THE USE OF SOFTWARE COMPONENTS TO CONSTRUCT E-COMMERCE WEB-SITES

GRAHAM DUNCAN BELL

submitted in partial fulfilment of the requirements of Napier University for the degree of Bachelor of Engineering with Honours in Software Engineering

School of Computing
May 2001
Abstract

There are many phrases employed to describe business conducted across the Internet, terms such as “E-Commerce”, “E-Business” and “Web-site” are all widely used to indicate some level of presence on the “net”. However, what is meant by these terms – and can they be regarded as referring to the same thing? The project commences by clarifying what meaning should be attached to such terms and demonstrates that while “web-sites” and “e-commerce” represent important steps, it is “e-business” that should be the overall aim of a business that truly wants to maximise the advantages that the electronic age presents.

Retailers are becoming increasingly aware of the opportunities the Internet presents to display their products to a lucrative worldwide market. Having clarified the terms used, the next step was to investigate what stage businesses in the United Kingdom have reached in achieving “e-business” status. This involved, amongst other things, analysis of there present level of Internet usage and also examination of the motivators and inhibitors that influence business commitment towards electronic business mediums in general.

While the larger retailer may be able to justify the initial cost of commissioning a tailor made solution, the small to medium sized retailer may not be in a position to commit the capital necessary all at once. The most important concept that is suggested by “e-business” is that it can be achieved through a staged progression. Most companies are likely to already have computer software packages that they utilise to hold the very information, such as stock levels and pricing, which would be included in a potential e-business. The project investigates how by utilising software components within a technology such as Active Server Pages, not only can existing data stores be incorporated into the overall “e-business” plan, but the software components can also assist in other areas to provide a staged progression that is considered most effective.

After an investigation of the various technologies that are presently available to construct an e-commerce site, a practical example is used to demonstrate how such a site could be developed for a financial institution. The need for generic software components is identified and these are then designed and built to provide the functionality that the site requires.
CONTENTS

1.0 Introduction

1.1 Aims and Objectives 7
1.2 How the Aims and Objectives will be Achieved 7

2.0 Electronic Commerce

2.1 Introduction 9
2.2 Adoption Ladder 10
  2.2.1 Messaging 10
  2.2.2 Online Marketing 11
  2.2.3 Online Ordering 11
  2.2.4 Online Payments 11
  2.2.5 Order progress/Online sales support 12
  2.2.6 E-Business 12
2.3 Adoption 12
2.4 Motivators 13
2.5 Competitiveness 13
2.6 Inhibitors 14
2.7 Conclusions 14

3.0 Theory

3.1 Introduction 16
3.2 Available Technologies 17
3.3 Mark-up Languages 19
  3.3.1 Standardised Generalised Mark-up Language (SGML) 20
  3.3.2 Hypertext Mark-up Language (HTML) 20
  3.3.3 Extensible Mark-up Language (XML) 22
3.4 Active Server Pages 23
  3.4.1 Static Web Pages 24
  3.4.2 Dynamic Web Pages 24
  3.4.3 Contents of an ASP page 26
3.5 Components 27
  3.5.1 Benefits of Using Components 28
  3.5.2 Component Object Model (COM) 29
  3.5.3 Classification of Components 31
3.6 Network Application Architectures 32
  3.6.1 Two-Tier (or Client-Server) Systems 32
3.6.2 Three-Tier Systems  
3.6.3 Location of ASP Pages within Architecture  
3.6.4 n-Tier Applications  
3.7 Conclusions  

4.0 Requirements Analysis and Design  
4.1 Introduction  
4.2 System Specification  
4.3 Requirements Analysis  
4.4 Design of the User Interface  
  4.4.1 The Display Design Rational  
  4.4.2 Gulf of Execution/Gulf of Evaluation  
  4.4.3 Website Storyboard  
4.5 Design of Components  
  4.5.1 Introduction  
  4.5.2 Description of Component Formation  
  4.5.3 Conceptualisation of Component Model  
  4.5.4 Design of Individual Components  
  4.5.5 Order of Execution  
4.6 Error Checking Within Components  
4.7 Conclusions  

5.0 Testing and Validation  
5.1 Introduction  
5.2 Unit or Component Testing  
  5.2.1 White-Box Testing  
  5.2.2 Black-Box Testing  
5.3 Validation Testing  
5.4 Conclusions  

6.0 Implementation  
6.1 Introduction  
6.2 Technologies Employed  
6.3 The Architecture for the E-Commerce Site  
6.4 Component Implementation  
  6.4.1 The Calculator Component  
  6.4.2 The Profile Component  
  6.4.3 The Quote Component  
  6.4.4 The Format Component  
6.5 Demonstration of the Components
6.6 Conclusions

7.0 Conclusions

7.1 Introduction 78
7.2 E-Business 78
7.3 Technologies 78
7.4 Software Components 79

Appendix 1 Storyboard for a Financial Services Web Site 80
Appendix 2 Test Specifications for Testing Components 83
Appendix 3 Component Parameters and Functions 109
Appendix 4 Error Checking Within Components 113
Appendix 5 Component Implementations 117

List Of Tables 124
List Of Figures 125
List Of Equations 127
References 128
Bibliography 130
Acknowledgements

I would like to acknowledge my Project Supervisor, Dr. William Buchanan for his support and guidance throughout the year.
1.0 Introduction

1.1 Aims and Objectives
The Internet is increasingly being seen as a medium where business transactions can be undertaken. This has led to businesses, which would not traditionally have been expected to maintain a substantial presence on the Internet, recognising the sales potential that the Internet offers. A business that has a presence on the Internet has a potential worldwide customer base.

However, this increase in interest from commercial organisations has led to an ever-increasing demand for more and more functionality to be provided within web sites and one of the effects of this is that web sites are increasing in complexity. Furthermore, as competition to attract customers to their sites increases, businesses expect web sites to be updated and amended as quickly as possible. Any delay is seen as a loss of potential sales.

It is against this background this report will investigate the following:

- The expansion of the Internet has resulted in the use of new terms such as “e-commerce” and “e-business”, however, what is the meaning of these terms and how do they differ from a traditional web site?
- The complexity of web sites has increased; can software components be used as a way to encapsulate functionality into separate pieces of business logic?
- Can software components be designed in such a way that they are generic and thus reusable in other applications?

1.2 How the Aims and Objectives will be Achieved
The report is split into seven chapters, including this one, and these will be used to examine the issues raised above.

Chapter 2 outlines terms such as “e-commerce” and “web site” and how these fit into the overall concept of an “e-business”. This will involve an investigation into a recent Department of Trade and Industry (DTI) report entitled “Business In The Information Age, International Benchmarking Study 2000” [1]. This will provide a clearer understanding of what is required from an “e-business” and thus give an overview of the setting in which the software components will be required to operate in.

Chapter 3 is concerned with the various technologies that are currently available for use in constructing web sites. It supplies a broad overview of a number of technologies and then goes on to describe in greater detail the technologies that will be used within the remainder of the project. Among the areas covered will be an investigation of software design, software architectures and also various issues relating to component construction.
Chapter 4 discusses the requirements analysis and design for a potential web site. It then goes on to look at the specific requirements of a particular set of four components that will provide part of the functionality of the site. Also, it will demonstrate that with careful consideration to the design of components, they will be able to be combined in different configurations to produce the desired functionality in different parts of the site. In addition, the functionality can be expanded with the addition of further components in the future.

Chapter 5 investigates the methods available to test and validate components and defines a test plan for the four components that have been implemented.

Chapter 6 is concerned with the implementation of the design and it begins with a look at the various technologies that will be employed. It then details the architecture of the site before detailing the implementation of the individual components. The chapter finishes with a description of how the four components can be combined together to provide the functionality that allows a visitor to a web site to obtain a quotation for a product such as a loan. It highlights the need to provide error checking in the correct component and that how error codes can provide developers with assistance in tracing errors.

The final chapter provides a summary of the conclusions reached after carrying out the project.
2.0 Electronic Commerce

2.1 Introduction
Prior to going into detail of the various technologies and techniques that can be employed to implement an e-commerce strategy, it would be helpful to define what e-commerce is understood to be and to take a brief look at both the progress companies in the United Kingdom are making to achieving e-business status and also what their attitudes are towards e-business. For the purposes of this project, the definition that has been chosen is one that describes e-commerce in its widest terms. The reasons for using this definition is that it suggests a longer-term strategy that involves the business as a whole and also it promotes the notion that e-commerce can be achieved using a staged development program.

The definition and the other information in this section are based on the “Business In The Information Age, International Benchmarking Study 2000”, which is a report commissioned by the Department of Trade and Industry (DTI). It defines electronic commerce as “any business process carried out over an electronic network, such as exchanging data files, having a website, using other businesses’ websites or buying and selling goods online.” [2]

The survey is based on the results from nearly 6,000 telephone interviews with businesses of all sizes. The businesses were grouped into the following classifications to reflect all sizes of businesses:

- Micro businesses 0-9 employees.
- Small businesses 10-49 employees.
- Medium businesses 50-249 employees.
- Large businesses 250+ employees.
2.2 Adoption Ladder
The report identifies the steps that a business can take to achieve true e-business status. These steps are described as an “adoption ladder”, which provides a “stepped” structured approach of progression, as illustrated in Figure 1.

![Adoption Ladder Diagram]

Figure 1. Adoption Ladder

2.2.1 Messaging
This is the use of electronic mail (e-mail) to send text messages, either to provide internal communication between staff, or to allow communication between businesses and their suppliers and customers. In e-commerce, the focus is on the external use of e-mail. Some elements of electronic data interchange (EDI) can also be regarded as messaging, such as sending a request for a quotation:

- **Use of PCs** - The primary access device to e-mail, Internet and other network-based services is currently the PC. Currently, almost every business has at least one computer and over 90% of businesses have employees making use of PCs on a daily basis [3].

- **E-mail** - Electronic mail can be used both internally to share information between employees, and also externally to exchange information with other businesses and with customers. Almost 90% of businesses use e-mail (either internal or external) on at least a daily basis. There was no dominant single reason for businesses not using e-mail, and businesses will react to customer and supplier demand for e-mail connectivity [4].

- **Internet access** – The use of the Internet by businesses is widespread at 90%, however, only just over 30% of employees actually use the Internet on at least a monthly basis, which would appear to suggest that the existence of an Internet connection does not necessarily equate to frequent use [5].
Network access - In order to be online, a business must be able to receive and transmit data along the network. A key measure in comparing the efficiency of different methods of transmitting data is the speed of transmission.

Increasingly businesses are enabling staff, such as sales managers and technical support staff, to access e-mail and other applications when away from the office and when working at home. It is now relatively easy to forward e-mail messages to a web e-mail box for access over the Internet. Almost 60% of businesses using basic connectivity technologies provide remote access to their employees [6].

2.2.2 Online Marketing
This is the establishment by a business of a website or e-mail list to publish information about products and services, so that customers can access this information online. The website can publish marketing information, prices, and stock levels. Checking the availability of products and services can also be done online.

It is an important step towards e-enabling a business’s customer-facing activities, enabling a higher level of interaction between a business and its customers. The level of adoption of marketing websites by businesses is just under 70% with almost 80% of such businesses publishing most of their marketing material on it [7]:

- Updating website content - It is important that a website is kept up-to-date, rather than updated frequently *per se*. The frequency with which a website is updated will depend on the rate of change of the information published on the site. So data relating to the frequency with which websites are updated needs to be viewed in this light. However, such data can be useful in identifying the level of commitment to keeping information on a website current.

Currently, 40% of businesses update their website at least weekly, however, updating websites at longer intervals may be sufficient. The important principle is that a customer should have confidence in the reliability of the information on the website [8].

- Intention to set up website – Of the 34% of businesses that do not presently have a website, almost a third of these are in the process of setting one up [9].

- On-line availability checking - The provision of a capability for customers to check the availability of a product online is not a facility that is currently widely offered with only just over 20% of businesses currently offering this facility [10].

2.2.3 Online Ordering
This is the online interaction between a business and its customers, or a business and its suppliers, for the placement of an order.

- On-line ordering - Online ordering can be accomplished by sending an e-mail or through a website. Almost 30% of businesses allow their customers to order on-line, 70% allow this via e-mail and almost 60% offer ordering directly through the website [11].
• **Value of on-line orders** - An important measure of the extent to which customers are making use of e-commerce is the value of the orders that they place online. Overall, it would seem that although companies are offering the ability to order online, uptake by customers forms only a small part of overall purchasing behaviour.

• **On-line invoicing** - Overall the level of online invoicing is quite low, with fewer than 10% of businesses providing customers with online invoices [12].

2.2.4 **Online Payments**

A transaction can be split into two separate online activities, firstly, issuing or receiving an invoice and secondly, its completion online by an electronic payment. This can be undertaken through the use of debit and credit cards, electronic cash, electronic funds transfer, or through an EDI service.

At present, just over 10% of businesses allow their customers to make payments online. Of these, nearly 60% accept cards, and almost 70% use electronic funds transfers [13].

• **Value of on-line payments** - The majority of businesses that offer their customers the ability to pay online estimate that around one-fifth of there customers’ purchases were paid for online [14].

2.2.5 **Order progress/Online sales support**

This involves the use of e-commerce to support the business relationship between a customer and a supplier, for example through the provision of interactive order progress tracking or online post-sales support.

• **Post-sales support** - The purchasing experience does not end with a purchase being made, and post-sales support is seen as an important function that can be provided through e-commerce activity. Post-sales support can be provided by e-mail, through a website, or through an Extranet.

The percentage of businesses that offer customers the facility to track the progress of orders online is low with fewer than 10% of businesses offering this capability [15].

2.2.6 **E-Business**

The previous steps describe online activities between businesses and customers and suppliers. The theoretical end point comes when a business has fully integrated Information and Communication Technologies (ICT) into its operations, potentially redesigning its business processes around ICT or completely reinventing its business model. Step 6, which the benchmarking report, labels E-Business, is understood to be the integration of all these activities with the internal processes of a business through ICT.

2.3 **Adoption**

Attitudes of the businesses towards e-commerce are now explored, in order to gain an understanding of the motivators, and barriers, to the adoption of e-commerce.
At the present moment very few businesses are totally reliant on e-commerce – less than 3% [16]. However, the majority have implemented some elements of e-commerce alongside existing business methods.

While 70% of businesses have implemented some form of e-commerce this means that 30% of businesses have no e-commerce activity at present, although almost all intend to use e-commerce in the future. While a future intention is no guarantee that a business will in fact develop e-commerce activities there is clearly a strong overall commitment in the UK towards e-commerce. Taking the two figures together, 94% of UK companies either use or intend to use e-commerce – only 6% have no intention to do so [17].

2.4 Motivators
The study looked at two categories of motivators towards e-commerce in the context of a business strategy.

The first category is based on cost/efficiency drivers, which are:

- Increasing speed with which supplies can be obtained.
- Increasing speed with which goods can be despatched.
- Reduced sales and purchasing costs.
- Reduced operating costs.

The second category of drivers is related to the competitive position of the business. The competitiveness drivers are:

- Customer demand.
- Improving the range and quality of services offered.
- Avoid losing market share to businesses already using e-commerce.

In general, businesses placed slightly higher emphasis on the competitiveness drivers when compared to the cost-effectiveness drivers. This could indicate that businesses recognise that the potential benefits of e-commerce are longer-term [18].

2.5 Competitiveness
The study also sought to assess perceptions of both the current and future importance of e-commerce to their own business’ competitiveness. Fewer than half of businesses regard e-commerce as currently being very important, however, almost three quarters see it as more important in three years time than at present. In the UK, 73% of businesses regard e-commerce as very important/essential to their competitive position [19].
2.6 Inhibitors
Although there are some strong drivers to the development of e-commerce, there are also a number of inhibiting factors that need to be taken into account. Inhibitors could include the following:

- No tangible benefits.
- E-commerce is not relevant to the business.
- The technology costs are too high.
- Concern about fraud.
- Concern about confidentiality.
- Telecommunications infrastructure.
- Credit card clearing.

Across all sizes of business the main inhibitors for users are concerns about confidentiality and about fraud [20].

2.7 Conclusions
This report shows that while very few businesses are totally reliant on e-commerce, 70% do make some use of e-commerce. Of the remaining businesses, 80% of them said that they intended to move into e-commerce within the next 12 months. Overall the motivators towards initiating or raising the level of e-commerce seem to be mainly competitiveness drivers, rather than cost-efficiency drivers. In other words, businesses seem to have decided that for strategic reasons, and in particular the need to remain competitive in global markets, they have to trade through e-commerce, and not try to justify the investment on a profit basis. The main inhibitors are concerns about confidentiality and fraud.

The “adoption ladder” clearly demonstrates the longer-term approach that businesses require to take if they wish to be classed as fully e-business. This type of staged progression lends itself to a software component type solution. A system of components would enable a business to expand their system gradually through the controlled addition and replacement of components. The benefits of this approach would include:

- It aids the business in establishing an overall plan for achieving total “E-Business” which can be paralleled with the DTI’s Adoption ladder.
- It should reduce the cost of progression towards e-business as existing components can be reused. Other components can then either be purchased from within the software market-place or developed in-house, and then be integrated into the existing framework to provide the desired functionality.
• It provides the business with the possibility of utilising existing systems to achieve a particular step, for example, the business may already have a computerised database system for maintaining stock levels and product prices. This system could be incorporated into the e-business by using a particular component to connect it to the other systems. Thus information already held in one place will become available throughout the business.

• It will assist in gaining staff acceptance of changes in working practices. This will be achieved by the use of a core framework which will be expanded as new systems are incorporated into the e-business. This leads to users of the systems becoming familiar with the system as it grows and gaining confidence in the benefits that new additions to the system will provide based on past experience of using the system.

• If more secure or efficient components become available then they can be integrated into the system to improve security, without changing the remainder of the system.


3.0 Theory

3.1 Introduction
The previous chapter introduced the concepts of “e-business” and “e-commerce”. In this chapter the various technologies and methodologies that can be combined to achieve the goal of becoming an “e-business” are examined. This will be achieved by initially briefly describing various options and then the technologies used within this project will be described in greater detail. Areas that will be covered in some depth include, Active Server Page (ASP) technology, the theory of using components within web site design and a discussion of the development of component architectures. The chapter will finish with a summary of the conclusions drawn from this investigation.

Figure 2 shows that a true “e-business” has integrated all of its business systems allowing all of the businesses interested parties to conduct their transactions electronically, for example, customers can gain access to details of products and orders placed, suppliers can be contacted to order new stock and employees can access accounting information. Such transactions are passed between the business and its customers, suppliers and employees via browsers which are pieces of software located on the users computer. The information is sent between the browser and the central server, via either an internal or external network. The server, which is likely to be located at the business premises holds the web pages that contain the information that will be displayed on the users screen, together with the components which provide the functionality to the web pages, for example to carry out a calculation on data input by the user. Access to persistent data stores can be gained through the use of specialised software components. These stores may or may not be located on the same computer as the server is located on.

![Figure 2. E-Business Layout](image-url)
3.2 Available Technologies

The technologies available for constructing an e-commerce site are numerous and varied. In this section the main alternatives have been grouped together within general headings that attempt to demonstrate the role that they fulfil within the overall construction of the site using important criteria such as:

- **Security** - This covers the transfer of data and information between the client and the server.

- **Data Storage** - The data store is the storage area for persistent data and traditionally it has taken the form of a database.

- **Web Browser Language** - This is the language used within the web pages that the client browser turns into display for the screen.

**Hypertext Mark-up Language (HTML)** - HTML is a system for describing the content and appearance of documents and is known as a page description language. It can be seen as the “glue” that holds Web pages together and almost all current Web pages are built using it. An HTML file contains the text that appears on the page, together with the embedded codes that control other aspects such as how the text looks and what else appears on the page. Images, sounds, video clips, programs and other elements can be specified within the HTML, and the browser will load these files too, weaving them into the text as appropriate.

**Extensible Mark-up Language (XML)** - File incompatibility is a problem that XML seeks to solve. For example, at present, data created in one program may not be transferable to another program, as the second program does not understand the file formats the source program knows. Furthermore, old files created in propriety formats become unusable once the programs that created them become defunct. In the world of long-term document archival, this presents problems that may lead to fewer and fewer people being able to access such archive files. The solution has been to use Standard Generalised Mark-up Language (SGML), which allows a program to save data in a standardised, structured, vendor-independent way. XML is a simplified SGML that has been developed for use on the web.

- **Dynamic Web Pages** - These are the technologies that allow scripting language code to be intermingled within a web page containing HTML or XML.

**Active Server Pages (ASP)** – Embedded in the HTML are scripts that run on the server when the page is requested. The scripts deal with user input from forms and generate content. So an ASP is really a web page template that executes code to fill itself in. ASP scripts are not seen by the browser as the server-sided scripting is replaced by generated HTML prior to it being sent.

**Java Server Pages (JSP)** - Allows Java code to be intermixed with HTML and also to call upon the services of JavaBeans. JavaBeans are packaged Java program modules analogous to ActiveX controls. A Bean is actually a type of Java class file that is written in a way that allows it to be manipulated, through its properties and methods, as an object.
JSP look very similar to ASP pages and they both utilise the `<% and %>` tags to enclose server side code. The main reasons for running JSP with IIS, when ASP comes as part of that package, is that JSP is less manufacturer dependent and therefore provides the flexibility of being able to be migrated to non-Microsoft platforms in the future.

While JSP pages are ultimately compiled into servlets, they are different from Java servlets. Java servlets are used in CGI programming, see page 19 below, where a Java-authored program module generates the entire HTML for a Web page. This means that the programmer has to write an output statement for every piece of HTML. It also means that the code is not easily reusable and if the page design changes, the code must be altered. JSP reduces the amount of coding as the static portion of HTML is already in place and can be altered without necessarily leading to a change in the code.

