

Developer Productivity Study – EGL vs. .NET

Comparing IBM® Developer Tools to Microsoft® Developer Tools

Prepared by:

Branham Group Inc.

100 Constellation Crescent, Suite 915

Ottawa, ON K2G 7E6

Tel: (613) 745-2282

Fax: (613) 745-4990

www.branhamgroup.com

March 2008

Important Notice

This document is copyrighted © by Branham Group, Inc. and is protected by Canadian and international copyright laws. This document was developed based on information and sources deemed to be reliable. While efforts were made by the Branham Group Inc. to verify the completeness and accuracy of the information contained in this documentation, this documentation is provided "as is" without any warranty whatsoever and to the maximum extent permitted, Branham Group Inc. disclaims all implied warranties, including without limitation the implied warranties of merchantability, non-infringement and fitness for a particular purpose, with respect to the same. Branham Group shall not be responsible for any liability or damages, including without limitation, direct, indirect, consequential or incidental damages, arising out of the use of, or otherwise related to, this documentation or any other documentation.

Trademarks

- IBM, DB2, Rational, WebSphere, System z, System i, z/OS, and i5/OS are either registered trademarks or trademarks of IBM Corporation in the United States, other countries, or both.
- Microsoft, Windows, BizTalk, SharePoint, Visual Studio, and Visio are either registered trademarks or trademarks of Microsoft Corporation in the United States, other countries, or both.
- Other company, product or service names may be registered trademarks, trademarks, or service marks of others.

Table of Contents

1.0 Disclosure	1
1.1 About Branham Group Inc.	1
1.2 Commissioned Research.....	1
2.0 Executive Summary	2
2.1 Overview	2
2.2 Goals of this Study.....	2
2.3 The Tools Compared	3
2.4 Productivity Study Results	3
3.0 Introduction	5
3.1 Ground Rules Revisited	5
3.2 Introducing – On Demand Insurance Company	5
3.3 The Applications and the Tools Used.....	6
4.0 The Tools Productivity Methodology	8
4.1 The Environment Methodology.....	8
4.2 The Measurement Methodology	8
5.0 The Productivity Results	9
5.1 Summary of Productivity Times	9
5.2 Summary of Requirements Met	9
5.3 Productivity Detail for each Application	10
6.0 Conclusions	17
6.1 Productivity	17
6.2 Quality	17
6.3 Requirements.....	18
6.4 The Value of EGL	18
7.0 Appendix	20
7.1 Application Requirements	21
7.2 Application Prototypes	30
7.3 Detailed Steps to build the IBM Applications.....	40
7.4 Detailed Steps to build the Microsoft Applications.....	42
7.5 Glossary	45

1.0 Disclosure

1.1 About Branham Group Inc.

Branham Group provides "Go to Market" direction to global Information Technology products and services companies.

Since its inception in 1990, the widespread vision of Branham's professional team has taken the company to all corners of the globe to assist valued clients. Branham's clients are based primarily in the United States, with a large percentage in Canada and Europe. Recently the company's international reach has touched Latin American countries and companies in the Asia Pacific markets.

Branham Group acts as an information channel for the future of business in a wired world, helping clients understand and leverage emerging and emergent technologies.

As enterprises expand worldwide in search of new opportunities, the need for global and local business intelligence increases. All of Branham's analysis, strategies and recommendations are based on its own primary research that covers the global market. Branham uncovers and documents the latest in IT developments, following user trends and next generation IT leaders.

Over the last decade, Branham has assisted world leaders in software, hardware and services. Through its vast understanding of the IT Market, Branham has been able to deliver meaningful insight to its clients in the areas of planning, marketing, and partnering.

1.2 Commissioned Research

IBM® commissioned Branham Group Inc. to perform an objective study evaluating the development productivity differences required to build a useful working application, using tools that support IBM's Enterprise Generation Language (EGL) and Microsoft®'s .NET. The application, in its completed form allows customers to apply for a homeowner insurance policy as well as view existing insurance quotes.

This study leverages parameters from the original Branham Group Productivity Study that was completed in 2005 (also commissioned by IBM) and compared developer productivity differences between J2EE and .NET (freely available at http://www.branhamgroup.com/tools_study). Over the course of this study (EGL vs. .NET), IBM and Microsoft applications were reviewed, designed, built, recorded and carefully timed. Where necessary, Branham Group updated the Microsoft developed applications to address updated procedures within the Microsoft development environment (as compared to the 2005 study). For the IBM EGL applications, IBM developers first designed and created the necessary components, and Branham Group verified that the design and approach was considered best practice and re-built and timed each component, respectively. Branham Group used both IBM and Microsoft certified developers to review the design of the components as well as build them.

Branham Group wholly supports the integrity and the methodology of how this study was conducted as well as the accuracy of the results and the conclusions. Branham Group and IBM have made every attempt to provide a fair, honest, and unbiased research study of developer productivity through a real world working application.

2.0 Executive Summary

2.1 Overview

This paper focuses on comparing various productivity efficiencies and techniques between the IBM EGL and Microsoft .NET programming environments. Direct measurements are made between equivalent tools from both environments that accomplish specific tasks in building a fully functional web based application.

Using several of the parameters from the original Branham Group Productivity Study that was completed in 2005 as the basis for this research, this new study looks at the effort required to complete the first two applications, plus one additional application, using EGL and .NET. Specifically, this study looks at the development of a browser based UI, web based processing, creation of a Web Service from scratch, and a new requirement to retrieve existing customer data from a z/OS located VSAM file (i.e. rather than duplicating customer data to a local database via batch process as done in the previous study). Virtual Storage Access Method (VSAM) is a mainframe z/OS access method that allows the organization of data into files for access via index, relative record numbers or sequential order. The introduction of z/OS enterprise data into the mix increased development complexity but provides a more realistic scenario for enterprise IT environments.

Ultimately, Microsoft tools were not able to directly address the third application (the retrieval of customer information from the VSAM file) without the use of third party tools. Rather than stifle the productivity comparison for the first two applications, which Microsoft is capable of completing, a local database was used as the data source for the first two applications similar to the original study. This allowed Branham Group to complete the development of the web application and Web Service using both the IBM and Microsoft tools.

2.2 Goals of this Study

This paper reflects on the results of an intensive study that takes an objective look at the productivity differences between specific IBM and Microsoft tools to build a “real world” multi-tier web-based application. The specific goals of this study included:

1. Define an objective methodology that would allow a fair “apples to apples” comparison of different development techniques using IBM and Microsoft tools.
2. Build all the application components necessary to create a working application.
3. Provide an objective measurement methodology to measure the productivity differences between IBM tooling and Microsoft tooling when building the application components.
4. Document the productivity measurement differences as well as any issues that were observed between IBM and Microsoft tools in the course of the study.

In summary:

Which development environment is more productive for constructing robust, enterprise-level, server side applications?

IBM or Microsoft

2.3 The Tools Compared

In order to build the web-based application based on the specified requirements, the following IBM and Microsoft tools were used. This list does not include additional support software that was required, such as IBM DB2 or Microsoft SQL Server. In all cases, only the latest supported developer tools, from both IBM and Microsoft, were used. Beta or “early” versions of developer tools were not allowed within the scope of this productivity study.

Table 1 - Tools used in the Productivity Study	
IBM Tools	Microsoft Tools¹
<ul style="list-style-type: none"> • Rational® Application Developer V7.0.0.3 • Rational Business Developer Extension V7.0 	<ul style="list-style-type: none"> • Visual Studio® .NET 2005 Service Pack 1

2.4 Productivity Study Results

In an effort to provide additional granularity with respect to measurable levels of productivity, the application testing was divided into smaller steps. A detailed description of this breakdown is documented in the Appendix, Section 7.3 (IBM) and Section 7.4 (Microsoft).

Summary of Results

Compared to the previous 2005 study, Microsoft related times decreased significantly for the application parts that were created. This is attributable to hardware advances and code optimization, as well as changes in development practices. Specifically, instead of creating the necessary HTML templates and CSS files from scratch within the development tools (as done previously), these were supplied by a fictitious design firm and simply imported into the tools, typifying more current web application development practices. While there is a significant decrease in Microsoft related times compared to the previous study, these fundamental changes eliminate any ability to draw conclusions or comparisons to the J2EE related development times from the previous study. Key findings from this year’s study as they relate to EGL and .NET development productivity include the following:

1. The timings for the creation of web based applications and Web Services through EGL or .NET were essentially equivalent for the applications that were built. There were no significant measured productivity differences. There were a couple of interesting observations made however, which may have an effect on productivity on a larger scale. First, Microsoft still requires the manual creation of scaffolding for the testing of its Web Services, adding time and complexity. Second, while the overall times for IBM EGL and Microsoft .NET were similar when building the web application, given IBM EGL’s use of a MVC architecture, the bulk of the logic (model of MVC) was completed in the initial test case. This resulted in the individual pages (view of MVC) taking significantly less time to complete compared to Microsoft, culminating in similar development times overall. If the web application was significantly larger in scope, this reduced development time could have resulted in a more favorable productivity measurement compared to Microsoft .NET.
2. In general, as the application requirements became more complex and required access to existing enterprise assets, the Microsoft development environment would have required more tools (third party) than the IBM development environment. An increase in tools can typically be equated to an increase in complexity in development particularly when these tools are not integrated. Similarly, increased complexity in tools is not conducive to ensuring high quality applications.
3. The Rational Business Developer Extension and EGL allowed developers to create both Java and COBOL applications through a single high-level language, a single development environment, and a

¹ The Microsoft environment includes the .NET framework V2.0 with Service Pack 1

single debugging environment without a need to know a single line of Java or COBOL to complete the applications.

Table 2 – Summary of the Productivity Study			
Application	IBM	Productivity Difference	Microsoft
1. Web Application for the homeowner policy quote	63	1.0X	66
2. Web Service built from scratch	10	1.0X	10
3. COBOL/CICS server program to manipulate VSAM data file	NM	N/A	Unsupported

NOTE(s):

- All times noted are in minutes
- NM = Not Measured
- N/A = Not Applicable
- Unsupported = Unable to build the required functionality through Microsoft tools only.
- Since Application #3 could not be built using Microsoft tools because of product functionality limitations, the testing for the same application using the IBM tools was not completed

Conclusion:

For simple web application development, IBM EGL productivity is similar, if not better than, Microsoft Visual Studio 2005 .NET.

Additionally, IBM EGL provides higher productivity for cross-platform, enterprise-level application development through a single IDE and language.

3.0 Introduction

In 2005, Branham Group completed an exhaustive study comparing the productivity of IBM (J2EE) and Microsoft (.NET) development tools for building real world web applications. This particular study revisits the original productivity study from a new IBM perspective; the EGL development language and associated development environment. Specifically, this study compares the productivity of IBM EGL development versus that using Microsoft Visual Studio 2005 .NET.

