxies in .NET:

```
// Create proxy object
WorkItemInputService service = new WorkItemInputService();
// Call web service - submitting work request
Service.InputTransaction("Grant Loan", xmlPayload);
```

Figure 7.4.2: Using a proxy to call a web service

### 7.5 Workflow Engine

To use the functionality expose by the Workflow Engine, developers must include the namespace `WorkStream.Engine` in their assemblies.

Each derived transaction controller must provide a constructor specifying two properties. The first is an `IList` (.NET collection interface) containing a series of activity placeholders, which inform the Base Transaction Controller of the activities it must process (this list is constructed by the Controller Factory, the workings of which are mentioned later). Secondly, the current Work Item Transaction instance is provided so that the Base Controller knows what transaction it is responsible for.

The Base Transaction Controller exposes two key methods; `NextActivity()` and `Commit()` – the latter must be overridden. The former provides automatic activity management for its derivatives using the activity placeholders passed to it during initialisation. Following instantiation, the `NextActivity()` method is immediately called which loads and executes the first activity – if the activity requires a GUI, the controller’s base property `GetActivityPage` is populated with the relevant ASP page address, so that the UI framework knows what to load (`PageBase` will automatically check this). Transactions are progressed by calling this single method, which ensures physical activities/page locations are never hard coded. Once the final activity is complete the subsequent call to `NextActivity()` will return `NULL`. This signifies the end of the process and so the Base Transaction Controller calls the overridden `Commit()` method in the derived class, which contains the logic required to persist the outcome of the transaction to the database.

#### 7.5.1 Transaction Controller Factory

Each Transaction Controller is instantiated by providing the Controller Factory with the Work Item Transaction entity that is to be processed. By examining the Work Item Transaction’s `TransactionType` property the factory can find the relevant entry in the process definition (by looking up the “Transaction Name” field) and thereby load the correct Transaction Controller and
child activities (which are Activity Placeholder’s at this stage). The XML below (Figure 7.5.1.1) contains an extract from the process definition for the Update Balance transaction.

```xml
<Transaction name="Update Balance" class="UpdateBalanceTC">
  <Activity name="Retrieve Party Details" class="RetrievePartyDetailsAC"
    page="RetrievePartyDetails.aspx"/>
  <Activity name="Enter Balance" class="EnterBalanceAC"
    page="EnterBalance.aspx"/>
</Transaction>
```

Figure 7.5.1.1: Process Definition Extract

Parsing this definition allows the activity placeholders to be created. These are simply an array of `ActivityHolder` objects that contain the string properties `ActivityName`, `PageName` and `ClassName`. `ActivityHolder` also exposes a `Create()` method which returns a concrete instance of the corresponding activity to the calling Transaction Controller.

### 7.5.2 State Management

The Base Transaction Controller provides a series of publicly accessible methods which provide state functionality to related classes; such as activities, derived transaction controllers, ASP code-behind classes and the like – this, above all, allows for safe inter-component communication. Transactional state has been implemented as a class capable of adding, retrieving and removing objects from an underlying hash table. To allow transactions to be “handed off” to a supervisor and processed at a later date, the contents of state can be serialised as a string and saved to the `WorkItemTransaction.TransactionalData` property, which has been implemented as a field in the database of type `text` (this can hold up to 2GB of data). This syntax shows the `Serialize()` method on the `state` class.

```csharp
// Formatter for serializing hash table to binary
BinaryFormatter b = new BinaryFormatter();
// Memory Stream to hold the binary once serialized.
MemoryStream ms = new MemoryStream();
// Serialize the Hashtable to the memory stream
b.Serialize(ms, state);
// Convert stream into a byte array
byte[] data = ms.ToArray();
// Return the Base64 string of that byte array
return Convert.ToBase64String(data);
```

Figure 7.5.2.1: Serialisation of Hashtable to Base64 encoded string

Conversely, `Deserialize()` converts the persisted string back into a hashtable, allowing the stored objects to be used again as normal.
7.6 Activity Block

The business-logic element of the Activity block was implemented quite simply as a Base Activity Controller, of which custom business activities are derived. As previously mentioned, activities will be instantiated by their parent Transaction Controller which will provide the activity’s constructor with three parameters: the calling transaction controller (i.e. \texttt{this}), the activity name and the page location as obtained from the process definition.

For those activities not requiring UI interaction; the developer will simply need to place the relevant code within the constructor ensuring each custom activity method gets called. For activities requiring a GUI, the behaviour is a little different, with the ASP code-behind (user events) responsible for calling each method – as will be explained in the following section.

7.7 Web-based User Interface

The user interface component has been implemented using a combination of ASP pages, C# Code-behind, Cascading Style Sheets (CSS) and HTML. Together, these techniques provide a GUI element for workflow activities and general navigation - allowing the user to log on etc.

7.7.1 HTML Layout

The application view is created in HTML from the execution of ASP pages - which contain a mixture of HTML tags and a small element of C# code acting as placeholders for dynamic text – which are basically method calls to the code-behind file. For example, to display the name of the current page (normally the activity name), the following is inserted within the HTML: \texttt{<%=base.PageName %>}. Each ASP page is informed of its code-behind class by the following line:

\begin{verbatim}
<%@ Page language="c#" Codebehind="CreateUser.aspx.cs"
AutoEventWireup="false"
Inherits="WorkStream.Deployment.Transactions.CreateUser" %>
\end{verbatim}

7.7.2 User Management

To facilitate the management of workflow participants (i.e. users) the UserValidator class has been created. This class has two main roles; the creation of new users and the validation of existing user credentials. User details are stored in the WorkStreamUser table (or entity); which contains their username, password and user level (i.e. clerk, administrator etc). As the storage of passwords in plain text is incredibly bad practice [30], the UserValidator class uses a SHA-512 hashing algorithm to encrypt the password, storing it as a Base-64 encoded string. This computation is shown below (Figure 7.7.2.1):
using (HashAlgorithm hashAlg = HashAlgorithm.Create("SHA512"))
{
    //Convert password string into an array of bytes
    byte[] pwdBytes = Encoding.Default.GetBytes(password);
    byte[] hash = hashAlg.ComputeHash(pwdBytes); //Compute Hash
    return System.Convert.ToBase64String(hash); //Return as Base64 string
}

Figure 7.7.2.1: Hash computation

Whenever a user attempts to log on, the ORM is used to retrieve the WorkStreamUser instance by username (which will be unique). The hash of the supplied password is then calculated and compared to the one stored in the database – if they match the user is logged on by placing their credentials within ASP.NET session state, else an error message is displayed.

7.7.3 Page Base Implementation

The Page Base class forms the corner stone of the GUI implementation, with each view’s code-behind file inheriting from it (forming the class hierarchy illustrated in 6.6.2.2).

Security:

The PageBase class has been implemented to include a method that confirms whether or not a user is logged each time a derived page is loaded. This ensures unauthorised users can’t just enter an address string and gain access to an activity view, for example. On such occasions, the base page simply redirects the current user to the designated Logon screen.

Error Management:

To allow developers to raise errors in a consistent manner, the standard ASP.NET label class has been sub-classed to create the aptly named ErrorDisplay control. Using the RaiseError() method in PageBase, developers can raise custom errors and display them in a user-friendly way. This works by placing the error message within ASP.NET session state, under the key “Current Error”. When an ErrorDisplay control is placed on a page it will check (on server Postback’s) for the existence of this state key – when this is present the label’s text is set to the error message(s) and rendered into HTML. When no error has been raised this control simply is not rendered, and therefore does not show up on the page (or for that matter, use system resources). The screen-shot below shows the implemented control in action.

![You have supplied an invalid username or password. Please try again.](Image)

To start your session please enter your login credentials below.

Figure 7.7.3.1: ErrorDisplay ASP Control
Activity Management:

The PageBase class plays a central role in joining the GUI component to the Transaction and Activity Controllers. Key to this is the method StartTransaction(), which can be called by any derivative (i.e. the Transaction Selector component – Section 7.7.4) to start a transactional process by providing it with a WorkItemTransaction instance. Using this, PageBase can call the Transaction Controller Factory; obtain a reference to the relevant Transaction Controller, and so each activity. As ASP pages are stateless, this reference is kept in session state and re-assigned to the PageBase.CurrentTransaction property each time a page is (re)loaded - i.e. on each ASP “Post Back” event. This ensures page reloads always take the user to the correct activity page (and mean they can’t stray off). Once the user has finished entering the required information (normally indicated by clicking a submit button), activity UI pages deriving from PageBase can simply call base.CurrentTransaction.NextActivity() to move on.

