# Developer Productivity Study

## Comparing IBM® WebSphere® Developer for System z to Traditional IBM Mainframe Development 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

November 2006

**Important Notice**



**Trademarks**

- IBM, CICS, DB2, Rational, WebSphere, and System z are either registered trademarks or trademarks of IBM 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 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 for the mainframe. The application, in its completed form allows staff and eventually external brokers and associates to request a quote for a homeowner insurance policy as well as view existing insurance quotes. To build all of this functionality, a COBOL application (consisting of eight programs), two web services, and a web-based application were reviewed, designed and built by Branham Group Inc. Branham Group used a seasoned COBOL/CICS/DB2 application developer to review the design of the components as well as build them. Finally, Branham Group followed a strict methodology (detailed in section 4.0) to assure unbiased results.

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 WebSphere Developer for System z (WDz), based on the Rational Software Development Platform, and traditional IBM mainframe development tools for COBOL, CICS, and DB2 based applications, based on the z/OS platform. Direct measurements are made between equivalent tools from both environments that accomplish specific tasks in building fully functional applications. Starting with rudimentary functionality, this application increases in sophistication through three phases of implementation. Each phase builds on the previous in both complexity and capability in an attempt to demonstrate the realistic portrayal of an inevitable increase in requirements based on customer demand and the discovery of new opportunities within the organization. This scenario provides an accurate measurement of the productivity of the examined tools, and their ability to respond to these additional requirements.

## 2.2 Goals of this Study

This paper reflects on the results of a four (4) month intensive study that takes an objective look at the productivity differences between specific tools to build a “real world” mainframe 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 mainframe related 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 the tooling when building the application components.
4. Document the productivity measurement differences as well as any issues that were observed between IBM WDz and traditional mainframe development tools in the course of the study.

### In summary:

**Which development environment is more productive for constructing robust User Interfaces, Connectivity via Web Services and Business Services?**

**IBM WDz or Traditional Mainframe Development Tools**

## 2.3 The Tools Compared

In order to build the applications based on the specified requirements, the following tools were used. This list does not include additional support software that was required. For the entire list, please refer to section 4.0. In all cases, only the latest supported developer tools 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**

| Rational SDP | Traditional Mainframe Development Tools |
|---|---|
| • WebSphere Developer for System z V6.0.1<br>• Subcomponents include:<br>• Remote System Explorer (RSE)<br>• BMS Editor<br>• z/OS LPEX Editor<br>• Service Flow Modeler (SFM)<br>• Host Connection Emulator (HCE)<br>• Operations Editor<br>• Mapping Editor | • TSO/E<br>• ISPF<br>• Screen Definition Facility II (SDF 2) |

## 2.4 Productivity Study Results

In an effort to provide additional granularity with respect to measurable levels of productivity, application development was divided into additional sub stages. For example, the entire mainframe based application was divided into eight separate functional applications and the productivity measurements focused on each of these eight applications. A detailed description of each application is documented in Section 3.3 Table 3.

### Summary of Results

1. For simple COBOL applications, where there were few if any coding errors, traditional mainframe development tools and WDz where relatively similar in productivity. This means that after a short initial learning curve, WDz can make a seasoned mainframe programmer as efficient in WDz as traditional mainframe development tools, for simple applications.
2. In more complicated real-world applications, WDz demonstrated higher productivity, particularly in the area of compile, test, and debug. Where coding and syntax errors existed, WDz was significantly more productive, being up to 44% faster than traditional IBM mainframe development tools.
3. The WDz tools were able to meet and exceed the requirements for all components of the applications, including the web services generation and the creation of the web application.
4. Without WDz, the development of Web Services from the new COBOL application would require a significant amount of hand coding or multiple third party tools. WDz significantly eased this operation through its Web Services generation wizard, without a requirement for manual coding.
5. The WDz environment created test environments for the developer such as when using the Web Services Explorer for testing newly created and deployed Web Services. Otherwise, these tests would have required manual coding lowering productivity.
6. The traditional mainframe development environment was unable to easily* meet the development of the Web Services and the web applications. While it would be possible, the manual creation would have required hand coding in conjunction with significant knowledge in the development of Web Services. While possible, it would be ill advised to do so.

**easily*** = There was no built in feature or wizard to provide this function in the traditional mainframe development environment. Given enough time and expertise, any function could be hand coded. In some cases, where the function was missing in the mainframe tooling and where it would have been very complex and time consuming to hand code the function, the decision was made to state the function could not "easily" meet the requirements.

| Table 2 – Summary of the Productivity Study | | | |
|---|---|---|---|
| **Application** | **WDz** | **Productivity Difference** | **Traditional Tools** |
| 1. COBOL/CICS/DB2 Application | 199 | ← **1.2X** | 228 |
| 2. Web Service built from existing Code | 5 | Unsupported[1] | N/A |
| 3. Build a Service Flow | 47 | Unsupported[1] | N/A |
| 4. Build a web based interface to Web Service | 20 | Unsupported[1] | N/A |

**NOTE:** All times to develop the necessary applications recorded in minutes.

[1] While it is possible to do this development by hand, this is not realistic and as such was deemed as an unsupported feature of the traditional IBM mainframe development tools.