**Personal Home Page (PHP)** – This is intended for use in an Active Server Page type model where server-side scripts are embedded in HTML files. It resembles the C programming language with object oriented features and elements taken from Perl, see page 19 below, and Java added. Written with web development in mind, it is a scripting language that operates under a run-time compiler and does not require Microsoft or Java technologies to use it.

- **Scripting Language** - These are contained within the web browser language and enable dynamic web pages to be created.

  **JavaScript** - JavaScript’s syntax was modelled on Java’s, giving JavaScript its name; however, despite this the two are entirely different. Sun invented Java with JavaScript being invented by Netscape. Java is used to write self-contained applications, while JavaScript is used to extend HTML enabling interactivity and computational power to be added to Web pages.

  Another difference between Java and JavaScript is that Java arrives as a byte code package separate from the HTML of the Web page, and is sent to the Java Virtual Machine (JVM) for execution. JavaScript is embedded in HTML as plain text between `<script>` and `</script>` tags, or downloaded as a plain text source file if the code is used by several pages around the site. It is interpreted by a scripting language that has nothing to do with the JVM. JavaScript can either be run on a server (server-side scripting) or as an extension of HTML that runs from the browser on the client (client-side scripting).

  **VBScript** - As well as JavaScript (JScript), Internet Explorer also supports VBScript. VBScript works very much like JScript in that scripting code is embedded in the HTML. VBScript is capable of the same kinds of things as JScript; the main difference between the two is that VBScript is a member of the Microsoft Visual basic family. A possible problem with this is that Microsoft products mainly support VBScript and this means that while scripts will execute perfectly well in Internet Explorer, the page may not work in other browsers. However, this does not mean that VBScript can never be used to construct pages that will be viewed on the Web. For example, VBScript is used in Microsoft
server products and the brand of browser used by the visitor does not matter if the VBScript is running server-side and generating customised HTML that does not contain VBScript.

- **Common Gateway Interface (CGI)** – This is the traditional system for building dynamically generated web pages. Several languages can be plugged into CGI but Perl is probably the most popular. In CGI a browser request can be delegated to a self-contained program or script. HTML generated this way is the output from the program, with the equivalent of a ‘PRINT’ statement producing every line on the page.

**Practical Report and Extraction Language (Perl)** – This was originally written as a Unix-based tool for producing reports. Perl took off as a niche product and several iterations later, it became a general purpose language available on other systems including Linux and Windows. Version 5 was the first version with object-oriented features.

Having provided a brief overview of the available technologies, a more detailed description will now be given of the technologies that it is envisaged will be utilised to construct the e-commerce site.

### 3.3 Mark-up Languages

A mark-up refers to anything put on a document that adds special meaning or provides extra information. For example, highlighted or bold text is a form of mark-up. To convey the meaning of the mark-up, a set of rules must be stipulated that covers both what constitutes mark-up and also exactly what the mark-up means.

**Characteristics of Mark-up Languages**

Mark-ups can be classified as one of four types:

- **Stylistic Mark-up** – This indicates how the document is to be presented. When bolding or italics is used on a word processor it is stylistic mark-up. In Hypertext Mark-up Language (HTML) the `<FONT>`, `<I>`, `<B>` and `<U>` tags are all stylistic.

- **Structural Mark-up** – This conveys how the document should be structured. The `<Hn>` where n is a number, `<P>` and the `<DIV>` tags are examples of structural mark-up, which indicate a heading, paragraph and container section respectively.

- **Semantic Mark-up** – This is used to relay details of the content of the data, for example, `<TITLE>` and `<CODE>` are semantic mark-up in HTML.

- **Functional Mark-up** – This adds functionality to the data that is marked-up, such as hyper-links and pointers, sound files etc.

Mark-up languages define the mark-up rules that add meaning to the style, structure and content of documents. They are the grammar and syntax which specify how a language should be ‘spoken’. An example is HTML, which is a mark-up language that enables documents to be written for display on the web.
Tags and Elements
Tags are the angled brackets, known as delimiters, and the text between them. For example some of the tags used in HTML are:

- `<P>` is a tag that marks the beginning of a new paragraph.
- `<I>` is a tag indicating that the following text should be rendered in italic type.
- `</I>` is a tag that indicates the end of a section of text to be rendered in italic type.

Elements refer to the tags plus their content, so the following is an example of an element:

- `<B>Here is some bold text</B>`

In general terms a tag is a label that tells a user-agent, such as a browser, to do something to whatever is encased in the tags.

Attributes
Any tag can have an attribute as long as it is defined. They take the form of name/value pairs, in that the element can be given an attribute, with a name, and the attribute must carry a text value surrounded by quotation marks. They take the form:

```
<tagname attribute="value">
```

### 3.3.1 Standardised Generalised Mark-up Language (SGML)

This is a mark-up language that is used to create other mark-up languages, with the most famous being HTML. HTML is known as an application of SGML; the difficulty with using SGML in itself is that it is very complicated. Extensible Mark-up Language (XML) is a simplified version of SGML that retains much of SGML’s functionality and is designed for use on the web.

As a language SGML is very powerful, but with its power comes complexity, and many of the features are rarely used. It is very difficult to interpret an SGML document without the definition of the mark-up language, which is kept in a Document Type Definition (DTD). The DTD is where all the rules for the language are kept and it has to be sent with, or included in, the SGML document so that the custom created tags can be understood. SGML has tended to be adopted in industries where large amounts of documentation have to be marked up.

### 3.3.2 Hypertext Mark-up Language (HTML)

HTML was originally an SGML application. Its purpose is to describe how information is to be prepared for the World Wide Web. HTML is just a set of SGML rules and as such it also has a DTD. In fact there are several DTDs; ones for loose and tight structured HTML, ones for different versions and so on. HTML is far simpler than SGML and is a fraction of its size. This has made it easier to learn and therefore more popular and widely adopted by all sorts of people.

HTML uses a protocol called HTTP (Hypertext Transfer Protocol) to transfer information across the Internet. It is one of a number of protocols used on the Internet, which are collectively known as the Internet Protocol Suite. HTTP has gained an edge over other protocols due to its relative ease with which it can be used to retrieve other documents. The combination of an easy to use protocol and a simple to learn language is an attractive proposition that has ensured the rapid spread of systems implementing HTML and HTTP.
Historically, as HTML usage increased and web browsers started to become readily available, non-scientific users started to create web pages. These non-scientific users became increasingly concerned with the aesthetic presentation of the material. This led the manufacturers of browsers to start offering different tags that allow web page authors to display their documents with more creativity than was possible using plain ASCII text.

With the new tags came new problems - different browsers implemented the new tags inconsistently. This has led to sites that state, “Best viewed through Netscape Navigator” or “Designed with Internet Explorer”. In addition, users now expect to be able to produce web pages that resemble documents created on the most sophisticated desktop publishing system.

The browser’s potential as a new application platform was recognised and web developers started creating distributed applications for businesses, using the Internet as a medium for information and financial transactions.

Users now want HTML to fulfil an ever-increasing variety of new and more complex tasks, which has led to its weaknesses becoming more apparent:

- **HTML has a fixed tag set** – Tags cannot be created that can be interpreted by others.

- **HTML is a presentation technology** – It does not carry information about the meaning of the content held within its tags.

- **HTML is “flat”** - You cannot specify the importance of tags, so a hierarchy of data cannot be represented.

- **Browsers used as an application platform** – HTML does not provide the power needed to create advanced web applications at the level at which developers are currently aiming. For example, it does not readily offer the ability for advanced retrieval of information from documents marked up in HTML and, because the text is only marked up for display, it is not easy to process the data within the document.

- **High traffic volumes** – HTML documents that are used as applications clog up the Internet with high volumes of client-server traffic. For example, sending large general sets of data across a network when only small amounts are required.

HTML is a very useful way of marking up documents for display in a web browser, however it tells the user very little about its actual content. For most documents to be useful in a business environment, there is a need to know about the content of the document. With this information it then makes it possible to perform generalised processing and retrieval on that file. It is then no longer only suitable for one purpose; rather it can be used for display in a web page and also be used as part of an application. Marking up data in this way conveys its content and thus makes it self-describing and this leads to it being re-useable in different situations. SGML made this possible, but it is now also possible with XML; which is far simpler to use.
3.3.3 Extensible Mark-up Language (XML)

The browser manufacturers had no intention in fully supporting SGML and its complexity prevented many people from learning it. So, a simplified version for use on the web was created, thus signalling a return to documents being marked up according to their content. This allowed developers to create their own tags and mark-up languages so that they could mark-up whatever they wanted; however they wanted, with the intention of making it self-describing.

XML got its name because it is not a fixed format like HTML, users can create their own tags so that they actually describe the content of the element.

XML in itself is just a way of describing the data, so there are several other things that a developer needs to know in order to use XML productively and these include:

- **Schemas** to define what each tag actually means.
- **Style sheets** for presenting the XML in an attractive way.
- **Linking rules** since XML has no built in hyper linking mechanism.

**Benefits of using XML**

Information is increasingly being stored electronically and the tendency has been to think of the majority of such business data as residing in relational databases such as SQL Server, Oracle or DB2. However, in reality, data is stored in a number of formats:

- Quotes and reports in word processor formats, such as Word or Word Perfect.
- Web pages in HTML.
- Presentations in PowerPoint.
- Mail and memos in mail servers such as Microsoft Exchange and Lotus Notes.

The ubiquity of the Internet means that more and more of this data is being shared between people in different physical locations. Figure 3. demonstrates how XML allows this to be achieved efficiently with the data only being stored once and then reformatted to meet the particular requirements of a user application. It also allows the data content to be examined and for example, particular elements of the data can be retrieved as required, thus reducing the amount of data transmitted across a network.
Style Sheets
This means using a separate language, in a separate file, to declare how the document should be presented, which can be achieved using Cascading Style Sheets (CSS). The advantages of using style sheets are:

- Improved clarity of the document.
- Can help to reduce download time, network traffic and server load.
- Allows the presentation of several files to be changed by altering one style sheet.
- Allow the same data to be presented in different ways for different purposes.

With style sheets all of the rules are kept in one file and the source document simply links to this file. This means that if several pages use the same type of display then the style rules do not need to be repeated on each page. Furthermore, if the style of the site needs to be changed, it is only the style sheets that are changed and not each page. The same data can be displayed in different ways for different purposes by applying different style sheets.

3.4 Active Server Pages (ASP)
ASP is a Microsoft technology for creating dynamic web pages. It provides the functionality of a programming language that allows program code to be written that will dynamically generate HTML for a web page. In practice this means that when a client browses to a web site and requests an ASP page, the web server processes the ASP code at that time. This processing generates the HTML, which is then passed to the browser and used to create the page itself on the client machine.

Two important characteristics of ASP can be drawn from this: firstly, the HTML is not created until the user wants to see the web page (it is created dynamically), and secondly, as the processing takes place on the web server, the generated HTML can be displayed using any modern browser. ASP code is executed on the web server and
generates pure HTML; therefore the client machine does not need to provide any kind of ASP support.

ASP is not a language in the sense that Pascal and C++ are languages, although it does make use of existing scripting languages such as VBScript or JavaScript. Neither is it an application in the sense that Microsoft FrontPage and Microsoft Word are applications. Instead, it is described in a more ambiguous term as a ‘technology’ for building dynamic and interactive web pages.

To understand the benefits of utilising a technology such as ASP to produce dynamic web pages, it is helpful to briefly examine the differences between static and dynamic web pages.

### 3.4.1 Static Web Pages

A static web page is a page whose content consists of some HTML that was typed directly into a text editor and saved as an ‘.htm’ or ‘.html’ file. The author of the page has already completely determined the exact content of the page in HTML at some time before any user visits the page.

Figure 4. demonstrates the route that static, pure-HTML pages follow to appear on a client browser:

1. A web author writes the page of pure HTML and saves it within an ‘.htm’ file.
2. Sometime later, a user types a page request into their browser, and the request is passed from the browser to the web server.
3. The web server locates the ‘.htm’ page.
4. The web server sends the HTML stream back across the network to the browser.
5. The browser processes the HTML and displays the page.

![Figure 4. Static Web Page Generation](image)

### 3.4.2 Dynamic Web Pages

In dynamic web pages, part of the HTML is replaced with a set of instructions that will be used to generate HTML for the page at the time the user actually requests the page. So the page is created dynamically on request, this adds an additional step to the
process described above for static web pages. Figure 5. demonstrates what this now becomes:

1. A web author writes a set of instructions for creating HTML, and saves these instructions within a file.

2. Sometime later, a user types a page request into their browser, and the request is passed from the browser to the web server.

3. The web server locates the file of instructions.

4. The web server follows the instructions in order to create a stream of HTML.

5. The web server sends the newly-created HTML stream back across the network to the browser.

6. The browser processes the HTML and displays the page.

So the process of serving a dynamic web page is only slightly different from the process of serving a static web page. However, the difference is crucial in that the HTML that defines the web page is not generated until after the web page has been requested. This allows web page authors to capture various pieces of information that are not known at the time the instructions are written, for example:

- The user’s identity and personal preferences.
- The type of browser the user is using.
- Other information provided by the user request.
- Information contained in databases, text files, XML files, etc.

The HTML-generation instructions can be written in such a way that they use this newly captured information to create up-to-the-minute, personalised, interactive web pages, which serve fresh information every time they are requested.
### 3.4.3 Contents of an ASP page

An ASP page is likely to be composed of a combination of three types of syntax: some parts ASP, some parts HTML tags and some parts pure text. The constituent parts of the ASP page are saved together in a file with an ‘.asp’ extension and Table 1 summarises the three ingredients together with their purpose and appearance:

<table>
<thead>
<tr>
<th>Type</th>
<th>Purpose</th>
<th>Interpreter</th>
<th>Hallmarks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Text</td>
<td>Hard-coded information to be shown to the user.</td>
<td>Viewer’s browser on their PC shows the text.</td>
<td>Simple ASCII text.</td>
</tr>
<tr>
<td>HTML tags</td>
<td>Instructions to the browser about how to format text and display images.</td>
<td>Viewer’s browser on their PC interprets the tags to format the text.</td>
<td>Each tag within &lt; &gt; delimiters.</td>
</tr>
<tr>
<td>ASP statements</td>
<td>Instructions to the web server running ASP about how to create portions of the page to be sent out.</td>
<td>Web site host’s web server software with ASP extensions performs the instructions of the ASP code.</td>
<td>Each ASP section contained within &lt;% %&gt; delimiters.</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>ASP statements contain the structures of more 'traditional' programming languages, such as Visual Basic and Java, as they have features such as variables, decision trees, cyclical repetitions etc.</td>
</tr>
</tbody>
</table>

**Table 1. Contents of an ASP page**

ASP is processed on the web server to generate HTML, while HTML is processed solely on the browser. The advantages of performing actions on the web server first are:

- It allows programs to be run in programming languages that are not supported by the client browser.
- It enables the development of dynamic web applications that are browser independent and without recourse to client-side programming features such as Java applets, Dynamic HTML, ActiveX controls, all of which are browser specific.
- It can be used to provide the client browser with data that does not reside at the client.
- It often makes for quicker loading times than with client-side dynamic web technologies such as Java applets or ActiveX controls, because the client is actually only downloading a page of HTML.
• It provides improved security, since the ASP code is never viewable from the client browser. For example, this stops clients viewing details that the author wants to keep secret:
  
  o ASP pages that use a database query to get some data require code to connect to the database. These connection details may contain sensitive information such as a username and password.
  
  o An ASP page may contain an algorithm to calculate an insurance quote.

However, a disadvantage of using server side technology is that it increases the workload on the server, which could lead to the host having to upgrade their hardware as the numbers of visitors to their site increases.

**The resources needed to run ASP**
The resources necessary to facilitate web pages containing ASP to be written, and then published on a web server are as follows:

• The pages can be written using either a simple text editor, such as notepad or by using a specialised web development tool. Visual InterDev (VI) comes as part of Microsoft’s suite called Visual Studio. VI is a tool for designing dynamic web applications and is a collection of useful tools and utilities. Microsoft FrontPage 2000 is part of Microsoft’s Office 2000 suite and is another tool for creating and designing web pages, but it does not offer all of the functionality of VI.

• In order to publish the pages, a web server that supports Active Server Pages is required. The latest version of ASP is version 3.0 and this comes with a web server called Internet Information Server 5.0 (IIS 5.0), which in turn is part of the Windows 2000 operating system.

However, for the purposes of this project, the version of ASP that will be used is 2.0, which comes with the previous version of Internet Information Server 4.0 (IIS 4.0). These are the latest versions of ASP and IIS that allow the continued use of the Windows NT 4.0 operating system. Another alternative would be to use Microsoft’s Personal Web Server as this also supports ASP 2.0 and can be run on Windows NT 4.0, Windows 98 or Windows 95 operating systems.

• The completed web pages can be viewed and tested within a web browser. ASP is processed on the server; therefore any web browser should be suitable for viewing the pages.

### 3.5 Software Components
A software component (component) can be thought of as a small, self-contained piece of code that sits at a well-defined point in the system architecture and performs a certain range of related tasks. The theory to using components within software is similar to the approach used to tackle any complex problem: firstly, break up the problem into smaller parts, or components, each of which is self-contained and can be easily understood. Secondly, solve each of these smaller parts or use a ready-written solution and then thirdly, assemble the smaller parts together to create the overall solution.
Components, once written, only require a small amount of application ‘glue’ to join them together. The ‘glue’ calls the components into being, provides them with the information they need to perform their tasks, and then awaits the results. In this sense, components can be seen as ‘black boxes’ of functionality. Users do not need to know how each component achieves its results; they only need to know what functionality it offers and what input it requires to achieve it.

3.5.1 Benefits of Using Components
The main advantages of packaging functionality into components rather than using script are as follows:

- **Code Reuse and Distribution**
  Components allow complex applications to be broken up into manageable chunks as mentioned above. The resultant chunks can be packaged into components that offer common functionality that can be reused more than once.

  When code is packaged into a COM component, see section 3.5.2 below, this provides a clear definition of precisely how to use the functionality of that component, with the specifics contained within the relevant documentation. This also assists in ease of distribution of the component as users do not need to understand the code behind the component to get it to work – they only need to know what functionality it performs together with what input it requires. COM components are language neutral - they can be written in one language and then be used in another.

- **Maintenance**
  Assuming that the replacement component provides at least the same functionality as the previous one, then components can be replaced without disrupting the rest of the application. This means that, for example, if a component is produced to perform a repetitive task such as form validation, it can be reused throughout the application, then if subsequently a problem with the validation code comes to light, the faulty component can be replaced with a suitably amended one.

- **Commercially Available Components**
  There are an ever-growing number of components that are available commercially. This means that if a developer has a programming task to achieve; it is quite possible that there is already a component in the market place that will help provide a solution. The benefit of buying a third-party component is that the user does not need to research, write and test the component. This in turn should lead to shorter development times as the component can be used immediately; in addition, the cost of development and testing has been replaced with a fixed-price for the pre-written code. Highly specialised components tend to require consultation with specialists in the field and again this cost is avoided by utilising pre-written components.

- **Performance Advantages**
  Scripting languages that are used within ASP pages are known as interpreted languages. It takes longer to execute complicated pieces of code using these languages compared to compiled languages. Components are usually already
compiled, which means that they have already been converted into binary format and can therefore be executed straight away. The result is that components often execute much more quickly than plain scripts do.

• **Hiding Sensitive Code**
  Code that is distributed can be read and understood by the recipient and then subsequently changed by them. It may be the case that the developer wants to hide the code; this can be achieved by using compiled components as this produces a binary representation, which protects the source code, thus secures the business logic contained in the code.

• **Splitting Tasks Into Distinct Areas**
  Overall application development time can be reduced as a result of splitting the application into discrete chunks that can be developed independently.

• **Ease of Debugging**
  The Script debugger is a useful tool, however, it is not nearly as sophisticated as those available for Visual Basic and Visual C++. For example, the debugger in Visual C++ allows the user to examine the processes right down into the computer’s memory to see what is being stored where.

### 3.5.2 Component Object Model (COM)

COM is Microsoft’s favoured solution for solving the complex problems relating to large applications and code reuse on the Windows platform. It provides the necessary framework for creating and using components. The foundation of COM is a binary specification that defines how the code using a component – usually called a client – can use the functionality that the component makes available. It also defines how a component exposes that functionality to a client. Because COM is a binary specification, it allows language neutrality. As long as a given language can produce compiled code that compiles with the binary specification, it can then be used to write COM components. Furthermore, it will also be able to interact with COM components written in any other COM enabled programming language. Suitable languages include: Visual Basic, C++, Visual J++, SmallTalk and Delphi. CORBA and JavaBeans are two other specifications for components, however, as this project is mainly involved with components for ASP, the focus will be on COM.

### ActiveX and COM

The term ActiveX in most situations can be thought of as COM and an ActiveX component is the same as a COM component. In addition, an ActiveX control is the same as a COM control and an ActiveX server is the same as a COM server. ActiveX is actually a brand name that was introduced by Microsoft a few years ago to redirect developers towards the Internet. To facilitate this, it revamped some of the technologies to meet the low-bandwidth requirements of the Internet. Essentially, however, in modern day language and as far as ASP component development is concerned, ActiveX implies COM.

### Components and Objects

When an actual concrete instance of a component is required, the component is used as a blueprint to create an entity, this entity is known as an object. The process of
creating an object from a component is called ‘instantiation’ and an object is a single instance of a component. The object is created in the image of the component, and is the vehicle by which the developer can make use of the component’s functionality. It is possible to create several instances of a single component at any one time either within different applications or within a single application.

Interfaces and Implementation
One of the key characteristics of all COM components is that they are able to perform tasks without the client being aware of how the task is performed. To achieve this, it must define what methods it exposes, the parameters that each method expects, and the return value from each method. In order to facilitate this, COM distinguishes the description of a component’s functionality from its internal workings.