ASP web forms work according to an event-driven model, with each user interaction firing a specific event. The act of clicking a button to submit an activity is classed as a server event and as such causes a Postback. A handler on the server then executes the method and returns the page. This execution-reload process is known as the “round trip” of a web page. The table below details the key stages within the ASP.NET page life cycle beginning at the point when a page is loaded.

<table>
<thead>
<tr>
<th>Event/Method</th>
<th>Event Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Page initialization</td>
<td>All the controls that are statically declared in the .aspx file will be initialized with the default values.</td>
</tr>
<tr>
<td>Load</td>
<td>During load, if the current request is a postback, control properties are loaded with information recovered from view state and control state.</td>
</tr>
<tr>
<td>Postback event</td>
<td>If the request is a postback, any event handlers are called. It is at this stage that the request to move onto the next activity is dealt with.</td>
</tr>
<tr>
<td>Rendering</td>
<td>This is where the HTML is rendered, creating the final page view.</td>
</tr>
</tbody>
</table>

Taking this ordering into account, the PageBase implementation has overridden the PreRender() method, which as the name suggests occurs just before the HTML rendering stage. This overridden method contains added code which performs a check to determine whether NextActivity() has been called – if so, the current page (which is being reloaded) will not match the current activity (which changed during the Postback stage). This check allows the PageBase class to invisibly re-direct the user to the next activity page. In addition to automatic page re-direction, the PreRender() method also calls the virtual EnterTask() method. This serves to allow developers one last chance to make any final changes to the page controls before they are rendered to HTML, this functionality is deliberately called here to ensure any Postback event changes can be reflected.
**State Management:**
Finally, state management has been implemented by providing the developer with two key methods: `SavePageToState()` and `RestoreState()`. On transactions where a handoff is applicable, the developer will need to explicitly call the `SavePageToState()` method just before progressing onto each UI activity – this method automatically saves what the clerk has entered (in text boxes etc) to transactional state. Thus, when `PageBase.HandOff()` is called, all this information can be saved “offline” (in the DB) along with the transaction until a supervisor can pick up where the clerk left off. This is done by calling `RestoreState()`, which loads each activity page with all controls populated with exactly what the clerk previously entered.

**7.7.4 ASP.NET Custom Controls**
The following custom ASP.NET controls were produced, providing additional ready made functionality which can be “dragged & dropped” into a business application.

**Transaction Selector:**
This control provides a customisable, pre-made mechanism by which to view and select a given transaction for processing. The control was implemented by sub-classing `System.Web.UI.Control` and overriding the `Render()` method to output table as shown below.

<table>
<thead>
<tr>
<th><strong>Transaction Name</strong></th>
<th><strong>Created</strong></th>
<th><strong>Options</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>Change Name</td>
<td>14/02/2006</td>
<td>More Info</td>
</tr>
<tr>
<td>Create User</td>
<td>14/02/2006</td>
<td>More Info</td>
</tr>
<tr>
<td>Update Balance</td>
<td>04/02/2006</td>
<td>More Info</td>
</tr>
</tbody>
</table>

*Figure 7.7.4.1: Transaction Selector ASP Control*

This was achieved by retrieving the user credentials from session state and then using the ORM to retrieve a ‘Work List’ specific to that user’s level (this code was used in Figure 7.3.2.1 to demonstrate ORM usage). Clicking “More Info” will open a pop-up window containing a Note Viewer Control (as discussed in next section). Selecting “Select & Process” forces a re-direct to the same page housing the control (i.e. SelectTransaction.aspx) passing the query string “?TransactionID=ID” – this is caught by `PageBase`, and the transaction begins (i.e. `PageBase.StartTransaction()` is called).

**Note Viewer:**
This control provides a pre-made mechanism by which to view the notes associated with a given transaction. Again, this control was implemented by sub-classing `System.Web.UI.Control` and overriding the `Render()` method to produce the output shown below (Figure 7.7.4.2). It works by examining either:

- **Session State** – This allows a user view notes whilst processing a transaction.
• A query string – This allows notes to be displayed independently of any transaction, i.e. if “More Info” has been clicked in the Transaction selector control.

![Figure 7.7.4.2: Note Viewer ASP Control](image)

The relevant notes are retrieved using the ORM, thus:

```csharp
using (EntityFactory myDB = new EntityFactory(DBAccessMode.Read))
{
    Set notes = myDB.RetrieveCollection(typeof(Note),
        "WorkItemTransaction = " + transactionID);

    if (notes.Count == 0) // If transaction does not have notes
    {
        this.writeNoNotesMsg(writer); // Write appropriate message
    }
    else // If transaction has notes
    {
        foreach (Note n in notes)
        {
            this.writeNoteTitle(writer, n.Title);
            this.writeNoteText(writer, n.Text);
        }
    }
}
```

![Figure 7.7.4.3: Code for note retrieval](image)

### 7.8 Work Item Scheduler

Unlike the other components, which have been implemented as dynamic link libraries (DLL’s) the scheduler has been implemented as a stand-alone application – which can be executed and left to run on a given machine. Before the scheduler can be used the business/developer must specify the transaction types they wish to schedule along with the task start time and its frequency, as depicted in Figure 7.8.1 below.

```xml
<Task>
    <Start starts="0" hours="0" minutes="0" seconds="0"/> - immediate start
    <Sleep starts="0" hours="4" minutes="0" seconds="0"/> - every 4 hours
    <Transactions>
        <Transaction type = "Change Name"/>
        <Transaction type = "Non-GUI Transaction 2"/>
    </Transactions>
</Task>
```

```xml
<Task>
    <Start starts="0" hours="8" minutes="0" seconds="0"/> - after 8 hours
    <Sleep starts="1" hours="0" minutes="0" seconds="0"/> - every day
    <Transactions>
        <Transaction type = "Create User"/>
    </Transactions>
</Task>
```
Once executed, a SAX parser (the .NET XmlTextReader class) is used to extract the details of each task – with a new TaskList instance being created for each time-slot; for example, the above XML would result in two new TaskList instances.

Standard .NET timers (System.Threading.Timer) have been used to keep track of each TaskList (more appropriately, its time span), with a TimerCallback delegate being used to handle calls from each timer when it expires. Upon expiry, the TimerCallback delegate calls a static Notify() method using a separate thread from the thread pool, passing its respective TaskList instance. This then results in each incomplete transaction of the given types being retrieved from the database with a TransactionTask instance being created for each one. Finally, each TransactionTask instance executes the transaction by creating the relevant transaction controller.
8. Testing

8.1 Introduction

As this project has aimed to provide a working framework it is necessary to put the implementation, and to some degree the design, through three specific forms of testing; Unit, Integration and End-to-end.

8.2 Unit Testing

The purpose of Unit Testing is to ensure that each unit in the system functions properly. This testing strategy was applied to each object in the system following first iteration implementation (as per the chosen methodology), with any problems being fixed during the second iteration. To facilitate unit test a C# .NET test class was introduced into each component assembly – which was compiled as an executable (*.exe) during testing (not a DLL). The test class worked simply by calling each method with an assortment of parameters to test whether the actions performed were appropriate.

8.2 Integration Testing

The purpose of performing Integration Testing is to ensure that the different portions that make up the system work correctly when put together – effectively, this considers whether they “talk to each other” appropriately. Integration testing was performed directly after the second iteration phase, with any issues remedied instantly. Much like unit test, use was made of each component’s test class, which contained logic pulling each completed component together – the final component, the ASP.NET Web Application, formed the final stage of integration test which tested all components together (except the Work Item Scheduler & Entity forward engineering tool – which are standalone applications).

8.3 End-to-End Testing

End-to-end testing involves testing the framework in an environment that mimics real-world use. As such, a series of sample workflow transactions and corresponding activities were created in order assess the effectiveness of the framework. It should be noted that before this testing process began the database already contained the relevant core Framework tables (as in Figure 6.2.1.1) and two workflow participants – clerk & supervisor.

8.3.1 Business Entity Creation

Using the Forward Engineering tool Business related entities (Business Entities) were produced with relative ease; with the only real development effort required to create the corresponding DB tables and compiling them. The following additional entities were created to mimic real-life usage:
• Party – Holding details about a given party, i.e. a customer.
• Address – A party’s address.
• Account – A party’s bank account.