IBM's Enterprise Generation Language (EGL) is a high-level language that provides an additional layer of abstraction from lower level programming languages and architectures. Specifically, EGL is a language that is not tied to a specific technology. It provides platform neutrality allowing a deployed application to be generated for a number of target environments. Developers are not required to learn multiple languages, but instead use the neutral language and transform it into the language that best suites the selected target environment (COBOL, Java). Code generation predictively reduces the cost and time needed to become proficient in designing and implementing applications in multiple languages and environments.

Given the passage of time since the 2005 study, the release of new versions of the Microsoft development tools, changes in development technologies, and changes in real world web based application development practices, the Microsoft tools were retested in this study to provide a true, accurate, and current comparison to EGL. By way of retesting the latest Microsoft tools, this process also eliminates any indiscretions related to improvements in hardware and other performance optimizations, ensuring a fair comparison between both environments.

It should be noted however, that while the Microsoft development tools were retested, the IBM Java tools used in the original study were not. The point of this study is to only consider the productivity differences and similarities between the respective IBM EGL and Microsoft .NET development tools.

3.1 Ground Rules Revisited

In the original 2005 productivity study there were specific ground rules that were followed. These same ground rules apply to this study and consist of the following:

1. Tools must be generally available from IBM and Microsoft at the time of the study and supported as current products. No beta or early release products were allowed in this comparison.
2. No third party tools or solutions were allowed. Only IBM and Microsoft supported tools.
3. Productivity is measured by the time it takes to complete a working and tested component.
4. No un-documented features or procedures were used. For the building of all components, fully documented "best programming practices" were used.
5. Try to mitigate developer skill factor or experience differences with respect to component timings by allowing developers to achieve their peak productivity efficiency. Developers 'practiced' building the application components multiple times to achieve peak productivity efficiency, only then was the time to build components officially recorded.

3.2 Introducing – On Demand Insurance Company

On Demand Insurance Company's (ODIC) core business has always been homeowners insurance. Over the past several years, the company has grown via acquisition, acquiring various additional lines of business such as auto and life insurance, including the problems associated with integrating these acquired data systems.

In the past, ODIC has used paper centric processes for its homeowner policies. The company is trying to streamline its homeowner's policy processes through automation. Where it has already been successful implementing the application process for auto and life on its internet portal site, ODIC now wishes to build a proof of concept for a homeowner's policy on the internet. The company starts by sending mail to existing preferred customers (of either auto or life policies) to invite them to look into a home policy at a

special web site (non-portal). If the trial period is successful, the company plans to move its non-portal application to the main portal site and open the program to the general public.

3.3 The Applications and the Tools Used

ODIC management initially wants to test what the demand from customers will be for a web site that will allow them to apply for a homeowner policy. The company decided to “test the waters” by developing a simple web application that will ask a few questions about the customers home and will return an “informal quote” of how much a homeowners policy will cost. ODIC decided that if it sees a high demand for this informal quote, it will then invest in a more formal quote process.

Table 3 provides a description of the particular applications built for this productivity study.

Table 3 – Description of the Applications	
Name of Application	Description of Application
1. Web Application for the homeowner policy quote	The web user interface and application used to apply for an informal insurance quote for home insurance.
2. Web Service built from scratch	This web service encapsulates a credit score check for customers requesting quotes, which is used in a risk assessment calculation. This web service is built from scratch.
3. COBOL/CICS server program to manipulate VSAM data file	This server side application provides the necessary facilities to retrieve customer data directly from the mainframe Virtual Storage Access Method (VSAM) data file.

NOTE: While IBM EGL can be used to generate the necessary COBOL/CICS application, timings for this particular application were not performed due to the fact that Microsoft tools cannot complete these steps.

The first application provides a web based user interface allowing a customer to request a formal quote. Based on customer information held within an existing mainframe VSAM file and additional information submitted by the customer, a risk factor is calculated allowing an informal quote to be returned to the user. Similarly, the second application uses the customer information from the VSAM file to perform a credit check for that customer. The third application provides the necessary server program to retrieve this customer information.

It should be noted however, that ideally, Application 3 should be created first, so that Application 1 and 2 can use the retrieved data to complete their tasks. The issue however, is that Microsoft tools alone cannot create the necessary COBOL/CICS code to create this particular application. In this case, this caveat would have stifled the creation of Application 1 and 2 using only Microsoft tools. As such, for Application 1 and 2 in this study, the customer data was duplicated to a local data source that was accessible to both IBM and Microsoft tools to allow the completion of these applications. In the case of the IBM tools, the data source made available by Application 3 could just as easily have been used in lieu of the local database with comparable effort. While the introduction of z/OS (or i5/OS) enterprise data into the mix increases development complexity, it does provide a realistic scenario given the amount of data that exists within these assets.

Note: This particular Application 3 caveat was also circumvented in the initial 2005 productivity study by using a batch process (created with non-Microsoft tools) that downloaded the VSAM customer information to a local database on an hourly basis and was described as “beyond the scope” of the study.

Finally, Table 4, on the following page, provides a list of the corresponding vendor tools that were used to create each of the applications.

Table 4 – Applications and Tools used

Name of Application	IBM Tools Used	Microsoft Tools Used
1. Web Application for the homeowner policy quote	<ul style="list-style-type: none"> • Rational Application Developer V 7.0.0.3 • Rational Business Developer Extension V7.0 	<ul style="list-style-type: none"> • Visual Studio .NET Enterprise Architect 2005 SP1
2. Web Service built from scratch	<ul style="list-style-type: none"> • Rational Application Developer V 7.0.0.3 • Rational Business Developer Extension V7.0 	<ul style="list-style-type: none"> • Visual Studio .NET Enterprise Architect 2005 SP1
3. COBOL/CICS server program to manipulate VSAM data file	<ul style="list-style-type: none"> • Rational Application Developer V 7.0.0.3 • Rational Business Developer Extension V7.0 	<ul style="list-style-type: none"> • Not Supported

NOTE: While IBM EGL can be used to generate the necessary COBOL/CICS application, timings for this particular application were not performed due to the fact that Microsoft tools cannot complete these steps.

4.0 The Tools Productivity Methodology

4.1 The Environment Methodology²

In order to achieve an identical and repeatable operating environment for the EGL and .NET productivity testing, both the IBM and Microsoft developers used identical hardware with identical processor speeds and memory. The developer environment was further controlled with identical VMWare images that were installed on the hardware. These VMWare images were “cloned” such that an exact operating environment (Microsoft Windows XP Professional 2002 SP-2) existed for both the EGL and .NET development environments. Other than the development tools themselves, this eliminated any differences between the software environments since both were using a duplicated environment.

4.2 The Measurement Methodology

Based on the foundation of the original Productivity Study that was completed in 2005, this version of the study leverages many of those same core application development requirements. Section 7.1 of the Appendix documents in detail all the functionality that the entire Insurance application needs to achieve. Section 7.2 of the Appendix also includes the prototypes of each web page as they should look in their final form.

The application code for the components from the 2005 study also existed for the .NET environment. However, given the evolving and updated tool set, Branham Group made modifications to update and address the changing Microsoft related development processes. Simultaneously, IBM developers designed and built the necessary components using the EGL development language. Each component was then independently reviewed by Branham Group to assure that it was considered best practice in the development community, that the approach was well documented and understood, and that it remained in line with the final requirements and expected results of ODIC.

Although developers were picked that had extensive experience with each of the IBM and Microsoft tools, to eliminate bias based on experience levels, each developer built each of the components multiple times such that they became proficient and achieved an optimal level of productivity. Once the optimal level of productivity was reached, each component was timed in order to document the productivity measurement.

² Proper licensing was obtained and was in place for both IBM and the Branham Group for the usage of VMWare, all the IBM products, and all the Microsoft products.

5.0 The Productivity Results

5.1 Summary of Productivity Times

The productivity results were measured multiple times for both the EGL and .NET environments using different developers. Each developer practiced building each application to give them “expert status” on each tool before recording their final times. Table 5 represents the average time to complete each application.

Table 5 – Summary of the Productivity Study			
Application	IBM	Productivity Difference	Microsoft
1. Web Application for the homeowner policy quote	63	1.0X	66
2. Web Service built from scratch	10	1.0X	10
3. COBOL/CICS server program to manipulate VSAM data file	NM	N/A	Unsupported*

NOTE(s):

- All times noted are in minutes
- NM = Not Measured
- N/A = Not Applicable
- Unsupported = Unable to build the required functionality through Microsoft tools only.
- While IBM EGL can be used to generate the necessary COBOL/CICS application, timings for this particular application were not performed due to the fact that Microsoft tools cannot complete these steps.

Conclusion:

For simple web application development, IBM EGL productivity is similar, if not better than, Microsoft Visual Studio 2005 .NET.

5.2 Summary of Requirements Met

In addition to the productivity times recorded, it is also important to note if the basic requirements outlined by On-Demand Insurance Company, were met for each of the applications. A detailed description of application requirements is available in the Appendix (section 7.1 and 7.2). While both vendors provide the facilities to create the first two applications, Microsoft tools cannot address the requirements of Application #3. This is summarized in Table 6 below:

Table 6 – Summary of the Requirements that were met		
Application	IBM	Microsoft
1. Web Application for the homeowner policy quote	Yes	Yes
2. Web Service built from scratch	Yes	Yes
3. COBOL/CICS server program to manipulate VSAM data file	Yes	No

Requirements that Microsoft tools did not meet:

3. COBOL/CICS server program to manipulate VSAM data file

Major Concern(s):

Microsoft does not provide tools for creation of COBOL/CICS server programs. While it is not possible for Microsoft tools to create the necessary COBOL/CICS server program to manipulate customer data from a VSAM file, if the necessary program was built using additional tools, then the process could likely be completed. For example, if the COBOL/CICS server program was created and exposed through Web Services, Microsoft tools could use this Web Service to complete the task. This however, defies the ground rule that no third party tools or solutions are allowed as part of this study.

Conclusion:

IBM tools can fulfill real world cross platform heterogeneous enterprise-level application requirements better than Microsoft.

5.3 Productivity Detail for each Application

This section takes a detailed look at each of the three applications. The following is covered for each application, taking into consideration that the third application was not actually built:

- a. Summary of productivity and requirements
- b. How it was/could be built with IBM tools
- c. How it was/could be built with Microsoft tools
- d. Lessons learned and issues encountered
- e. Bottom line points

1. Web Application for the homeowner policy quote

Brief Description:

This is a web-based application that a customer uses to apply for an informal insurance quote for homeowner's insurance. For each quote, the user supplies information regarding the property they wish to insure, and the application generates a risk factor. The informal quote process uses this risk factor to provide an immediate quote to the customers. The layout of the pages and the detailed requirements, including the look and feel prototypes, are available in section 7.2 of the Appendix.

One modification between this and the previous version of the study, is that developers were no longer required to create the HTML based templates themselves. It is now common practice to have a design firm or department create the basic layout and supply the developers with the necessary templates, images, and cascading styles sheets. This allowed the developers to simply import the graphics and templates into the corresponding development tools. As compared with the previous study, this practice can also account for the significant drop in development times.