- The description of the component’s functionality is defined by the component’s interface. A component can have many interfaces, but related methods are generally grouped together within the same interface.

- The ‘internal workings’ of the methods and properties – that is, the code that allows them to perform their tasks – is generally referred to as the component’s implementation.

An interface provides a list of methods, properties and events that let clients know how to use it. The interface does not give any details of the component’s implementation, but its existence implies a promise that the functionality it describes will always be available. It is this distinction between interface and implementation that allows an existing component to be replaced with a new one that has a different implementation, provided that the new component offers the same interface as the old one.

Methods, Properties and Events
Methods allow a client to perform a task without worrying what is going on inside the object. Some methods need additional information or parameters that influence the behaviour of the method.

Properties are the settings or stored values that are contained within an object, some of which are exposed to the user. There are three types of properties: read-only properties, write-only properties and read-write properties.

Events are the object’s way of letting a client know that something has happened. The object can react to an event in a prescribed way and this is known as event handling.

Lollipop Diagrams
To represent the interfaces that a COM component supports a simple pictorial technique is used called lollipop diagrams as shown in Figure 6. A lollipop diagram represents the component in the form of a box; the interfaces extrude from the left side of the component. A name that appears within the box is the name of the component. The straight line protruding from the top of the box represents an interface called IUnknown, which is a special interface in COM. IUnknown is implemented by every component; however, languages such as Visual Basic shield the user from this and certain other COM-specific workings.
Identifying the Component
In the above example "IDrive" is the name given to an interface, however, as this is an intuitive name to give to a drive interface, another developer may also use the same name to describe another drive interface. To stop confusion, COM resolves this by ensuring that each COM interface has a ‘real’ name that is guaranteed to be unique. An interface’s unique name is called an interface identifier (IID), and a component’s unique name is called a class identifier (CLSID). IID and CLSID are both types of globally unique identifier (GUID); a GUID is a 128-bit number that can be generated with a special utility supplied by Microsoft.

Location of Components
To run a program normally requires an executable file, which might in turn, calls up some other files called dynamic-link libraries (DLLs) to perform some tasks. A DLL is like an executable file, in that it contains instructions that the computer can run directly, but it differs from an executable file in that it cannot be run independently. A DLL is like a library that can be called up by any executable that is already running.

COM is designed to allow any application or component to call up any other component, no matter where the other component is. This means that COM components can be stored within either executable files or as DLLs. There are several factors to consider when choosing which type to choose, but broadly speaking an executable offers greater security, while a DLL can give greater performance. This is because an executable will run in a separate process, which means that in order to use the component within the calling application, COM has to spend time passing data back and forth between the application’s process and the component’s process. A DLL-hosted component does not run in a separate process, so for components hosted in DLLs, this overhead is not an issue. Because of this, components hosted in DLLs are referred to as in-process components, while those located in executables are referred to as out-of-process components.

3.5.3 Classification of Components
To try to demonstrate the range of different tasks which components can provide solutions for, they can be broadly categorised according to their nature:
• **Generalised or Universal Components** are designed for use in a variety of different environments. An example of this is ActiveX Data Objects (ADO) components that provide programmatic access to data stores. These components can be used from programming languages such as Visual Basic or Visual C++ or alternatively, they can be accessed via Microsoft Office, Windows Script Host (WSH) or Internet Information Server (IIS).

• **Environment-Specific Components** are designed specifically for use in a particular environment, and rely on some of the facilities of that environment in order to operate. For example, the standard Browser Capabilities component that is available with ASP is designed specifically for use within IIS and ASP, and cannot be used out with these environments.

• **Visual Interface Components** are more often described as controls and the term is generally used to indicate that a component has some sort of visible representation that forms a significant part of its operation. The ActiveX Calendar control that comes with MS Office is one example of this type of component. Controls are usually implemented on client machines rather than the server as part of an ASP script.

• **Business Rules Components** contain the rules to which the operation of the component, application or system must adhere. These rules may include areas such as checking that customers remain within agreed credit limits, that discounts are applied in a uniform manner, or that a manufacturing cycle is correctly scheduled. Business rules components are mainly concerned with the internal logic of the application and have no visual interface.

• **Transactional Components** are utilised in situations where an application requires multiple, separate actions to take place in order to achieve the desired result. For example, when transferring monies from one bank account to another, both the debit and the credit parts of the transaction must be completed successfully otherwise the database will be incorrect. This is called a transaction, and components can be built to work with a transaction monitor to ensure the completion or complete undoing of each transaction.

• **Utility/Commercial Components** is the catchall that covers all the components that do not fall into the other categories. For example, components that are designed to perform specific tasks within other components.

3.6 **Network Application Architectures**

While ASP is a relatively new technology, network-based applications have been used for a number of years. In order to see where components fit into the overall system architecture the following sections look at the options available for network development.

3.6.1 **Two-Tier (or Client-Server) Systems**

Historically network applications have invariably been structured to consist of a server, which carries out the main computational processing of the application and multiple clients, which connect to that server, as shown in Figure 7. The clients provide instructions to the server and have access to the results that it produces.
clients are referred to as thin clients because they do not carry out a great deal of the processing as most of the functionality is hosted on the server.

As computing spread into more areas of the workplace this led to greater demands being placed on computer-based applications. This resulted in larger more complex applications that are often referred to as enterprise applications, which incorporated many servers and data stores. Such applications tend to be organic in that they are always growing and they can be difficult to maintain. To simplify the administration of these complex arrangements, applications started to be divided up into layers or tiers. These tiers can be either physical, whereby the tier actually resides in a distinct physical place or logical, which is a more conceptual way of thinking about them. Logical tiers assist in the planning and viewing of different parts of an application without the concerns about where they actually physically reside. Ultimately, this led to an expansion of the architectural model from a two-tier model to a three-tier model. Subsequently the three-tier model has been further expanded and generalised to an n-tier model.

![Figure 7. Two-Tier (Client-Server) System](image-url)
3.6.2 Three-Tier Systems

The client-server model is extended in the three-tier model by the addition of a middle layer that sits between the client and the data sources, as shown in Figure 8.

The three physical tiers break down as follows:

- The presentation or client tier is the front-end tool with which the end user interacts. It contains all the presentation logic that is all the code responsible for displaying data to the end user, and also for retrieving data from the end user. The presentation tier may take the form of any type of user interface, such as a Visual Basic application, however in ASP applications, the client tier usually takes the form of a web browser. In any case, applications and components in the presentation tier run on the client’s machine, and therefore consume the client’s resources. This does not particularly fit the case of ASP applications as the ASP pages in the middle tier generate the code that the user interacts with and this is where n-tier systems, which are described section 3.6.4, come in.

The presentation tier has two key characteristics:

- It only contains presentation logic, it does not concern itself with the inner workings of the application.
- It does not access a data source directly; it must pass requests to the business tier, which performs data access on the client’s behalf.
This level of separation and encapsulation is important, because it shields the browser from the complexity of the application’s inner workings. As a consequence, changes can be made in the middle tier without having to update the client.

- The **middle tier** (also known as the **business tier**) represents most of the logic that makes the application functional. This is where the various business components that model the business rules reside and there is an intention that this should model the real business that it is representing, as shown in Figure 9. By isolating the rules in the business components, no matter where the objects are used, the same business rules will be enforced. This is important as it ensures consistency throughout distributed applications. As each component has modelled a part of the real business rules, should one area of the task change, only one component will have to be altered rather then finding all of the appropriate changes that would need to be made in a linear application.

The client tier collects information required from the user so that the middle tier can perform whatever processing is necessary to achieve the task the application is intended to perform. Sometimes this will require additional information from a data store, or it may require information to be written to some form of persistent storage, both of which are handled by the third tier.

- The **data tier** represents the storage mechanism used to hold persistent data. As more and more data is kept in electronic formats, this tier encompasses an ever-growing number of storage formats, not only relational databases, but also text-based files, mail-servers etc. The separation of the data storage from the application of the business rules in the middle tier enables the application to accept data from any number of sources without concern for their type or location.

![Figure 9. Three-Tier Models](image)

**3.6.3 Location of ASP Pages Within the Architecture**

ASP does not fit easily into the physical three-tier model this is because the business tier will include the ASP-dependent and ASP-independent components together with the ASP pages themselves. ASP pages are responsible for creating visual output for a client, which means they have presentation elements, so, why do they reside in the middle tier? In physical tier terms, the presentation tier is effectively the client machine, and Active Server Pages are not part of this presentation tier, although they do have a tendency towards it. Having both visual and non-visual elements as part of a single tier is confusing, therefore the middle tier of ASP applications is subdivided into two tiers: the user-centric and the data-centric tiers. This gives a typical ASP application four tiers, which is where n-tier applications come in.
3.6.4 n-Tier Applications
‘n-Tier application’ is a generic way of referring to applications that have three or more tiers. In ASP the case where the middle tier is split to form a user-centric and data-centric tier is an example of an n-tier application.

The user-centric tier of an ASP application contains the ASP pages and environment dependent ASP components that help to render HTML pages to the presentation tier. Environment dependent ASP components generally make use of the ASP object model, so they are effectively tied to the user interface in some fashion to generate HTML, access and create cookies, etc. These components generally enforce basic business rules that are unlikely to change, for example, validation of user input.

The data-centric tier contains the components that do not depend upon the ASP environment. These are responsible for performing database manipulation, such as adding, querying and updating records in a table. These components are in no way dependent upon the ASP environment.

An Example of a Four-Tier ASP Architecture
A brief example will assist in showing how a four-tier architecture could be applied to an ASP application. The scenario is that an application is required to enable a web browser to be used to retrieve lists of customer details that are stored on a central database. ASP pages are used to provide the interface between the end-user and the components that retrieve and return the customer details from the database. A three-tier system would likely divide the application as follows:

- A presentation tier consisting of the client browser.
- The middle tier consisting of the ASP pages together with the retrieval component.
- The data tier consisting of the database that stores the customer details.

In Figure 10, the middle tier has been split into two logical tiers, with the ASP page residing on the user-centric tier and providing an interface for the user on the client browser. It would also collect the information from the user interface on the client and pass it on to the data-centric tier, which is responsible for accessing data from the database. The return values are then passed back to the ASP so that they can be formatted for the client to see.

The list object does not rely on the ASP object model as the form values are passed through the ASP page; therefore there is no reason why the application cannot have different types of client interfaces. Rather than always using a web browser, a Visual Basic application could be better for some clients. Furthermore, if it became necessary to replace or upgrade the database, then only the component that accessed the database would require to be changed.
3.6 Conclusions

This chapter has covered a wide range of areas and demonstrates the large number of alternatives that are available to web site builders. The technologies are still evolving which makes it difficult to keep abreast of new developments. However, it is important to adopt a longer-term strategy when considering an “e-business”. This is perhaps easier to achieve by opting for technologies from one particular source as hopefully, these will be compatible with each other and will continue to be supported in later versions from the same manufacturer. This approach narrows the alternatives down to three broad alternatives - Microsoft related products, Java related products or Unix/Linux products. All three offer viable solutions and it is possibly down to personal preference as to which one is used. A project of this size cannot fully explore all such alternatives and after providing a broad overview, the project will concentrate on Microsoft technologies.

Whichever combination of technologies are chosen, the other major considerations revolve around the architecture of the application and the design of individual components. As an e-commerce site increases in complexity, it becomes vitally important to ensure that sound design and architectural processes have been utilised in its construction.
4.0 Requirements Analysis and Design

4.1 Introduction
This section starts with a brief description of the anticipated requirements that the system would be expected to fulfil. This is intended to be a high level view of what is required of the application as seen from the perspective of the e-business and thus does not initially look at how the solution will be implemented from a technical perspective. Once these main objectives have been established, the second part of this section will deal with how an appropriate application can be developed that implements these objectives.

4.2 System Specification
The application being developed is aimed at a financial services business, which may or may not already have a presence on the web, but in either case it is looking to establish a framework to aid in its progression towards e-business status. The definition of e-business is taken from the DTI report and the framework is based on the “Adoption Ladder”[1], which is described in the same report. The main guiding criterion from this report is that an e-commerce site should be seen as only part of an overall e-business strategy. The end goal of such a strategy should be seen as developing a business that has a fully integrated system, which facilitates all dealings between itself and suppliers, customers and staff to be carried out electronically.

Due to the competitive and fast-changing nature of the financial services market, it is important that the e-commerce site is flexible and allows the financial services provider to alter various weightings and interest rates with the minimum of delay. Also, the products offered by the provider may vary through time and again it is envisaged that the site should be able to accommodate this with the minimum of disruption and with the maximum of reusability.

Customer access to the site will be via the Internet using a standard web browser such as Internet Explorer or Netscape Navigator. It is difficult to ascertain the computer literacy of the end users, however, it is envisaged that they will posses at least basic keyboard and mouse skills. The site should be developed with a view to making it easy to use from an end-users perspective. This will include factors such as speed of response, ease of navigation around the site and how intuitive the site is to use.

The actual business requirements of the site will be to initially provide product quotations, based on details input by the user, for example, typical services might include loans, mortgages and insurance policies. Based on the details input, the user will be provided with a breakdown of the total cost that they would have to pay for the selected product. They should then be given the choice to complete an application form to apply for the product described in the quotation. It is at this stage that copies of the clients details may be required to be held in the businesses computer system. It is envisaged that the details issued will be stored in a manner that will be accessible for other purposes e.g. sending out statements, marketing and opening the account etc.
The site should encourage “cross-selling” of products, for example, a user requesting a holiday loan should have the opportunity to view details of travel insurance.

The main objectives of the application will therefore be to:

1. Enable the company to provide potential clients with the opportunity to obtain quotations, via the Internet, on the costs and repayment amounts of various products that the company has to offer.

2. The end-user should be able to obtain quotations and apply for products by completing appropriate application forms on-line.

3. The application should be intuitive, fast and easy to use from an end users perspective.

4. The site design should take into consideration the “cross-selling” opportunities that the products provide.

5. The company should be able to easily and quickly alter the product parameters. They should also be able to offer new products as and when necessary.

6. Customer details may require to be maintained by the company and these should be held in a form that is accessible to other applications.

4.3 Requirements Analysis

The objectives described above will now be examined to ascertain how best they can be achieved.

The objectives appear to fall into two broad categories; the first category covers the visual presentation of the information and the second category deals with how the information is produced.

The first 4 objectives described in section 4.2 all relate to the way in which the company and customer interact and fall into the first category. Customer access to the site will be via the Internet with the use of a standard browser such as Microsoft’s Internet Explorer or Netscape’s Network Navigator. The design of this part of the application will be based on the principles used within multimedia design, as the requirements of the web interface closely resembles those of a multimedia application, for example, ease of use and presentation of information etc.

On the other hand, the other 2 objectives relate more closely to how the underlying data is processed and stored and this is where software components can assist. The benefits of using components have been detailed in section 3.5.1, however to briefly recap; precompiled components should provide increased speed of execution over script and components also promote reusability and maintainability by allowing components to be exchanged without affecting the remainder of the application. This is achieved by utilising standard interfaces for the components and developing each component to interact through these interfaces rather than directly with the underlying implementation of the component. For a full description of software components see Chapter 3, section 3.5.
The architecture of the application will be an n-tier one, as described in Chapter 3, in this case n being 4. The first or presentation tier will take the form of the Web Browser that will be used to display the HTML code. The second or middle tier is split into two layers, the ASP pages that collect the users input, and secondly, the various components that perform calculations and formatting on the data together with the data connection component that provides access to the persistent data store. The third or data tier is the actual data store, which in this case will be a relational database that will store the user details that are input to application forms.

It is envisaged that only the presentation tier will be physically located on the users computer, with both the middle and data tiers being physically located on the web server.

From Figure 11, it can be seen the end-user communicates only directly with the ASP pages and not directly with the components or data store. This provides flexibility to the application as it allows components to be replaced as and when necessary, for example, to provide the additional functionality to the site or to rectify a bug in a particular component. These changes can be accomplished with the minimum of disturbance to the remainder of the system. Furthermore, the principle of adding a distinction between the components and the data store also increases the flexibility as the components are only interfacing to the data connection and not directly to the data store itself. Therefore the data store could be replaced with a different type of store by simply substituting the data connection with another one that interfaces to the new data store. From the components viewpoint, as long as the new connection offers the same interface to them, then they will continue to function unchanged.

![Figure 11. n-Tier Application Layout](image-url)
4.4 Design of the User Interface
It is envisaged that the site will initially offer four categories of products as follows:

1. **Mortgages** – Within this category will fall a selection of the various mortgage products that the business currently has, for example, “Endowment” type, “Capital and Interest” type and “Fixed Rate” type mortgages.

2. **Insurance** – The insurance products offered will cover areas such as “Home”, “Travel” and “Car” insurance.

3. **Loans** – The types of loans offered will vary over time, but will include “Car”, “Personal” and “Holiday” loans.

4. **Accounts** – This category includes the various accounts that the business provides such as “Current Accounts”, “Saving Accounts” and “Term Deposit Accounts”.

Over time further categories such as “Credit Cards” may be added to the site.

Each individual product within each of the four categories will be required to provide an option to obtain further “Product Information” and to complete an “Application Form”. In addition, all the products, apart from those within the “Accounts” category, will need to be able to produce a quotation, based on details input by the user.

The “cross-selling” of products means that links should be established between “Home” insurance and “Mortgages” and between “Insurance” and “Loans”.

The user should also be able to access a general “Help” screen from any point in the site.

The overall layout of the site is shown in Figure 12. with Figures 13. to 16. providing details of each of the four categories.
Figure 12. Financial Services Web Site
Figure 13. Financial Services Web Site (Insert 1)

Figure 14. Financial Services Web Site (Insert 2)
Figure 15. Financial Services Web Site (Insert 3)

Figure 16. Financial Services Web Site (Insert 4)
4.4.1 The Display Design Rational
This section looks at areas for consideration when designing the visual presentation of the data.

Screen Content:

1. While higher resolutions are now fairly common, the design of all screens has been carried out using the assumption that a 800x600 display will be used to view them. This is a compromise to ensure that the display is accurate on as many users screens as possible, while maintaining display quality.

2. All main screens have a cream background. This colour was chosen to provide a good contrast with the other graphics and text that is included on each screen. The background colours of other screens varies based on the area – see “colours” below.

3. The page content has been kept to a minimum to assist in clarity and increase the speed the pages will be loaded.

4. All screens have been constructed around the same basic framework as shown in Figure 17:

![Figure 17. Display Screen Layout](image)

- The top of the frame contains the heading of the particular page and the middle of the frame contains the information that is to be conveyed to the user. Any buttons are placed in the bottom of the frame.

- The text in the headings is all of bold “Comic Sans MS” font using size 18pt. This font is used throughout the piece and gives the system a less formal appearance and this font size is used to highlight the headings.

- Subheadings within the bottom part of the frame are also bold “Comic Sans MS” font but the size is only 14pt. This difference in size emphasises the difference between headings and subheadings, while still making both legible on the screen.
• Any other text that appears in the bottom part of the frame is plain “Comic Sans MS” of size 12pt. This allows the Sans Comic MS font to be used and the smaller font size allows more information to be displayed at one time.

**Colours** - Each individual area of the site has a different background colour; these colours were chosen to assist the user in navigating around the system. All text and graphics within each area are of the allotted colour. This has been carried through to buttons that appear on other pages which transfer the user to another page e.g. the “Quotation” area is constructed in a cream colour, therefore the text on buttons pointing to that area are also written in cream. This has been used to assist the user in associating buttons with areas and thus speed up operation and reduce errors.

**Buttons** - All buttons are of a rectangle design and as stated previously, are coloured based on the area they represent. Buttons that appear on more than one page always appear in the same location within each form that they appear. Furthermore once an area has been used on one form to display a particular button, no other form will display a different button in the same location e.g. the “Help” button appears in the bottom right hand corner, so no other button can appear in the bottom right hand corner of any page. This is to try and minimise errors by users pressing the wrong button.

### 4.4.2 Gulf of Execution/Gulf of Evaluation

• The gulf of execution has been minimised by trying to model the behaviour of the system as closely as possible to that of the real world. This is achieved by modelling all screen “buttons” to look like real buttons and to name them so that new users are given clues as to their functions. Furthermore a short description is given, if the cursor is moved over the button, of what the result will be of clicking on the button.

• Using a shopping type list to display the “Loan Types” selected by the user and again through the use of a suitable named button to gain access to it reduces the gulf of evaluation.

### 4.4.3 Website Storyboard

The website layout has been prototyped using a storyboard and is attached to this report by way of Appendix 1. Storyboards provide a graphical depiction of the outward appearance of the intended system without any accompanying system functionality. This method of prototyping the presentation layer was chosen as the inclusion of the presentation layer within this project is primarily for completeness only. The main objectives of the project relate to the design and development of the components that predominately provide the functionality of the application.
4.5 Design Of Components

4.5.1 Introduction
In this section the design of the components that will provide the functionality of the application will be described.

The component design is based primarily on an object-oriented approach, however in order to provide the reusability that is required, the object-oriented approach has been altered slightly to accommodate this.

Szyperrski[22] argues that object oriented programming comprises primarily of a combination of polymorphism, late binding, information hiding and inheritance. However, while inheritance within individual components is acceptable, inheritance between components can lead to difficulties if subsequently changes are made to parent classes that affect more than one component.

4.5.2 Description of Component Formation
The approach that has been taken in the actual design of the components is to initially view the application as a “black-box”, which means only being concerned with the inputs and outputs and not how the actual processing is achieved, as is shown in Figure 18. This enabled the functionality that application required to provide to be ascertained. The result was that three main outputs of the system were identified, which were:

- Quotations for the various products.
- Application forms to apply for these products.
- Agreements such as loan agreements for signature once the provision of the product have been agreed.