8.3.2 Transaction & Activity Creation
Implementing a new Work Item Transaction (type) is effectively a three-stage process. Firstly, a new WorkItemTransactionType instance must be persisted to the database; this includes a (FK) relationship to the relevant user level (i.e. the level of user authority required). Then, a Transaction Controller must be created (and logic implemented for the commit method), followed by the creation of one or more Activity Blocks (unless already present from a previous task). Finally, the process definition is created which links the activities with its parent Work Item Transaction Type.

This process was carried out for the creation of 4 sample transactions:
• Create User
• Update Balance
• Generic Non GUI
• Change Name (Non-GUI)

The first two were tested within the GUI framework and the latter two tested using the Work Item Scheduler. The desired behaviour was observed in all cases.

8.3.3 Work Request Submission (Web Service)
This was tested both using the proxy approach detailed in section 7.1 and as the web service supported HTTP-GET requests, could be invoked directly from a web browser, as shown below.

![Test](to-be-inserted)

Figure 8.3.3.1: Browser invocation of web service

Once invoked the service returned the XML string shown below – as expected, this also created the relevant WorkItemTransaction and Note instances in the DB.

```xml
<?xml version="1.0" encoding="utf-8" ?>
<string xmlns="http://tempuri.org/">Transaction, 274 - Create User, was created at 10/04/2006 17:19:09</string>
```

Figure 8.3.3.2: SOAP response from web service
8.3.4 Web-based UI Integration

Two forms of testing were necessary here. The first tested that any ASP page with the appropriate assembly (DLL) references could communicate with any component. The second tested that the GUI of any activity block inheriting from `PageBase` would be subject to automatic user authorisation (i.e. a check ensuring the appropriate user has logged on), and automatic activity navigation, which ensures the Transaction Controller has ultimate control over what page is displayed. Both tests proved successful.

9. Evaluation

9.1 Introduction

This chapter aims to examine the success of the project by revisiting the minimum requirements, then going on to evaluate each project stage, implemented extensions, possible enhancements and finally looking at feedback from a .NET professional. The evaluation will comprise of both personal experiences and feedback from a professional with several years of workflow system development experience.

9.2 Aims, Objective and Minimum Requirements

**Requirement One:** To produce a series of interlinked base-components that can be extended by business developers to produce a bespoke workflow system.

This requirement effectively sums up the general qualitative aim of the project, which by itself proves rather difficult to measure. As such, Chapter 1 detailed five components intended to satisfy this requisite and associated each one with an underlying aim. Chapter 4 then expanded upon these; using the background research findings to tightly stipulate the detailed requirements and evaluation criteria of each component:

**Component 1: Data Access Layer**

As documented in Chapters 6 & 7, the DAL allows for fully configurable access to a database. The ORM approach followed ensured the complexities inherent in standard data access were abstracted away from the developer; allowing for a neat object-orientated approach which is widely regarded to speed up development. It allowed for the CRUD of persisted data, and permitted information to be queried using object-orientated concepts/notation. Finally, a series of core workflow tables were produced.
The ORM aspect in particular gained positive feedback from the reviewer (Appendix E) – who praised the simplicity of its use and acknowledged the large technical challenge its production represented. The author feels that the ORM approach went beyond the minimum requirement, producing overall a better quality component.

**Component 2: Work Request Submission**
The web service implemented presents a fully platform independent way of entering work items onto the system. Providing an XML payload allowed for the specification of transactional notes, but is also flexible enough to accommodate the addition of further information as per specific business requirements.

**Component 3: Workflow Engine**
A configurable Base Controller was provided, which, providing inherited appropriately, will automatically manage the behaviour of a transactional process by parsing a given process definition, which was also designed as part of this project.

**Component 4: Workflow Activity Block**
The workflow activity block was designed and implemented as a conceptual block loosely coupling GUI and business logic together following the sound principles laid down by the MVC.

**Component 5: Web-based User Interface**
This was achieved by producing a series of components capable of being “plugged” into any ASP.NET application. A mechanism was provided for workflow participants to log onto the system, view & select transactions and view transactional notes using the Note Viewer component. Central to the UI design was the `PageBase` class which can pull the various constituents of any activity block together.

**Requirement Two:** *To demonstrate and test the usage of these components by utilising the framework to create a series of sample workflow transactions.*
This was achieved by mimicking several different workflow transactions that would collectively test each element of the framework’s functionality. A discussion surrounding this can be found in Section 8.

**9.3 Project Enhancements**
The following two enhancements were also implemented which contributed as both a measure of success and as an additional improvement to framework usability.
Entity Forward Engineering Tool:
To facilitate the easy addition of new ORM compatible entities a graphical entity designer was provided with full code generation support (discussion in section 7.3.1)

Transaction Scheduler:
This allowed those transactions comprised totally of non-GUI activities to be automatically executed a time specified by a business/developer (discussion in sections 6.8 & 7.8).

9.4 Evaluation of Project Approach
The RUP methodology was found to be highly suited to this project. The two-stage iterative approach adopted allowed for continuous evaluation to be performed throughout the system life cycle, rather than it simply being bolted on at the end. This meant problems could be easily spotted and instantly remedied – the major advantages this afforded became glaring obvious towards the end of the development phase, where bugs often took much longer to fix as they were either deep-seeded in the solution or shared among multiple components. Many texts cite that it is much easier to make changes at the beginning rather than at the end – this was certainly found to be accurate. Another advantage was that building the process of continuous review into the development phase actually meant there was relatively little to test, as the majority of problems had already been fixed during the earlier iterations.

The only real criticism worth making is that the RUP places heavy emphasis on a large amount of initial design modelling and re-modelling following the lessons learnt during each iteration. The projects tight timeframe simply did not allow for this. As such, only the elements of RUP most suitable were adopted – it is the authors opinion that this selective approach worked well and allowed for minimal hiccups.

Overall, the predicted project schedule (Appendix B) was in line with actual project progress. In part, this is probably due to splitting the design/implementation phases into smaller components, which made timeframes easier to visualise and therefore predict. There was, however, one notable exception in relation to implementing the DAL. This process took an additional 6 days (^54%) due to several unforeseen problems implementing ORM functionality. However, rather than impacting the overall schedule, time allotted for January module revision was re-distributed accordingly, allowing for the completion of this component. This re-schedule was deemed workable due to good module coursework results (requiring lower exam performance) and ultimately sensible if it meant delivering the overall solution on time – which it did.
Finally, had this been within a professional environment with a paying customer expecting delivery of a final solution, a peer review would ideally be carried out to ensure the quality of the implementation. However, this did not fall within the scope of this project.

9.5 Evaluation of Background Research

Throughout the project a conscious effort was made to ensure the background research was gathered from a wide variety of sources, including; books, journals and the web. Whilst books and journals offer reliable reading as they are well researched and checked before publishing, the reliability of web sources on the other hand is often questionable and could affect the quality of findings. As such, web references were only used where appropriate or when the other sources were not available – this was often found to be the case when researching Workflow, with very few current books available. Ideally, web sources should have been used for guidance only, with authority being obtained from more reliable sources such as journals and books.

9.6 Evaluation of Design

The importance of a sound design applies to all forms of software engineering, however when producing a development framework the importance of a sound design is absolutely fundamental – indeed as Erich Gamma points out; “Frameworks emphasize design reuse over code reuse” [28]. As demonstrated in Chapter 6, a great deal of careful thought was put into the design of each component in order to accommodate a high degree of reuse. This meant adopting tried & tested software design patterns such as those by the Gang of Four and Martin Fowler to ensure problems were solved in the best possible way. Core object-orientated paradigms like Meyer’s Open-Closed Principle and Liskov’s Substitution Principle were revisited and studied in relative detail to help understand the history behind concepts such as abstraction, which feature heavily in the design. These mature techniques and concepts were applied in parallel along with newer techniques and concepts such as object-relational mapping and web services to ensure the design was both architecturally sound and technologically current. Finally, the work of Boyce Codd was followed to make certain the database conformed to an appropriate normal form.