## Conclusion:

**IBM WDz tools are more productive for building robust mainframe and web based applications**

# 3.0 Introduction

## 3.1 Ground rules of the study

Ground rules that were strictly observed during the entire study included:

1. Tools must be generally available 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 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. 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. The company now wishes to provide some traditional access to its mainframe allowing a wider range of employees to access existing quotes and request new quotes on customers’ behalf. If this implementation is successful, the company plans to make this quoting process available to its larger external partner brokers through a web service. Again, if this process is successful, the company wishes to build a web-based application to make this quote request process available through its portal site, to smaller associates and potentially customers.

The applications built will focus on the following areas as depicted in

Diagram 1:

1. Corporate IT – where claims data, payment and records of the policy are stored
2. Risk Management – will use Insurance based Risk Assessments to calculate the policy quote
3. Human Resources – where staff can request new quotes for customers on behalf of its brokers
4. Business Partner Relations – where business partners access the process through a web service
5. Corporate Portal – where associates and potentially customers can access the new quote functionality through a web based application

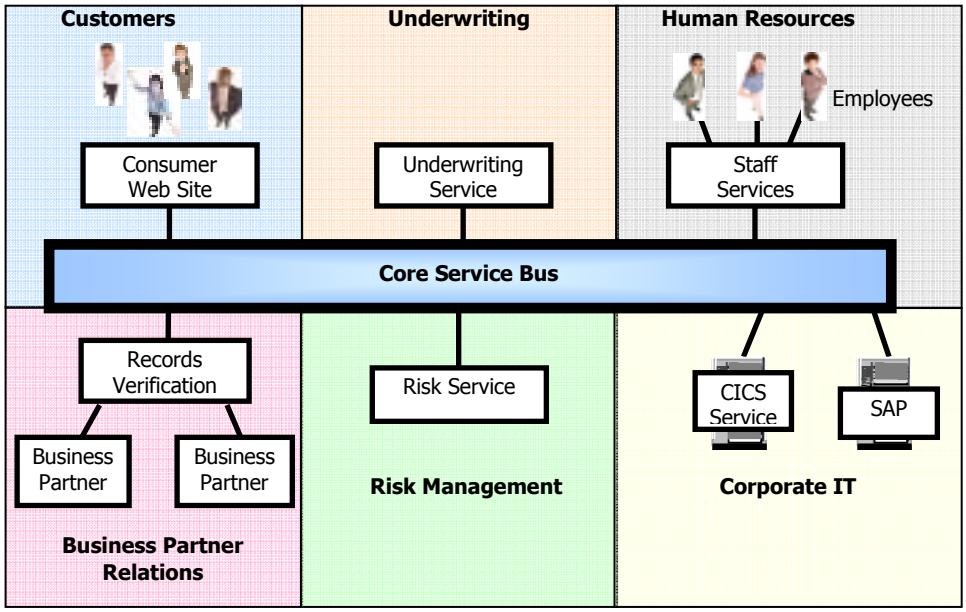



**Diagram 1 – On Demand Insurance Company**

## 3.3 The Phases of the Application

In order to make the application development more reflective of real world scenarios, new application functionality was implemented in three phases. Each phase assumed the previous phase was successful and built upon the previous phase by adding functionality to the overall application. Each phase also added additional challenges for the developers.

### Phase 1

ODIC initially wanted to make a terminal based application so that internal brokers and employees can access customer data and their existing quotes. ODIC also wanted to give brokers the opportunity to provide their customers with new quotes based on information provided by customers. ODIC decided that if this implementation were successful, it would implement Phase 2, making the information and quote process available to partner brokers.

### Phase 2

ODIC saw great success and a significant decrease in response times to customers through a successful implementation of Phase 1. As such, ODIC wanted to open these processes to its partner brokers through the availability of a web service. This would allow external partner brokers to design their own front end to the applications, while directly accessing customer details and quote processes. If Phase 2 were successful, then ODIC would invest in Phase 3, which would extend this same feature functionality to the web.

### Phase 3

ODIC experienced a large increase in new homeowner policies being granted which was a direct result of applications being quickly processed due to the Phase 2 implementation. The company decided to extend the Phase 2 functionality to the web, making it easier for field agents and smaller external brokers to access the same processes developed in Phase 1. Eventually, this could be extended even further to provide the ability for a customer to review their profile and any claims information on-line.

## 3.4 The Applications that were built

Table 3 provides a description of each of the applications built, and to which of the three phases the applications correspond.

| Table 3 – Description of the Applications | | |
|---|---|---|
| **Phase** | **Name of Application** | **Description of Application** |
| **1** | **1. COBOL/CICS/DB2 application** | This application allows staff members to enter customer details for calculating a new quote, or reviewing existing quotes. It consists of eight programs. |
| **2** | **2. Web Service built from existing code** | This web service extends the COBOL/CICS/DB2 application allowing larger external broker partners to request new quotes and view existing quotes on their customers' behalf. This eliminates the paper based process and the requirement to contact ODI directly via telephone, significantly reducing customer response times. |
| **2** | **3. Build a service flow** | This web service helps to automate multiple steps in a business process, reducing the development requirements for the end user application. |
| **3** | **4. Web application built for existing portal** | This application uses the web services built in Phase 2. This page calls the web service allowing associate brokers to access the new quote process without the requirement to build their own front end application to call the web service. |

## 3.5 The Tools that were used

The following requirements were adhered to for the developer tools that were used in this comparison study:

1. The tool had to be generally available and fully supported by IBM.
2. No beta code or early code was allowed.
3. The tool and the procedure used had to be fully documented. No undocumented procedures or techniques were allowed.