Table 7 below summarizes the average results of the multiple timings conducted by the different developers.

Table 7 - Summary of Productivity and Requirements

Description	IBM	Microsoft	Difference
Average Time in minutes to build entire working application	63	66	1.0X
Did it meet the application requirements	yes	yes	N/A

How it was built with IBM tools

(see Appendix 7.3 for the detailed steps)

Tools used included:

- IBM Rational Application Developer with Rational Business Developer Extension
- Templates

Developers accomplished this task using IBM Rational Application Developer with the Rational Business Developer extension. First, an EGL Web Project was created and all the necessary the necessary data access EGL was written. This allowed read and write access to the associated database objects. The second step was creating each of the individual pages and their navigation. In each case, the individual page (JSP) was created using the provided JSP template (to comply with the look and feel requirements). The required components for the individual page were created by dragging and dropping the widgets from the pallet of available parts. These widgets were connected to EGL data access functionality created earlier, again using drag and drop.

How it was built with Microsoft tools

(see Appendix 7.4 for the detailed steps)

Tools used included:

- Visual Studio .NET 2005
- ASP.NET
- .NET Framework 2.0
- Data bound controls calling ADO.NET
- Forms Authentication

Developers accomplished this task using the Visual Studio .NET 2005 product. First, an ASP.NET Web Application project was created in Visual Studio .NET. The common UI layout and image files were imported and ASP.NET User Controls were created to encapsulate the UI layout.

Once the style was finalized, Web form controls were drag and dropped into the content cells, and validation controls were added and properties set. Additionally, event handling code was written to handle event logic as well as the various insurance parameters that were stored in the sessions in order to calculate the risk factor.

Lessons Learned and Issues Encountered

Microsoft has traditionally been a leader in developer productivity for building web applications like this one. The ability to drag and drop web forms on to a web page makes this a fairly easy and productive task. The ability to create the build instructions as part of VS.NET makes it easier to build the solution with minimal compiler knowledge. Further, with the introduction of Master pages, Microsoft has closed the gaps to IBM templates.

While the overall timings are very comparable between these two development environments, there were some interesting differences in times at the page level. Specifically, IBM EGL required more initial setup and configuration of the application before the creation of the individual pages than Microsoft. However, once the necessary libraries were created, the time required to create the individual IBM pages was significantly less than Microsoft. Given this finding, it begs the question, given a requirement to create

even more pages, would IBM EGL have provided more positive results in productivity compared to the Microsoft product?

IBM

1. The re-usable templates made this a very nice productivity feature. Comparatively, overall developer productivity would likely have been significantly higher for a much larger application with significantly more pages.
2. Binding JSF and EGL components made for a very productive and error free programming environment.
3. Validation rules can be associated to a data item. Every time that item is used in a particular context the validation is automatically applied and enforced. This can significantly reduce repetitive and error-prone coding.
4. For users familiar with EGL, the application was easily written without intimate knowledge of Java or JSP programming.

Microsoft

1. Overall, this was a straightforward web application to build and something that Visual Studio .NET does well. Microsoft has increased its capabilities with respect to building data bound web forms. With the introduction of master pages, the common look and feel was able to be implemented with ease.
2. One issue came up, which depending on the nature of the application requirements, could have had a very negative effect on the productivity times. Although the issue did not greatly impact this study, developers should be aware of it.

Issue 1: ODIC naming convention issue.

A curious bug arose when making use of the DataSet. When a dataset was added and named ODIC it worked as expected. Tables were added along with specialize queries and all compiled well. However, when used in a page, it would generate a peculiar error:

```
Warning 1: The type 'ODIC' in 'c:\Users\AppData\Local\Temp\Temporary ASP.NET
Files\odic\ef03441c\53cc7fc3\App_Web_xv2upsqc.0.cs' conflicts with the imported type 'ODIC' in
'c:\Users\AppData\Local\Temp\Temporary ASP.NET
Files\odic\ef03441c\53cc7fc3\App_Code.sinhm2g1.dll'. Using the one in
'c:\Users\AppData\Local\Temp\Temporary ASP.NET
Files\odic\ef03441c\53cc7fc3\App_Web_xv2upsqc.0.cs'. c:\Users\AppData\Local\Temp\Temporary
ASP.NET Files\odic\ef03441c\53cc7fc3\App_Web_xv2upsqc.0.cs 163
```

This is coincidentally caused by having a masterpage (ODIC.master) the same name as the dataset. Had the specific Microsoft recommended naming practice been followed, the dataset should have been named ODICDataSet and this problem would have never occurred.

Bottom Line Points

1. The introduction of Microsoft master pages provides a method for implementing page templates, increasing productivity compared to its previous incarnation.
2. The margin of difference in productivity timings was very small, illustrating that EGL can provide similar productivity for these types of web based applications.
3. While the initial test case for IBM EGL did take a little longer than Microsoft Visual Studio, this initial time significantly reduced the times for building the individual pages. While the overall timings were similar for the two vendors, given the reduced time for individual page construction by IBM vs Microsoft, the creation of a larger application could have illustrated more favorable results for IBM EGL as the number of pages increased.

2. Build Web Service from Scratch

Brief Description:

This Web Service performs a credit report check of the applicant using a Web Service from one of the “big” three credit agencies. This web service will eventually be used within a workflow (beyond the scope of this study) to further refine the customer Risk Factor for a formal quote request. This Web Service was created from scratch.

Table 8 summarizes the final results of multiple timings by different developers.

Table 8 - Summary of Productivity and Requirements			
Description	IBM	Microsoft	Difference
Average Time in minutes to build entire working application	10	10	1.0X
Did it meet the application requirements	yes	yes	N/A

How it was built with IBM tools

(see Appendix 7.3 for the detailed steps)

Tools used included:

- IBM Rational Application Developer with Rational Business Developer Extension

In this application, developers created a web service built from EGL code written from scratch. Once this EGL code was written to perform the business logic, the EGL Deployment Descriptor was modified, the Web Service wrapper generated and the application deployed as a Web Service. Tests were run on the built in (the server is built into the Rational tool) WebSphere Application Server; also known as the Unit Test Environment (UTE).

How it was built with Microsoft tools

(see Appendix 7.4 for the detailed steps)

Tools used included:

- Visual Studio.NET
- .NET Framework 2.0

First, a new ASP.NET web service project was created, generating a “Hello World” skeleton example within Visual Studio .NET. Manual coding was then required to write the Web Service methods that provided the functionality required, and needed to work with the provided classes - Person, CreditReport, CreditScore and Address. It was required to instantiate these classes within a Web Service method and call the appropriate methods. The Credit Report Web Service uses complex data types, which contain a number of private fields. Visual Studio .NET 2005 does not automatically generate property fields corresponding to private fields so these private fields were manually mapped for each class property. For testing a web service, Microsoft Visual Studio 2005 .NET provides an HTTP test page but no SOAP tester. The HTTP based test page provided cannot test complex data types. Thus, it was necessary to build a rudimentary test scaffold to test the web service.

Lessons Learned and Issues Encountered

IBM

1. Deployment of EGL as a Web Service required only the modification of the EGL Deployment Descriptor and the generation of the Web Service wrapper. This was a simple process.
2. The Rational Application Developer environment provided a fully functional built in Unit Test Environment.

Microsoft

1. Although there was a “hello world” sample skeleton to get started, there were not any wizards to help build the classes. The various web service methods had to be hand coded.
2. In Microsoft, property fields are used to access data (as opposed to getters and setters in Java). Visual Studio .NET does not automatically generate these property fields so manual mapping for every private field to the property fields was required.
3. There is no functional SOAP tester available, so developers are required to create a custom one.

Bottom Line Points

1. The IBM development environment is highly productive for creating web services from scratch for either primitive or complex data types. Minimal effort was required for Web Service deployment.
2. The Microsoft development environment relies on a sample skeleton code to get started as well as manual hand coding to build the complex data type web service. While an HTTP test of the web service can be run with a browser, for full SOAP testing, developers must build their own test form. For more complicated or a larger number of web services, this would become more time consuming.

3. COBOL/CICS server program to manipulate VSAM data file

Brief Description:

In order to calculate risk factors for generating quotes, customer data must be retrieved from a z/OS VSAM file. In the 2005 study, customer data was downloaded to a separate ODIC database using an hourly batch process. Instead of duplicating this information, ODIC would like to retrieve the data directly from the VSAM file. This is more realistic since most of the datasets and legacy code called by the server program will typically be located within z/OS under CICS.

Table 9 - Summary of Productivity and Requirements			
Description	IBM	Microsoft	Difference
Average Time in minutes to build entire working application	NM	Unsupported	N/A
Did it meet the application requirements	yes	no	N/A

NM = Not Measured

N/A = Not Applicable

Note: While IBM EGL can be used to generate the necessary COBOL/CICS application, timings for this particular application were not performed due to the fact that Microsoft tools cannot complete these steps.

How it could be built with IBM tools

Tools used would include:

- IBM Rational Application Developer with Rational Business Developer Extension

Given the inability to create this application using Microsoft tools only, this particular application was not built, but rather left as a written exercise. As such, the following provides some details with regard to the process required to create this application using IBM tools.

While it is possible to create all components under the initial Web project, the EGL server component is created in a separate EGL project to provide a clear separation between the server portion of the application from the client side.

Note: It is good practice to code an application from "back to front". Specifically, the "back" of an application, its data access and working areas, are completed first, moving to the programmatic logic, and finally the user interface areas. Using this “back to front” practice, before the new JSF and EGL components built in Application 1 in the EGL Web project can be created, the EGL components created for the server side (this application scenario) need to be visible to pass the data. Since the Microsoft tools

themselves could not create the components for this application, this “back to front” practice was not followed in order to allow Applications 1 and 2 to be completed by Microsoft (using the local data source).

After creating the new EGL project, the preferences for the data connections are set and the EGL parts created. For example, a linkage part defines how the generated Java client code should invoke a called EGL program, such as using a J2C connector to put all parameters in the CICS commarea. The application is tested using the EGL debugger, at which point no Java or COBOL code has yet been generated for the EGL server program. Once the program has been thoroughly tested and debugged, the appropriate build descriptors are created and the COBOL/CICS code generated, submitting the mainframe components to create the executables. During the COBOL program generation, EGL by default also produces a build plan as an input to the step that prepares the generated output for run time.

Further, this is only one way that the application could be built. Instead of building the web based component of the application to invoke the EGL services directly, the generated COBOL/CICS server program can be generated and implemented as a Web Service including the associated WSDL files. While this does provide additional flexibility and value, it also increases the overhead typically associated with Web Services. For performance reasons, the use of an EGL service is preferred.

It should also be noted that this capability is not limited to the System z platform. For example, organizations with customer data stored in a System i i5/OS logical file, instead of a z/OS VSAM file can just as easily utilize these existing facilities in a similar manner.