It is felt that by adopting the above, well-researched, approach to the various design problems a high quality overall design was produced. This was certainly highlighted when implementing the Work Item Scheduler – which integrated into the existing components with relative ease.
9.7 Evaluation of Implementation

9.7.1 Choice of Technology Used

In some respects the technology choices were relatively easy to make. Because the nature of the project was to produce a .NET specific framework it was not necessary to contrast different languages or even web technologies. C# was an obvious choice as the author has previous experience (plus transferable Java knowledge) and ASP.NET is the core web technology provided by .NET.

As no specific operating system was stipulated in the requirements, a choice of platform was available – Windows or Linux (using Mono). Windows was adopted for two reasons; firstly because the author did not have Linux installed and had several years of experience using the Microsoft Visual Studio development environment – which is not available for Linux. Although, it should be noted that the latter was by far the biggest consideration – with the choice being made to use Visual Studio primarily as the author was most comfortable using it. Above all, this meant that development could progress instantly (without learning a new environment) and allowed for an efficient development by leveraging Visual Studio’s excellent features; such as automatic code-completion, advanced debugging tools and the visual ASP.NET page designer.

Whilst the database requirements of the framework itself were relatively simplistic, a feature-rich database, namely Microsoft SQL Server 8.0, was selected after considering wider future business uses beyond the scope of this project (as discussed in Section 5.3.3). Whilst the author maintains this was a sensible decision, the choice does impact the portability of the DAL between different database systems. This is because a specific ADO.NET MS SQL Server data provider was used (System.Data.SqlClient) due to it providing optimised access for MS SQL Server 7.0 and above. As such, those businesses not using MS SQL Server would need to re-compile the DAL after modifying the data provider, for example using System.Data.ODBC instead (which in most cases would only involve a simple find & replace operation). DAL portability would certainly feature in any future development.

The distributed platform independent Web Services approach chosen for implementing the Work Request Input component was found to be extremely effective. It provided the best possible mechanism for meeting the evaluation criteria for that component (as discussed in Section 5.5), and was remarkably easy to implement within .NET – literally just a case of providing a [WebMethod] attribute above the given function.
9.7.2 Development Effort & Software Quality

The implementation of the final product required successful operation between multiple layers; DB, DAL, Workflow Engine & Activity blocks and ASP pages. As mentioned in section 2.2.5, the iterative, component-based approached afforded by the chosen methodology provided a great mechanism by which to break down these development tasks into more manageable chunks. As such, the author was never overwhelmed by the complexity of the undertaking.

The most challenging component by far was implementing an object-relational Data Access Layer. The complexity inherent in such conversions meant large parts of the .NET Framework had to be employed and utilised by over 1800 lines of code to ensure they co-operated seamlessly. As mentioned in the previous section, this represented a large amount of additional unforeseen work, and duly affected the project plan. Despite the additional complexities and planning implications, it is the author’s opinion that this extra work allowed for a much simpler overall implementation and increased the general quality of the solution.

Whilst a good design concentrates on re-use, a good framework implementation concentrates on being in a position to be put to work immediately [28]. Despite the DAL portability issues, testing on various machines indicated that the Framework could be successfully deployed within around 30 minutes – at which point a business developer could start writing their transactions. Given this, the implementation is considered a success.

9.8 Possible Future Enhancements

Although a solution has been produced which meets and in some cases exceeds the minimum requirements, future development should consider implementing the following functional updates, although it should be noted that this list is by no means exhaustive:

**DAL portability** – As mentioned in the previous section, the current DAL will only function on Microsoft SQL Server. Future development should consider either adopting ODBC connectivity or allowing the developer to specify their chosen database, thus allowing the DAL to communicate with a single interface which deals with data provider issues invisibly.

Allow **Work Item Scheduler** to operate as a **Windows Service** – This would allow the scheduler to run automatically, without explicit user execution and regardless of whether anyone was logged on to the hosting machine. Future development could involve modifying the scheduler to accommodate said functionality by extending the .NET ServiceProcess.ServiceBase class.
Active Directory Support – User authorisation is currently provided by the WorkStreamUser table, using SHA-512 hashes to store credentials securely – entering such credentials would require a separate business effort. It would be desirable in the future to integrate authentication automatically with the operating system, specifically with Active Directory services – which many businesses would already have set up with each user’s details. This would be achieved by making use of the functionality provided by the .NET System.DirectoryServices namespace.

In addition to the functionality mentioned above, the following non-functional improvements could also be made:

To guide developers around the Framework a user manual could be produced. This would provide the necessary introductions, examples and act as a readily accessible point of support for queries.

Ideally, the user manual should be backed up with more detailed designs and models. This would allow developers to fully understand the architecture of the Framework, which would be of particular use if the Framework required customising in some way to meet additional business needs.

9.9 Evaluation from .NET Workflow Professionals

With no immediate end user to provide specific feedback about the system, it was necessary to seek constructive feedback from another source. An obvious choice was a .NET professional working within the Workflow domain. Appendix E contains said feedback after the professional (who works for large IT consultancy group) had reviewed the design and working implementation.

The vast majority of this feedback was extremely positive and it is hoped the feedback will add to the success of this project. Although, of course, the real test of any system is to examine how it functions within a real working environment.
References


[31] AMBLER, S.W. The Object-Relational Impedance Mismatch. [Cited 01/03/06]. Available From: http://www.agiledata.org/essays/impedanceMismatch.html


[40] APACHE.ORG. Axis User’s Guide. [Cited 02/04/06]. Available From: http://ws.apache.org/axis/java/user-guide.html#WSDL2JavaBuildingStubsSkeletonsAndDataTypesFromWSDL

Appendix A – User Reflection

Overall, this project has proved both an enjoyable and rewarding experience. From a skills perspective, the lessons learnt by experiencing each stage of a project’s lifecycle firsthand have been invaluable. My knowledge of the .NET Framework and indeed of software in general has improved ten-fold, which has increased both my confidence and ability to solve software-related problems.

In addition to the numerous technical abilities gained or advanced, my time management skills have no doubt improved, thanks, in part, to some painful ‘reality checks’ about exactly what can be accomplished within a given timeframe. That said, this assignment has also been about pushing myself by deliberately choosing, what I believe, has been a challenging project. Indeed, opting to implement an ORM from scratch was a difficult and sometimes frustrating undertaking, but very intellectually rewarding – something which I have always regarded as deeply important in any task.

Below, are 4 lessons that I learnt which may prove valuable to other students in the future:

**Choose something interesting**
The key to a successful project is ensuring you pick a topic that interests you. I found that you are more likely to stay focused and motivated if the subject area is intriguing, indeed you will probably find the whole experience far more enjoyable.

**Don’t lose sight of the goal**
It if often very easy to stray off course by getting wrapped up in the specifics of one particular problem-area, or devoting time to non-core activities, such as making fancy additions. Whilst I would encourage fellow students to always think two-steps ahead, they must strike a careful balance between enhancing the quality of their solution and over delivering. It is good practice to ask yourself the question; ‘is this relevant at this stage’ – and only make additions when the core deliverable is complete.
Remember the “Six P’s” – Perfect planning prevents [pitifully] poor performance
I would urge students to break down their project into smaller, measurable chunks. This way, if the first ‘chunk’ takes longer than expected you can adjust the schedule early on, thus avoiding any frenzied late nights, as the deadline creeps closer. Good time management will make for a relatively stress-free and relaxed project!

Don’t underestimate the importance of sound design
For any project involving software development it is vital that extra care is given to the design phase. Firstly, I would strongly urge any student to make sure they fully understand their chosen language and the libraries/functionality it exposes – take time to read a book on the subject. I would also advise reading a book on software design, which contain tried & tested solutions to many design problems – although, these shouldn’t be blindly copied, you should apply the concepts learnt to your specific scenario. Sound design makes understanding the final solution, development and producing enhancements so much easier.