Table 4 provides a summary of the base tools used in the comparison. For a complete list including any additional software used, considered support software, please see "The Tools Productivity Methodology" section within 4.0.

| Table 4 - Tools used in the Productivity Study | |
|---|---|
| **Rational SDP** | **Mainframe Environment** |
| • WebSphere Developer for System z V6.0.1 containing:<br>  • BMS Editor<br>  • LPEX Editor<br>  • Service Flow Modeler | • ISPF<br>• SDF II |

# 4.0 The Tools Productivity Methodology

## 4.1 The Environment Methodology[1]

In order to achieve an environment that was typical of today's mainframe environment and development practices, the following were used.

1. A Mainframe running z/OS version 1.6, and other supporting software included CICS, DB2, etc.
2. For the development using a traditional mainframe development environment, a TSO connection through a 3270 emulator was used.
3. The WDz environment was created within a VMWare image so that the environment could be easily duplicated between testing, allowing for incremental snapshots and the ability to revert to previous states.

### Base and Support Software Required

The following base and support software were used in this study to meet the requirements of the insurance application.

| Table 5 - Base Software | |
|---|---|
| **WDz Environment** | **Mainframe Environment** |
| • Windows XP Professional SP2<br>• WebSphere Developer for System z V6.0.1 | • z/OS V1.6<br>• DB2 Universal Database for z/OS V8.1<br>• Enterprise COBOL for z/OS V3.3 (This level required to support WDz)<br>• CICS Transaction Server for z/OS V3.1 (This level required for native Web Services support)<br>• USS |

| Table 6 - Support Software | |
|---|---|
| **WDz Environment** | **Mainframe Environment** |
| • 3270 emulator | • SDF II<br>• ISPF<br>• 3270 emulator |

## 4.2 The Measurement Methodology

The Branham Group designed and built a fully functional COBOL/CICS/DB2 application using both traditional mainframe development and WDz tools. The Insurance application performed the following functions:

1. Staff access a CICS session and apply for an informal homeowner quote to insure their home.
2. Staff change various homeowner parameters, accumulate different homeowner quotes, and view them.
3. Staff eventually request a homeowner insurance quote.

[1] Proper licensing was obtained and was in place for both IBM and the Branham Group for the usage of VMWare and all the IBM products including WDz, DB2, CICS TS, z/OS, etc.

IBM Developers created a detailed requirements document, which was reviewed by Branham Group developers that outlined all the functionality that the entire Insurance application needed to achieve. This document includes the prototypes of each application, as it should look in its final form. This requirements document is available separately. For details on downloading this separate document, please see the Appendix of this paper.

In order to provide all of the required functionality, Branham Group designed and built 11 components within a traditional mainframe development environment (where possible) and IBM WDz. The 11 components consisted of 8 COBOL/CICS/DB2 based applications (included browse and input screens for customers, building materials, proximity to a fire hydrant, etc.), 2 web services, and 1 web based application. Appendix sections 8.2 and 8.3 described each component in detail. Each component was reviewed to assure that it met the following criteria:

1. The approach was considered best practice in the development community.
2. The approach was using the latest technology provided by both environments.
3. The approach was well documented and understood.

Although the Branham developer had extensive experience with traditional IBM mainframe development tools and no experience with WDz, to help eliminate bias based on experience levels, each of the 8 COBOL/CICS/DB2 components were built multiple times to help reduce any learning curve associated with the WDz development environment. Once the optimal level of productivity was reached, each component was timed in order to document the productivity measurement.

# 5.0 The Applications that were built

Table 7 provides a list of the eleven components and applications that were built, including the corresponding developer tools that were used for each application. The first eight components collectively form the COBOL/CICS/DB2 application.

**Table 7 – Components, Applications and Tools used**

| Name of Component/Application | WDz Tools Used | Mainframe Tools Used |
|---|---|---|
| **1. Menu Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **2. Customer Browse** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **3. Policy Browse Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **4. Quotation Browse Policy** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **5. Dwelling Type Browse Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **6. Materials Browse Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **7. Heat Source Browse Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **8. Request Quotation Panel** | • WebSphere Developer for System z V 6.0.1 | • ISPF<br>• SDF II |
| **9. Request Quotation Web Service** | • WebSphere Developer for System z V 6.0.1 | N/A |
| **10. Customer Browse Service Flow** | • WebSphere Developer for System z V 6.0.1 | N/A |
| **11. Request Quotation Web Application** | • WebSphere Developer for System z V 6.0.1 | N/A |

# 6.0 The Productivity Results

The Branham developer was new to WDz but was seasoned in traditional IBM mainframe development tools. As a result there was a slight learning curve for the WDz tool. However, for the creation of simple applications with little if any source errors, the developer was virtually equally productive in both the traditional mainframe and WDz development environments.

Where the differences in these development environments started to appear was in more real world, larger and more complex applications. Specifically, the WDz environment proved to be more productive. Where WDz showed significant productivity gains was in its edit, compile, and debug functionality. To illustrate this, a comparison of procedures for traditional mainframe development versus WDz development is required.