How it could be built with Microsoft tools

In short, Microsoft tools do not provide the facilities to create the necessary COBOL/CICS server programs to manipulate the VSAM files containing the customer data. Out of the box, Microsoft Visual Studio .NET 2005 does not provide CICS integration. In essence, in the case of manipulating VSAM files, it is impossible to remain true to the ground rule of only using solutions from a single vendor (i.e. Microsoft). At a minimum, mainframe development tools are required to develop the necessary components. These might include TSO/E, ISPF, SDF2, etc. In order to provide integration with CICS, the destination system would have to implement a Web-Service, a COM+ interface, or use Microsoft Host Integration Server 2006 to make a CICS client available to a .NET component.

A Host Integration Server license is part of a BizTalk license, which is Microsoft’s SOA solution for integrating .NET to legacy systems. BizTalk 2006 R2 licenses cost between \$500 and \$35K per processor. This can add significant costs for those organizations looking to purchase BizTalk server licenses for the operational environment.

Lessons Learned and Issues Encountered

IBM

1. IBM can meet all the requirements allowing developers to create, test and debug the mainframe related COBOL/CICS server program through the same interface used for the JSF components. Ultimately, all the developed components can be tested together through a single development and testing environment.
2. Development and testing of mainframe code can be performed locally on the workstation without a requirement to deploy.

Microsoft

Although this application cannot currently be built solely using Microsoft tools, it is possible to create the necessary components through additional tools. Depending on how the components are to be implemented, this can dramatically effect the associated costs. While the implementation of a COBOL/CICS Web Service will theoretically allow the direct connection, additional overhead will be added. Alternatively, additional costs would be associated with additional Microsoft licenses.

Bottom Line Points

1. IBM Rational Application Developer provides developers with a single environment for creating not only the user facing and server side Java code, but also the necessary code to capitalize on existing enterprise assets residing on the System z platform.
2. IBM Rational Application Developer provides a testing environment that allows all the separate components to be tested together. This allows developers to see the call statements and both sides of execution in the debugger. It provides VSAM remote access from the remote workstation allowing it to be accessed by the debugger. In this way, the entire application can be tested before any Java or COBOL code is generated. Specifically, developers are not required to regenerate the Java or COBOL code every time editing of the EGL source occurs.
3. The Microsoft approach was rated as “unsupported” since it could not be completed solely with Microsoft related tools. While there are approaches that could be taken to implement a solution using Microsoft technology – such as in the 2005 study that used a batch process to duplicate the customer data to the ODIC database – this would also require additional third party tools and potentially additional developer resources and cost.

6.0 Conclusions

6.1 Productivity

As with the original 2005 productivity study, this study was also completed with four goals in mind.

1. Define an objective methodology that would allow an “apples to apples” comparison of different development techniques using IBM and Microsoft tools.
2. Build all the application components necessary to create a working application.
3. Provide an objective measurement methodology to measure the productivity differences between IBM tooling and Microsoft tooling when building the application components.
4. Document the productivity measurement differences as well as any issues that were observed between IBM and Microsoft tools in the course of the study.

After extensive work, the results of this study strongly indicate that even for the most basic web application development (Application 1 and 2), EGL and the IBM development tools are on par with, or potentially more productive than, Microsoft tools. Even with the significant advancements between versions of the Microsoft Visual Studio releases, EGL remains competitive as a development language for web based applications.

However, where EGL and the IBM development tools really shine in comparison to Microsoft and potentially other third party development tools is cross-platform enterprise-level application development (Application 3). Through a single language and development environment, developers can create all the necessary components for all tiers of the application, without a requirement to know the individual underlying run-time technologies. The ability to generate Java, COBOL, CICS, IMS DB/DC and more, and deploy on standard Java application servers, System z, and System i can significantly increase productivity levels.

Although not tested extensively within the scope of this project, IBM Rational Business Developer also offers additional productivity gains through end-to-end debugging of all application parts. Specifically, IBM Rational Business Developer incorporates the testing of what would eventually become the outmost web pages, COBOL code, and the CICS invocation simultaneously. The EGL debugger allows developers to debug their code without a requirement that they first generate output. This can provide significant productivity improvements over separate unit testing and integration of application parts created by different developers in separate development environments.

In short, IBM and EGL support the ability to build robust enterprise-level cross platform applications in an ever changing on demand environment. In a z/OS environment, it is possible to deploy the server program (the third tier in the Web application) using CICS. This is more realistic since most of the datasets and legacy code called by a server program can typically be located within z/OS under CICS.

6.2 Quality

Although measuring the quality of the application is always a difficult subject, the following observations were made.

1. The IBM development environment had the debugging and test harnesses embedded into the development tools. This was not always the case for the Microsoft development environment. Even the “simple” application of constructing a web service did not provide a built in SOAP test environment. Microsoft developers are required to construct their own SOAP test environment.
2. In general, as the application requirements became more complex and required access to existing enterprise assets, the Microsoft development environment would have required more tools (third party) than the IBM development environment. An increase in tools can typically be equated to an increase in complexity in development particularly when these tools are not integrated. Similarly, increased complexity in tools is not conducive to ensuring high quality applications.

6.3 Requirements

The requirements for the application components were developed before they were built. To make this application as realistic as possible, in addition to the viewing of many existing on-line Insurance application sites, individuals within the Insurance industry were consulted. The requirements for the application were created starting with use cases, and eventually the detailed requirements were created, which are available in Appendix 7.1. These requirements outline a “realistic” insurance application as defined by the Insurance consultants and the researched on-line Insurance sites.

The IBM development environment was able to meet all of the application requirements while the Microsoft environment fell very short where access of existing enterprise assets (VSAM files) was a requirement.

This highlights the fact that the Microsoft development environment is very easy to use to build basic, and some level of advanced web applications, but cannot provide heterogeneous platform support found in larger more demanding enterprise environments.

This study indicates that the IBM development environment has the ability to build demanding and robust enterprise-level web based and business applications for multiple platforms, while reducing the amount of manual coding required, ultimately reducing the requirement to engage highly experienced developers to perform all the tasks.

6.4 The Value of EGL

Through the IBM Rational developer workbench and EGL tools, IBM provides a single development environment for developing applications that span distributed heterogeneous environments. The addition of EGL through the Business Developer extension provides developers with a significant tool that can increase productivity while simultaneously expanding capabilities. With a single high level language, developers can create Java/J2EE applications deployed to the web, GUI and Java batch applications running in a networked computing environment, and COBOL, CICS, and IMS DB/DC applications deployed to System z and System i platforms, without a requirement to know the details of the underlying run-time technologies.

EGL is continuously being enhanced with new language constructs, integration with new technologies (such as JSF), and new code generation drivers for new runtime platforms. Through the same high-level language, developers can generate code for these new platforms with minimal training. While this study used the retrieval of customer data from a VSAM file, EGL provides support for additional data sources. Developers are not required to be proficient in SQL data manipulation language, JDBC, or other SQL access programming technologies. Using a small set of polymorphic verbs, developers can write full function services accessing DB2, IDS, Oracle, Microsoft SQL Server, Derby, i5/OS integrated relational database and i5/OS logical files, VSAM, and IMS/DB (DL/I).

Using a single development platform, developers can build a solution that is best suited to their IT infrastructure. Similarly, costs associated with migration from one platform to another are significantly reduced as the same EGL can be used to generate code for the new platform. For example, generated COBOL is supported on z/OS and i5/OS while Java (as would be expected) is supported on z/OS, i5/OS, AIX, HP-UX, Sun Solaris, Microsoft Windows (2000, 2003, XP) and Linux. Developers can code in response to current platform requirements, and migrate in the future without worrying about migration details.

While code generators do make it easier and quicker than hand-coding, they often require a special runtime which makes it difficult to achieve the security, scalability, and reliability offered by native servers. Applications generated with EGL can deploy as native Java, CICS, i5/OS, or IMS programs, exactly as if they were hand-written. Specifically, while a web application can be built without knowing a line of Java, the EGL generates Java code that can run on a standard implementation of Apache Tomcat as the application server. While the use of runtime libraries may be misleading to this fact, this is simply commonly used code that would not make sense to repeat as generated code in each and every program; a common software engineering practice already in use by developers hand-coding applications.

Through EGL, IBM still provides customers with choice of implementation. There are multiple ways in which web based applications can be implemented, allowing developers to use the method best suited to

their particular requirements. In our example, the COBOL/CICS server program could have been implemented as a standards based Web Service, providing additional flexibility and access beyond the scope of our requirements.

Admittedly, EGL may not be suited for all development requirements. Specifically, it does not provide low-level APIs to operating systems and subsystems, and as such is not suited for systems development. However, EGL is particularly suited for transactional business services and applications development. This includes the creation and consumption of service oriented architectural services, web applications, reports (through integration with open source reporting engines such as BIRT), traditional text UI applications (green screen interfaces), and GUI applications.

Without writing a single line of Java or COBOL code, developers implemented an application that used JSF. Even if this simple application were more sophisticated and complex, the process to code, test, debug, and deploy would be very similar. When coding in EGL, the developers didn't need to deal with all the plumbing and configuration "sit-ups". EGL provides an easy way for programmers to move into the J2EE world (or mainframe world) without knowing all of the skills that are necessary for these complex implementations.

7.0 Appendix

This Appendix contains the details of the Insurance application that was built for the purpose of the tools comparison study. It is organized as follows:

7.1 Application Requirements

This section contains the detailed application requirements.

7.2 Application Prototypes

This section includes the graphic user layout (prototypes) of the web customer applications to help guide the developer on what the application must look like when development is completed.

7.3 Detailed steps to build the IBM Applications

This section contains the step-by-step procedures that were timed for each of the IBM applications.

7.4 Detailed steps to build the Microsoft applications

This section contains the step-by-step procedures that were timed for each of the Microsoft applications.