To conclude, I feel that this project has been a tremendously positive experience. It has given me the opportunity to put all my acquired skills into practice, in an area which I found interesting. I am in no doubt that the lessons learnt will prove invaluable in my future career – which will always hold the importance of continuously learning new proficiencies in high regard.
## Appendix B – Project Schedule

### Initial (Pre-research) Gantt Chart

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
<th>Nov</th>
<th>Dec</th>
<th>Jan</th>
<th>Feb</th>
<th>Mar</th>
<th>Apr</th>
<th>May</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Research</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Methodology selection &amp; evaluation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Note creation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Requirements</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Requirements Modeling</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Analysis &amp; Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 1: Core Components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 2: Core Components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 1: Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 2: Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 1: Sample Transactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 2: Sample Transactions</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Construction</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 1: Core Components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 2: Core Components</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 1: Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bar 2: Functionality</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Test &amp; Evaluate</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Unrelated Project Events</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christmas</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christmas events &amp; revision</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Report tasks</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Main report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Evaluation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>TOC &amp; draft chapter submission</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Submit pdf report</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
## Predicted Progress

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Background Research</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Locating research</td>
<td>01-Mar</td>
<td>25-Nov</td>
<td>25 days</td>
<td></td>
</tr>
<tr>
<td>Methodology, selection &amp; evaluation</td>
<td>06-Mar</td>
<td>12-Mar</td>
<td>5 days</td>
<td></td>
</tr>
<tr>
<td>Note creation</td>
<td>16-Mar</td>
<td>27-Mar</td>
<td>20 days</td>
<td></td>
</tr>
<tr>
<td><strong>Requirements Analysis</strong></td>
<td></td>
<td></td>
<td></td>
<td>M1</td>
</tr>
<tr>
<td>Requirements Analysis</td>
<td>07-Dec</td>
<td>11-Dec</td>
<td>5 days</td>
<td></td>
</tr>
<tr>
<td>Requirements Modeling</td>
<td>12-Dec</td>
<td>20-Dec</td>
<td>9 days</td>
<td></td>
</tr>
<tr>
<td><strong>Component Design</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Data Access Layer</td>
<td>21-Dec</td>
<td>23-Dec</td>
<td>3 days</td>
<td></td>
</tr>
<tr>
<td>2. Data Access Layer</td>
<td>01-Jan</td>
<td>02-Jan</td>
<td>2 days</td>
<td></td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18-Jan</td>
<td>18-Jan</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>4. Work Request Input</td>
<td>20-Jan</td>
<td>20-Jan</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>5. Workflow Engine + Activity Book</td>
<td>21-Feb</td>
<td>31-Jan</td>
<td>11 days</td>
<td></td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>09-Feb</td>
<td>09-Feb</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>7. Web-based GUI</td>
<td>12-Feb</td>
<td>19-Feb</td>
<td>7 days</td>
<td></td>
</tr>
<tr>
<td>8. Web-based GUI</td>
<td>07-Mar</td>
<td>10-Mar</td>
<td>4 days</td>
<td></td>
</tr>
<tr>
<td>9. Sample Transactions</td>
<td>11-Mar</td>
<td>16-Mar</td>
<td>4 days</td>
<td></td>
</tr>
<tr>
<td>10. Sample Transactions</td>
<td>26-Mar</td>
<td>27-Mar</td>
<td>2 days</td>
<td></td>
</tr>
<tr>
<td><strong>Component Implementation</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>11. Data Access Layer</td>
<td>23-Dec</td>
<td>31-Dec</td>
<td>9 days</td>
<td>M2</td>
</tr>
<tr>
<td>12. Data Access Layer</td>
<td>02-Jan</td>
<td>03-Jan</td>
<td>2 days</td>
<td>M3</td>
</tr>
<tr>
<td>13. Work Request Input</td>
<td>21-Jan</td>
<td>24-Jan</td>
<td>3 days</td>
<td>M4</td>
</tr>
<tr>
<td>14. Workflow Engine + Activity Book</td>
<td>01-Feb</td>
<td>07-Feb</td>
<td>7 days</td>
<td></td>
</tr>
<tr>
<td>15. Web-based GUI</td>
<td>09-Feb</td>
<td>11-Feb</td>
<td>3 days</td>
<td></td>
</tr>
<tr>
<td>16. Web-based GUI</td>
<td>31-Mar</td>
<td>06-Mar</td>
<td>14 days</td>
<td></td>
</tr>
<tr>
<td>17. Sample Transactions</td>
<td>20-Mar</td>
<td>20-Mar</td>
<td>5 days</td>
<td>M5</td>
</tr>
<tr>
<td>18. Sample Transactions</td>
<td>26-Mar</td>
<td>30-Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td><strong>Unclassified Project Events</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Christmas</td>
<td>24-Dec</td>
<td>27-Dec</td>
<td>4 days</td>
<td></td>
</tr>
<tr>
<td>Christmas exams &amp; revision</td>
<td>03-Jan</td>
<td>18-Jan</td>
<td>16 days</td>
<td></td>
</tr>
<tr>
<td><strong>Final Exams</strong></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Final Exam Report</td>
<td>27-Mar</td>
<td>05-Mar</td>
<td>12 days</td>
<td>M7</td>
</tr>
<tr>
<td>Main Report</td>
<td>01-Apr</td>
<td>26-Apr</td>
<td>7 days</td>
<td></td>
</tr>
<tr>
<td>TOC &amp; draft chapter submission</td>
<td>10-Mar</td>
<td>18-Mar</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>Submit Report</td>
<td>19-Mar</td>
<td>26-Mar</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>Submit Draft Report</td>
<td>16-Apr</td>
<td>03-May</td>
<td>1 days</td>
<td></td>
</tr>
</tbody>
</table>
### Actual Progress

<table>
<thead>
<tr>
<th>Task Name</th>
<th>Start</th>
<th>End</th>
<th>Duration</th>
<th>Milestone</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Get-Go Research</td>
<td>01 Nov</td>
<td>05 Nov</td>
<td>5 days</td>
<td></td>
</tr>
<tr>
<td>2. Methodology Selection</td>
<td>03 Nov</td>
<td>12 Nov</td>
<td>1 day</td>
<td></td>
</tr>
<tr>
<td>3. Noteworthy</td>
<td>15 Nov</td>
<td>07 Dec</td>
<td>3 days</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Requirements Analysis</td>
<td>07 Dec</td>
<td>11 Dec</td>
<td>2 days</td>
<td></td>
</tr>
<tr>
<td>2. Requirements Meeting</td>
<td>12 Dec</td>
<td>20 Dec</td>
<td>3 days</td>
<td>M1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Design</td>
<td>21 Dec</td>
<td>23 Dec</td>
<td>3 days</td>
<td></td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>05 Jan</td>
<td>07 Jan</td>
<td>2 days</td>
<td></td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>15 Jan</td>
<td>1 day</td>
<td></td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>31 Jan</td>
<td>11 days</td>
<td></td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>05 Feb</td>
<td>1 days</td>
<td></td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>8 days</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Implementation</td>
<td>23 Dec</td>
<td>05 Jan</td>
<td>15 days</td>
<td>M2</td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>07 Jan</td>
<td>08 Jan</td>
<td>2 days</td>
<td>M2</td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>19 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>21 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>07 Feb</td>
<td>11 days</td>
<td>M1</td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>7 days</td>
<td>M5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M5</td>
</tr>
<tr>
<td>1. Same Transactions</td>
<td>15 Mar</td>
<td>19 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td>2. Same Transactions</td>
<td>25 Mar</td>
<td>27 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Design</td>
<td>23 Dec</td>
<td>05 Jan</td>
<td>15 days</td>
<td>M2</td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>07 Jan</td>
<td>08 Jan</td>
<td>2 days</td>
<td>M2</td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>19 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>21 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>07 Feb</td>
<td>11 days</td>
<td>M1</td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>7 days</td>
<td>M5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M5</td>
</tr>
<tr>
<td>1. Same Transactions</td>
<td>15 Mar</td>
<td>19 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td>2. Same Transactions</td>
<td>25 Mar</td>
<td>27 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Design</td>
<td>23 Dec</td>
<td>05 Jan</td>
<td>15 days</td>
<td>M2</td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>07 Jan</td>
<td>08 Jan</td>
<td>2 days</td>
<td>M2</td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>19 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>21 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>07 Feb</td>
<td>11 days</td>
<td>M1</td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>7 days</td>
<td>M5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M5</td>
</tr>
<tr>
<td>1. Same Transactions</td>
<td>15 Mar</td>
<td>19 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td>2. Same Transactions</td>
<td>25 Mar</td>
<td>27 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Design</td>
<td>23 Dec</td>
<td>05 Jan</td>
<td>15 days</td>
<td>M2</td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>07 Jan</td>
<td>08 Jan</td>
<td>2 days</td>
<td>M2</td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>19 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>21 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>07 Feb</td>
<td>11 days</td>
<td>M1</td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>7 days</td>
<td>M5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M5</td>
</tr>
<tr>
<td>1. Same Transactions</td>
<td>15 Mar</td>
<td>19 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td>2. Same Transactions</td>
<td>25 Mar</td>
<td>27 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>1. Component Design</td>
<td>23 Dec</td>
<td>05 Jan</td>
<td>15 days</td>
<td>M2</td>
</tr>
<tr>
<td>2. Date Access Layer</td>
<td>07 Jan</td>
<td>08 Jan</td>
<td>2 days</td>
<td>M2</td>
</tr>
<tr>
<td>3. Work Request Input</td>
<td>18 Jan</td>
<td>19 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>4. Workflow Engine - Activity Block</td>
<td>21 Jan</td>
<td>21 Jan</td>
<td>1 days</td>
<td>M3</td>
</tr>
<tr>
<td>5. Web-based GUI</td>
<td>03 Feb</td>
<td>07 Feb</td>
<td>11 days</td>
<td>M1</td>
</tr>
<tr>
<td>6. Web-based GUI</td>
<td>15 Feb</td>
<td>20 Feb</td>
<td>7 days</td>
<td>M5</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>M5</td>
</tr>
<tr>
<td>1. Same Transactions</td>
<td>15 Mar</td>
<td>19 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td>2. Same Transactions</td>
<td>25 Mar</td>
<td>27 Mar</td>
<td>5 days</td>
<td>M6</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Completion</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Related Events</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Report Tasks</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Notes</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total Summary</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Appendix C – Design