*Traditional Procedure:* Using traditional interfaces, once developers finish editing their applications they exit the source. The next step is to edit the JCL and submit the compile job. Developers can then swap to SDSF, select the job, locate any error message, and find the offending code line. Swapping back to the edit session, developers exit the JCL, enter the source code for the applications, find the line of code with the error, make the necessary changes, and then repeat these steps.

*WDz Procedure:* From within the WDz environment, developers simply edit their source, perform a syntax check, and then double click on any errors to jump to the exact error location for editing. Error checking within the WDz environment also similarly locates errors within linked copy books. Thanks to its multiple windowed interface, WDz allows developers to multitask, viewing multiple windows and having multiple files open simultaneously. This allows faster switching between tasks resulting in additional productivity gains.

Finally, WDz incorporates functionality for generating web services from existing COBOL/CICS/DB2 applications and tools for building web applications that are not available through traditional mainframe development tools. While web services and web-based applications could technically be built within a simple text editor, this is far from economical, and any tools that provide this feature functionality are typically cost effective in comparison to hand coding.

In traditional environments, there are multiple pain points that developers experience. The following provides a more detailed description of some of these pain points, how WDz helps mainframe developers be more productive, and finally some additional recommendations for helping make a developers new experience with WDz even more productive.

## Pain Points Related to Traditional Mainframe Development

There are several areas where productivity on the z/OS platform could be improved. These have been addressed historically by the composition of several vendor products at additional licensing costs. The vendor products available may not provide all of the solutions and are not integrated, leading to inconsistent interfaces and usability concerns. Some of these products have seen dramatic escalation in license fees in recent years.

CICS has been the first choice for interactive applications on the zSeries platform due to excellent performance and reliability in high volume systems and richness of terminal control features. The interactive capabilities are delivered through the Basic Mapping Support (BMS) component, which has evolved over time to support a variety of specialized hardware devices in addition to the IBM 3270 series display terminals. Today however, developers still experience significant issues including:

- ⇨ Exploitation of the performance capabilities requires the development of complex PSEUDO-CONVERSATIONAL logic that involves splitting the process at terminal wait points and maintaining context so that the application can terminate and restart with each interaction. This process delivers a virtual threading capability that increases the number of in-flight transactions orders of magnitude beyond the actual number of real threads configured in the CICS system. Complexity increases

when a panel transition occurs since application fragments from the next panel must be concatenated to the completion of the current panel. This scenario must be reversible to enable return to a prior panel.

- ⇨ Interaction involves the development of Basic Mapping Services (BMS) definitions for each panel with detail for each field along with complex logic to interpret Attention Identification (AID) and a multitude of flags associated with each field on the panel. These flags indicate whether the operator modified the field, used the erase key, the length of data which was entered and the current cursor position (*focus* in GUI terms). The next process for an application can depend on input data, a terminal control function or a combination of both (and sometimes a network function external to the application).
- ⇨ BMS definitions are coded in MACRO Assembler, which leads to an error-prone and tedious process. Although BMS may be developed using MACROs, this is rarely used except for very simple display presentations. Most installations use a tool like Screen Definition Facility II (SDF2), which delivers a character-based graphics editing environment for BMS development.
- ⇨ The most time consuming component of interactive development is the editing of input data, especially when the requirement is to allow flexible and intuitive input of currency, dates and non-numeric coded fields.
- ⇨ Syntax checking in the ISPF environment involves scanning the compile listing for errors and then searching for the related source line in the main program or copy book source. Since ISPF is limited to two windows, only one program file at a time may be viewed while maintaining a view of the compile listing. Unless several TSO logons are available to the developer, this means that repeated file open and closing is required.
- ⇨ CICS has a built-in debugging facility provided by the CEDF transaction. One advantage of this facility is that a production program does not necessarily need to be re-compiled for debugging. The break point process with CEDF is not very productive. Break points can be set by source statement number but this is not practical when large numbers of copy books are used and the numbering sequence is irregular. Usually a break point is created by inserting an "EXEC CICS SUSPEND" statement in the source and then re-compiling. A re-compile is required for each change of the break points. CEDF does not provide symbolic access to data areas. Program data areas must be located by binary offsets in the working storage. The alternative is a vendor tool at additional cost.
- ⇨ When using the ISPF environment the developer may need to download/upload a file for local editing on the workstation or for loading test data. The file transfer capability of ISPF is tedious and must be configured for each file or program element to be transferred. When a file transfer error occurs, the screen is often automatically cleared or over-written before the error conditions can be noted. Even when the error condition is known, several different configurations may have to be tried before the problem is corrected.
- ⇨ ISPF does have some customizing features, which may be used by an experienced developer to improve the cycle time. For example, a sequence of commonly used commands may be assigned to a function key to renumber, save, submit, swap windows and check JES2 status of a compile in one keystroke.
- ⇨ Most z/OS programming languages are still based on 80-col card image standards. ISPF provides specialized exclusion and column-wise editing features, which can be highly productive in special circumstances. This is used to shift or delete a selected column area of a source file or apply a change to a symbol only when it starts in a specific column or only when it appears on the same line as another symbol. This may be accomplished with other text editors, but usually a complex script must be written to perform a similar function.