7.5 Glossary

7.1 Application Requirements

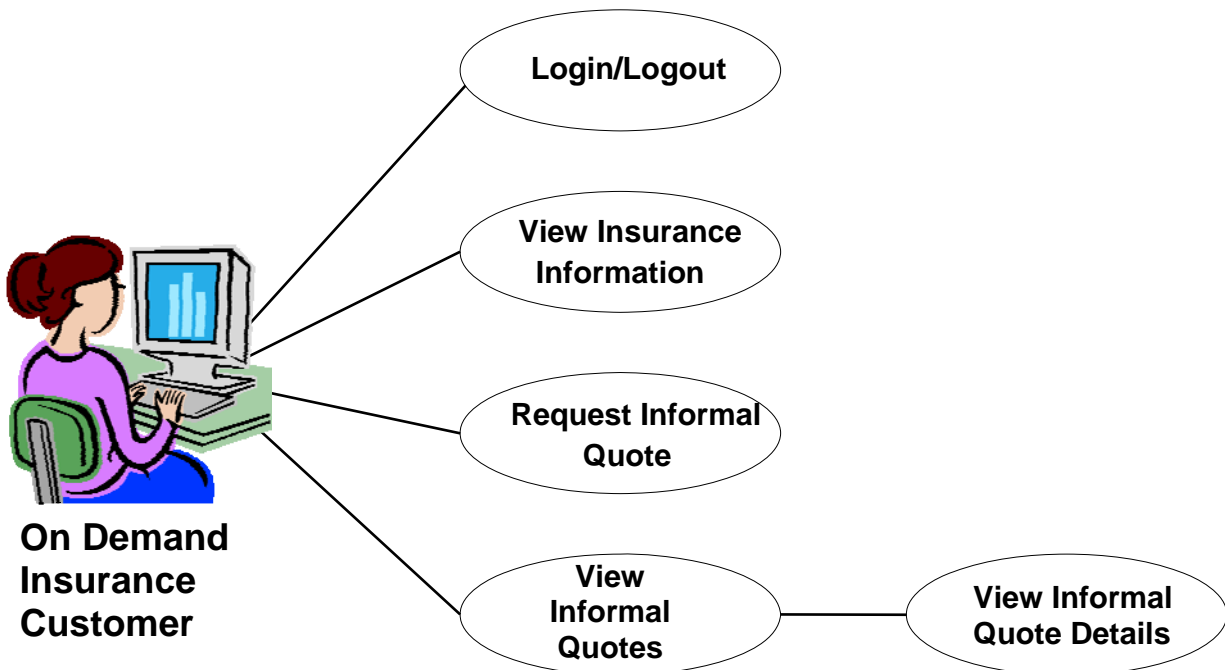
This section contains the requirements information that the IBM and Microsoft development teams used to build the required Insurance applications. The requirements are organized as follows:

- Vision Statement
- Use Case Model
- Architecture Diagram
- Database Schema
- Documented Features

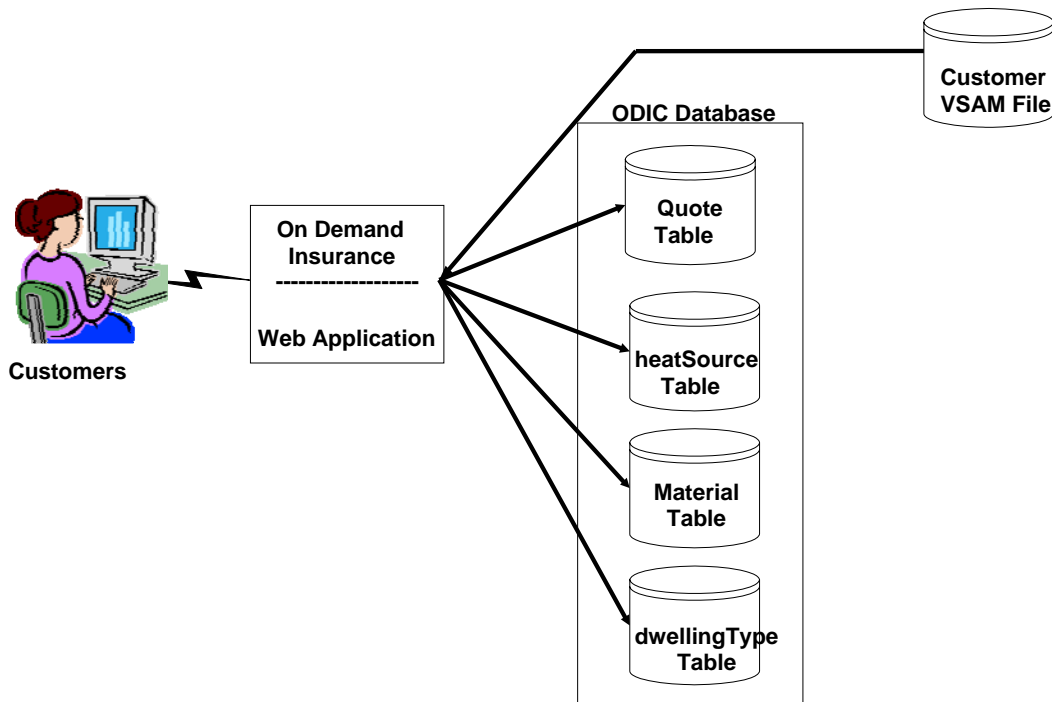
Vision Statement

The ODIC web application provides preferred customers with three essential features. They can view general insurance information, request an informal homeowner insurance policy quote (please see Glossary for further information), and they can view all of the quotes that they have requested previously as well as details about those previous quotes.

Use Case Model



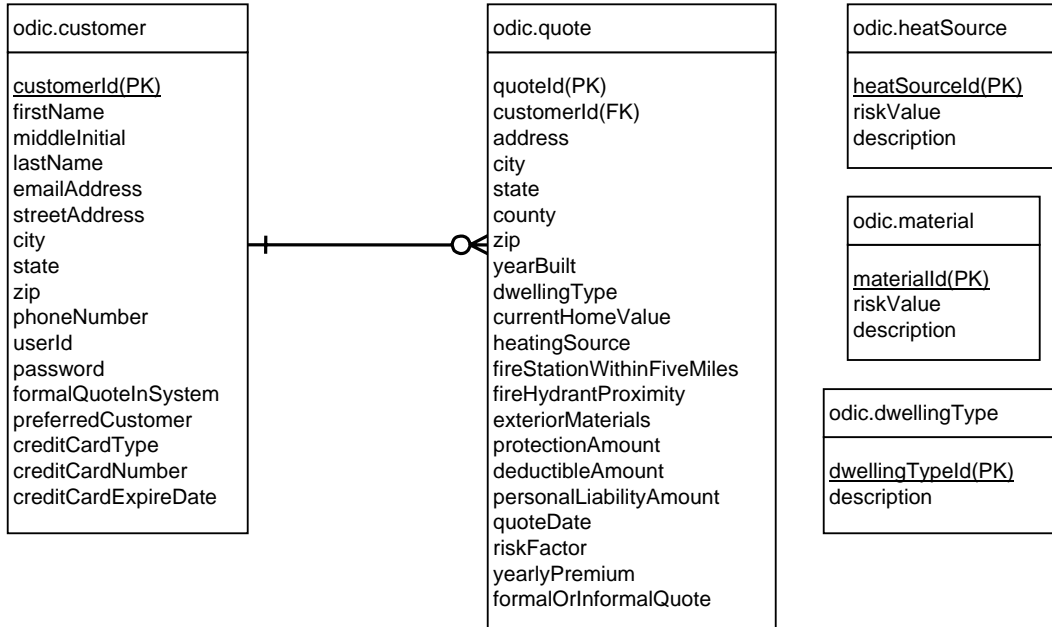
Architecture Diagram



Note: All the customer input from the web application is stored in the "Quote Table". ODIC DB is SQL Server for Microsoft Test Cases - SQL Server 2005 Enterprise Edition SP2. ODIC DB is DB2 for IBM Test Cases - DB2 UDB Workgroup Edition V 9.1.

Database Schema

ODIC	
Description: On Demand Insurance	
Database: DB/2	Rev: 1.1
Filename: ODIC Schema	Company: ODIC



Features and Use Cases

Feature 1: Application startup

Preferred customers - henceforth referred to as customer(s) - can access the web application by entering the URL provided to them in the promotional letter sent by ODIC.

Use Case 1.0 Basic Flow:

The customer enters the URL in the browser and the system displays the Login page.

Feature 2: Authentication

Customers supply their userid and password at the Login page and are authenticated by the system. The Login page is the initial page presented to the customer.

Use Case 2.0 Basic Flow:

The customer is prompted for userid and password and inputs them. The system validates that information and then presents the Main page. The system stores pertinent information about the customer (customer ID, name) in session scope.

Use Case 2.1 Alternate Flow:

The customer only supplies one of the two required parameters. The system provides a message indicating the missing parameter and redisplay the Login page.

Use Case 2.2 Alternate Flow:

The customer supplies a userid that is not in the database. The system provides a message indicating that and redisplay the Login page.

Use Case 2.3 Alternate Flow:

The customer supplies a password that does not match the provided valid userid. The system provides a message indicating that and redisplay the Login page.

Feature 3: Application termination

Customers can exit the system.

Use Case 3.0 Basic Flow:

Customer selects the Logout link from any page where it is displayed and the system logs the customer out and displays the Login page. The Logout link is displayed on all pages except the Login page.

Feature 4: View insurance information

Customers can view general insurance information by selecting the Insurance Info link on any page after logging on.

Use Case 4.0 Basic Flow:

From any page after logging on the customer selects the Insurance Info link and the system displays that page.

Feature 5: Request informal quote

Customers can request an informal homeowner insurance policy quote from any page after logging on. This request will return an immediate response to the customer based on the information that the customer provides, insurance data, and a risk factor calculation.

Use Case 5.0 Basic Flow:

From any page following logging on, the customer selects the Homeowner Quote link and the system displays that page. The customer then provides information regarding the home and insurance coverage needed. This information is gathered from several pages. The last page will contain a Request Quote button that will allow the customer to submit the request. Each page preceding the page with the Request Quote button will contain a Continue button, which allows the customer to proceed to the next page in the process.

The customer provides the following information:

- Dwelling protection amount (e.g. 250,000)
- Deductible amount (100, 250, 500, 1000, 2000)
- Personal liability amount (100,000, 300,000, 500,000, 1,000,000)
- Dwelling type (Single family, duplex, apartment, condominium, townhouse, mobile home)
- Location (Address, city, state, county, zip)
- Heating source (Oil, natural gas, electric, wood stove)

- Year dwelling built
- Current home value
- Exterior materials (Brick, wood frame, vinyl or aluminum, hardy plank, stucco, log, cinder block, stone)
- Fire station within five miles (yes, no)
- Fire hydrant proximity (In feet)

Use Case 5.0.1 Request Informal Quote:

Immediately following Use Case 5.0, the system calculates a risk factor (see algorithm below).

Use Case 5.0.2 Risk Factor Acceptable – Less Than 41:

The system evaluates the risk factor calculated in Use Case 5.0.1 and it is less than 41. The risk factor is acceptable. The system calculates the premium and displays the following information on the Results page:

- Message stating that ODIC would be happy to provide homeowner insurance to this customer. System displays a telephone number for the customer to pursue the matter further.
- Dwelling protection amount (e.g. 250,000)
- Deductible amount (100, 250, 500, 1000, 2000)
- Personal liability amount (100,000, 300,000, 500,000, 1,000,000)
- Yearly insurance premium

The system stores all of the data gathered and calculated as part of this quote in the quote table. The system provides an Another Quote button for the customer to request another quote.

Use Case 5.0.3 Risk Factor Not Acceptable – Greater Than or Equal to 41:

The system evaluates the risk factor calculated in Use Case 5.0.1, and it is greater than or equal to 41. The risk factor is not acceptable. The system displays the following information on the results page:

- Message stating that ODIC cannot offer a quote based on the information provided.