Core Entity Requirements

Table: Work Item Transaction

Description: This table details each item of work on the system, whether it be complete or not

Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Item Transaction ID</td>
<td>Primary key – uniquely identifies each transaction.</td>
</tr>
<tr>
<td>Creation Date</td>
<td>The date the transaction was created.</td>
</tr>
<tr>
<td>Transaction Type</td>
<td>A foreign key relationship indicating the type of transaction.</td>
</tr>
<tr>
<td>Transaction Status</td>
<td>A foreign key relationship indicating the status of the transaction.</td>
</tr>
<tr>
<td>Transactional Data</td>
<td>This is a text field containing serialised data relating to the transaction. Its purpose is to provide a mechanism for transactions to be paused, or handed off.</td>
</tr>
</tbody>
</table>

Transaction Type

Description: The type of work. Each transaction is of one type or another, for example; Update Balance, Change Address etc

Contents

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Item Transaction Type ID</td>
<td>Primary key – uniquely identifies each type.</td>
</tr>
<tr>
<td>Type Name</td>
<td>The name of the type. Must be unique.</td>
</tr>
<tr>
<td>User Levels</td>
<td>The user levels permitted to carry out a transaction of the given type.</td>
</tr>
</tbody>
</table>

Note:

Description: A generic textual comment that can be appended to a transaction. This could, in theory, also contain extra information payloads to assist the automation of certain transactional activities.

Contents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Note ID</td>
<td>Primary key – uniquely identifies each Note.</td>
</tr>
<tr>
<td>Title</td>
<td>The title of the note.</td>
</tr>
<tr>
<td>Text</td>
<td>Note text.</td>
</tr>
<tr>
<td>Work Item Transaction</td>
<td>The transaction the note applies to.</td>
</tr>
</tbody>
</table>
WorkStream User:

Description: Information relating to a given user.

Contents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Stream User ID</td>
<td>Primary key – uniquely identifies each Note.</td>
</tr>
<tr>
<td>User Name</td>
<td>The user’s user name.</td>
</tr>
<tr>
<td>Password</td>
<td>The user’s password</td>
</tr>
<tr>
<td>Level</td>
<td>Foreign key relationship, used to indicate user level.</td>
</tr>
</tbody>
</table>

User Level:

Description: Information relating to the user’s level, i.e. whether they are a clerk, supervisor etc

Contents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>User ID</td>
<td>Primary key – uniquely identifies each User.</td>
</tr>
<tr>
<td>Level Name</td>
<td>The name assigned to the level, i.e. supervisor.</td>
</tr>
<tr>
<td>Level Rank</td>
<td>A rank used to indicate level of authority, higher level = increased authority.</td>
</tr>
</tbody>
</table>

Work Item Transaction Status:

Description: This contains information regarding the status of the transaction. For example, whether it is complete, awaiting supervisor authorisation etc.

Contents:

<table>
<thead>
<tr>
<th>Name</th>
<th>Role</th>
</tr>
</thead>
<tbody>
<tr>
<td>Work Item Transaction Status</td>
<td>Primary key – uniquely identifies each Status level.</td>
</tr>
<tr>
<td>Status</td>
<td>The status, i.e. Complete.</td>
</tr>
</tbody>
</table>
Entity-Relationship Diagram

- **Note**
  - Consists of
  - **Work Item Transaction**
    - Has
    - **Work Item Transaction Status**
      - **User Level**
        - Has status
          - **Work Stream User**

- **Work Item Transaction Type**
  - Requires status
  - **User Level**
    - Has status
### ORM Classes

#### Factory Functionality

<table>
<thead>
<tr>
<th>Method Name</th>
<th>Parameters</th>
</tr>
</thead>
<tbody>
<tr>
<td>EntityFactory</td>
<td>(in accessMode : DBAccessMode)</td>
</tr>
<tr>
<td>readConfigs ()</td>
<td></td>
</tr>
<tr>
<td>Retrieve (in type : Type, in ID : object) : EntityBase</td>
<td></td>
</tr>
<tr>
<td>executeCustomQuery (in query : string, in parameters : IList) : IDataReader</td>
<td></td>
</tr>
<tr>
<td>executeCustomQuery (in query : string, in parameter : SqlParameter) : IDataReader</td>
<td></td>
</tr>
<tr>
<td>RetrieveReference (in parentObject : object, in requiredType : Type) : EntityBase</td>
<td></td>
</tr>
<tr>
<td>RetrieveCollection (in t : Type, in obj : object) : Set</td>
<td></td>
</tr>
<tr>
<td>executeGetQuery (in table : string, in ID : object) : IDataReader</td>
<td></td>
</tr>
<tr>
<td>Update (in obj : object)</td>
<td></td>
</tr>
<tr>
<td>New (in t : Type) : object</td>
<td></td>
</tr>
<tr>
<td>getConnection () : IDbConnection</td>
<td></td>
</tr>
<tr>
<td>getParameter (in pInfo : PropertyInfo, in obj : object) : SqlParameter</td>
<td></td>
</tr>
<tr>
<td>Dispose ()</td>
<td></td>
</tr>
<tr>
<td>Commit ()</td>
<td></td>
</tr>
<tr>
<td>UpdateCache (in entityType : string, in propertyName : string, in entityID : int, in propertyValue : object)</td>
<td></td>
</tr>
<tr>
<td>AddToCache (in entity : EntityBase)</td>
<td></td>
</tr>
</tbody>
</table>

### DBAccessMode

- Read
- ReadWrite
### Query Processing

**QueryHelper**

- `typeMappings : DBTypeMappings`
- `joinCache : Hashtable`
- `configs : Configs`

+ `QueryHelper (in configs : Configs )`  
+ `Dispose ()`  
+ `GetQueryOperator (in query : string ) : string`

- `getColumnPath (in t : Type , in query : string ) : string`

+ `BuildJoin (in t : Type , in query : string ) : string`

+ `BuildQueryParam (in t : Type , in propName : string , in propValue : string , in incrementor : int ) : SqlParameter`

+ `getSQLOperator (in op : string ) : string`

+ `GetEntityDBColumns (in t : Type ) : string`

+ `GetFinalColType (in t : Type , in query : string ) : Type`

+ `GetMappedColumn (in t : Type , in query : string , in queryOperator : string ) : string`