## WDz Solutions for Development

WDz provides an integrated environment for development or re-engineering of System z applications. As a consolidation tool, it can replace several non-integrated vendor tools once the developers are conversant and comfortable with all of the capabilities of WDz. WDz provides a visual development platform for z/OS host applications that is comparable to visual development tools available for other GUI platforms. Some of the pain points addressed by WDz include:

- The remote system explorer provides file transfer capabilities through drag and drop for a single file, or for a large group of files in different formats. Configuration requirements are minimal and limited to the definition of a filter to select the host file subset (This assumes that z/OS File System Mapping has been pre-configured to local installation standards). This provides ease-of-use, reliability and productivity that is superior to ISPF file transfer.
- The Remote Syntax Check is one of the most valuable features of WDz. An application is syntax checked using all of the actual host resources that would be in effect for a host compile. Errors are corrected by clicking on the syntax message. WDz automatically opens a window for the related program file or copy book, directing focus immediately to the affected source line. This saves manual search time and allows several windows to be opened at once for correction of related errors.
- The WDz LPEX editor (with the ISPF option set) is a good replacement for the ISPF editor. Since editing is localized, it may be faster than the ISPF editor, which requires communication for every function key. The LPEX editor provides a level of instant syntax checking (as you go) on input and highlighting of syntax structures that is not available with ISPF. Local history is maintained by the editor, which permits several levels of UNDO functionality compared to only one level with ISPF. In fact, the single UNDO level in ISPF is available only if the file has not been saved. After a SAVE, there is no UNDO available in ISPF and developers are required to retrieve previous versions from the library management system in use. With WDz, there is no such limitation. Developers can browse through previously saved versions of the file and revert to any of them. A compare capability also provides the ability to display multiple versions and the differences between them.
- Content assistance is available for COBOL and PL/1, which provides a pop-up reference for standard COBOL statements and can populate statements with defined variables. This helps eliminate typing errors resulting in fewer mistakes.
- WDz generates custom compile JCL for each developer for languages where host compile is available. Customization is derived from Project, Folder, File and PDS Member properties (cascaded and inherited).
- WDz works with the Debug Tool for z/OS to provide visual, source level debugging for z/OS host programs, including CICS. Once compiled for the debug tool, an application may be debugged through a GUI on WDz (Remote Debug) or through the CICS or TSO interface on the host (Local Debug). Visual debugging includes the facilities for easily adding breakpoints, watching variables, and even changing variables during execution. Although not used within the scope of this project, WDz offers end-to-end debugging incorporating the outermost web pages, into the COBOL code, and into the CICS invocation.
- The "Enable Enterprise Web Service" wizard can take a CICS COMMAREA application and generate wsdl, XML messages and a driver interface to provide almost instant access to the application from the web. Generated files must be uploaded to the host, the driver must be compiled and the PIPELINE SCAN must be PERFORMED to complete the implementation. The prerequisites are that Web Services are defined in CICS and USS folders are available and accessible. This process is useful for a simple one message function.
- CICS applications, which have separated the communications functions from the business functions, are the best candidates for COMMAREA web conversion. Traditional CICS applications may have communications and business function intertwined in order to deliver a more intuitive user interface. The COMMAREA for these applications usually combines PSEUDO-CONVERSATIONAL context and business function parameters. These applications can still be enabled, however some additional effort is required to default the context information and select business parameters from

the messages. The application may also require alteration to recognize that a terminal facility is not available.

- ⇨ WDz does not address the effort required to edit BMS user input data, beyond building of edit statements with content assist, however it does export this functionality to a different platform when web enablement is used. This permits the use of outboard editing with a client script to reduce network traffic and the use of "Regular Expression" logic to simplify the editing process. From the lines of code counts, it is evident that when COPY books are used for the bulk of redundant logic, most of the coding that is unique to each application function is related to BMS processing and the dynamics of the end-user interface.

## Maximizing WDz Productivity

Any tool, which is not configured correctly or used correctly, can lead to a decline in productivity rather than an improvement. The following guidelines are derived from experience with WDz and should help to achieve the benefits of WDz and shorten the learning curve for new users.

- ⇨ The installation of WDz should include the creation of template workspaces that are pre-configured to include installation standards wherever applicable.
- ⇨ Project properties may be specified at the Project, Folder, File or PDS member levels. The properties are cascaded and inherited. It is not visible at the lower levels that a property is inherited or distinctive. It is also not visible as to which level the inherited property originated. Installation standard properties should be set at the project level and available as an importable project template for each type of project. Properties should be set distinctive only when necessary at lower levels to avoid unexpected results.
- ⇨ Apply installation standards to the BMS pallet items in the workspace preferences of the template workspace.
- ⇨ Provide pre-defined Host Connections with URLs for commonly used Host nodes.
- ⇨ Modify the canned examples to adopt local naming standards and install supporting demo services at nodes referenced by the examples (i.e. Host services) so that the examples are fully functional for tutorial purposes.
- ⇨ Adopt a standard convention to identify properties that cannot be defaulted and must be customized by each user. Use this to identify items such as Host Userid, JES Account, JES output Routing, Departmental High Level Index, etc.
- ⇨ Modify JCL PROCs used by the WDz generated JCL to automatically incorporate the current release of each product as designated by host systems support.
- ⇨ Set host software default parameters to installation standards where possible or modify JCL PROCs to minimize the PARM field overrides required in WDz generated JCL. The JES PARM field is limited to 100 characters and is easily exceeded when non-abbreviated options are concatenated. If necessary, incorporate standard "PROCESS" options at the beginning of each source stream to reduce the override requirements.
- ⇨ Set default template workspace preferences to be appropriate for the target users. The ISPF mode should be defaulted for the LPEX editor for ISPF users.
- ⇨ Insure that Host components have the necessary maintenance to support features used by WDz.
- ⇨ The installation of WDz should include a shortcut to the internal Error Log. The log is referenced on many of the error messages but the existing path to the log from the toolbar is very long.
- ⇨ Do not include unnecessary filters in the Remote Systems Explorer and keep the filter results as short as possible. Large filter results can lead to prolonged expand times when opening or reconnecting a workspace.
- ⇨ Turn off Automatic Rebuild unless working with a small Local project. The background build activity can affect response times and cause periodic lockouts.