The system does not store any data related to this quote in the quote table. The system provides an Another Quote button for the customer to request another quote.

Use Case 5.1 Alternate Flow:

On any page within Use Case 5.0 the customer may select the Logout link. The system logs the customer out and displays the login page.

Use Case 5.2 Alternate Flow:

On any page within Use Case 5.0 the customer may select the Insurance Info link, the Homeowner Quote link, or the View Quotes link from the navigation area and the system displays the appropriate page.

Risk Factor and Premium Algorithm:

This algorithm determines the risk factor as well as the yearly premium amount to be offered to a customer.

Risk will be determined by five factors – age of home, fire hydrant proximity, heat source, fire station proximity, and exterior materials used. Each of the values within each of these factors will have a weight associated with it as follows:

Home Age in Years

0-5	add 1
6-10	add 5
11-20	add 8
> 20	add 10

Fire Hydrant within 1000 Feet

Yes	add 0
No	add 2

Heat Source

Natural gas	add 10
Electric	add 12
Oil	add 14
Wood stove	add 16

Fire Station within Five Miles

Yes	add 0
No	add 2

Material House was built with

Mostly brick	add 10
Mostly stone	add 10
Hardy plank	add 10
Stucco	add 12
Cinder block	add 12
Vinyl or aluminum	add 14
Mostly wood frame	add 16
Log	add 16

The calculation begins by assigning zero to the riskFactor (RF). Then, each of the values that the customer provides are compared with the five categories above and the “add” amounts are added to RF. The first part of the algorithm is complete and the risk factor number is known. Next, this number is used to calculate the premium.

RF is then divided by 20,000 to get the adjustedMultiplier (AM).

The protection amount that the customer desires is then multiplied by the AM. The liability amount that the customer desires is also multiplied by the AM. These two products are then added together to get the preliminaryPremium (PP).

PP is for the lowest deductible (\$100), so it needs to be adjusted for the actual deductible that the customer has requested.

For each level of deductible higher, decrease the PP by .025.

Deductibles can only be one of the following amounts -100, 250, 500, 1000, 2000.

Developer Productivity Study

Example 1 – Lowest Risk Factor Use Case		
Category	Value	Points
Protection	\$250,000	N/A
Liability	\$300,000	N/A
Deductible	\$500	N/A
Age of Home	1 Year	1
Fire Hydrant	Yes	0
Heat Source	Natural Gas	10
Fire Station	Yes	0
Materials	Mostly Brick	10
		RF = 21

Calculation for Example 1 – Lowest Risk Factor Use Case

AM = 0.00105, (21/20,000)
 250,000 * AM = 262.50
 300,000 * AM = 315.00
 PP = 577.50, (262.50 + 315.00)
 Deductible (\$500) is two units above the base, so
 deductibleAdjustment (DA) is (.025)(2)(577.50) = 28.87
 Actual Premium (AP) = 577.50 - DA
 AP = \$548.62

Example 2 – Highest Risk Factor Use Case		
Category	Value	Points
Protection	\$250,000	N/A
Liability	\$300,000	N/A
Deductible	\$500	N/A
Age of Home	> 20 Years	10
Fire Hydrant	No	2
Heat Source	Wood Stove	16
Fire Station	No	2
Materials	Log	16
		RF = 46

Calculation for Example 2 – Highest Risk Factor Use Case

AM = 0.0023, (46/20,000)
250,000 * AM = 575.00
300,000 * AM = 690.00
PP = 1265.00, (575.00 + 690.00)
Deductible (\$500) is two units above the base, so deductibleAdjustment (DA) is (.025)(2)(1265.00) = 63.25
Actual Premium (AP) = 1265.00 - DA
AP = \$1201.75

Feature 6: View quote information

Customers can view all of the quotes that they have previously requested in a line item display and also view further details about each of the quotes.

Use Case 6.0 Basic Flow:

From any page following logging on, the customer selects the View Quotes link and the system displays a list of all of the quotes that the customer has previously requested. The customer then selects one of the line items to see more details about that particular quote. The system displays the rest of that quote's details.

Use Case 6.1 Alternate Flow:

From any page following logging on, the customer selects the View Quotes link and the system displays a list of all of the quotes that the customer has previously requested. The customer then selects one of the line items to see more details about that particular quote. The system displays the rest of that quote's details. The customer clicks on the Back to Quotes button. The system displays the page with the quote line items on them. This cycle can repeat indefinitely.

Use Case 6.3 Alternate Flow:

At any point within Use Case 6 the customer can select the Logout link. The system logs the customer out and displays the login page.

Use Case 6.4 Alternate Flow:

At any point within Use Case 6.0 the customer may select the View Insurance Info link, the Homeowner Quote link, or the View Quotes link from the navigation area and the system displays the appropriate page.

Non Functional Requirements

NF Requirement 1: Credential Storage

Userid and password used for authentication are stored in the ODIC relational database. Checking customer provided values against entries in the database performs authentication.

NF Requirement 2: Customer Data Access

ODIC customer information used to calculate the Risk Factor for an informal quote is stored within a VSAM file that is also used as input to other applications outside the scope of the productivity study.

Developer Productivity Study

The following describes the fields from the VSAM file and the associated information.

<i>VSAM File Field</i>	<i>Description</i>
Number	Customer id
Name	First name, middle initial, last name
Address	Address, city, state, zip
Phone	Phone number
Date	Date of formal risk factor calculation
Amount	Premium for formal risk factor
Comment	Formal risk factor value

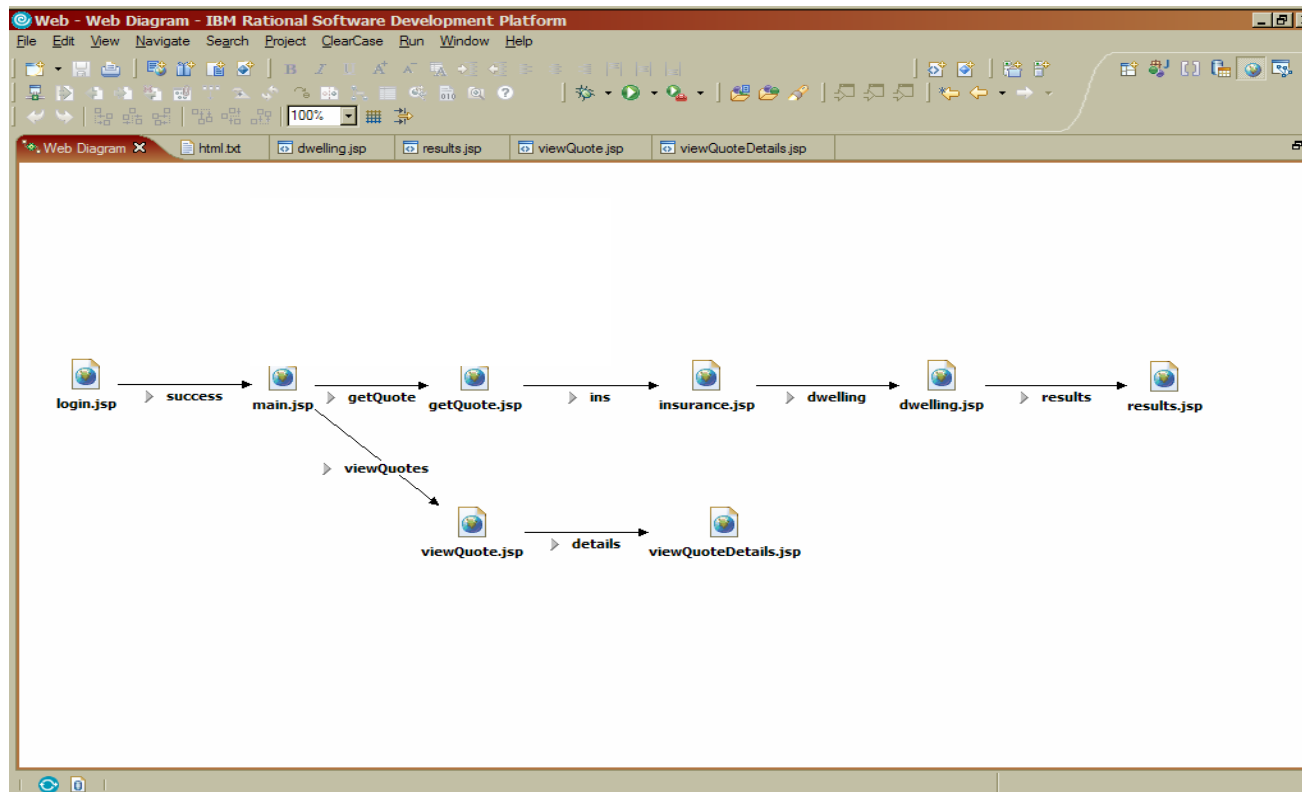
7.2 Application Prototypes

The follow provides the screens of an application prototype that was designed before development began. Of the applications built, Application 1, the Homeowner Policy Quote, was an end user web application and needed to be defined in detail; page by page.