+ `GetTableMappings (in tableName : string , in dr : IDataReader ) : Hashtable`

### SingleFilter

- `myDB : EntityFactory`
- `configs : Configs`

  - `join : string = string .Empty`

  - `propName : string = string .Empty`

  - `propValue : string = string .Empty`

  - `finalQuery : string = string .Empty`

  - `dBParam : SqlParameter = null`

  - `selectQuery : string = `"SELECT (0)ID FROM (0) "`

+ `SingleFilter (in DB : EntityFactory , in config : Configs )`

+ `PerformSingleFilter (in t : Type , in query : string ) : IDataReader`

### MultipleFilter

- `myDB : EntityFactory`
- `configs : Configs`

  - `selectQuery : string = `"SELECT (0)ID FROM (0) "`

  - `criteria : Hashtable`

  - `requireJoins : ArrayList`

  - `simpleQuery : ArrayList`

  - `parameters : ArrayList`

  - `propName : string = string .Empty`

  - `propValue : string = string .Empty`

  - `queryOperator : string = string .Empty`

  - `queryHelper : QueryHelper`

  - `uniqueCounter : int = 1`

+ `MultipleFilter (in DB : EntityFactory , in config : Configs )`

+ `PerformMultipleFilter (in t : Type , in query : string ) : IDataReader`

+ `calculateJoinRequirements (in query : string )`

+ `processJoins (in t : Type )`

+ `processSimpleQueries (in t : Type )`

### DBTypeMappings

- `typeMap : Hashtable`

+ `DBTypeMappings []`

+ `GetType (in t : Type ) : SqlDbType`
**Set**
- Add (in value : EntityBase )
- AddRange (in list : IList )
- Sort (in propertyName : string )

**DBTypeMappings**
- typeMap : Hashtable
- Add (in value : EntityBase )
- AddRange (in list : IList )
- Sort (in propertyName : string )

**Object handling**

**ObjectHelper**
- myDB : EntityFactory
- SetObjectHelper (in DB : EntityFactory )
- GetPopulatedPropertiesCount (in type : Type ) : int
- getConstructorIndex (in constructor : ConstructorInfo [], in propertyCount : int ) : int
- getConstructorValues (in constructor : ConstructorInfo [], in constructorIndex : int , in valueMappings : Hashtable ) : object []
- getPropertyValue (in obj : object , in propName : string ) : string

**ObjectCache**
- objectMappings : Hashtable
- Add (in obj : object )
- Remove (in obj : object )
- Update (in obj : object )
- Update (in entityType : string , in propertyName : string , in entityID : int , in propertyValue : object )
- GetObjectCache () : IList

**Configuration**

**Configs**
- OPERATOR_EQUALS : string = "="
- OPERATOR_MORETHAN : string = ">"
- OPERATOR_LESSTHAN : string = "<"
- OPERATOR_LESSTHAN_EQUALTO : string = "<="
- OPERATOR_MORETHAN_EQUALTO : string = ">="
- OPERATOR_NOT_EQUALTO : string = "!="
- NameMappings : Hashtable

**ConfigReader**
- ConfigReader ()
**Workflow Engine**

**Base Classes**

```csharp
BaseTransactionController
```

**Entities**

### DTOSerializer

```csharp
-myDB : EntityFactory
+DTOSerializer (in DB : EntityFactory )
+Serialize (in type : Type , in entity : EntityBase ) : object
+Deserialize (in dto : DTOBase ) : EntityBase
```

### EntityBase

```csharp
#myDB : EntityFactory
-isClone : bool = false
+EntityBase ()
#UpdateCache (in propertyName : string , in obj : object )
+Clone () : EntityBase
```

**XmlParser**

```csharp
-reader : XmlTextReader
-nameMappings : Hashtable
+XmlParser ()
+Parse (in fileName : string ) : Hashtable
+doRead ()
+getMemberDetails (in typeName : string )
+getDBField () : string
```

**BaseTransactionController**

```csharp
#state : TransactionState
#transaction : WorkItemTransactionDTO
-enumerator : IEnumerator
#Activities : IList
#callingPage : Page
#currentActivity : BaseActivityController

+BaseTransactionController (in activities : IList , in workItemTransaction : WorkItemTransactionDTO )
#RestoreState ()
+RetrieveFromState (in key : string ) : object
+AddToState (in key : string , in obj : object )
+RemoveFromState (in key : string )
+Serialize ()
+Deserialize (in base & String : string )
+NextActivity ()
+GetActivityPage () : string
+GetActivityName () : string
+Commit ()
+Transaction () : WorkItemTransactionDTO
+CurrentActivity () : BaseActivityController
```
Controller Creation

```
BaseActivityController
+ActivityName : string
+PageName : string
+BaseActivityController ()
+EnterTask ()
+RetrieveFromState (in key : string) : object
+AddToState (in key : string, in obj : object)
+RemoveFromState (in key : string)

#transactionController : BaseTransactionController
+PageName : string

TransactionControllerFactory
+TxnControllerFactory ()
+GetController (in workItemTransaction : WorkItemTransactionDTO) : BaseTransactionController

ActivityHolder
+ActivityHolder (in activityName : string, in className : string, in pageName : string)
+Create () : BaseActivityController
+ActivityName () : string
+ClassName () : string
+PageName () : string

TransactionHolder
+TransactionHolder (in className : string, in activities : IList)
+ClassName () : string
+Activities () : IList

ActivityParser
+xtr : XmlTextReader
+ActivityParser ()
+ParseActivites () : Hashtable
+getActivites () : IList

Transactional State

TransactionState
+state : Hashtable
+TransactionState ()
+Add (in obj : object, in key : string)
+Retrieve (in key : string) : object
+Remove (in key : string)
+Clear ()
+Serialize () : string
+Deseralize (in base 64 String : string)
```
## Work Item Scheduler

```csharp
class Scheduler
{
    -tasks : TaskList []
    -keepAlive : ArrayList

    +Scheduler ()
    -Main (in args : string [])
    -Notify (in obj : object )
    -msg ()
    -readConfig () : IList
    -schedule ()
}

class TaskParser
{
    -xtr : XmlTextReader

    +TaskParser ()
    +Parse () : TaskList []
    -doRead () : TaskList []
    -getStartTime () : TimeSpan
    -getSleepTime () : TimeSpan
    -getTransactions () : ArrayList
}

class TaskList
{
    -tasks : ArrayList

    +threadStart : TimeSpan
    +threadSleep : TimeSpan
    +transactionTypes : string []

    +TaskList (in transactionTypes : string [], in threadStart : TimeSpan, in threadSleep : TimeSpan )
    +ExecuteTasks ()
    +createTransactionTasks ()
    +Attach (in task : TransactionTask )
    +DetachAll ()
}

interface ITask
{
}

class TransactionTask
{
    -workItemTransactionDTO : WorkItemTransactionDTO
    -factory : TxnControllerFactory

    +TransactionTask (in workItemTransactionDTO : WorkItemTransactionDTO, in factory : TxnControllerFactory )
}
Web-based GUI

PageBase

-CurrentTransaction : BaseTransactionController
-pageName : string
-redirectOnLoginError : bool = true

+PageBase ()
+EnterTask ()
+SavePageToState ()
+SaveTransactionalState ()
+RestoreState ()
+HandOff ()
+CurrentUser () : string
+PageName () : string
+OnPreRender (in e : EventArgs )
+OnLoad (in e : EventArgs )
-starTransaction (in transactionID : int )
+RaiseError (in errorText : string )
-validateUser () : bool
+LogOut ()
+Navigate (in uRL : string )
+ProcessNextActivity ()
-endTransaction ()

SelectTransaction

#Menu : WSMenuDrawer
#ErrorDisplay 1 : ErrorDisplay
#WorkItemGrid 1 : WorkItemGrid
#Form 1 : HtmlForm

-Page_Load (in sender : object, in e : EventArgs )
#OnInit (in e : EventArgs )
-InitializeComponent ()

UserValidator

+ValidateUser ()
+ValidateUser (in userName : string, in password : string ) : WorkStreamUserDTO
-retrieveUser (in userName : string ) : WorkStreamUserDTO
-checkHash (in userDTO : WorkStreamUserDTO, in password : string ) : bool
-getHash (in password : string ) : string
+CreateUser (in userName : string, in password : string )

WSLogin

#lblUserName : Label
#lblPassword : Label
#txtUserName : TextBox
#txtPassword : TextBox
#btnLogin : WSButton

-Page_Load (in sender : object, in e : EventArgs )
#OnInit (in e : EventArgs )
-InitializeComponent ()
-btnLogin_Click (in sender : object, in e : EventArgs )
-doError ()

NoteView

#noteviewer : NoteViewControl

-Page_Load (in sender : object, in e : EventArgs )
#OnInit (in e : EventArgs )
-InitializeComponent ()
Entity Forward Engineering Tool

Graphical Component