The next stage was to identify what information was required to enable these outputs to be produced. The result was that a combination of user inputs and business input was identified as being required, which were as follows:

- The business has to configure the system with details of the terms and conditions that apply to each product (a Product Profile).
- The user has to be able to select the product they require.
- The user also should be able to specify criteria such as the amount and term etc.
- If an application is to be made, then the users personal details will be required to prepare loan agreements etc.
Having established the inputs and outputs of the system, the next step is to ascertain what processing requires to be carried out within the application to convert the input data into the necessary output data.

- **Quotations** – These provide a breakdown of the cost to the user for a particular product or service, as shown in Figure 19. To achieve this they need to have access to the users selection of input requirements and also the profile for that particular product as specified by the business. A combination of these two sources of information will then provide a set of data from which the quotation can be calculated. This data will then require various calculations applied to it to ascertain items such as the loan repayments or insurance premiums etc.

- **Application Forms** – Figure 20 demonstrates how application forms enable users to apply for products or services that the business supplies. This requires the user to supply personal details that will be used by the business to ascertain whether they are willing to provide the requested product. The business decision making
process will include the application of a credit scoring system to the users details. Assuming the application is successful, the details supplied by the user should be transferred to a persistent data store, such as a database.

![Diagram of Application Process](image)

**Figure 20. Initial view of production of “Applications”**

- **Agreements** – These are produced once the provision of a product or service has been approved, as shown in Figure 21. They include details of the user together with details of the product being supplied. These details should already be held within the data store as per the “application” procedure discussed above.

![Diagram of Agreement Process](image)

**Figure 21. Initial view of production of “Agreements”**

From this analysis it is apparent that some overlap exists between the three functions, for instance, the information that is used to produce the quotation then forms part of the information that is required within the application form. Despite this, it might still be beneficial to store both parts of the information separately because the quotation details relate to a specific request while the other personal details are of a more general nature and as such are likely to relate to any product that the user requests. It would appear that if both sets of information were to be stored within a relational database and then linked together, this would enable the general personal details to be
held only once and then reused for all applications from that particular user, as shown in Figure 22.

Table 1 – Customer Details
Customer1

Table 2 – Quotation Details
Customer1 Quote1
Customer1 Quote2

Thus “Customer1” personal details can be combined with either “Customer1 Quote1” or “Customer1 Quote2” to produce an application or agreement form. In addition the personal details could be used for other purposes such as to send statements or other literature etc.

**Figure 22. Relational Database**

Another area of overlap is in the presentation of the data – all of the functions require data to be displayed on the users screen and therefore the provision of a “Formatting” component would be beneficial. This component would exhibit polymorphic behaviour and base its output on the type of input it receives.

The combined model showing the individual components as coloured boxes, together with the interactions between them is shown in Figure 23. This diagram includes one further component that allows the database to be separated from the other components.
Figure 23. Component Layout (Loans, Insurance and Mortgages)
4.5.3 Conceptualisation of Component Model

One of the goals of this design is to develop components that are reusable. To achieve this, individual components will require to operate independently of each other and will demonstrate encapsulation of information. All interactions between components will be via the public interfaces, thus simplifying the replacement and/or reuse of individual components – a developer then only requires to know the functionality that is provided on the interface to utilise the component and is not concerned with the internal processing of the data. As the components are being developed to be independent, it would be easier to conceptualise the purpose of each component if the information being passed between them was contained within an appropriate object. This may lead to some duplication of data, however, having a clearer distinction between the components outweighs this disadvantage. For example, in the case of a request for a quotation, the quotation is seen as an object that will eventually contain the details necessary to provide the user with the quotation. When the quotation is first initialised it may only have some of its parameters set, the remaining parameters are set as it passes from component to component, as shown in Figure 24.

Once the quotation has reached the end of this cycle it will either be sent to the formatting component, which will provide appropriate formatting and send it to the screen or it will be sent to an application object, which will transfer the values to the database. In either case the quotation object will be destroyed at this point.
4.5.4 Design of Individual Components

The high-level design process has identified a number of components that will be required to provide the functionality to the overall system. The design of several of the key components will now be examined to demonstrate the techniques that have been adopted.

To achieve reusability, all of the components have been designed to be as generic as possible. To achieve this aim, the functionality of each component has been kept to a fairly narrow width; accordingly, the components can be considered to be “fine-grained”. The components that accept an object as an input have been developed to demonstrate polymorphism and late binding, as they do not know what type of object they will receive until execution and they exhibit the appropriate behaviour based on the object they receive. For example, the calculation component receives quotation objects and base on the type of quotation; it performs the calculations in different ways.

The design of the following components will now be described; this then allows a set of components to be implemented that can be used to demonstrate a quotation being generated:

- **Product Specification (or Profile)**– The product specification component provides a means for the business to create product profiles in which they can declare the various key parameters that will apply to specific products. This is then used by a quotation object to check that the user inputs are allowable for the product requested, and if not they adjusted accordingly. This component will initially be utilised to specify the criteria for loans and mortgages; however, it could also be used for other categories of products such as credit cards.

  The first step is to identify the parameters that will have to be set. This involves an examination of existing products and discussions with potential users of the component. In addition to these parameters, certain other parameters were identified as necessary to act as internal flags that would be used to confirm whether certain actions have been performed.

  The next step is to identify the functions that the component has to offer on its interface to enable it to provide the functionality that the business requires from it.

  A summary of the parameters and functions that will require to be provided are detailed in Figure 25, a fuller description of these is available in Appendix 3.
Figure 25. Summary of “Profile” component

- **Calculation** – The calculation component is required to perform calculations on the quotation data based on the value of calculation property that has been set in the quotation by the profile component described above.

After examination of various existing borrowing types and discussions with people from within the financial services industry, it has been ascertained that the methods to calculate repayments of borrowing fall into three types of calculation:

1. **Capital and Interest** (interest calculated on full sum borrowed) – This type of borrowing calculates the interest figure without taking into consideration the monthly repayments of capital.

2. **Capital and Interest** (interest calculated on amount outstanding) – This type of borrowing calculates the interest on the actual amount of capital outstanding i.e. it takes into account that the sum borrowed is reducing month by month.

3. **Interest Only** – This type assumes that only interest is payable on a monthly basis and that the capital is repaid at the end of the term in one lump sum.

This component does not maintain any properties with all calculation values being passed in and out of the calculation object by way of parameters. All of the methods require the same input parameters; being the borrowing type, the capital amount, the interest rate, the administration fee and the repayment term.
The public methods that the component provides are as follows, and are summarised in Figure 26, a full description of each method or function is included in Appendix 3:

<table>
<thead>
<tr>
<th>“Calculation” Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public Methods:</strong></td>
</tr>
<tr>
<td>Calculation</td>
</tr>
<tr>
<td>Annual Percentage Rate (Borrowing Type, Capital, Interest Rate, Administration Fee, Term) APR</td>
</tr>
<tr>
<td>Amount Repayable (Borrowing Type, Capital, Interest Rate, Administration Fee, Term) Total Amount</td>
</tr>
<tr>
<td>Monthly Repayments (Borrowing Type, Capital, Interest Rate, Administration Fee, Term) Monthly Amount</td>
</tr>
<tr>
<td>Interest Payable (Borrowing Type, Capital, Interest Rate, Administration Fee, Term) Total Interest</td>
</tr>
</tbody>
</table>

| **Private Properties:** |

- **Quotation** – The quotation component is used to create instances of a quotation that are then used to represent an individual client request and is summarised in Figure 27, a full description of the properties and functions is included in Appendix 3. When the instance is created no values are set within it. The user inputs are captured through accessing the ASP object model and then it is the “Profile” component that adds to and adjusts these prior to the “Calculation” component carrying out and storing the values for APR, repayments etc. It is therefore important that the components are utilised in the correct order and the order is discussed more fully below. The completed quotation is then passed to the “Format” component to be formatted for display to the users screen.

<table>
<thead>
<tr>
<th>“Quotation” Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public Methods:</strong></td>
</tr>
<tr>
<td>Quotation</td>
</tr>
<tr>
<td>Collect User Input ()</td>
</tr>
<tr>
<td>Set Profile Details (Profile)</td>
</tr>
<tr>
<td>Set Calculation Details (Calculation)</td>
</tr>
</tbody>
</table>

| **Private Properties:** |
| Calculation Type |
| Purpose |
| Interest Rate |
| Capital |
| Term |
| Fee Amount |
| Fee Rate |
| APR |
| Repayments |
| Total Interest |
| Check Term Bounds |
| Check Capital Bounds |
| Check Multiple |
| Check Fee Bounds |

**Figure 26. Summary of “Calculation” component**

**Figure 27. Summary of “Quotation” component**
- **Format** – This component removes all the responsibility for formatting the display from the ASP page into a separate component and is summarised in Figure 28, a full description of the methods and functions is included in Appendix 3. The format component will have a different method for each type of display that is required. For example a method to display the contents of a quotation and one to display the contents of an application form etc.

<table>
<thead>
<tr>
<th>“Format” Class</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Public Methods:</strong></td>
</tr>
<tr>
<td>Format</td>
</tr>
<tr>
<td>Write Quote (Quotation)</td>
</tr>
<tr>
<td><strong>Private Methods:</strong></td>
</tr>
<tr>
<td>Page Heading ()</td>
</tr>
<tr>
<td>Draw Line ()</td>
</tr>
<tr>
<td>Start Table ()</td>
</tr>
<tr>
<td>Display Capital (Capital, Check Capital Bounds, Check Multiple)</td>
</tr>
<tr>
<td>Display Term (Term, Check Term Bounds)</td>
</tr>
<tr>
<td>Display Interest Rate (Interest Rate)</td>
</tr>
<tr>
<td>Display APR (APR)</td>
</tr>
<tr>
<td>Display Interest (Interest)</td>
</tr>
<tr>
<td>Display Fee Rate (Fee Rate)</td>
</tr>
<tr>
<td>Display Fee Charge (Fee Amount)</td>
</tr>
<tr>
<td>Display Total Repayable (Total Repayable)</td>
</tr>
<tr>
<td>Display Repayments (Repayments)</td>
</tr>
<tr>
<td>End Table ()</td>
</tr>
<tr>
<td>End Page ()</td>
</tr>
<tr>
<td>Table Entry (Narrative, value) Narrative</td>
</tr>
</tbody>
</table>

**Figure 28. Summary of “Format” component**

### 4.5.5 Order of Execution

It is important that the components are called in a specific order, for example, the calculation cannot be performed until the profile and user input have been gathered. The order that the processes should adhere to is as follows, and is demonstrated in Figure 29:

1. Create instances of each different profile.

2. Create a quotation instance.

3. Call the “Collect User Input” method on the quotation instance.

4. Check the value of the “purpose” parameter of quotation instance.

5. Based on purpose value, pass the appropriate profile to the quotation using the “Set Profile Details” method.

6. Create an instance of the calculator component.
7. Call the “Set Calculation Details” method of quotation and pass it the calculation instance.

8. Create an instance of the format component.

9. Call the “Write Quote” method of format and pass it the quotation instance.

10. Destroy all the objects.

**Figure 29. Order of Execution**

**4.6 Error Checking Within Components**

The components have been designed to be re-usable and therefore the amount of checking of input has been limited to the checking that requires to be carried out in each particular component. For example the “Profile”, “Quote” and “Calculation” components all deal with a “CalculationType”, however, it is only within the “Calculation” component that the actual value that has been input requires to be confirmed. This is because the value within this component directly influences the method of calculation and therefore if the value is not one of the acceptable ones, then an error will occur. If error checking for “CalculationType” were performed in other components, this would have the effect of limiting the reusability of these components, as the values of acceptable inputs would require being “hard-wired” into them. The value is thus checked within the “Calculation” component and if an error is discovered an appropriate error code is based back to the calling component.
The error codes that are passed between components have been designed to provide as much information as possible to the end user, to enable errors to be easily identified. A full description of the error checking performed in each component, together with the error codes is described in detail in Appendix 4.

A summary of the error checking that is performed within each of the Profile, Quotation and Calculation Components is shown in Figure 30. The APR, Repayments and TotalInterest within the Quotation are calculated values from the Calculation component and thus assuming other inputs correct then these values do not need to be checked. The Purpose will be checked within the ASP code to ensure that it matches a valid profile.

**Figure 30. Error Checking Within Components**

### 4.7 Conclusions

This chapter starts with the design of a potential Financial Services web site and from this identifies the functionality that such a site will require. The chapter demonstrates that the functionality of a site can be developed separately from the user interface. The user interface will be constructed with a combination of HTML or XML code with embedded scripting code such as VBScript providing access to the underlying site functionality, which is supplied using generic components. However, to make components generic takes a considerable amount of design effort and it is important to split up functionality and ensure that each component only performs a small part of the overall functionality. If this approach is not taken, then the components are likely to become too specific to a particular application and therefore less reusable. Another aspect to consider is error checking within components. If unnecessary error checking is built into a component, this also reduces the reusability of the component. Error checking should only be to data that is critical to the operation of the component and components should be able to pass such error codes to other components. Error codes should be provided that assist the developer in identifying the source of the error.
5.0 Testing and Validation

5.1 Introduction
Having established the requirements of the system it is now important to define the testing and validation strategy that will be used to establish whether the system meets these requirements and also that it functions as expected. This strategy involves unit or component testing and then validation testing.

5.2 Unit or Component Testing
This tests each component of the software as implemented in source code. These tests focus on each component individually, ensuring that it functions properly as a unit. The testing method predominantly used at this stage is ‘white-box’ testing, as this ensures that all possible paths through the unit are implemented and that the maximum error detection rate is achieved.

The tests cover:

- The component interface is tested to ensure that information correctly flows into and out of the unit – if data does not enter and exit properly, all other tests are moot.

Myers [21] proposes a checklist for interface tests, which includes:

1. Number of input parameters equal to number of arguments?
2. Parameter and argument attributes match?
3. Parameter and argument units systems match?
4. Input-only argument altered?
5. Global variable definitions consistent across modules?

- The local data structure is examined to ensure that data stored temporarily maintains its integrity during all steps in an algorithm’s execution. In addition, the impact of global data on a module should be ascertained during component testing.

The local data structure is a common source of errors and test cases should be designed to uncover errors in the following:

1. Improper or inconsistent typing.
2. Erroneous initialisation or default values.
3. Incorrect (misspelled or truncated) variable names.
4. Inconsistent data types.

5. Underflow, overflow and addressing exceptions.

- The boundary conditions are tested to ensure that the module operates properly at boundaries established to limit or restrict processing.

When testing this we should include areas such as:

1. What happens when the nth element of an n-dimensional array is processed?

2. What happens when the ith repetition of a loop with i passes is invoked?

3. What happens when the maximum or minimum allowable value is encountered?

- All independent paths through the module are exercised to ensure that all statements in the module have been executed at least once.

- All error-handling paths are tested. Error conditions should have been anticipated and error-handling paths set up to re-route or cleanly terminate processing when an error does occur.

Tests should cover situations such as:

1. Error description is unintelligible.

2. Error noted does not correspond to error encountered.

3. Error condition causes system intervention prior to error handling.

4. Exception-condition processing is incorrect.

5. Error description does not provide enough information to assist in the location of the cause of the error.

After the code has been developed, reviewed and verified for correct syntax, component test case design begins. A review of the design information provides guidance for establishing test cases. It is vital that each test case should be coupled with a set of expected results.

Driver and/or stub software must be developed to simulate the module interaction with other modules during the tests.

5.2.1 White-Box Testing
Each method in each component should be white-box tested before integrating them into the component.

This is achieved by:
Firstly identifying the control flow points in a particular method and numbering them.

Then construct a flow graph.

Next calculate the number of independent paths through the method.

Devise test cases, including drivers, to demonstrate that all of these paths have been exercised. Test data should try to cover ‘likely’ hot spots e.g. at boundaries.

To demonstrate this type of testing, the following method has been extracted from the “Calculation” component. This method examines the “CalculationType” input parameter and based on its value a different calculation is performed on the other inputs and the result is returned.

The numbers on the left relate to the flow graph.

```vbnet
Function AnnualPercentageRate (CalculationType As String, Capital As Double, InterestRate As Double, AdminFee As Double, Term As Integer) As Double
{
    intCheck = 1
    dblErrorCode = ErrorCheck(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm)

    1 If (dblErrorCode < 0)
        {
            intCheck = 0
            dblAPR = dblErrorCode
        }

    2 If (CalculationType = "CapIntMort") And (intCheck = 1)
        {
            For (intAmount = 0 to Term)
                {
                    dblTotalInt = dblTotalInt + ((Capital - ((Capital / Term) * intAmount))
                        * ((InterestRate / 12) * (1 / 100)))
                }

            10 dblTotalCost = InterestPayable(CalculationType, Capital, InterestRate, AdminFee, Term) + AdminFee
            dblAPR = (((InterestPayable(CalculationType, Capital, InterestRate, AdminFee, Term) + AdminFee) / dblIntPayable) * InterestRate)
        }

    3 ElseIf (CalculationType = "CapInt") And (intCheck = 1)
        {
            dblTotalAmountBorrowed = Capital + AdminFee

            6 For (intAmount = 0 To Term)
                {
                    dblIntPayable = dblIntPayable + ((dblTotalAmountBorrowed - ((dblTotalAmountBorrowed / Term) * intAmount))
                        * (InterestRate / 12) * (1 / 100)))
                }

            11 dblAPR = (((InterestPayable(CalculationType, Capital, InterestRate, AdminFee, Term) + AdminFee) / dblIntPayable) * InterestRate)
        }

    4 ElseIf (CalculationType = "IntOnly") And (intCheck = 1)
```
The number of distinct paths that need to be tested can be calculated from Figure 31: 

Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes and \( E \) is the number of flow graph edges is:

\[
\v(G) = E - N + 2 \quad \text{[Equation 13.]}
\]

Therefore above graph \( v(G) = 15 - 11 + 2 \)

\[= 6\]

This means that there should be 6 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 9
Path 2 = 1 to 2 to 3 to 4 to 9
Path 3 = 1 to 2 to 3 to 6 to 11 to 9
Path 4 = 1 to 2 to 3 to 6 to 8 to 6 to 11 to 9
Path 5 = 1 to 2 to 5 to 10 to 9
Path 6 = 1 to 2 to 5 to 7 to 5 to 10 to 9

To do this a driver needs to be constructed that will enable various values to be input and the results checked against the expected results. This can be simplified by ensuring that functions that are utilised by other functions are tested first, this then enables them to be used within the testing of these other functions. For example, the “ErrorCheck” and “InterestPayable” functions are used by the other functions and therefore should be tested first. In addition, to provide confirmation of which path has been executed, several additional lines of code have been inserted into the function body. These extra lines merely output a message to the screen based on which part of code has been executed and also display what value has been returned to the calling program.

Assumptions made when deciding on test data:

- The test data that will be used is based on boundary analysis.
- To reduce the number of combinations that require to be checked, it will be assumed that if an error occurs with one particular parameter value, then an error will occur irrespective of the values of the other parameters.
- The components will be built to accept certain types of data for certain parameters, for example “Capital” is of type “Double”. Accordingly, all values supplied to the component for “Capital” must be of type “Double”. Therefore it is deemed the responsibility of the author of the ASP page, which utilises the component, to ensure that data is of the correct type. The test data will assume that such tests have been made and therefore only data of the correct type will be tested here.

![Flow Graph for APR function](image)

**Figure 31. Flow Graph for APR function**

<table>
<thead>
<tr>
<th>Parameter Name</th>
<th>Test</th>
<th>Path</th>
<th>Value: CalcType</th>
<th>Value: Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculationType</td>
<td>Correct input</td>
<td>3</td>
<td>CapInt</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>CapInt</td>
<td>4</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>5</td>
<td>CapIntMort</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>6</td>
<td>CapIntMort</td>
<td>4</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>2</td>
<td>IntOnly</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Blank</td>
<td>1</td>
<td></td>
<td>-990000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Incorrect input</td>
<td>1</td>
<td>CapAndInt</td>
<td>1</td>
<td>-990000</td>
</tr>
</tbody>
</table>

**Table 2. APR function – “CalculationType” input test**
5.2.2 Black-Box Testing
The next stage is to integrate the separate methods to form each component and then to integrate the components to form the desired functionality, which predominately uses ‘black-box’ testing.
Integration can be achieved using either a top-down, bottom-up or sandwich strategy depending on the final structure of the application. The application under construction does not have a substantial depth of layers and so a top-down strategy will be utilised, adding components in a depth first manner. This strategy has been chosen as it allows components to be added more or less in line with the way they are perceived to operate on the “quotation” objects.

As the number of components included within the application grows, this could result in the number of levels also increasing. If this occurs, then a sandwich approach could be utilised as this can both reduce the number of stubs required and also allow clusters of components to be added in a similar way to the way the components are grouped to form a “quotation”.

5.3 Validation Testing
Tests that requirements in the original software requirement analysis are validated against the software that has been constructed. To do this, a set of ‘high-order, black-box’ tests are conducted to provide final assurance that the software meets all functional, behavioural, and performance requirements.

After each validation test case, one of two possible conditions exists:

1. The function or performance characteristics conform to specification and are accepted, or
2. A deviation from specification is uncovered and a deficiency list is created.

Table 8 shows a set of validation tests that have been devised to test whether the “Quotation” application meets all the requirements in the original requirements document. These tests are ‘Black-Box’ in nature.