Homeowner Policy Quote Application

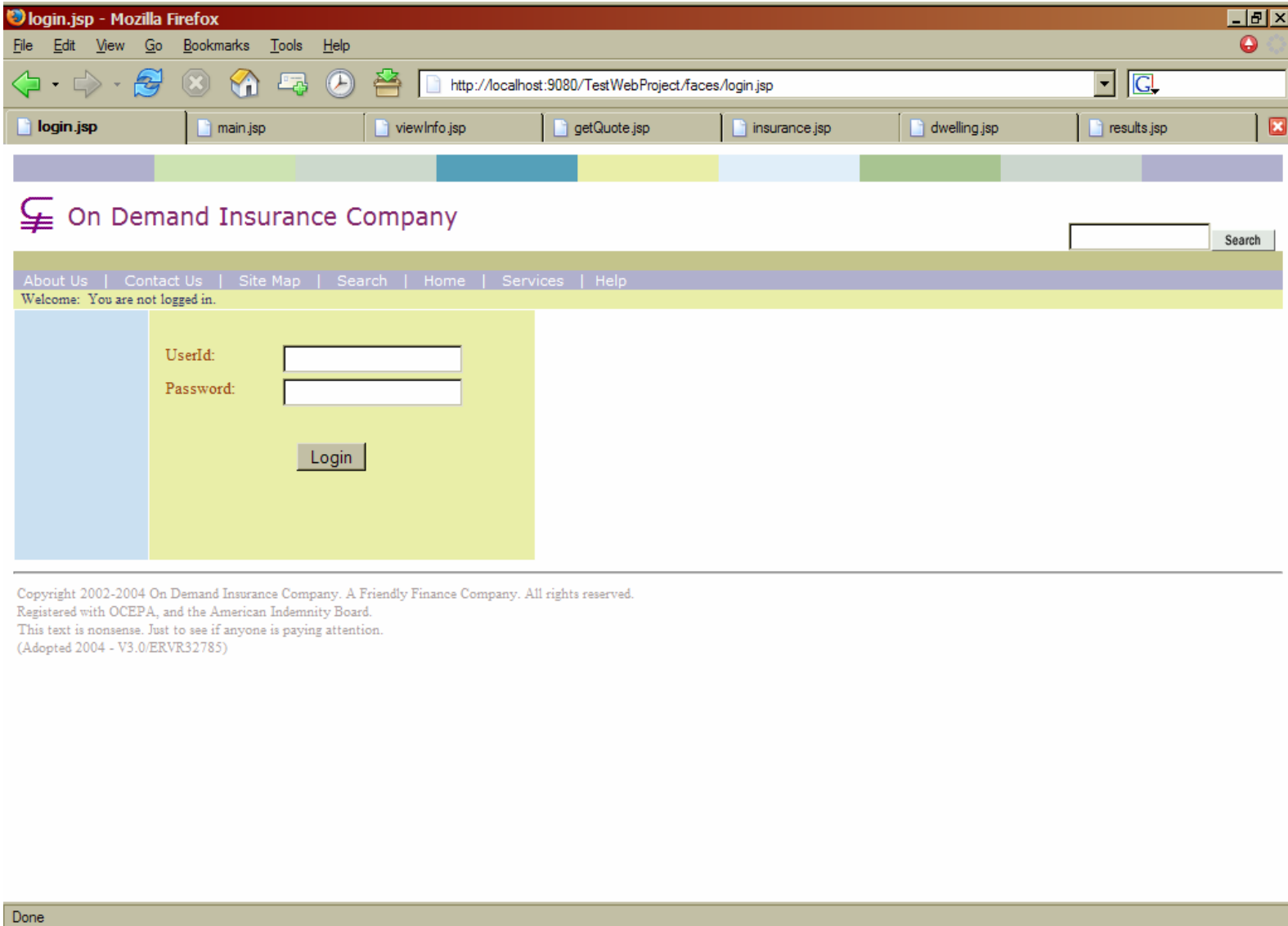
This flow diagram illustrates the flow of the individual On Demand Insurance web application pages. This was designed using the IBM Rational Application Developer tool. The flow is the same for both the IBM and Microsoft environments.

Informal Quote



The next nine pages are the prototype pages that were designed before any coding began. They detail the requirements as to what the customer will see as they step through each part of the application. The prototype pages were originally designed using the IBM Rational Application Developer. Both the IBM and Microsoft developer teams had to build their applications to this prototype.

1. Login Page



2. Main page - Personalized greeting for customer

The screenshot shows a Mozilla Firefox browser window with the title 'main.jsp - Mozilla Firefox'. The address bar displays 'http://localhost:9080/TestWebProject/faces/main.jsp'. The browser's tab bar shows several open tabs: 'login.jsp', 'main.jsp', 'viewInfo.jsp', 'getQuote.jsp', 'insurance.jsp', 'dwelling.jsp', and 'results.jsp'. The main content area features a purple logo for 'On Demand Insurance Company' and a search bar. A navigation bar includes links for 'About Us', 'Contact Us', 'Site Map', 'Search', 'Home', 'Services', and 'Help'. A personalized greeting reads 'Welcome: Billy Corgan.' with a 'Logout' link. A sidebar on the left contains links for 'Insurance Info', 'Homeowner Quote', and 'View Quotes'. The main content area has a yellow background with the heading 'Paying Too Much for Homeowners Insurance?' and the text 'Get a Free Quote! Select Homeowner Quote Link to the Left.' The footer contains copyright information: 'Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved. Registered with OCEPA, and the American Indemnity Board. This text is nonsense. Just to see if anyone is paying attention. (Adopted 2004 - V3.0/ERVR32785)'. The status bar at the bottom shows 'Done'.

3. GetQuote page

The screenshot shows a Mozilla Firefox browser window with the address bar set to `http://localhost:9080/TestWebProject/faces/getQuote.jsp`. The browser's tab bar shows several open tabs: `login.jsp`, `main.jsp`, `viewInfo.jsp`, `getQuote.jsp` (active), `insurance.jsp`, `dwelling.jsp`, and `results.jsp`. The page content includes a logo for 'On Demand Insurance Company' and a search bar. A navigation bar contains links for 'About Us', 'Contact Us', 'Site Map', 'Search', 'Home', 'Services', and 'Help'. Below the navigation bar, a welcome message reads 'Welcome: Billy Corgan.' with a 'Logout' link. The main content area is a form titled 'Please Enter Home Information' with the following fields: 'Address', 'City', 'County', 'State' (a dropdown menu currently showing 'AL'), 'Zip Code', and 'Year Built'. A 'Continue' button is positioned below the form. On the left side of the form, there is a vertical menu with links for 'Insurance Info', 'Homeowner Quote', and 'View Quotes'. At the bottom of the browser window, a status bar displays the word 'Done'.

Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved.
Registered with OCEPA, and the American Indemnity Board.
This text is nonsense. Just to see if anyone is paying attention.
(Adopted 2004 - V3.0/ERVR32785)

4. Insurance page

insurance.jsp - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:9080/TestWebProject/faces/insurance.jsp

login.jsp main.jsp viewInfo.jsp getQuote.jsp insurance.jsp dwelling.jsp results.jsp

On Demand Insurance Company Search

About Us | Contact Us | Site Map | Search | Home | Services | Help

Welcome: Billy Corgan. [Logout](#)

Please Enter Insurance Information

Protection Amount: Personal Liability Amount: \$100,000

Current Home Value: Deductible Amount: \$100

Continue

Insurance Info

Homeowner Quote

View Quotes

Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved.
Registered with OCEPA, and the American Indemnity Board.
This text is nonsense. Just to see if anyone is paying attention.
(Adopted 2004 - V3.0/ERVR32785)

Done


5. Dwelling page

dwelling.jsp - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:9080/TestWebProject/faces/dwelling.jsp

login.jsp main.jsp viewInfo.jsp getQuote.jsp insurance.jsp dwelling.jsp results.jsp

 On Demand Insurance Company

Search

About Us | Contact Us | Site Map | Search | Home | Services | Help

Welcome: Billy Corgan. [Logout](#)

Please Enter Dwelling Information

Fire Hydrant Proximity (In Feet): Dwelling Type:

Heat Source: Exterior Materials:

Fire Station Within Five Miles:

Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved.
Registered with OCEPA, and the American Indemnity Board.
This text is nonsense. Just to see if anyone is paying attention.
(Adopted 2004 - V3.0/ERVR32785)

Done

6. Results page - success

The screenshot shows a Mozilla Firefox browser window with the address bar set to `http://localhost:9080/TestWebProject/faces/results.jsp`. The page title is "results.jsp - Mozilla Firefox". The browser's menu bar includes File, Edit, View, Go, Bookmarks, Tools, and Help. The address bar contains the URL and a search icon labeled "dlink drivers". The browser's tab bar shows two tabs: "results.jsp" and "dwelling.jsp".

The main content of the page is for "On Demand Insurance Company". It features a search bar and a navigation menu with links for About Us, Contact Us, Site Map, Search, Home, Services, and Help. A welcome message reads "Welcome: Billy Corgan." with a "Logout" link. A sidebar on the left contains links for Insurance Info, Homeowner Quote, and View Quotes.

The main content area displays a congratulatory message: "Congratulations! We can offer you a competitive quote on homeowners insurance. Here are the details:". Below this is a table of quote details:

Protection Amount:	\$250,000.00	Liability Amount:	\$300,000.00
Deductible Amount:	\$500.00	Yearly Premium:	\$548.62

Below the table is a button labeled "Another Quote".

At the bottom of the page, there is a copyright notice: "Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved. Registered with OCEPA, and the American Indemnity Board. This text is nonsense. Just to see if anyone is paying attention. (Adopted 2004 - V3.0/ERVR32785)".

The browser's status bar at the bottom shows "Done".

7. Results page - rejection

The screenshot shows a Mozilla Firefox browser window with the following details:

- Browser Title:** results.jsp - Mozilla Firefox
- Address Bar:** http://localhost:9080/TestWebProject/faces/results.jsp
- Open Tabs:** results.jsp, dwelling.jsp
- Page Content:**
 - Header:** On Demand Insurance Company
 - Navigation:** About Us | Contact Us | Site Map | Search | Home | Services | Help
 - Message:** We are terribly sorry, but based on the information that your provided, we cannot offer you a quote at this time. Please choose the Another Quote button to try again with different input.
 - Button:** Another Quote
- Footer:** Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved. Registered with OCEPA, and the American Indemnity Board. This text is nonsense. Just to see if anyone is paying attention. (Adopted 2004 - V3.0/ERVR32785)

8. View quote page

viewQuote.jsp - Mozilla Firefox

File Edit View Go Bookmarks Tools Help

http://localhost:9080/TestWebProject/faces/viewQuote.jsp

login.jsp main.jsp viewInfo.jsp getQuote.jsp insurance.jsp dwelling.jsp results.jsp viewQuote.jsp