## 6.1 Summary of Productivity Gains

This section provides a summary of the productivity gains realized with WDz in comparison to traditional mainframe development tools. Instead of an overall view of the amalgamated times to complete all the necessary components of the required applications, this section breaks down the productivity gains to help provide a glimpse into which areas WDz provides the largest productivity gains.

**Compile, Test, and Debug**

One of the largest productivity gains experienced with the use of WDz was with respect to compiling, testing, and debugging. Even for these simple application components, WDz realized productivity gains of up to 44% compared to traditional IBM mainframe development tools. Branham found that on average, compiling, testing, and debugging within WDz was 25% faster than traditional mainframe development tools, or reversely, traditional mainframe development tools were 34% slower than WDz.

It does need to be re-iterated that Branham developers practiced the development process a couple of times to help eliminate product learning curves. This resulted in a side effect where the code used was logically error free, with the exception of typos when writing code. This virtually eliminated additional productivity gains that would otherwise have been realized during the compile, test, and debug phase. As a result, these productivity gains are actually lower than what developers should expect to see with new coding endeavors.

Table 8 provides a summary of the differences seen with respect to compiling, testing, and debugging between WDz and traditional mainframe development tools.

**Table 8 – Compile, Test and Debug**

| Component | IBM WDz | Productivity Difference | Traditional Tools |
|---|---|---|---|
| 1. Menu Panel | 9 | ← **1.7 X** | 16 |
| 2. Customer Browse | 22 | ← **1.2X** | 26 |
| 8. Request Quotation Panel | 22 | ← **1.3X** | 29 |

**NOTE:** All debugging times recorded in minutes.

Given the limited number of applications created, it is not possible for Branham to state whether this gain in productivity will be linear or exponential when extrapolated to much larger applications. However, utilizing the worse case scenario that this productivity growth only remains linear, developers can still expect to see a significant productivity savings. For example, organizations that specialize in providing mainframe development services can significantly add to their bottom line, potentially taking only three months to develop applications that might have traditionally taken four months.

**Coding**

The productivity gains realized when coding the programs built by Branham provided mixed results. Specifically, for the very simple programs, there was very little difference between WDz and the traditional mainframe development tools. Where the productivity gains started to appear was in the more complex programs. Specifically, features such as content assist provides a pop-up reference for standard COBOL statements and the population of statements with defined variables helps to increase productivity while coding applications. Again, the process of building the applications multiple times subsequently created a significant familiarity with the code diminishing the need for the content assist functionality, which reduced the productivity gains realized in this particular area.

### BMS Maps

Overall, BMS map creation occupies a small percentage of the development time devoted to building applications. Similarly, the need to create BMS maps is decreasing as an increasing number of new development projects are moving toward web-based interfaces instead of traditional interfaces. The creation of BMS maps however, cannot be discounted, as there are still many applications being built that utilize them. WDz provides developers with a drag and drop interface for creating BMS maps that can provide significant productivity savings for newer, less experienced, mainframe developers. In Branham's case, the mainframe developer was seasoned in traditional development tools resulting in very efficient BMS map creation through both development environments. As a result, the timings for building the corresponding maps were similar. For Branham's developer, in some cases the WDz BMS Editor took slightly longer because of a requirement to create filters for multiple map sets. As the number of maps within the set decreased, the time to create filters was reduced. In this case, WDz pulled ahead of the traditional mainframe development tools.

Effectively, Branham's developer was as productive with traditional mainframe development tools as with WDz. While this situation might not necessarily look favorable for WDz, it should be noted that newer developers recently out of school will not have the same proficiency with the traditional mainframe development tools, resulting in a situation where WDz is again more productive. Again, BMS map creation occupies such a small percentage of development time compared to the larger application that as the size of the programs increase, this equality in productivity diminishes in comparison to the other productivity gains.

### Web Enablement

The web enablement of existing applications was also a requirement for this project. For a developer to perform this task through traditional mainframe development tools, a significant amount of hand coding and expertise would be required. As such, WDz provides a significant productivity advantage compared to traditional mainframe development tools in this respect. Table 9 provides the times required for WDz to extend the constructed COBOL/CICS/DB2 application to the web.