<table>
<thead>
<tr>
<th>Original Requirement</th>
<th>Validation Test</th>
</tr>
</thead>
</table>
| 1. User should be able to specify the following parameters:  
  • Amount Borrowed  
  • Term of Borrowing  
  • Specify borrowing type | An ASP page will be constructed to enable the user to specify these parameters. The details input will then be made available to the quotation components for use within the quotation calculation via the ASP object model. |
| 2. Business should be able to define a number of product profiles that will be used in preparing the quotation, these will specify values for:  
  • Interest rate  
  • Fee rate  
  • Maximum and minimum fee  
  • Maximum and minimum amount of borrowing  
  • Maximum and minimum repayment term | A component entitled “Profile” will be built that allows the business to create instances of a profile, each of which will have the specified parameters that can then be set. |
3. The system should be able to calculate relevant details relating to:
   - APR
   - Total Interest Payable
   - Total Fee Payable
   - Total Amount Payable
   - Monthly Repayments

A component entitled “Calculator” will be built that will calculate the various details using one of three different calculation methods:
   - Capital and Interest (Loan)
   - Interest Only
   - Capital and Interest (Mortgage)

The calculation method that will be employed when calculating a particular product quotation will be specified within the profile referred to in “2” above.

4. The business should be able to add to or amend the profiles referred to in “2” above with the minimum of delay.

Profiles will be created and configured via an ASP page. Any alterations to that page will be reflected within the quotation.

5. The data accumulated while calculating a quotation should be available for re-use within the remainder of the application if so required.

A component entitled “Quotation” will be built that enables objects to be created that hold all the details required to provide a particular quotation. These objects can then be used to produce output for the users screen or they could be passed to another component to, for example, produce an application form.

6. The quotation system should be designed with flexibility and re-usability in mind. Thus promoting the possibility that the system could be amended or re-used in other situations in the future.

The quotation system will be designed using components, each of which provide specific functionality. The components should be generic so that individual components could be replaced or re-used without adversely affecting the other components.

7. The quotation system should simplify the inclusion of such functionality within a website.

The number of lines of ASP code will be greatly reduced and the components encapsulate the details of the calculations. Within the web page, only visible sign will be the creation and configuration of the instances of the components, which are more intuitive to use.

8. The results from the quotation calculations should be able to be displayed on the users web browser.

A component entitled “Format” will be produced that enables the results of the quotation to be prepared for presentation on the users screen.

<table>
<thead>
<tr>
<th>Table 8. Validation Tests</th>
</tr>
</thead>
</table>

5.4 Conclusions

This chapter investigated the testing and validation required to be carried out on the components. It demonstrated the considerable amount of tests that require to be performed on each function within each component. However, while this takes considerable effort at the development stage, it should ensure that errors are identified thus allowing future reuse of the components with the minimum of testing being required at that time. When testing individual functions, prior to assembling the complete component, it is beneficial to into account which components/functions utilise other components and functions and then to test these first as this greatly reduces the need to produce additional test software.
6.0 Implementation

6.1 Introduction
This chapter details the implementation of the design of the four components discussed in the previous chapter. It also will demonstrate how the components can then be utilised together to provide a quotation system for the financial services e-commerce web site.

6.2 Technologies Employed
The technologies that will be used to implement the design are all based around Microsoft products. Having investigated a number of possibilities, such as Java, Unix etc. it became apparent that the number of differing technologies available merely complicated the underlying purpose of this project, namely to show how a component approach to software development could simplify the construction of e-commerce sites. By constructing components that have COM interfaces, the choice of other technologies become less important, as COM provides a universal standard for interface access; the components are thus able to function within any configuration of technologies. Similarly COM is language neutral meaning components can be written in any computer language and should still be able to interact with other components even if they are written in a different language. The choice of utilising predominately Microsoft products has therefore been taken to simplify the process of building, testing and demonstrating the components, rather from any perceived performance gain.

The development has been carried out utilising the following configuration of products/technologies:

- **Microsoft Windows NT 4 Server** – This is the operating system that is used to host all the other technologies that have been used.

- **Microsoft’s Web server, Internet Information Server 4 (IIS4) -** This has been used to host the e-commerce site that demonstrates and tests the components.

- **Microsoft Internet Explorer 5 (IE 5) –** This is the web browser that is used to view the ASP pages.

- **Active Server Pages (ASP) –** A scripting environment that facilitates server-side program execution under the control of IIS.

- **VBScript** – This is the scripting language that is used within the ASP pages. However, another scripting language such as JavaScript could just as easily be used to provide this role.

- **Component Object Model (COM)** – This provides an interface standard, together with a set of operating system services, that facilitates interoperability between software components.
• **Microsoft Visual Basic 6 (VB 6)** – This is the programming language that is used to actually construct the components.

• **Microsoft Transaction Server (MTS)** - A technology that eases the development and administration of scalable, secure N-tier applications. However, this is not required in the development of the quotation components.

• **ActiveX Data Objects (ADO)** - A generic data access layer that permits applications to be indifferent as to which vendor’s database they are using.

### 6.3 The Architecture for the E-Commerce Site

The components that will be implemented will operate within an architecture similar to that in Figure 32, although there is presently no requirement for access to be made to a database, when the application is developed further to accept customer details etc. as discussed in the previous chapter, there will then be a requirement for database access.

The four components have been built using Microsoft Visual Basic Version 6 (VB6), which provides a simplified means of constructing COM components that hides some of the more involved parts of writing COM interfaces. VB6 demonstrates the performance advantages of utilising compiled code over interpreted scripts, however, to improve performance further, the finalised versions of the components would ideally be built using a language such as C++. The components have been built into a dynamic link library (DLL), this allows the component to be loaded only if required during execution and thus providing a possible performance improvement. A further advantage of using a DLL is that it runs within the same execution memory space as the host application and therefore does not require the switching of processes in and out of memory, which again leads to improvements in performance. A potential difficulty in allowing the DLL to execute in the same memory space is that if the DLL crashes, it means the whole application will crash, it is therefore important to ensure that the DLL has been fully tested before use.

The components were implemented in parallel based on the designs described in the “Design and Requirements Analysis” chapter. This assisted in the testing of the components and also provided confirmation that the development was progressing satisfactorily. As functions were developed and tested and then integrated into a particular component, that functionality could then be utilised to test other related functions in other components, thus reducing the need for developing additional software to allow testing.
6.4 Component Implementation

6.4.1 The Calculator Component
This component carries out calculations based on the values supplied to it. All of the values required to carry out the calculation are likely to be available directly from the client and therefore all of the values are passed to the instance of the component when the particular method is called. This means that no values are held as global variables. However, to simplify the implementation of the methods, they employ calls to other methods within the component. This means that parameters passed to the first method may not be directly utilised within that method but may be required to call the additional method. This has allowed the reuse of code and the simplification of the methods. Each of the methods requires the same set of parameters to be set, these parameters are all specified to be of a particular type and this ensures that only parameters of that type can be passed as a parameter, for example, Capital must be of type Double and so the component will not accept a String as a value for Capital. This assists in the error checking within the component, as only inputs of the specified type require to be tested, invalid inputs should be identified out with the component, for example within an ASP page. The names and types of the input parameters are as follows:

- **CalculationType** type String.
- **Capital** type Double.
- **InterestRate** type Double.
- **AdminFee** type Double.
• **Term** type Double.

The methods all return a value of type Double, which can be used either hold the result of a calculation or to pass an error code.

Three different “CalculationTypes” are available, which are explained in detail in Appendix 5, and the way each calculates the APR, Interest Payable and monthly repayments is based on the following:

- **Capital & Interest (Loan)** – Interest is calculated as if the money was being borrowed for the full term of the loan i.e. no allowance made for the fact that repayments of capital are being made on a monthly basis.

- **Capital & Interest (Mortgage)** – Interest is calculated based on the outstanding borrowing on a daily basis i.e. capital repayments are taken into consideration.

- **Interest Only (Loan or Mortgage)** – Interest is calculated on the outstanding borrowing on a daily basis. No capital is repayable during the term of the borrowing.

The methods that the Calculator provides to allow access to the various calculations are as follows, full descriptions of these methods are included in Appendix 5:

- **ErrorCheck**
- **AnnualPercentageRate**
- **AmountRepayable**
- **MonthlyRepayments**
- **InterestPayable**

### 6.4.2 The Profile Component

This component is likely to be utilised by the business to specify criteria that will then be applied when a client requests a quotation. For this reason the various parameters are initially set as global variables when an instance of the component has been created. It is then these values that the client requirements are compared against prior to the quotation being produced.

The methods provided by this component are as follows, with full descriptions of these and the variables being provided in Appendix 5:

- **ErrorCheck**
- **LoanProfile**.
- **CheckCapital**
- **CheckBorrowingMultiples**.
• AdministrationFee.
• CheckTerm.

6.4.3 The Quotation Component
This component is used to store all the information required to produce a quotation. As it is envisaged that it will interact with other components to gather and set the various parameters, it uses global variables to store the values. This is the most general of the components as it could be used in many different situations to provide different quotes. Because of this the checking of input is passed to the other components, for example, the Quotation has a variable that holds a “CalculationType”, however the Quotation does not validate the value that is set, provided that it is a String. It is only within the “Calculation” component that the value of the “CalculationType” is checked. The “Quote” does not pass any judgement on the input, as it does not know what use is going to be made of it.

The methods provided by this component are as follows, with full descriptions of these and the variables being provided in Appendix 5:

• CollectFormDetails
• SetLoanProfile
• SetLoanCalculations

It is important that the methods be used in the correct order; the “CollectFormDetails” must be called prior to “SetLoanProfile” and then “SetLoanCalculations” to ensure that the necessary values are set correctly.

6.4.4 The Format Component
This component is designed to take the data that is contained within an object such as a Quotation and to then display it onto the client computer screen. This process has been broken down into several steps to assist in the future reuse of the component. At present the component only displays one public method and that is “WriteQuote” and it is through this method that the user can display the quotation. However, “WriteQuote” uses the private methods within the component to construct the display. Each element of data that is to be displayed has a separate private method that is used to produce two strings, the first string holds the formatted value and the second string holds any narrative that is to appear next to the value on the screen. These two strings are then passed to another private method that places them within a suitably formatted line of table output. This formatted table line is then returned to the “WriteQuote” and is sent to the client machine by using the ASP Scripting Context object.

For example, say the value of “IntRate” is 10.555676 and it is to be displayed on the screen. It would be passed to the “DisplayInterestRate” private method, which produces two strings, the first would hold “10.56%” and the second would hold “The flat rate of Interest is: “. These two strings are then passed to the “TableEntry” private method, which inserts the two strings into a formatted line of table output and it is this line that is then sent to the client screen.
By dividing up the formatting process across several methods it is envisaged that further public formatting methods could be added that utilise the same private methods to generate output.

The “Format” component does not maintain any global variables; it obtains all of the values that it requires from the object that is passed to the “WriteQuote” method.

The various private methods that produce the strings for display via the “TableEntry” are as follows, with full descriptions being provided in Appendix 5:

- DisplayCapital
- DisplayInterestRate
- DisplayFeeRate
- DisplayTerm
- DisplayInterest
- DisplayRepayments
- DisplayTotalRepayable
- DisplayAPR
- DisplayFeeCharge

The strings produced by the above private methods are then passed to the “TableEntry” method that adds any background formatting, sets the font and places the two strings in a format that can be displayed in a table. The output from this method is a string, which is returned to the “WriteQuote”.

6.5 Demonstration of the Components

To demonstrate how the four components can be combined to produce a quotation, three ASP pages have been produced:

1. The first page, entitled “TestPage1.asp”, is required to capture the user selections for “Purpose”, “Amount” and “Term” of the borrowing. In the demonstration, the formatting on this page has been omitted to highlight the actual code required to capture the user selections – see Figure 33.
2. The second page, entitled “TestPage2.asp”, is a server side include and contains the details required to create and configure the profiles that the user can choose from, see Figure 34. This page is automatically loaded when the third page is loaded.

Figure 34. TestPage2.asp

<!--
'create our component instances
Set objProfile1 = Server.CreateObject("Loans.Profile")
Set objProfile2 = Server.CreateObject("Loans.Profile")
Set objProfile3 = Server.CreateObject("Loans.Profile")
'and configure the various properties

dblCode1 = objProfile1.LoanProfile("CapInt", 6, 50, 20000, 5000, 1.5, 75, 25, 120, 12)
dblCode2 = objProfile2.LoanProfile("IntOnly", 7, 50, 15000, 3000, 1, 100, 35, 240, 6)
dblCode3 = objProfile3.LoanProfile("CapIntMort", 7.5, 100, 100000, 10000, 2, 150, 50, 480, 12)
'catch any error codes returned and display them
If((dblCode1<0)Or(dblCode2<0)Or(dblCode3<0))Then
  Response.Write ("dblCode1 = \\
  Response.Write ("dblCode2 = \\
  Response.Write ("dblCode3 = \\
End If

%>

Figure 33. TestPage1.asp

```html
<HTML>
<HEAD>
<TITLE>Graham's Bank</TITLE>
</HEAD>
<BODY>
<form name="Form1" action="TestPage3.asp" method="post">
Enter type of borrowing <select name="purpose">
  <option value="1">Personal Loan</option>
  <option value="2">Mortgage</option>
  <option value="3">Car Loan</option>
</select><br>
Enter amount to borrow £ <input type="text" name="capital"><br>
Enter term of borrowing in months <input type="text" name="term"><br>
<input type="submit">
</form>
</BODY>
</HTML>
```
3. The third page entitled “TestPage3.asp” contains the ASP that will be used to produce the HTML that will be displayed on the client screen, see Figure 35.

```
<%@ Language=VBScript %>
<!-- #include file="TestPage2.asp" -->

<%
'create quote and capture form details into object
Set objQuote = Server.CreateObject("Loans.UserQuote")
objQuote.CollectFormDetails

'check which loan profile chosen by user
dblPurpose = objQuote.Purpose
IF (dblPurpose = 1) Then
    set objProfile = objProfile1
ElseIf (dblPurpose = 2) Then
    set objProfile = objProfile2
ElseIf (dblPurpose = 3) Then
    set objProfile = objProfile3
End If

'check input details against profile criteria and update "UserProfile"
dblErrorCheck1 = objQuote.SetLoanProfile (objProfile)
If(dblErrorCheck1 < 0) Then
    Response.Write("Error code: " & dblErrorCheck1 & " received from SetLoanProfile" & "<BR>"
Else
    'create and then pass details to calculate quote
    Set objCalculator = Server.CreateObject("Loans.calculator")
    dblErrorCheck2 = objQuote.SetLoanCalculations (objCalculator)
    If(dblErrorCheck2 < 0) Then
        Response.Write("Error code: " & dblErrorCheck2 & " received from SetLoanCalculations" & "<BR>"
    Else
        'create and then pass details to format quote
        Set objFormat = Server.CreateObject("Loans.Format")
        objFormat.WriteQuote (objQuote)
    End If
End If

'dispose of objects
Set objQuote = Nothing
Set objCalculator = Nothing
Set objProfile = Nothing
Set objProfile1 = Nothing
Set objProfile2 = Nothing
Set objProfile3 = Nothing
Set objFormat = Nothing
%
```

Figure 35. TestPage3.asp

This demonstrates how instances of the four components can be produced and used to combine to provide the functionality for providing quotations on various types of borrowing. The number of lines of code that have to be written within the ASP pages
is minimised, which makes each page easier to understand. Additional borrowing profiles can easily be added to the system by creating a further instance of the profile component in the TestPage2.asp and then specifying the various values. Then the other two ASP pages would only have to be changed to add the new profile to the list of options.

Figure 36 shows an example of the correct output of a quotation. This shows that the various values have been calculated and displayed onto a formatted screen.

![Figure 36. Loan Quotation](image)

The error codes that are generated assist in identifying the source of the error, for example, in Figure 37, an incorrect value for capital has been input. The error code is generated when a negative value for “Capital” is entered, the actual code returned reflects that errors occurred in “Capital” and that “Capital + AdminFee < 0”.

![Figure 37. Error Screen 1](image)

Another example of an error code being generated from another point is when an error is encountered in the values being set for the profile, Figure 38 demonstrates this situation. In this case the error is identified when the profile values are set within “TestPage2.asp”, the interest rate was set as “-7” instead of “7” and the code identifies that the error is related to the interest rate.
A final example demonstrates that an error will be highlighted only if it affects the actual component concerned. In this case the “CalculationType” within the profile is misspelled, however no error is raised until the value is passed to the calculation component:

**Figure 39. Error Screen 3**

The error code identifies that an incorrect “CalculationType” has been passed to the calculator component.

### 6.6 Conclusions

This chapter has covered the implementation of the four components and describes the functionality that each one provides. The components divide up the responsibility of producing a quotation for a loan or mortgage. One component entitled “Quotation” provides instances that are used to hold all the necessary information to formulate a quotation. These instances have a number of parameters or properties that start of empty and are then filled in when the instance is passed to instances of the other components. The “Profile” component holds all the business logic for a particular product and is fully configurable and thus reusable for various loan products. The “Calculator” component performs the arithmetic calculations on the data based on a choice of three different types of loan. The algorithms that are used by each type of calculation are described in Appendix 5. Finally there is a “Format” component that takes the completed “Quote” instance and formats the data for output to the client screen. This approach has produced four components that can be further developed independently of each other to provide additional functionality. Also, the output from any of the components could be passed to different components to produce a different result; for example, the format component could be replaced with a component that writes the data to a data store. By dividing up the task of producing a completed quotation, the individual components are reusable, however, the amount of code and
testing required is greater than if all the functionality was contained within one component. Also the transfer of data from component to component may introduce complications however, in the longer term this approach should reduce both development time and testing for future applications.
7.0 Conclusions

7.1 Introduction
This report has covered a wide area which has meant that it has been necessary to concentrate on only certain aspects of the concept of “e-business”. However, the report is still able to provide an overview of the whole topic.

7.2 E-Business
The concept of “e-business” as an organisation wide approach, encompassing all aspects of business not only with customers but also with suppliers and employees, is an important one. While an “e-commerce” web-site may increase business exposure in the short-term, and thus lead to increased sales, it is the integration of all of the business systems that will provide the major benefits through increased efficiency.

The “Business In The Information Age, International Benchmarking Study 2000” [1], demonstrates that businesses in the United Kingdom are receptive to the concept of “e-business” and that the majority of them, irrespective of size, have already in place at least part of the infrastructure that is necessary to progress towards full “e-business”.

7.3 Technologies
The results from the DTI report [2] indicate that there is a demand for the technologies that are used to build “e-commerce” sites, and in the second part of the report a number of the technologies and methodologies were examined. This is still a developing field and therefore there are a number of different alternatives available. The three main ones revolve around the use of Java products, the use of Microsoft products or the use of a system running on a Unix or Linux operating system. All of these systems should provide satisfactory solutions to constructing an e-commerce site, however this was not attempted within this project as it was discovered that each system required its own particular software systems to be installed before it was usable. For this reason, the project concentrated on the products of one particular manufacturer, namely Microsoft. Microsoft was chosen as it not only has a wide range of software products available, which should allow future expansion of any system developed, but it also provides the operating system that the majority of potential users would be familiar with and already have available. A further consideration was that many of the systems that would in the future be integrated into the overall system may already be running on Microsoft products and while this does not necessarily mean they cannot be integrated into other solutions, it was decided that it made sense to concentrate with the same manufacturer, although this could lead to the business being dependant on a single source for future developments.

The Microsoft solution to providing dynamic web-pages involves constructing pages that are stored on the server and then converted into HTML and sent to the user browser when they are requested. The construction of the individual pages involves combining HTML code, which deals mainly with the static parts of the page, and ASP code, which provides the dynamic behaviour, within the same page. For simple pages this is fairly straightforward, however, when the site starts to expand and the functionality required from the pages increases, this combination of HTML and ASP
becomes extremely difficult to maintain and understand as the amount of coding increases.

7.4 Software Components

The solution is to use a multi-layered architectural design and to remove the functionality from the individual web pages into a separate layer from the presentation layer. This is what software components seek to achieve, this offers a number of benefits in addition to simplifying the individual page design.

Individual components are designed and built using the Component Object Model (COM), this provides a binary standard that allows the internal functionality to be accessed via their interfaces, irrespective of the language in which they were developed.

There are two approaches to developing components, they can either merely encapsulate a particular piece of functionality within a single component or they can be developed in a generic way that allows them to be reused in future applications. The first choice is likely to result in larger, “coarse-grained” components that have to be changed every time an alteration to the processing method is required. However, these components are usually easier and quicker to develop. On the other hand, the generic approach is initially likely to produce a number of “fine-grained” components that have taken longer to produce, but are reusable. When taking the “e-business” view of the future, the extra time and effort in developing the generic components should prove beneficial in the longer term.

Other benefits of software components include:

- **Security of source code** – Components are compiled and therefore the source code is hidden.

- **Speed of Execution** – Components will normally execute faster than scripting languages that are used within ASP pages.

- **Develop/Purchase** – Standard functionality may be available to purchase from Software Developers, which reduces development time and costs. Then only specialised components need to be developed in-house.