On Demand Insurance Company

Search

About Us | Contact Us | Site Map | Search | Home | Services | Help

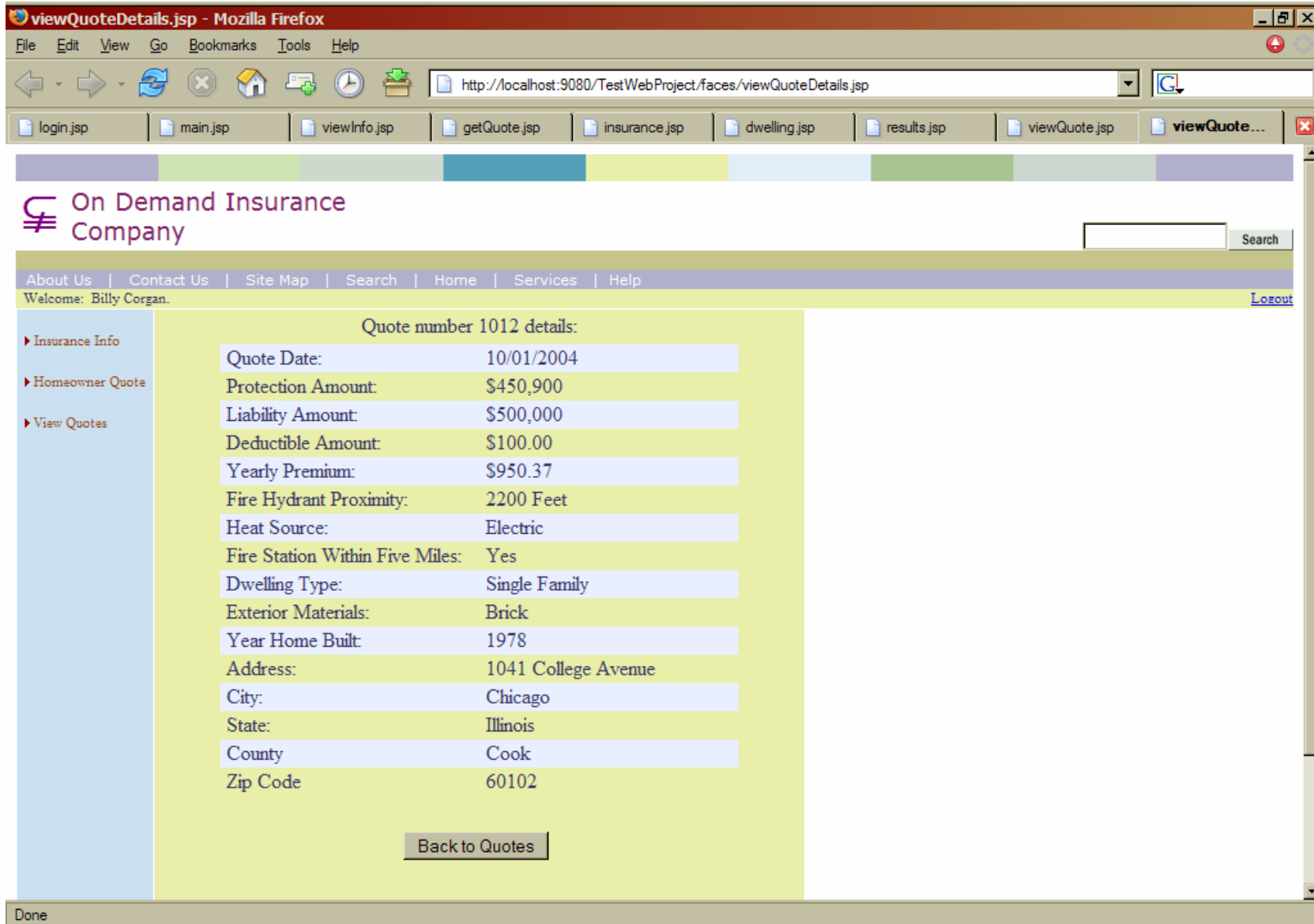
Welcome: Billy Corgan. [Logout](#)

Quote Id	Quote Date	Protection Amount	Liability Amount	Deductible Amount	Yearly Premium
1245	11/15/2004	\$250,000.00	\$300,000.00	\$200.00	\$548.62
1244	11/15/2004	\$250,000.00	\$300,000.00	\$500.00	\$524.85
1094	10/18/2004	\$500,000.00	\$500,000.00	\$100.00	\$845.23
1031	10/02/2004	\$300,000.00	\$300,000.00	\$200.00	\$775.50
1012	10/01/2004	\$450,900.00	\$500,000.00	\$100.00	\$950.37

Copyright 2002-2004 On Demand Insurance Company. A Friendly Finance Company. All rights reserved.
 Registered with OCEPA, and the American Indemnity Board.
 This text is nonsense. Just to see if anyone is paying attention.
 (Adopted 2004 - V3.0/ERVR32785)

Done

9. View quote detail page



7.3 Detailed Steps to build the IBM Applications

The following tables represent both of the applications that developers built within the IBM development environment. The individual line items within each table document the steps used for each of these two applications and provide a reference for each timed step.

IBM Application 1 – Building the Web App for the homeowner policy quote		
Test Case	Object being Created	Detailed Steps to Create the Object
Test Case 1A	Application Setup	1. Create Dynamic Web Project
		2. Setup database connection
		3. Modify data definitions
		4. Modify Dwelling EGL
		5. Modify Heat Source EGL
		6. Modify Material EGL
		7. Modify Customer EGL
		8. Modify Quote EGL
		9. Generate EGL Source
		10. Redirect Configuration
		11. Import Page template
		12. Create login status code
		13. Create common EGL functions
Test Case 1B	Login Page	14. Create JSP file from template
		15. Create business logic
Test Case 1C	Main Page	16. Build main page from two templates
		17. Layout html correctly
		18. Place text in right table
Test Case 1D	GetQuote Page	19. Create page from template and third template
		20. Create business logic
Test Case 1E	Insurance Page	21. Create page from template and third template
		22. Create business logic
Test Case 1F	Dwelling Page	23. Create page from template and third template
		24. Create business logic
Test Case 1G	Results Page	25. Rejection
		26. Success
		27. Create Success business logic

IBM Application 1 – Building the Web App for the homeowner policy quote		
Test Case	Object being Created	Detailed Steps to Create the Object
Test Case 1H	ViewQuotes Page	28. Create page from template and third template
		29. Create business logic
Test Case 1I	ViewQuoteDetails Page	30. Create page from template and third template
		31. Add business logic and navigation

IBM Application 2 - Building a Web Service from scratch		
Test Case	Object being Created	Detailed Steps to Create the object
Test Case 2A	Creation of Implementation Code for Web Service	1. Workspace configuration
		2. Creation of Dynamic Web Project
		3. Creation of Package Structure
		4. Import of package
		5. Creation of Classes
Test Case 2B	Generation of Services	6. Creating Web Service
	Starting of Server	7. Selecting properties on Web Service
Test Case 2C	Testing Web Service	8. Start Web Service Explorer

7.4 Detailed Steps to build the Microsoft Applications

The following tables represent both of the applications that developers built within the Microsoft .NET development environment. The individual line items within each table document the steps used for each of these two applications and provide a reference for each timed step.

Microsoft Application 1 – Building the Web Application for the homeowner policy quote		
Test Case	Object being Created	Detailed Steps to Create the object
Test Case 1A	Application Setup	1. Create ODIC database, run sql scripts to create schema and load data
		2. Create the ODIC_login id that the app will use for logging into SQL Server
		3. Add Connection to Server Explorer
		4. Create ASP .NET Web Application project in VS .NET
		5. Create project structure
		6. Create DataLayer
		7. Create SessionHelper Class
		8. Create Business Rule Helper
		9. Import layout.htm, the common UI layout (provided by a fictitious graphics design team)
		10. Import image files (provided by a fictitious graphics design team)
		11. Create ASP .NET User Controls to encapsulate the UI layout
		12. Create Login Status Line
		13. Create a stylesheet with the various styles used in the application (import CSS provided by a fictitious graphics design team)
		14. Create MasterPage
Test Case 1B	Login Page	15. Add new ASP .NET web form titled login.aspx to project using ODIC.master page
		16. Drag and drop Web form controls into the content cell. This step lays out UI elements specific to the page (labels, textboxes, button).
		17. Write event handling logic for the click event of the Login button
		18. Edit web.config to enable Forms authentication
		19. Create Secure Pages area

Microsoft Application 1 – Building the Web Application for the homeowner policy quote		
Test Case	Object being Created	Detailed Steps to Create the object
Test Case 1C	Main Page	20. Add new ASP .NET web form titled main.aspx to project and use ODIC master Page
		21. Drag Quote image to the page
Test Case 1D	GetQuote Page	22. Add new ASP .NET web form titled getQuote.aspx to project using ODIC masterpage and place in SecurePages area
		23. Drag and drop Web form controls into the content cell. This step lays out UI elements specific to the page (labels, textboxes, dropdown list and a button)
		24. Populate Drop Down List
		25. Write event handling code
Test Case 1E	Insurance Page	26. Add new ASP .NET web form titled insurance.aspx to project using ODIC masterpage and store in SecurePages section
		27. Drag and drop Web form controls into the content cell
		28. Populate Drop Down Lists
		29. Write event handling code
Test Case 1F	Dwelling Page	30. Add new ASP .NET web form titled dwelling.aspx to project using ODIC master page and SecurePages
		31. Drag and drop Web form controls into the content cell
		32. Populate Drop Down Lists
		33. Write event handling code
Test Case 1G	Results Page	34. Add new ASP .NET web form titled results.aspx to project using ODIC template and Secure Pages
		35. In PageLoad event: Calculate risk
		36. Create NoQuote.aspx page
		37. Create QuoteResult.aspx page
		38. Add Link Button to NoQuote.aspx
Test Case 1H	ViewQuotes Page	39. Add new ASP .NET web form titled viewQuotes.aspx to project, ODIC master page and SecurePages
		40. Drag and drop a DataGrid control into the content area. Bind the DataGrid to the dataset.
		41. Set background, alignment, forecolor and other properties for the DataGrid as well as for the Grid's individual columns. Note Currency columns use {0:c2} format and date columns use {0:d}

Microsoft Application 1 – Building the Web Application for the homeowner policy quote		
Test Case	Object being Created	Detailed Steps to Create the object
Test Case 1I	ViewQuoteDetails Page	42. Add new ASP .NET web form titled viewQuoteDetails.aspx to project using ODIC masterpage and SecurePages.
		43. Set up table with header and footer information. Add Link button (Back To Quotes, viewQuote.aspx).
		44. Create a DataView control, bind it to the DataSource using the query param of quote ID. Set alignment, forecolor and other properties for the labels. Order fields.

Microsoft Application 2 - Building a Web Service from scratch		
Test Case	Object being Created	Detailed Steps to Create the object
Test Case 2A	Develop a Web Service from scratch	1. Create WebService Project
		2. Create Business Objects (Person, Address, CreditReport)
		3. Create Service methods: <ul style="list-style-type: none"> a. GetData (private) b. GetCreditReport (webmethod) c. GetScore (webmethod) d. GetAddress (webmethod)
	Create Test Harness	4. Add New Project (ASP.NET). Add reference to WebService.
		5. Add PropertyHelper class
		6. Add table and enter labels and text boxes
		7. Add and bind 3 buttons to call each of the functions in the webservice. Output the response objects to the response label.

7.5 Glossary

ODIC – On Demand Insurance Company (a fictitious company outlined for the purpose of this study).

HO Policy – Homeowner Policy.

Basic Customer – Constitutes an On Demand Insurance Company customer with less than two policies through the organization (Auto, Life, Homeowner).

Formal Quote – A formal quote has the same risk factor calculation as the informal quote, but has the exception that the result is not shown to the customer. Instead, the formal quote risk factor is passed to a workflow for further processing (beyond the scope of this study). A formal quote is asynchronous; customer receives a message stating that their information has been received and that On Demand Insurance Company will respond within 24 hours.

Informal Quote – This type of quote provides an automated response to customer requests and contains a price quotation for homeowner's insurance. This determination is provided by a risk factor calculation utilizing specific information provided by the customer such as the level of insurance desired and the type of dwelling to be insured. Based on the calculation, the application determines whether or not On Demand Insurance Company will grant the insurance policy, and if so, what the projected yearly costs will be. An informal quote has the following characteristics:

- Synchronous – Customer gets an immediate quote response.

Preferred Customer – Constitutes an On Demand Insurance Company customer that holds two or more insurance policies with the organization (Auto, Life, Homeowner).

Risk Factor – A number calculated in the business logic for Use Case 5. This number is used to determine whether or not a quote will be extended to the customer and is integral to the calculation of the premium amount. A quote will be offered to a customer if the risk factor is 40 or lower.