**Table 9 – Web Services Enablement**

| Component / Application | IBM WDz | Productivity Difference | Traditional Tools |
|---|---|---|---|
| 1. Request Quotation Web Service | 5 | Unsupported* | N/A |
| 2. CICS Service Flow | 46 | Unsupported* | N/A |
| 3. Request Quotation Web Application | 20 | Unsupported* | N/A |

**NOTE:** All times to develop the necessary applications recorded in minutes.

* While it is possible to do this development by hand, this is not realistic and as such was deemed as an unsupported feature of the traditional IBM mainframe development tools.

## Conclusion:

**The WDz environment is more productive for building real-world mainframe and web-based applications**

## 6.2 Summary of Requirements Met

In addition to the productivity times recorded, it is also important to note whether the basic requirements outlined by On-Demand Insurance Company were met for each of the applications. A detailed description of all application requirements is available in the Appendix (section 8.1 and 8.2). In three of the eleven instances, the traditional mainframe development tools fell short based on the application requirements. This is summarized in Table 10 below:

**Table 10 – Summary of the Requirements that were met**

| Application | WDz | Traditional |
|---|---|---|
| 1. COBOL/CICS/DB2 applications (all 8 parts) | Yes | Yes |
| 2. Web Service Built from Existing Code | Yes | N/A |
| 3. Create a Service Flow | Yes | N/A |
| 4. Building a web page that initiates a web service | Yes | N/A |

### Requirements that the traditional mainframe development tools met:

**1. Create COBOL/CICS/DB2 applications.**

Creation of traditional COBOL/CICS/DB2 applications with BMS panels with ISPF was used as the benchmark for evaluation of the development process. Although BMS panels may be created from MACRO Assembler source, this is rarely done due to the error-prone and tedious process. Screen Definition Facility II (SDF2) was used to represent current practice. Without SDF2 the process would be significantly longer. SDF2 provides a limited character-based graphics interface.

### Requirements that the traditional mainframe development tools did not meet:

**2. Web Service Built from Existing Code**

**Major Concern(s):**

The Web services support in CICS Transaction Server 3.1 enables CICS programs to be Web service providers and requesters. CICS supports a number of specifications including SOAP Version 1.1 and Version 1.2, and Web services distributed transactions (WS-Atomic Transaction). For the creation of web services from existing application code, developers can potentially utilize any of three scenarios including WDz GUI Web assistants, CICS Batch Web Assistants, and CICS Low Level Sockets. CICS supplies utility programs called Web Services Assistants that will take an existing language structure and generate WSDL directly. It will also generate a Web services binding file that contains information about the message, which CICS will use to convert the language structure and SOAP message dynamically. Because these assistants are being merged with WebSphere Developer for System z, the options focused on for the purpose of this study are CICS Low Level Sockets and WDz GUI Web Assistant. To attempt to web service enable CICS applications through low level hand coding however, would be unrealistic for the scope of this project given the length of time it would take. There may be additional tools available to provide some help in this area, but these again are beyond the scope of this guide.

**3. Build a Service Flow**

**Major Concern(s):**

The Service Flow Modeler component of the WebSphere Developer for System z provides the ability to build process flows that string together multiple CICS based applications. This helps easily automate more manual or multiple step processes, allowing these processes to be extended to the web through web

services enablement. The Service Flow Modeler also provides the ability to help generate web services for more complex applications, where the simpler Web Services Wizard cannot generate a web service from existing CICS applications. The traditional mainframe development environment does not provide functionality to simplify this process to the same extent.

**4. Building a web page that initiates a web service**

**Major Concern(s):**

While this particular initiative can realistically be coded by hand, with the number of tools available on the market, doing so would be a significant waste of time and resources. Because the focus of this guide is between WDz and traditional IBM mainframe development environments, the introduction of third party tools is not permitted. By no means does Branham mean to indicate that WDz is more (or less) productive than any other third party tool available for this particular development effort. Instead, Branham simply states that WDz provides developers with the necessary tools through a single integrated development environment without the requirement to introduce additional third party tools. The same thing cannot be said with respect to the traditional mainframe development environment.

**Note on Construction of CICS Web Interfaces (Legacy Web Interfaces)**

Web interfaces can be constructed as native CICS applications using the CICS Sockets interface. In fact, prior to the availability of CICS Web Services, this was common practice. The process required the development of sockets communication code for either a Listener in Server Mode or an outbound connection for Client Mode. The common code for sockets can be built as a reusable COPY book which would normally also include code page translation to and from EBCDIC. To complete the process, the developer would also have to create support for the HTTP protocol, which would be implemented as another reusable COPY book set. XML/SOAP support would require another major effort. The Sockets and HTTP protocol support COPY books require extensive error condition detection and recovery. Typically, a custom Listener would be constructed for each application that would interface to one or more CICS applications via the standard COMMAREA interface. The entire Web Support process would take about 6 months or more of development effort. In a way, these could be called "Legacy Web Interfaces".

The incentive for embarking on this type of development was usually the issue of early deployment of Web Services in a competitive environment. These interfaces did not require Unix System Services. Other factors that may mandate this approach:

- ⇨ The requirement to multi-thread transactions over one socket connection to interface to a specialized remote system (This feature is not available in any standard support packages or in MQ Series).
- ⇨ The requirement for the application to be sensitive and aware of real-time network conditions for transaction integrity, application controlled dynamic re-routing or automated recovery for high-reliability critical applications.