- **Reuse** – Generic components developed using COM are reusable in non-web applications, therefore they can be used within in-house applications that use for example Visual Basic interface rather than a browser to display information.
Appendix 1
Storyboard – Financial Services Web Site

1. Initial Screen

Welcome to AnyBank plc
Please select an area to visit:
- Accounts and Services
- Telephone Banking
- Share Dealing

Back  Help

2. Accounts and Services screen

AnyBank plc - Accounts and Services
Please select an area to visit:
- Mortgages
- Loans
- Insurance
- Accounts

Back  Help

3. Mortgages Screen

AnyBank plc - Mortgages Page
Please select an area to visit:
- Endowment
- Capital and Interest
- Fixed Rate

Back  Insurance  Help

4. Loans Screen

AnyBank plc - Loans Page
Please select an area to visit:
- Car Loan
- Holiday Loan
- Personal Loan

Back  Insurance  Help

5. Insurance Screen

AnyBank plc - Insurance Page
Please select an area to visit:
- Home Insurance
- Travel Insurance
- Car Insurance

Back  Help

6. Accounts Screen

AnyBank plc - Accounts Page
Please select an area to visit:
- Current Account
- Saving Account
- Term Account

Back  Insurance  Help

7. Capital and Interest Mortgages

AnyBank plc - Capital + Interest Mortgages
Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help

8. Endowment Mortgages

AnyBank plc - Endowment Mortgages
Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help
9. Fixed Rate Mortgages

AnyBank plc - Fixed Rate Mortgages

Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help

10. Information Capital + Interest Mortgages
(Similar pages for Endowment and Fixed Rate)

AnyBank plc - Capital + Interest Information

Please select an area to visit:
- Information on product 1
- Information on product 2
- Complete Application

Back  Help

11. Application – Capital + Interest Mortgages
(Similar Forms for Endowment + Fixed Rate)

AnyBank plc - Capital + Interest Application

Please complete details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Occupation</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Back  Help

12. Quotation for Capital and Interest Mortgages
(Similar Forms for Endowment + Fixed)

AnyBank plc - Capital + Interest Quotation

Please complete details:

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Term</th>
<th>Etc.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Back  Help

13. Car Loans

AnyBank plc - Car Loan

Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help

14. Personal Loans

AnyBank plc - Personal Loans

Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help

15. Holiday Loans

AnyBank plc - Holiday Loans

Please select an area to visit:
- Information on products
- Obtain quotation
- Complete application

Back  Help

16. Product Information – Car Loans

AnyBank plc - Car Loans Information

Please select an area to visit:
- Information on product 1
- Information on product 2
- Complete Application

Back  Help
17. Application Form – Car Loans

AnyBank plc – Car Loan Application Form

Please complete details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupation</td>
<td>Amount</td>
</tr>
</tbody>
</table>

Back | Help

18. Quotation for Car Loans

AnyBank plc – Car Loan Quotation Form

Please complete details:

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>Etc.</td>
</tr>
</tbody>
</table>

Back | Help

19. Home Insurance

AnyBank plc – Home Insurance

Please select an area to visit:

- Information on products
- Obtain quotation
- Complete application

Back | Help

20. Travel Insurance

AnyBank plc – Travel Insurance

Please select an area to visit:

- Information on products
- Obtain quotation
- Complete application

Back | Help

21. Car Insurance

AnyBank plc – Car Insurance

Please select an area to visit:

- Information on products
- Obtain quotation
- Complete application

Back | Help

22. Product Information – Home Insurance

AnyBank plc – Home Insurance Information

Please select an area to visit:

- Information on product 1
- Information on product 2
- Complete Application

Back | Help

23. Application Form – Home Insurance

AnyBank plc – Home Insurance Application

Please complete details:

<table>
<thead>
<tr>
<th>Name</th>
<th>Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Occupation</td>
<td>Amount</td>
</tr>
</tbody>
</table>

Back | Help

24. Quotation for Home Insurance

AnyBank plc – Home Insurance Quotation

Please complete details:

<table>
<thead>
<tr>
<th>Product</th>
<th>Amount</th>
</tr>
</thead>
<tbody>
<tr>
<td>Term</td>
<td>Etc.</td>
</tr>
</tbody>
</table>

Back | Help
25. Current Accounts

AnyBank plc – Term Account
Please select an area to visit:

- Information on products
- Complete application

Back   Help

26. Saving Accounts

AnyBank plc – Saving Account
Please select an area to visit:

- Information on products
- Complete application

Back   Help

27. Term Accounts

AnyBank plc – Current Account
Please select an area to visit:

- Information on products
- Complete application

Back   Help

28. Product Information – Current Accounts

AnyBank plc – Current Account Information
Please select an area to visit:

- Information on product 1
- Information on product 2
- Complete Application

Back   Help

29. Application Form – Current Accounts

AnyBank plc – Current Account Application
Please complete details:

Name   Address
Occupation   Amount

Back   Help
Appendix 2

Test Specifications for Function Testing of Components

Component - “Calculator”
Function - “ErrorCheck”
Type – “Private”

Private Function ErrorCheck(CalculationType As String, Capital As Double, IntRate As Double, AdminFee As Double, Term As Double) As Double
{
    Dim dblCheck As Double
    dblCheck = 1
    If ((CalculationType <> "CapInt") And (CalculationType <> "CapIntMort") And (CalculationType <> "IntOnly"))
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck - 90000
        }
        Else
        {
            dblCheck = -990000
        }
    } Else
    {
        dblCheck = -990000
    }
    If (Capital <= 0)
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck - 900
        }
        Else
        {
            dblCheck = -909000
        }
    } If (IntRate <= 0)
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck - 900
        }
        Else
        {
            dblCheck = -900900
        }
    } If ((Capital + AdminFee) < 0)
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck – 90
        }
12   Else
13       dblCheck = -900090
14   End If
15 If (Term <= 0)
16       If (dblCheck < 0)
17           dblCheck = dblCheck - 9
18       Else
19           dblCheck = -90009
20       End If
21 ErrorCheck = dblCheck
22 End Function

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph $v(G)$ with $N$ being the number of flow graph nodes
and $E$ is the number of flow graph edges is:

$$v(G) = E - N + 2$$

In the graph below each area within the dotted lines will be calculated separately
Therefore from the graph $v(G) = 5 - 4 + 2 = 3$

There are 5 groups of these, therefore total paths equals $5 \times 3 = 15$
This means that there should be 15 independent paths from which all other paths can
be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 4
Path 2 = 1 to 2 to 4
Path 3 = 1 to 2 to 3 to 4
Path 4 = 4 to 7
Path 5 = 4 to 5 to 7
Path 6 = 4 to 5 to 6 to 7
Path 7 = 7 to 10
Path 8 = 7 to 8 to 10
Path 9 = 7 to 8 to 9 to 10
Path 10 = 10 to 13
Path 11 = 10 to 11 to 13
Path 12 = 10 to 11 to 12 to 13
Path 13 = 13 to 16
Path 14 = 13 to 14 to 16
Path 15 = 13 to 14 to 15 to 16
Flow Graph of ErrorCheck Function:

Tests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: CalculationType</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculationType</td>
<td>Correct input</td>
<td>1</td>
<td>CapInt</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>1</td>
<td>CapIntMort</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>1</td>
<td>IntOnly</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Lower case</td>
<td>2</td>
<td>capint</td>
<td>-1</td>
<td>-990000</td>
</tr>
<tr>
<td></td>
<td>Upper case</td>
<td>3</td>
<td>CAPIXT</td>
<td>1</td>
<td>-990000</td>
</tr>
<tr>
<td></td>
<td>Blank</td>
<td>2</td>
<td>-1</td>
<td>-990000</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Incorrect input</td>
<td>3</td>
<td>CapAndInt</td>
<td>1</td>
<td>-990000</td>
</tr>
</tbody>
</table>
Component - “Calculator”  
Function - “InterestPayable”  
Type – “Public”

Function InterestPayable(CalculationType As String, Capital As Double, InterestRate As Double, AdminFee As Double, Term As Double) As Double
{
    strCalculationType = vCalculationType
    dblCapital = vCapital
    dblTerm = vTerm
    dblAdminFee = vAdminFee
    dblIntRate = vInterestRate

    'check that figure acceptable if not set error code and skip to end
    intCheck = 1
    dblErrorCode = ErrorCheck(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm)

    If (dblErrorCode < 0)
    {
        intCheck = 0
        dblTotalInt = dblErrorCode
    }
2  If ((strCalculationType = "CapIntMort") And (intCheck = 1))
  {
    'Take into account capital repaid and charges
    dblTotalAmount = dblCapital + dblAdminFee
    Dim dblAmount As Integer
    For dblAmount = 0 To dblTerm
      dblTotalInt = dblTotalInt + (((dblTotalAmount / dblTerm) * dblAmount)
                              * (1 / 100))
    Next dblAmount
  }
3  ElseIf ((strCalculationType = "CapInt") And (intCheck = 1))
  {
    'Declare local variables
    Dim dblAnnualInt As Double
    'To calculate the total interest payable the sum borrowed
    'including any charges) is multiplied by the interest rate
    dblAnnualInt = (dblCapital + dblAdminFee) * (dblIntRate / 100)
    'The result is then multiplied by the term in years
    dblTotalInt = dblAnnualInt * (dblTerm / 12)
  }
4  ElseIf ((strCalculationType = "IntOnly") And (intCheck = 1))
  {
    dblTotalAmount = dblCapital + dblAdminFee
    'Capital outstanding remains unchanged - so calculate interest for one year
    'and multiply by number of years
    dblTotalInt = (dblTotalAmount * (dblIntRate * 1 / 100)) * (dblTerm / 12)
  }
7  InterestPayable = dblTotalInt
}

The number of distinct paths that need to be tested can be calculated: 
Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes
and \( E \) is the number of flow graph edges is:
\[
v(G) = E - N + 2
\]
Therefore graph below \( v(G) = 10 - 7 + 2 \)
\[
= 5
\]
This means that there should be 5 independent paths from which all other paths can be
generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 7
Path 2 = 1 to 2 to 3 to 4 to 7
Path 3 = 1 to 2 to 3 to 7
Path 4 = 1 to 2 to 5 to 6 to 7
Path 5 = 1 to 2 to 5 to 6 to 5 to 6 to 7
Flow Graph of InterestPayable Function:

Tests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: CalculationType</th>
<th>Value: Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculationType</td>
<td>Correct input 3</td>
<td>3</td>
<td>CapInt</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 5</td>
<td>5</td>
<td>CapIntMort</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 4</td>
<td>4</td>
<td>CapIntMort</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 2</td>
<td>2</td>
<td>IntOnly</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Lower case    1</td>
<td>1</td>
<td>capint</td>
<td>2</td>
<td>-990000</td>
</tr>
<tr>
<td></td>
<td>Upper case    1</td>
<td>1</td>
<td>CAPINT</td>
<td>2</td>
<td>-990000</td>
</tr>
<tr>
<td></td>
<td>Blank         1</td>
<td>1</td>
<td></td>
<td>2</td>
<td>-990000</td>
</tr>
<tr>
<td></td>
<td>Incorrect input 1</td>
<td>1</td>
<td>CapAndInt</td>
<td>2</td>
<td>-990000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: Capital</th>
<th>Value: Term</th>
<th>Value: CalcType</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital</td>
<td>Correct input 3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>CapInt</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 5</td>
<td>5</td>
<td>999999999</td>
<td>2</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 4</td>
<td>4</td>
<td>999999.99</td>
<td>1</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 2</td>
<td>2</td>
<td>5555.55</td>
<td>2</td>
<td>IntOnly</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input 1</td>
<td>1</td>
<td>-999999.9999</td>
<td>2</td>
<td>CapInt</td>
<td>-909000</td>
</tr>
<tr>
<td></td>
<td>Zero input    1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>CapIntMort</td>
<td>-909000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: InterestRate</th>
<th>Value: Term</th>
<th>Value: CalcType</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterestRate</td>
<td>Correct input 3</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>CapInt</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 5</td>
<td>5</td>
<td>999999999</td>
<td>2</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 4</td>
<td>4</td>
<td>999999.99</td>
<td>1</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 2</td>
<td>2</td>
<td>5555.55</td>
<td>2</td>
<td>IntOnly</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input 1</td>
<td>1</td>
<td>-999999.9999</td>
<td>2</td>
<td>CapInt</td>
<td>-909000</td>
</tr>
<tr>
<td></td>
<td>Zero input    1</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>CapIntMort</td>
<td>-909000</td>
</tr>
</tbody>
</table>
Graham Bell, BEng (Honours) Software Engineering, 2001

Parameter | Test | Path | Value: AdminFee | Value: Term | Value: CalcType | Expected Result
---|---|---|---|---|---|---
AdminFee | Correct input | 3 | 1 | 2 | CapInt | Ok
Correct input | 5 | 9999999 | 2 | CapIntMort | Ok
Correct input | 4 | 99999.99 | 1 | CapIntMort | Ok
Correct input | 2 | 5555.55 | 2 | IntOnly | Ok
Negative input | 3 | -99999.99 | 2 | CapInt | Ok
Zero input | 5 | 0 | 2 | CapIntMort | Ok

Parameter | Test | Path | Value: Term | Value: CalcType | Expected Result
---|---|---|---|---|---
Term | Correct input | 3 | 2 | CapInt | Ok
Correct input | 5 | 9999999 | CapIntMort | Ok
Correct input | 4 | 1 | CapIntMort | Ok
Correct input | 2 | 5555.55 | IntOnly | Ok
Negative input | 1 | -99999.99 | CapInt | -900009
Zero input | 1 | 0 | CapIntMort | -900009

Test Calculations:

<table>
<thead>
<tr>
<th>CalculationType</th>
<th>Capital</th>
<th>InterestRate</th>
<th>AdminFee</th>
<th>Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapInt</td>
<td>9999</td>
<td>999</td>
<td>99</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>CapIntMort</td>
<td>999999</td>
<td>99</td>
<td>9</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>IntOnly</td>
<td>9999999</td>
<td>9</td>
<td>999</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

Component - “Calculator”
Function - “AmountRepayable”
Type – “Public”

Public Function AmountRepayable(CalculationType As String, Capital As Double, InterestRate As Double, AdminFee As Double, Term As Double) As Double
{
    strCalculationType = CalculationType
    dblCapital = Capital
    dblTerm = Term
    dblAdminFee = AdminFee
    dblIntRate = InterestRate

    'check that figure acceptable if not set error code and skip to end
    dblErrorCode = ErrorCheck(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm)
    1 If (dblErrorCode < 0)  
        {  
            dblTotalRepayable = dblErrorCode  
        }
    2 Else
        {  
            'Sum of the amount borrowed, the total interest payable and charges  
            dblTotalRepayable = dblCapital + InterestPayable(strCalculationType, dblCapital, dblIntRate,  
            dblAdminFee, dblTerm) + dblAdminFee  
        }

    'Assign to function variable
    3 AmountRepayable = dblTotalRepayable
}

90
Flow Graph of AmountRepayable Function:

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph $v(G)$ with $N$ being the number of flow graph nodes
and $E$ is the number of flow graph edges is:

$$v(G) = E - N + 2$$

Therefore from the above graph $v(G) = 3 - 3 + 2 = 2$

This means that there should be 2 independent paths from which all other paths can be
generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 3

Path 2 = 1 to 2 to 3

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: dblErrorCode</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblErrorCode</td>
<td>Correct input</td>
<td>1 1</td>
<td></td>
<td>Ok</td>
</tr>
<tr>
<td>Correct input</td>
<td>2</td>
<td>-900000</td>
<td></td>
<td>-900000</td>
</tr>
</tbody>
</table>

Test Calculations:

<table>
<thead>
<tr>
<th>CalculationType</th>
<th>Capital</th>
<th>InterestRate</th>
<th>AdminFee</th>
<th>Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapInt</td>
<td>9999</td>
<td>999</td>
<td>99</td>
<td>999</td>
<td></td>
</tr>
<tr>
<td>CapIntMort</td>
<td>999999</td>
<td>99</td>
<td>9</td>
<td>999</td>
<td></td>
</tr>
<tr>
<td>IntOnly</td>
<td>9999999</td>
<td>9</td>
<td>999</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

Component - “Calculator”
Function - “MonthlyRepayments”
Type – “Public”

Function MonthlyRepayments(CalculationType As String, Capital As Double, InterestRate As Double, AdminFee As Double, Term As Double) As Double

{}
strCalculationType = CalculationType
dblCapital = Capital
dblAdminFee = AdminFee
dblTerm = Term
dblIntRate = InterestRate
'check that figure acceptable if not set error code and skip to end
intCheck = 1
dblErrorCode = ErrorCheck(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm)
1 If (dblErrorCode < 0)
   { intCheck = 0
     dblMonthlyRepayments = dblErrorCode
   }
2 If ((strCalculationType = "CapIntMort") And (intCheck = 1))
   { 'Add together the total interest payable, the capital amount and charges
     dblTotalPayable = InterestPayable(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm) + dblCapital + dblAdminFee
     'then divide by the term of the loan in months
     dblMonthlyRepayments = dblTotalPayable / dblTerm
   }
3 ElseIf ((strCalculationType = "CapInt") And (intCheck = 1))
   { 'Add together the total interest payable, the capital amount and charges
     dblTotalPayable = InterestPayable(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm) + dblCapital + dblAdminFee
     'then divide by the term of the loan in months
     dblMonthlyRepayments = dblTotalPayable / dblTerm
   }
4 ElseIf ((strCalculationType = "IntOnly") And (intCheck = 1))
   { 'Only Interest repayable monthly with this type of borrowing assign to function variable
     dblMonthlyRepayments = InterestPayable(strCalculationType, dblCapital, dblIntRate, dblAdminFee, dblTerm) / dblTerm
   }
5 'assign result to function variable
MonthlyRepayments = dblMonthlyRepayments
}
End Function

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph v(G) with N being the number of flow graph nodes
and E is the number of flow graph edges is:
v(G) = E - N + 2
Therefore from the graph below v(G) = 10 - 8 + 2
= 4
This means that there should be 4 independent paths from which all other paths can be
generated. The paths that need to be tested are therefore as follows:
Path 1 = 1 to 5
Path 2 = 1 to 2 to 6 to 5
Path 3 = 1 to 2 to 3 to 7 to 5
Path 4 = 1 to 2 to 3 to 4 to 8 to 5
Flow Graph of MonthlyRepayments Function:

Tests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: CalculationType</th>
<th>Value: Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CalculationType</td>
<td>Correct input</td>
<td>3</td>
<td>CapInt</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 2</td>
<td>2</td>
<td>CapIntMort</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input 4</td>
<td>1</td>
<td>IntOnly</td>
<td>2</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Lower case</td>
<td>1</td>
<td>capInt</td>
<td>2</td>
<td>990000</td>
</tr>
<tr>
<td></td>
<td>Upper case</td>
<td>1</td>
<td>CAPINT</td>
<td>2</td>
<td>990000</td>
</tr>
<tr>
<td></td>
<td>Blank</td>
<td>1</td>
<td></td>
<td>2</td>
<td>990000</td>
</tr>
<tr>
<td></td>
<td>Incorrect input</td>
<td>1</td>
<td>CapAndInt</td>
<td>2</td>
<td>-990000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: Capital</th>
<th>Value: Term</th>
<th>Value: CalcType</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Capital</td>
<td>Correct input</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>CapInt</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>2</td>
<td>999999999</td>
<td>2</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>2</td>
<td>9999999.99</td>
<td>1</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>5555.55</td>
<td>2</td>
<td>IntOnly</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>1</td>
<td>-9999999.99</td>
<td>2</td>
<td>CapInt</td>
<td>-909000</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>CapIntMort</td>
<td>-909000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: InterestRate</th>
<th>Value: Term</th>
<th>Value: CalcType</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>InterestRate</td>
<td>Correct input</td>
<td>3</td>
<td>1</td>
<td>2</td>
<td>CapInt</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>2</td>
<td>999999999</td>
<td>2</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>2</td>
<td>9999999.99</td>
<td>1</td>
<td>CapIntMort</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>5555.55</td>
<td>2</td>
<td>IntOnly</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>1</td>
<td>-9999999.99</td>
<td>2</td>
<td>CapInt</td>
<td>-900900</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>1</td>
<td>0</td>
<td>2</td>
<td>CapIntMort</td>
<td>-900900</td>
</tr>
</tbody>
</table>
Parameter | Test | Path | Value: AdminFee | Value: Term | Value: CalcType | Expected Result
--- | --- | --- | --- | --- | --- | ---
AdminFee | Correct input | 3 | 1 | 2 | CapInt | Ok
Correct input | 2 | 9999999 | 2 | CapIntMort | Ok
Correct input | 2 | 99999.99 | 1 | CapIntMort | Ok
Correct input | 4 | 5555.55 | 2 | IntOnly | Ok
Negative input | 1 | -99999.99 | 2 | CapInt | Ok
Zero input | 2 | 0 | 2 | CapIntMort | Ok

Parameter | Test | Path | Value: Term | Value: CalcType | Expected Result
--- | --- | --- | --- | --- | ---
Term | Correct input | 3 | 2 | CapInt | Ok
Correct input | 2 | 9999999 | CapIntMort | Ok
Correct input | 2 | 1 | CapIntMort | Ok
Correct input | 4 | 5555.55 | IntOnly | Ok
Negative input | 1 | -99999.99 | CapInt | -900009
Zero input | 1 | 0 | CapIntMort | -900009

Test Calculations

<table>
<thead>
<tr>
<th>CalculationType</th>
<th>Capital</th>
<th>InterestRate</th>
<th>AdminFee</th>
<th>Term</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>CapInt</td>
<td>9999</td>
<td>999</td>
<td>99</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>CapIntMort</td>
<td>999999</td>
<td>99</td>
<td>9</td>
<td>99</td>
<td></td>
</tr>
<tr>
<td>IntOnly</td>
<td>9999999</td>
<td>9</td>
<td>999</td>
<td>999</td>
<td></td>
</tr>
</tbody>
</table>

Component - “Profile”

Function - “ErrorCheck”

Type – “Private”

Function ErrorCheck(IntRate As Double, BorrowMulti As Double, MaxCapital As Double, MinCapital As Double, FeeRate As Double, MaxFee As Double, MinFee As Double, MaxTerm As Double, MinTerm As Double) As Double

{ Dim dblCheck As Double

    dblCheck = 1

    1 If (IntRate <= 0) Then

    2     If (dblCheck < 0) Then

    3         dblCheck = dblCheck - 800000

    4     Else

    5         dblCheck = -8800000

    6     End If

    7 End If

    4 If (BorrowMulti <= 0) Then

    5     If (dblCheck < 0) Then

    6         dblCheck = dblCheck - 800000

    7     Else

    8         dblCheck = -8080000

    9     End If

10 End If

11 End Function
If ((MaxCapital <= 0) Or (MinCapital <= 0) Or (MaxCapital < MinCapital))
{
  If (dblCheck < 0)
  {
    dblCheck = dblCheck - 8000
  }
  Else
  {
    dblCheck = -8008000
  }
}