```
Designer

- components : IContainer
- tronList : ImageList
- classes : TreeView
- properties : ListView
- tbName : TextBox
- tbType : TextBox
- Modify : GroupBox
- c : Hashtable
- columnHeader 1 : ColumnHeader
- columnHeader 2 : ColumnHeader
- currentProperty : string
- currentClass : string
- label 1 : Label
- label 2 : Label
- label 3 : Label
- label 4 : Label
- linkLabel 1 : LinkLabel
- linkLabel 3 : LinkLabel
- linkLabel 4 : LinkLabel
- groupBox 1 : GroupBox
- progressBar 1 : ProgressBar
- label 5 : Label
- btnModify : Button
- btnAdd : Button
- btnLoad : Button
- btnGenerate : Button
- lblRemoveProp : LinkLabel
- pictureBox 1 : PictureBox
- dlgFolderBrowse : FolderBrowserDialog
- dlgOpenLib : OpenFileDialog

+Designer ()
#Dispose (in disposing : bool)
- InitializeComponent ()

- button 1_Click (in sender : object, in e : EventArgs)
- getClasses (in lib : string)
- updateClassList ()
- parseClass (in t : Type)
- classes_AfterSelect (in sender : object, in e : TreeViewEventArgs)
- UpdateList ()
- Form 1_Load (in sender : object, in e : EventArgs)
- properties_SelectedIndexChanged (in sender : object, in e : EventArgs)
- button 2_Click (in sender : object, in e : EventArgs)
- UpdateCurrent ()
- button 3_Click (in sender : object, in e : EventArgs)
- linkLabel 1_LinkClicked (in sender : object, in e : LinkLabelLinkClickedEventArgs)
- groupBox 1_Enter (in sender : object, in e : EventArgs)
- btnAdd_Click (in sender : object, in e : EventArgs)
- linkLabel 2_LinkClicked (in sender : object, in e : LinkLabelLinkClickedEventArgs)
- linkLabel 4_LinkClicked (in sender : object, in e : LinkLabelLinkClickedEventArgs)
- linkLabel 3_LinkClicked (in sender : object, in e : LinkLabelLinkClickedEventArgs)
- enableType ()
- disableTypeForAdd ()
- disableTypeForModify ()
```
**Code Generation**

<table>
<thead>
<tr>
<th>CodeGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>-\textit{p} : PropertyHolder</td>
</tr>
<tr>
<td>-\textit{c} : ClassHolder</td>
</tr>
<tr>
<td>-\textit{tw} : TextWriter</td>
</tr>
<tr>
<td>-\textit{writeFolder} : string</td>
</tr>
<tr>
<td>-\textit{constructorProps} : ArrayList</td>
</tr>
</tbody>
</table>

| +CodeGen (in className : string , in classes : Hashtable , in writeToFolder : string )  |
| +\text{Write} ()  |
| +\text{writeTopBlock} ()  |
| +\text{writeConstructors} ()  |
| +\text{buildLists} ()  |
| +\text{buildMainConstructor} () : string  |
| +\text{writeProps} ()  |
| +\text{prepareSetProperty} (in sb : StringBuilder , in type : string )  |
| +\text{prepareScalarProperty} (in sb : StringBuilder , in type : string )  |
| +\text{prepareEntityProperty} (in sb : StringBuilder , in type : string )  |
| +\text{writeMems} ()  |
| +\text{writeToDTOMethod} ()  |
| +\text{toCamel} (in text : string ) : string  |

<table>
<thead>
<tr>
<th>ClassHolder</th>
</tr>
</thead>
<tbody>
<tr>
<td>+\text{Name} : string</td>
</tr>
<tr>
<td>+\text{Properties} : Hashtable</td>
</tr>
<tr>
<td>+\text{PropertyCode} : string</td>
</tr>
<tr>
<td>+\text{MemberVariables} : string</td>
</tr>
</tbody>
</table>

| +ClassHolder (in name : string )  |

<table>
<thead>
<tr>
<th>PropertyHolder</th>
</tr>
</thead>
<tbody>
<tr>
<td>+\text{PropertyName} : string</td>
</tr>
<tr>
<td>+\text{PropertyType} : string</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DTOGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>-\textit{p} : PropertyHolder</td>
</tr>
<tr>
<td>-\textit{c} : ClassHolder</td>
</tr>
<tr>
<td>-\textit{tw} : TextWriter</td>
</tr>
<tr>
<td>-\textit{writeFolder} : string</td>
</tr>
</tbody>
</table>

| +DTOGen (in className : string , in classes : Hashtable , in writeToFolder : string )  |
| +\text{Write} ()  |
| +\text{writeTopBlock} ()  |
| +\text{writeConstructor} ()  |
| +\text{writeFields} ()  |
| +\text{writeCollection} ()  |
## Mapping Generation

<table>
<thead>
<tr>
<th>MappingGen</th>
</tr>
</thead>
<tbody>
<tr>
<td>-p : PropertyHolder</td>
</tr>
<tr>
<td>-current : ClassHolder</td>
</tr>
<tr>
<td>-classes : Hashtable</td>
</tr>
<tr>
<td>-writeFolder : string</td>
</tr>
<tr>
<td>-xtw : XmlTextWriter</td>
</tr>
</tbody>
</table>

+ MappingGen (in classes : Hashtable , in writeToFolder : string )
+ Write ()
+ writeElements ()
Appendix D – Implementation

Entity Forward Engineering Tool
Work Request Submission - WSDL

```xml
<?xml version="1.0" encoding="utf-8" ?>
xmlns:s="http://www.w3.org/2001/XMLSchema" xmlns:s0="http://tempuri.org/
xmlns:soapenc="http://schemas.xmlsoap.org/soap/encoding/
targetNamespace="http://tempuri.org/" xmlns="http://schemas.xmlsoap.org/wsdl/
<types>
<s:schema elementFormDefault="qualified" targetNamespace="http://tempuri.org/">
<s:element name="InputWorkRequest">
<s:complexType>
<s:sequence>
<s:element minOccurs="0" maxOccurs="1" name="transactionType" type="s:string" />
<s:element minOccurs="0" maxOccurs="1" name="xmlPayload" type="s:string" />
</s:sequence>
</s:complexType>
</s:element>
</s:schema>
</types>
<message name="InputWorkRequestSoapIn">
<part name="parameters" element="s0:InputWorkRequest" />
</message>
<message name="InputWorkRequestSoapOut">
```

72
<part name="parameters" element="s0:InputWorkRequestResponse" />
</message>
<portType name="WorkRequestInputSoap">
<operation name="InputWorkRequest">
<input message="s0:InputWorkRequestSoapIn" />
<output message="s0:InputWorkRequestSoapOut" />
</operation>
</portType>

.binding name="WorkRequestInputSoap" type="s0:WorkRequestInputSoap">
<soap:binding transport="http://schemas.xmlsoap.org/soap/http" style="document" />
<operation name="InputWorkRequest">
<soap:operation soapAction="http://tempuri.org/InputWorkRequest" style="document" />
<input>
<soap:body use="literal" />
</input>
<output>
<soap:body use="literal" />
</output>
</operation>
</binding>
<service name="WorkRequestInput">
<port name="WorkRequestInputSoap" binding="s0:WorkRequestInputSoap">
<soap:address location="http://localhost/WorkStream.Deployment/WorkRequestInput.asmx" />
</port>
</service>
</definitions>
Appendix E – Evaluation

Developer Feedback

Hi Adam,

Find the following feedback below, based on review of the chapters you supplied and viewing of the solution on your machine.

In brief, an impressive effort which met my high expectations from your work for me last year.

The development framework shows a clear, clean design with an easily understandable architecture. The ORM component in particular stood out.

In terms of constructive feedback - the solution is probably better suited to relatively small development projects. The simplicity of the solution in terms of its ‘good to go’ approach and Use Case Flow-Activity fit in fact make it particularly well suited to smaller projects whom wish to adopt a WfMS where they normally wouldn’t have the expertise to do so.

For use within a larger, corporate domain you would need to make several additions such as: distributable components (maybe with some form of load balancing), more power process definitions (which integrate logic to determine activity progression) and ensure thorough component performance testing.

Monitoring/reporting of workflows would be useful.

I hope this is useful. Any queries, let me know. Best wishes in your forthcoming exams - I forward to working with you again in the summer.

Kind Regards,

Gordon

Gordon Eddie
Accenture
UK Delivery Centre - Consulting
Global Delivery Network
gordon.eddie@accenture.com
+44 (0)7966 578767

This message is for the designated recipient only and may contain privileged, proprietary, or otherwise private information. If you have received it in error, please notify the sender immediately and delete the original. Any other use of the email by you is prohibited.