If (FeeRate < 0)
{
  If (dblCheck < 0)
  {
    dblCheck = dblCheck - 800
  }
  Else
  {
    dblCheck = -8000800
  }
}

If ((MaxFee <= 0) Or (MaxFee < MinFee))
{
  If (dblCheck < 0)
  {
    dblCheck = dblCheck - 80
  }
  Else
  {
    dblCheck = -8000080
  }
}

If ((MaxTerm <= 0) Or (MinTerm <= 0) Or (MaxTerm < MinTerm))
{
  If (dblCheck < 0)
  {
    dblCheck = dblCheck - 8
  }
  Else
  {
    dblCheck = -8000008
  }
}

ErrorCheck = dblCheck

} End Function
The number of distinct paths that need to be tested can be calculated:

Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes and \( E \) is the number of flow graph edges is:

\[
v(G) = E - N + 2
\]

In the above graph each area within the dotted lines will be calculated separately
Therefore from the above graph $v(G) = 5 - 4 + 2 = 3$

There are 6 groups of these, therefore total paths equals $6 \times 3 = 18$

This means that there should be 18 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 4  
Path 2 = 1 to 2 to 4  
Path 3 = 1 to 2 to 3 to 4  
Path 4 = 4 to 7  
Path 5 = 4 to 5 to 7  
Path 6 = 4 to 5 to 6 to 7  
Path 7 = 7 to 10  
Path 8 = 7 to 8 to 10  
Path 9 = 7 to 8 to 9 to 10  
Path 10 = 10 to 13  
Path 11 = 10 to 11 to 13  
Path 12 = 10 to 11 to 12 to 13  
Path 13 = 13 to 16  
Path 14 = 13 to 14 to 16  
Path 15 = 13 to 14 to 15 to 16  
Path 16 = 16 to 19  
Path 17 = 16 to 17 to 19  
Path 18 = 16 to 17 to 18 to 19

 Tests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: IntRate</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>IntRate</td>
<td>Correct input</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>1</td>
<td>999999999</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>1</td>
<td>999999.99</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>1</td>
<td>5555.55</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>3</td>
<td>-9999999.99</td>
<td>1</td>
<td>-80080000</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>2</td>
<td>0</td>
<td>-1</td>
<td>-88000000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: IntRate</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>BorrowMulti</td>
<td>Correct input</td>
<td>4</td>
<td>1</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>999999999</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>999999.99</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>4</td>
<td>5555.55</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>6</td>
<td>-9999999.99</td>
<td>1</td>
<td>-80800000</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>5</td>
<td>0</td>
<td>-1</td>
<td>-80800000</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Parameter: Capital</th>
<th>Test</th>
<th>Path</th>
<th>Value: MaxCapital</th>
<th>Value: MinCapital</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max Min</td>
<td>Correct input</td>
<td>7</td>
<td>55555</td>
<td>1</td>
<td>1</td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>9</td>
<td>999999999</td>
<td>-1</td>
<td>1</td>
<td>-80080000</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>9</td>
<td>-9999999.99</td>
<td>1</td>
<td>1</td>
<td>-80080000</td>
</tr>
<tr>
<td></td>
<td>Min&gt;Max</td>
<td>9</td>
<td>5555.55</td>
<td>200000</td>
<td>1</td>
<td>-80080000</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>8</td>
<td>-9999999.99</td>
<td>9999999</td>
<td>-1</td>
<td>-80080000</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-80080000</td>
</tr>
</tbody>
</table>
Parameter:  

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Test</th>
<th>Path</th>
<th>Value: FeeRate</th>
<th>Value: MinFee</th>
<th>Value: MaxFee</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>FeeRate</td>
<td>Correct input</td>
<td>10</td>
<td>1</td>
<td></td>
<td></td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>10</td>
<td>999999999</td>
<td></td>
<td></td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Correct input</td>
<td>10</td>
<td>99999.99</td>
<td></td>
<td></td>
<td>Ok</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>12</td>
<td>-999999.99</td>
<td></td>
<td></td>
<td>-8000800</td>
</tr>
<tr>
<td></td>
<td>Zero input</td>
<td>10</td>
<td>0</td>
<td></td>
<td></td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td>Negative input</td>
<td>11</td>
<td>-1</td>
<td></td>
<td></td>
<td>-8000800</td>
</tr>
</tbody>
</table>

Parameter:  

<table>
<thead>
<tr>
<th>Fee</th>
<th>Test</th>
<th>Path</th>
<th>Value: MaxFee</th>
<th>Value: MinFee</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Min</td>
<td>Correct input</td>
<td>13</td>
<td>55555</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>13</td>
<td>999999999</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>15</td>
<td>-999998.99</td>
<td>-999999.99</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min&gt;Max</td>
<td>15</td>
<td>5555.55</td>
<td>200000</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>14</td>
<td>-999999.99</td>
<td>999999</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero input</td>
<td>14</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
</tbody>
</table>

Parameter:  

<table>
<thead>
<tr>
<th>Term</th>
<th>Test</th>
<th>Path</th>
<th>Value: MaxTerm</th>
<th>Value: MinTerm</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Max</td>
<td>Min</td>
<td>Correct input</td>
<td>16</td>
<td>55555</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>18</td>
<td>999999999</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>18</td>
<td>-999999.99</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Min&gt;Max</td>
<td>18</td>
<td>5555.55</td>
<td>200000</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Negative input</td>
<td>17</td>
<td>-999999.99</td>
<td>999999</td>
<td>-1</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Zero input</td>
<td>17</td>
<td>0</td>
<td>0</td>
<td>-1</td>
</tr>
</tbody>
</table>

Component - “Profile”  
Function - “CheckCapital”  
Type – “Public”

Function CheckCapital(Capital As Double) As Double  
{
    Dim dblCapital As Double
    dblCapital = vCapital
    'check that "capital" is valid
    1 If (dblCapital <= 0)
    {  
    2    dblCapital = -80008000
    }
    3 Else
    {
        'check that "capital" within borrowing bounds and set variables accordingly
    4        If (dblCapital < mvarMinCapital)
    {  
    5            dblCapital = mvarMinCapital
    mvarCheckCapital = -1
    mvarCapitalMultiple = 0
    }
    6        ElseIf (dblCapital > mvarMaxCapital)
    {  
    7            dblCapital = mvarMaxCapital
    mvarCheckCapital = 1
    mvarCapitalMultiple = 0
    }
The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes
and \( E \) is the number of flow graph edges is:
\[
    v(G) = E - N + 2
\]
Therefore from the above graph \( v(G) = 8 - 6 + 2 \)
\[
    = 4
\]
This means that there should be 4 independent paths from which all other paths can be
generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 2 to 6

Path 2 = 1 to 3 to 6

Path 3 = 1 to 3 to 4 to 6

Path 4 = 1 to 3 to 5 to 6

Tests:

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: dblCapital</th>
<th>Value: minCapital</th>
<th>Value: maxCapital</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-80008000</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1</td>
<td>99</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>1</td>
<td>9</td>
<td>99</td>
<td>9</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>45</td>
<td>89</td>
<td>89</td>
</tr>
</tbody>
</table>
Component - “Profile”
Function - “CheckBorrowingMultiples”
Type – “Public”

Function CheckBorrowingMultiples(Capital As Double) As Double
{
    dblCapital = Capital
    'check that "capital" is valid
    If (dblCapital <= 0)
    {
        dblCapital = -80080000
    }
    Else
    {
        'check that "capital" is a multiple of "mvarBorrowingMultiples"
        dblAns = dblCapital Mod mvarBorrowingMultiples
        If (dblAns > 0)
        {
            dblInc = mvarBorrowingMultiples - dblAns
            dblCapital = dblCapital + dblInc
            mvarCapitalMultiple = 1
        }
    }
    CheckBorrowingMultiples = dblCapital
} End Function

Flow Graph of CheckBorrowingMultiples Function:

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph v(G) with N being the number of flow graph nodes
and E is the number of flow graph edges is:
\[ v(G) = E - N + 2 \]
Therefore from the above graph \( v(G) = 6 - 5 + 2 \)
\[ = 3 \]
This means that there should be 4 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 2 to 5

Path 2 = 1 to 3 to 5

Path 3 = 1 to 3 to 4 to 5

Tests:

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: Capital</th>
<th>Value: Borrowing Multiples</th>
<th>Value: dblAns</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-80080000</td>
</tr>
<tr>
<td>2</td>
<td>9</td>
<td>1</td>
<td>0</td>
<td>9</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>4</td>
<td>1</td>
<td>12</td>
</tr>
</tbody>
</table>

Component - “Profile”
Function - “AdminFee”
Type – “Public”

Function AdministrationFee(Capital As Double) As Double
{
    dblCapital = vCapital
    'check that "capital" is valid
    1 If (dblCapital <= 0)
    {
        2            dblFee = -8000080
    } Else
    {
        'check which admin fee applies - i.e. min, max or somewhere between
        4 If (dblFee <= mvarMinFee)
        {
            dblFee = mvarMinFee
            mvarCheckFee = -1
        }
        5 ElseIf (dblFee >= mvarMaxFee)
        {
            dblFee = mvarMaxFee
            mvarCheckFee = 1
        }
    }
    AdministrationFee = dblFee
} End Function
The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph $v(G)$ with $N$ being the number of flow graph nodes
and $E$ is the number of flow graph edges is:
$$v(G) = E - N + 2$$
Therefore from the above graph $v(G) = 8 - 6 + 2$
$$= 4$$
This means that there should be 4 independent paths from which all other paths can be
generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 2 to 6
Path 2 = 1 to 3 to 6
Path 3 = 1 to 3 to 4 to 6
Path 4 = 1 to 3 to 4 to 5 to 6

Tests:

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: Capital</th>
<th>Value: FeeRate</th>
<th>Value: dblFee</th>
<th>Value: MinFee</th>
<th>Value: MaxFee</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-8000080</td>
</tr>
<tr>
<td>2</td>
<td>99</td>
<td>10</td>
<td>9.9</td>
<td>1</td>
<td>15</td>
<td>9.9</td>
</tr>
<tr>
<td>3</td>
<td>99</td>
<td>10</td>
<td>9.9</td>
<td>12</td>
<td>20</td>
<td>12</td>
</tr>
<tr>
<td>4</td>
<td>99</td>
<td>10</td>
<td>9.9</td>
<td>1</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
**Component - “Profile”**
**Function - “CheckTerm”**
**Type – “Public”**

Function CheckTerm(Term As Double) As Double
{
    dblTerm = vTerm
    'check that "term" is valid
    If (dblTerm <= 0)
    {
        dblTerm = -80000008
    }
    Else
    {
        'check that the term is within bounds
        If dblTerm < mvarMinTerm
        {
            dblTerm = mvarMinTerm
            mvarCheckTerm = -1
        }
        ElseIf dblTerm > mvarMaxTerm
        {
            dblTerm = mvarMaxTerm
            mvarCheckTerm = 1
        }
    }
    CheckTerm = dblTerm
} End Function

**Flow Graph of CheckTerm Function:**

![Flow Graph of CheckTerm Function](image-url)
The number of distinct paths that need to be tested can be calculated: -

Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes and \( E \) is the number of flow graph edges is:

\[
v(G) = E - N + 2
\]

Therefore from the above graph \( v(G) = 10 - 8 + 2 = 4 \)

This means that there should be 4 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 2 to 8

Path 2 = 1 to 3 to 8

Path 3 = 1 to 3 to 4 to 5 to 8

Path 4 = 1 to 3 to 4 to 6 to 7 to 8

Tests:

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: Term</th>
<th>Value: MinTerm</th>
<th>Value: MaxTerm</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>-8000008</td>
</tr>
<tr>
<td>2</td>
<td>99</td>
<td>10</td>
<td>999</td>
<td>99</td>
</tr>
<tr>
<td>3</td>
<td>9</td>
<td>10</td>
<td>99</td>
<td>10</td>
</tr>
<tr>
<td>4</td>
<td>999</td>
<td>10</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>

**Component - “Profile”**

**Function - “LoanProfile”**

**Type – “Public”**

Function LoanProfile(CalculationType As String, IntRate As Double, BorrowMulti As Double, MaxCapital As Double, MinCapital As Double, FeeRate As Double, MaxFee As Double, MinFee As Double, MaxTerm As Double, MinTerm As Double) As Double

```vbnet
'check that figure acceptable if not set error code and skip to end
intCheck = 1
dblErrorCode = ErrorCheck(IntRate, BorrowMulti, MaxCapital, MinCapital, FeeRate, MaxFee, MinFee, MaxTerm, MinTerm)
1 If (dblErrorCode < 0)
   intCheck = 0
2   dblLoanProfile = dblErrorCode
   }
3 If (intCheck = 1)
   {
    mvarCalculationType = CalculationType
    mvarIntRate = IntRate
    mvarBorrowingMultiples = BorrowMulti
    mvarMaxCapital = MaxCapital
    mvarMinCapital = MinCapital
    mvarFeeRate = FeeRate
    mvarMaxFee = MaxFee
```

104
Flow Graph of LoanProfile Function:

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes and \( E \) is the number of flow graph edges is:

\[ v(G) = E - N + 2 \]

Therefore from the above graph \( v(G) = 6 - 5 + 2 = 3 \)

This means that there should be 3 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 3 to 5
Path 2 = 1 to 2 to 3 to 5
Path 3 = 1 to 3 to 4 to 5

Tests:

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Path</th>
<th>Value: dblErrorCode</th>
<th>Value: intCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>dblErrorCode</td>
<td>3</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>2</td>
<td>-9999999</td>
<td>0</td>
<td>Negative figure</td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>99999.99</td>
<td>-1</td>
<td>0</td>
</tr>
</tbody>
</table>
Component - “Quote”
Function - “SetLoanProfile”
Type – “Public”

Function SetLoanProfile(Profile As Object) As Double
{
    dblCheck = 0
    'check valid value and set the properties based on loan profile
    dblCapital = Profile.CheckCapital(mvarCapital)
    If (dblCapital > -8000000)
    {
        mvarCapital = dblCapital
    }
    Else
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck + dblCapital + 8000000
        }
        Else
        {
            dblCheck = dblCapital
        }
    }
    dblMulti = Profile.CheckBorrowingMultiples(mvarCapital)
    If (dblMulti > -8000000)
    {
        mvarCapital = dblMulti
    }
    Else
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck + dblMulti + 8000000
        }
        Else
        {
            dblCheck = dblMulti
        }
    }
    dblTerm = Profile.CheckTerm(mvarTerm)
    If (dblTerm > -8000000)
    {
        mvarTerm = dblTerm
    }
    Else
    {
        If (dblCheck < 0)
        {
            dblCheck = dblCheck + dblTerm
        }
        Else
        {
            dblCheck = dblTerm
        }
    }
dblFee = Profile.AdministrationFee(mvarCapital)

If (dblFee >= -8000000)
{
    mvarFeeAmount = dblFee
}
Else
{
    If (dblCheck < 0)
    {
        dblCheck = dblCheck + dblFee
    }
    Else
    {
        dblCheck = dblFee
    }
}

mvarCalculationType = vProfile.CalculationType
mvarIntRate = vProfile.InterestRate
mvarFeeRate = vProfile.AdministrationRate
mvarCheckTerm = vProfile.CheckTermBounds
mvarCheckCapital = vProfile.CheckCapitalBounds
mvarCheckMultiple = vProfile.CapitalMultiple
mvarCheckFee = vProfile.CheckFeeBounds
SetLoanProfile = dblCheck

The number of distinct paths that need to be tested can be calculated: -
Cyclomatic complexity of a graph \( v(G) \) with \( N \) being the number of flow graph nodes and \( E \) is the number of flow graph edges is:
\[
    v(G) = E - N + 2
\]
In the graph below each area within the dotted lines will be calculated separately
Therefore from the below graph \( v(G) = 7 - 6 + 2 \)
\[
    = 3
\]
There are 4 groups of these, therefore total paths equals \( 4 \times 3 = 12 \)
This means that there should be 12 independent paths from which all other paths can be generated. The paths that need to be tested are therefore as follows:

Path 1 = 1 to 2 to 6
Path 2 = 1 to 3 to 5 to 6
Path 3 = 1 to 3 to 4 to 6
Path 4 = 6 to 7 to 11
Path 5 = 6 to 8 to 10 to 11
Path 6 = 6 to 8 to 9 to 11
Path 7 = 11 to 12 to 16
Path 8 = 11 to 13 to 15 to 16
Path 9 = 11 to 13 to 14 to 16
Path 10 = 16 to 17 to 21
Path 11 = 16 to 18 to 20 to 21
Path 12 = 16 to 18 to 19 to 21
Flow Graph of SetLoanProfile Function:

Tests:

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: dblCapital</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>9999</td>
<td>0</td>
<td>9999</td>
</tr>
<tr>
<td>2</td>
<td>-8008000</td>
<td>0</td>
<td>-8008000</td>
</tr>
<tr>
<td>3</td>
<td>-8008000</td>
<td>-8000080</td>
<td>-8008080</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: dblMulti</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>4</td>
<td>9999</td>
<td>0</td>
<td>9999</td>
</tr>
<tr>
<td>5</td>
<td>-8008000</td>
<td>0</td>
<td>-8000008</td>
</tr>
<tr>
<td>6</td>
<td>-8008000</td>
<td>-8000080</td>
<td>-8008080</td>
</tr>
<tr>
<td>Path</td>
<td>Value: dblTerm</td>
<td>Value: dblCheck</td>
<td>Expected Result</td>
</tr>
<tr>
<td>------</td>
<td>----------------</td>
<td>-----------------</td>
<td>-----------------</td>
</tr>
<tr>
<td>7</td>
<td>9999</td>
<td>0</td>
<td>9999</td>
</tr>
<tr>
<td>8</td>
<td>-8000008</td>
<td>0</td>
<td>-8000008</td>
</tr>
<tr>
<td>9</td>
<td>-8000008</td>
<td>-8000080</td>
<td>-8000088</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Path</th>
<th>Value: dblFee</th>
<th>Value: dblCheck</th>
<th>Expected Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>9999</td>
<td>0</td>
<td>9999</td>
</tr>
<tr>
<td>11</td>
<td>-8000080</td>
<td>0</td>
<td>-8000080</td>
</tr>
<tr>
<td>12</td>
<td>-8000080</td>
<td>-8000008</td>
<td>-8000088</td>
</tr>
</tbody>
</table>
Appendix 3

Description of Component Parameters and Functions

Profile Component

The following list of properties or parameters should be available for configuration:

- **Interest Rate** - The flat rate of interest payable on any borrowing.

- **Calculation Type** – This specifies whether borrowing repayable by capital and interest repayments or only interest each month with a lump sum repayment of the capital at the end etc.

- **Borrowing Multiples** – Which specify, for instance, whether the borrowing must be in multiples of £50 or £100 etc.

- **Maximum and Minimum Borrowing** – This specifies the maximum sum that can be borrowed and also the minimum amount that can be borrowed.

- **Fee Rate** – This is the percentage rate that any administration fee will be charged (calculated on the amount borrowed).

- **Maximum and Minimum Fee** – This allows the product to have a maximum and minimum fee amount i.e. if the percentage fee produces a figure above or below the maximum or minimum then it is increased to the minimum or decreased to the maximum as appropriate.

- **Maximum and Minimum Repayment Term** - This allows the maximum and minimum length of time, in months, that the particular borrowing can be taken over. Similar to the maximum and minimum fee, the figure is adjusted if it falls out with the stipulated parameters.

The following parameters have been identified as necessary to act as internal flags that can be used to confirm whether certain actions have been performed:

- **Check Capital Bounds**– This records whether the user input figure for capital has been adjusted because it falls outside the allowed boundaries of this type of borrowing.

- **Check Multiples Bounds** – This records whether the user input figure for capital has been adjusted because it is not an allowed borrowing multiple.

- **Check Term Bounds** – This records whether the user input figure for repayment term has been adjusted because it falls outside the allowed boundaries of this type of borrowing.
• **Check Fee Bounds** - This is set to reflect whether the user input figure for capital produced an administration fee outside of the allowed boundaries of this type of borrowing and that therefore the figure has been adjusted.

The functions on its interface to enable it to provide the functionality that the business requires from it are:

- **Profile** – This is the constructor that creates an instance of the component.

- **Loan Profile** – This is used to set the values of various properties of the component referred to above.

- **Check Capital** – This accepts a numerical input, representing the requested capital to be borrowed and it checks whether it is within the maximum and minimum borrowing specifications. If it falls out with these, it is adjusted accordingly and the parameter to signify that the figure has been altered is set. This function returns the applicable capital sum.

- **Check Term** - This accepts a numerical input, representing the term in months that the borrowing is to be taken over and it checks whether it is within the maximum and minimum permitted repayment periods. If it falls out with these, it is adjusted accordingly and the parameter to signify that the figure has been altered is set. This function returns the applicable repayment term.

- **Check Administration Fee** - This accepts a numerical input, representing the capital borrowed, and it checks whether the fee payable when calculated using the percentage rate is within the maximum and minimum fee specifications. If it falls out with these, it is adjusted accordingly and the parameter to signify that the figure has been altered is set. This function returns the applicable fee payable.

It is important that the capital figure is checked prior to the administration fee being calculated as this could affect the fee payable.

**Calculation Component**

The functions on its interface to enable it to provide the functionality that the business requires from it are:

- **Calculation** – This is the constructor that create an instance of the component.

- **Annual Percentage Rate (APR)** – This calculates the equivalent APR based on the details input.

- **Amount Repayable** – This calculates the total amount repayable on the particular loan, taking into consideration the capital, interest and fee.

- **Monthly Repayments** – This calculates the monthly repayments on the borrowing.
• **Interest Payable** – This calculates the total amount of interest that will be repayable on the borrowing.

**Quotation Component**
The properties that a quotation has will be as follows:

• **Calculation Type** – Set by the profile component.

• **Purpose** - Initially set by user via the web page, it is envisaged that it will refer to the actual product e.g. car loan or endowment mortgage etc.

• **Interest Rate** – Set by the profile component.

• **Capital** – Initially set by user via the web page but may be subsequently altered by the profile component.

• **Term** - Initially set by user via the web page but may be subsequently altered by the profile component.

• **Fee Amount** – Set by the profile component.

• **Fee Rate** – Set by the profile component.

• **APR** – Set by the calculation component.

• **Repayments** – Set by the calculation component.

• **Total Interest** - Set by the calculation component.

• **Check Term Bounds** – Set by the profile component.

• **Check Capital Bounds** – Set by the profile component.

• **Check Multiples** – Set by the profile component.

• **Check Fee Bounds** – Set by the profile component.

The component requires methods to allow it to call the quotation and profiles to set its properties:

• **Collect User Input** – This method enables the users inputs to be captured by the quotation via the ASP object model.

• **Set Profile Details** – This method accepts a profile object and uses it to set the various properties.

• **Set Calculation Details** - This method accepts a calculation object and uses it to set the various properties.
Format Component
Initially the component has only one public method and no properties:

- **Write Quote** – This method accepts a quotation object and formats it to display on a screen as a page of HTML. This is achieved by using a number of private methods that perform individual formatting to each of the quotation object parameter. This design would allow further display patterns to be specified by altering the order of the private methods are called.

The private methods are as follows:

- **Page Heading** – This produces the start of the HTML page.
- **Draw Line** – This draws a horizontal line on the screen.
- **Start Table** - This provides the start of the display table.
- **Display Capital** – This displays the capital borrowed. The display will vary depending on the values of the optional parameters “Check Capital Bounds” and “Check Multiple”.
- **Display Term** - This displays the repayment term. The display will vary depending on the value of the optional parameter “Check Term Bounds”.
- **Display Interest Rate** - This displays the interest rate.
- **Display APR** – This displays the APR.
- **Display Interest** - This displays the total interest.
- **Display Fee Rate** - This displays the fee rate.
- **Display Fee Charge** – This displays the fee payable. The display will vary depending on the value of the optional parameter “Check Fee Bounds”.
- **Display Total Repayable** – This displays the total amount repayable.
- **Display Repayments** – This displays the monthly repayments.
- **End Table** – This provides the end of the display table.
- **End Page** – This produces the end of the HTML page.
- **Table Entry** – This produces the standard format that all entries in the table require. Each of the display methods above pass two values to this method and receive the formatted line back.
Appendix 4

Error Checking within Components

Assumptions made and Error Codes for each component

- **Quotation Component** – This is designed to be as re-usable as practical and therefore no error checking is performed on input to instances of this component. Each parameter or property has a type which must be adhered to, however the actual values input are not checked beyond this as at this level they do not affect the operation of the component. The properties and types are as follows:

  1. **CalculationType** could be any String.
  2. **Purpose** could be any Double.
  3. **InterestRate** could be any Double.
  4. **Capital** could be any Double.
  5. **Term** could be any Double.
  6. **Fee rate** could be any Double.
  7. **Fee amount** could be any Double.
  8. **APR** could be any Double.
  9. **Repayments** could be any Double.
  10. **TotalInterest** could be any Double.

- **Calculator Component** – This component requires to ensure that certain values are correct to enable it to perform correctly. Accordingly, these values are checked and if any errors are discovered the appropriate error code is relayed back. The values that are checked and the error codes are as follows:

  1. **CalculationType** must be one of those listed i.e. "CapInt", "CapIntMort" or "IntOnly".
  2. **Capital** must be greater than zero.
  3. **Interest Rate** must be greater than zero.
  4. **AdminFee** could be any figure zero or greater or any figure less than zero but also less than the value of capital (i.e. a negative value for AdminFee cannot be greater than the capital borrowed).
  5. **Term** must be greater than zero.
If an error is discovered the following codes are used to identify the source:

-9000000 and
-900000 CalculationType
-90000 Capital
-90000 IntRate
-9000 AdminFee
-9000 Term

For example if errors in CalculationType and Capital were discovered the error code would be:
-900000 To signify an error
-900000 To signify a CalculationType error
-90000 To signify a Capital error
-999000 The combined error code that will be returned

- Profile Component - again only performs basic error checking that directly affects the logic of this component. For example, the MaxCapital that can be borrowed cannot be less than the MinCapital that can be borrowed and it must also be greater than zero. The values that are checked within this component are as follows:

1. CalculationType could be any string.
2. InterestRate must be greater than zero.
3. BorrowingMultiples must be greater than zero.
4. MaxCapital must be greater than zero and greater than or equal to MinCapital.
5. MinCapital must be greater than zero and equal or less than MaxCapital.
6. FeeRate must be equal to or greater than zero.
7. MaxFee must be greater than zero and greater then MinFee.
8. MinFee must be less than MaxFee.
9. MaxTerm must be greater than zero and greater than or equal to MinTerm.
10. MinTerm must be greater than zero and less then or equal to MaxTerm.

The error codes that are generated are as follows:

-8000000 and
-8000000 IntRate
-800000 BorrowMulti or CheckBorrowingMultiples()
<table>
<thead>
<tr>
<th>Value</th>
<th>Condition</th>
</tr>
</thead>
<tbody>
<tr>
<td>-8000</td>
<td>MaxCapital or MinCapital or MaxCapital &lt; MinCapital or CheckCapital()</td>
</tr>
<tr>
<td>-800</td>
<td>FeeRate</td>
</tr>
<tr>
<td>-80</td>
<td>MaxFee or MinFee or MaxFee &lt; MinFee or AdministrationFee()</td>
</tr>
<tr>
<td>-8</td>
<td>MaxTerm or MinTerm or MaxTerm &lt; MinTerm or CheckTerm()</td>
</tr>
</tbody>
</table>
Appendix 5

Component Implementations

Calculation Component

- **Capital & Interest (Loan)** – Interest is calculated as if the money was being borrowed for the full term of the loan i.e. no allowance made for the fact that repayments of capital are being made on a monthly basis.

  Type of Borrowing: CapInt

  The monthly repayments ($MR$) are calculated by adding together the total interest payable ($TIP$) to the amount borrowed ($AB$) and then dividing by the term of the loan ($T$)

  \[
  MR = \frac{AB + TIP}{T} \quad \text{[Equation 1]}
  \]

  To calculate the total interest payable ($TIP$) - the sum borrowed ($AB$) is multiplied by the annual interest rate ($IR$), with the result then being multiplied by the length of the loan expressed in years ($T / 12$).

  \[
  TIP = (AB \times IR) \times \left(\frac{T}{12}\right) \quad \text{[Equation 2]}
  \]

  The total amount repayable ($TAR$) is the sum of the amount borrowed ($AB$) and the total interest payable ($TIP$).

  \[
  TAR = AB + TIP \quad \text{[Equation 3]}
  \]

  The APR is the annual percentage rate and is the equivalent interest rate when monthly repayments included in the calculation of interest payable. To calculate this, firstly calculate the amount of interest that should be paid when monthly capital repayments are included ($IP$). Then take the actual interest payable ($TIP$) and divide by the calculated interest ($IP$) and multiply by the flat rate of interest ($IR$) to get the equivalent rate.

  \[
  APR = \frac{\text{For (v=0; v<T; v++)}}{	ext{return } ((TIP / IP) \times IR)} \quad \text{[Equation 4]}
  \]
• **Capital & Interest (Mortgage)** – Interest is calculated based on the outstanding borrowing on a daily basis i.e. capital repayments are taken into consideration.

Type of Borrowing: - CapIntMort

The monthly repayments \((MR)\) are calculated by adding together the total interest payable \((TIP)\) to the amount borrowed \((AB)\) and then dividing by the term of the loan \((T)\)

\[
MR = (TIP + AB) / T \quad \text{[Equation 5]}
\]

To calculate the total interest payable \((TIP)\) – the monthly amount of interest payable is calculated taking into consideration the monthly capital repayments. The monthly capital repayments \((AB / T)\) and the monthly interest rate \((IR/12)\) are used within a loop to calculate and accumulate the total interest payable \((IP)\).

\[
TIP = \{
\quad \text{For } v=0; v<T; v++;
\quad \{
\quad \quad IP = IP + (AB - AB / T) \times ((IR / 12) \times (1/100));
\quad \}
\quad \}
\quad \}
\quad \}
\quad \return IP;
\}
\}
\}
\}
\}
\]

The total amount repayable is the sum of the amount borrowed \((AB)\) and the total interest payable \((TIP)\).

\[
TAR = AB + TIP \quad \text{[Equation 7]}
\]

As the interest is calculated based on the outstanding balance each month, then the APR is equal to the Flat interest rate \((IR)\)

\[
APR = IR \quad \text{[Equation 8]}
\]

• **Interest Only (Loan or Mortgage)** – Interest is calculated on the outstanding borrowing on a daily basis. No capital is repayable during the term of the borrowing.

Type of Borrowing: - IntOnly

This type of borrowing does not have monthly repayments of capital; therefore the repayments are interest only.

\[
MR = TIP / T \quad \text{[Equation 9]}
\]

To calculate the total interest payable \((TIP)\), first multiply the amount borrowed \((AB)\) by the interest rate \((IR)\) to get the interest paid in one year and then multiply the result by the term in years \((T / 12)\).

\[
TIP = (AB \times IR \times (1/100)) \times (T / 12) \quad \text{[Equation 10]}
\]
The total amount repayable is the sum of the amount borrowed (\(AB\)) and the total interest payable (\(TIP\)).

\[
TAR = AB + TIP \quad \text{[Equation 11]}
\]

As there are no repayments during the length of the borrowing, the amount outstanding remains unchanged; therefore, the \(APR\) is equal to the Flat interest rate (\(IR\))

\[
APR = IR \quad \text{[Equation 12]}
\]

The methods that the Calculator provides to allow access to the various calculations are as follows:

- **ErrorCheck** – This is a private method that is utilised by the other methods to confirm that the supplied parameter values are valid.

- **AnnualPercentageRate** – This calculates the APR based on the values input. A combination of the value input for “CalculationType” and the result from “ErrorCheck” decides whether a calculation is performed and if so which type. If an error is discovered by the error check then it generates an error code, which will be a number less than -900000 and it is this code that is returned to the client rather than the result of the calculation. Part of the AnnualPercentageRate calculation requires knowledge of the total interest payable, however this figure is already available within an existing method in the component, therefore it is this method that is utilised rather than repeating the code.

- **AmountRepayable** – This calculates the total amount repayable and includes capital, interest and any arrangement fees. The method of calculation is the same for all “CalculationTypes”. The input parameters are checked using the “ErrorCheck” and based on the result of this either the error code is returned or the calculation result.

- **MonthlyRepayments** – This calculates the monthly repayments and the method of calculation varies based on the value of “CalculationType”. The input parameters are firstly checked using the “ErrorCheck” and based on the result of this either the error code is returned or the appropriate calculation result.

- **InterestPayable** – This calculates the total interest that will be payable on the borrowing with the method of calculation again being decided by the “CalculationType”. The input parameters are firstly checked using the “ErrorCheck” and based on the result of this either the error code is returned or the appropriate calculation result.
Profile Component

The methods provided by this component are as follows:

- **ErrorCheck** – This is a private method that is utilised by the other methods to confirm that the supplied parameter values are valid.

- **LoanProfile** – This is the method used to set the values and it returns a value of type Double, which can be used to hold an error code if appropriate.

The variables and types that are set by this to configure the component are as follows:

- **CalculationType** type String.
- **IntRate** type Double.
- **BorrowMulti** type Double.
- **MaxCapital** and **MinCapital** type Double.
- **FeeRate** type Double.
- **MaxFee** and **MinFee** type Double.
- **MaxTerm** and **MinTerm** type Double.

The LoanProfile method firstly checks that the input values are valid by using the “ErrorCheck”.

The remaining methods within this component are available to enable the client inputs to be checked to see if they fall within the values set in the “LoanProfile”.

- **CheckCapital** – This accepts an input of Capital that should be of type Double and it returns a Double. It firstly checks that the input is not negative and sets the error code to -8008000 if it is. It then checks that the figure is greater than or equal to the minimum capital amount and less than or equal to the maximum capital amount. If the input figure requires to be amended to make it fall within these bounds then a global variable is set to reflect this. Then finally either the suitably amended figure for capital or the error code is returned to the client.

- **CheckBorrowingMultiples** – This also accepts an input of Capital that should be of type Double and returns a Double. It firstly checks that the input is not negative and sets the error code to -80080000 if it is. It then checks that the input figure is a multiple of the figure set in the “LoanProfile” and if not rounds the figure up to the nearest multiple and sets a global variable to indicate that the input has been changed. Finally either the suitably amended figure for capital or the error code is returned to the client.

- **AdministrationFee** – This accepts an input of Capital that should be of type Double and returns a Double. It firstly checks that the input is not negative and
sets the error code to -8000080 if it is. The next stage is to calculate the administration fee that would be payable based on the capital figure input and the fee rate set by the “LoanProfile”. The result of this calculation is then checked to ensure that it falls within the minimum and maximum fees allowable as set in the “LoanProfile”. If it does not fall within these bounds then it is adjusted and a global variable is set to reflect this. Finally either the suitably amended figure for the administration fee or the error code is returned to the client. It is important when using this method to calculate the fee that the “CheckCapital” and “CheckBorrowingMultiples” checks are applied first as these can alter the figure for capital and therefore the value on which the fee should be calculated.

- **CheckTerm** – This accepts an input representing the Term that should be of type Double and returns a Double. It firstly checks that the input is not negative and sets the error code to -8000008 if it is. It then ensures that the term is within the set boundaries and if not adjusts it and sets a global variable to reflect this. Finally either the suitably amended figure for the term or the error code is returned to the client.

The Quotation Component
The various variables are all empty when the instance of the Quotation is created and it is its interaction with other components that sets them. The variables and their types are as follows:

- **CalculationType** type String.
- **Purpose** type Double.
- **IntRate** type Double.
- **Capital** type Double.
- **Term** type Double.
- **FeeAmount** type Double.
- **FeeRate** type Double.
- **APR** type Double.
- **Repayments** type Double.
- **TotalInterest** type Double.
- **CheckTerm** type Integer.

The methods provided by this component are:

- **CollectFormDetails** – This enables the values of Purpose, Capital and Term to be captured directly from the ASP page by use of the ASP Scripting Context. This is achieved by capturing a reference to the Scripting Context when the page is first
loaded. At this stage the only stipulation for the values of the three parameters is that they should be of the correct type.

- **SetLoanProfile** – This method accepts an object and then has access to its methods thus allowing the values set in, for example, a profile object to be checked against those held in the “Quote”.

- **SetLoanCalculations** – This method accepts an object and allows the calculations to be performed and the appropriate values to be set.

It is important that the methods be used in the correct order; the “CollectFormDetails” must be called prior to “SetLoanProfile” and then “SetLoanCalculations” to ensure that the necessary values are set correctly.

**The Format Component**
The various private methods that produce the strings for display via the “TableEntry” are as follows:

- **DisplayCapital** – This produces two strings used to display the capital amount, it uses the two (optional) variables that were set within the “Profile” component to decide which narrative to include in the second string, this is based on whether the capital value has been altered due to it not conforming to the profile details.

- **DisplayInterestRate** – This produces two strings used to display the interest rate charged.

- **DisplayFeeRate** – This produces two strings used to display the administration percentage rate charged.

- **DisplayTerm** – This produces two strings used to display the borrowing term, it uses the (optional) variable that was set within the “Profile” component to decide which narrative to include in the second string, this is based on whether the term value has been altered due to it not conforming to the profile details.

- **DisplayInterest** - This produces two strings used to display the total interest payable.

- **DisplayRepayments** - This produces two strings used to display the monthly repayments on the borrowing.

- **DisplayTotalRepayable** - This produces two strings used to display the total amount repayable.

- **DisplayAPR** - This produces two strings used to display the APR on the borrowing.

- **DisplayFeeCharge** - This produces two strings used to display the administration fee payable, it uses the (optional) variable that was set within the “Profile” component to decide which narrative to include in the second string, this is based
on whether the fee value has been altered due to it not conforming to the profile details.

The strings produced by the above private methods are then passed to the “TableEntry” method that adds any background, sets the font and places the two strings in format that can be displayed in a table. The output from this method is a string, which is returned to the “WriteQuote”.
List Of Tables

| Table 1. Contents of an ASP page | 26 |
| Table 2. APR function – “CalculationType” input test | 63 |
| Table 3. APR function – “Capital” input test | 64 |
| Table 4. APR function – “InterestRate” input test | 64 |
| Table 5. APR function – “AdminFee” input test | 64 |
| Table 6. APR function – “Term” input test | 64 |
| Table 7. APR function – test calculations | 64 |
| Table 8. Validation Tests | 66 |
# List Of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Adoption Ladder</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>E-Business Layout</td>
<td>16</td>
</tr>
<tr>
<td>Figure 3</td>
<td>Example of the use of XML document</td>
<td>23</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Static Web Page Generation</td>
<td>24</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Dynamic Web Page Generation</td>
<td>25</td>
</tr>
<tr>
<td>Figure 6</td>
<td>Lollipop Diagram</td>
<td>31</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Two-Tier (Client-Server) System</td>
<td>33</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Three-Tier Architecture</td>
<td>34</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Three-Tier Models</td>
<td>35</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Four-Tier Architecture</td>
<td>37</td>
</tr>
<tr>
<td>Figure 11</td>
<td>n-Tier Application Layout</td>
<td>40</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Financial Services Web Site</td>
<td>42</td>
</tr>
<tr>
<td>Figure 13</td>
<td>Financial Services Web Site (Insert 1)</td>
<td>43</td>
</tr>
<tr>
<td>Figure 14</td>
<td>Financial Services Web Site (Insert 2)</td>
<td>43</td>
</tr>
<tr>
<td>Figure 15</td>
<td>Financial Services Web Site (Insert 3)</td>
<td>44</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Financial Services Web Site (Insert 4)</td>
<td>44</td>
</tr>
<tr>
<td>Figure 17</td>
<td>Display Screen Layout</td>
<td>45</td>
</tr>
<tr>
<td>Figure 18</td>
<td>Initial view of system as a “Black-Box”</td>
<td>48</td>
</tr>
<tr>
<td>Figure 19</td>
<td>Initial view of production of “Quotations”</td>
<td>48</td>
</tr>
<tr>
<td>Figure 20</td>
<td>Initial view of production of “Applications”</td>
<td>49</td>
</tr>
<tr>
<td>Figure 21</td>
<td>Initial view of production of “Agreements”</td>
<td>49</td>
</tr>
<tr>
<td>Figure 22</td>
<td>Relational Database</td>
<td>50</td>
</tr>
<tr>
<td>Figure 23. Component Layout (Loans, Insurance and Mortgages)</td>
<td>51</td>
<td></td>
</tr>
<tr>
<td>Figure 24. Population of “Quotation” instance</td>
<td>52</td>
<td></td>
</tr>
<tr>
<td>Figure 25. Summary of “Profile” component</td>
<td>54</td>
<td></td>
</tr>
<tr>
<td>Figure 26. Summary of “Calculation” component</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Figure 27. Summary of “Quotation” component</td>
<td>55</td>
<td></td>
</tr>
<tr>
<td>Figure 28. Summary of “Format” component</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td>Figure 29. Order of Execution</td>
<td>57</td>
<td></td>
</tr>
<tr>
<td>Figure 30. Error Checking Within Components</td>
<td>58</td>
<td></td>
</tr>
<tr>
<td>Figure 31. Flow Graph for APR function</td>
<td>63</td>
<td></td>
</tr>
<tr>
<td>Figure 32. The core architecture for Web applications</td>
<td>69</td>
<td></td>
</tr>
<tr>
<td>Figure 33. TestPage1.asp</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>Figure 34. TestPage2.asp</td>
<td>73</td>
<td></td>
</tr>
<tr>
<td>Figure 35. TestPage3.asp</td>
<td>74</td>
<td></td>
</tr>
<tr>
<td>Figure 36. Loan Quotation</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Figure 37. Error Screen 1</td>
<td>75</td>
<td></td>
</tr>
<tr>
<td>Figure 38. Error Screen 2</td>
<td>76</td>
<td></td>
</tr>
<tr>
<td>Figure 39. Error Screen 3</td>
<td>76</td>
<td></td>
</tr>
</tbody>
</table>
# List Of Equations

<table>
<thead>
<tr>
<th>Equation</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Monthly Repayments (Capital and Interest Borrowing)</td>
<td>117</td>
</tr>
<tr>
<td>2</td>
<td>Total Interest Payable (Capital and Interest Borrowing)</td>
<td>117</td>
</tr>
<tr>
<td>3</td>
<td>Total Amount Repayable (Capital and Interest Borrowing)</td>
<td>117</td>
</tr>
<tr>
<td>4</td>
<td>APR (Capital and Interest Borrowing)</td>
<td>117</td>
</tr>
<tr>
<td>5</td>
<td>Monthly Repayments (Capital and Interest Mortgage Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>6</td>
<td>Total Interest Payable (Capital and Interest Mortgage Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>7</td>
<td>Total Amount Repayable (Capital and Interest Mortgage Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>8</td>
<td>APR (Capital and Interest Mortgage Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>9</td>
<td>Monthly Repayments (Interest Only Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>10</td>
<td>Total Interest Payable (Interest Only Borrowing)</td>
<td>118</td>
</tr>
<tr>
<td>11</td>
<td>Total Amount Repayable (Interest Only Borrowing)</td>
<td>119</td>
</tr>
<tr>
<td>12</td>
<td>APR (Interest Only Borrowing)</td>
<td>120</td>
</tr>
<tr>
<td>13</td>
<td>Cyclomatic complexity Testing</td>
<td>62</td>
</tr>
</tbody>
</table>
References


Bibliography