WDz may be of use for modernization of these early web interfaces especially where the interface is reduced to a simple COMMAREA at some point in the existing process and where there are no restrictive requirements as listed above.

## Conclusion:

**IBM WDz tools are more productive for fulfilling real world on demand application requirements with a minimum amount of developer tools**

## 6.3 Productivity Detail for each Application

This section takes a detailed look at each of the eleven applications. The following is covered for each application:

a. Summary of productivity and requirements
b. How it was built with traditional IBM mainframe development tools (where applicable)
c. How it was built with WebSphere Developer for System z
d. Lessons learned and issues encountered
e. Bottom line points

### 1. COBOL/CICS/DB2 Application

**Brief Description:**

Internal ODI staff use this application. It provides the ability to enter and browse customer data, view and add dwelling and material types, and more. Consisting of a total of eight programs, the main component of this set provides the ability to calculate quotes for customers and partner brokers based on specific criteria. For each type of 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 application user.

The layout of the pages and the detailed requirements, including the look and feel prototypes, are available in section 8.2 of the Appendix.

Table 11 summarizes the average results.

**Table 11 - Summary of Productivity and Requirements**

| Description | WDz | Difference | Traditional |
|---|---|---|---|
| Average time in minutes to build working quote application (creation of BMS maps) | **39** | ← **1.4x** | **56** |
| Average time in minutes to build working quote application (COBOL coding, compiling, testing, and debugging) | **150** | ← **1.1x** | **172** |
| Did it meet the application requirements | **yes** | **N/A** | **yes** |

**How it was built with traditional mainframe development tools**

(see Appendix 8.3 for the detailed steps)

Tools used included:

- ISPF
- SDF II
- COBOL Compiler
- SQL Preprocessor

The quote review panel, which allows selection from a list, which goes to a detail panel, is one of the more complex scenarios in CICS. It requires selection of rows for one specific customer, followed by matching up the cursor location to the row. The detail record is then retrieved by a stored key for each row that was previously displayed.

Panel switching in CICS is non-trivial for a PSEUDO-CONVERSATIONAL program. In order to keep the dialog flowing, it is necessary to run part of program B at the end of program A when switching from panel

A to panel B. This ensures that the wait-for-input is done at the correct time. The previous panel state has to be stacked and restored when returning to the main panel.

In order to save time without trivializing the example, Branham decided to adapt some canned code that was previously used for BMS mapping and PSEUDO CONVERSATION management. The reused code made it easier to add selection menus and row browsing, and helped handle all of the conditions returned by each EXEC CICS command. Consisting of a few COPY BOOKS, this made the whole process faster and more realistic since common code is often used for these functions. The same COPY BOOKS were used for WDz development.

### How it was built with WDz

(see Appendix 8.3 for the detailed steps)

Tools used included:

- WebSphere Developer for System z including
- BMS Editor component
- LPEX Editor component

As noted above, pre-existing COPY BOOKS where used to provide a more realistic example. These COPY BOOKS were used in both traditional mainframe and WDz development environments. Additionally, some feature functionality available through WDz was capitalized on to help with development efforts. Specifically, some modifications to the IBM supplied MACROS were created so that SDF II compatible code could be generated directly by WDz. This was simply to eliminate the requirement to make additional hand coding modifications. If used effectively, this ability helps provide additional productivity in common development scenarios.

BMS Maps were created using the BMS Editor within WDz. This editor provides a drag and drop interface for field placement and field property modification.

Once the BMS MAPS were created, the LPEX editor was used to write the necessary business logic. This editor was placed into ISPF mode to help ease the transition from the traditional ISPF environment to the new editor within WDz.

For this project the DFHMDF MACRO was cloned and modified to generate nesting levels starting at 05 so that the map could be imbedded in a larger data item. The DFHMDF MACRO was cloned again as ODIOCC to provide another MACRO that would insert OCCURS levels to bracket the series of elements of an array. The ODIOCC MACROs were inserted in the BMS source after creating a Map with the BMS Editor.

### Lessons learned and Issues Encountered

ISPF has been the traditional mainframe development environment. Despite the claims that the mainframe would be phased out, the mainframe is not going away. In this respect, vendors are providing tools intended to help ease development on the mainframe. WDz is such a tool. The ability to drag and drop fields for BMS map creation and local syntax checking can not only help increase productivity of seasoned mainframe developers, but helps get newer, less experienced mainframe developers up to speed faster, helping increase productivity in these individuals faster than would be required to gain experience and familiarity with traditional mainframe development tools.

#### WDz

1. WDz eases BMS map creation through a drag and drop interface.
2. WDz speeds syntax checking providing the quick location of errors for corrective editing.

   **Issue 1: Disappearing MVS projects.**

   In a few cases, developers found that after closing and restarting WDz with the same workspace, that ALL of the MVS projects would disappear. Project views are a logical grouping of the files used for a specific project and it is these groupings that disappear, not the actual files themselves. The issue revolves around a bug in the metadata definition files which results in the inability to properly parse

them. Developers made regular backups so when this happened the files were easily replaced. IBM is aware of the issue and has stated that it will be fixed in the next release. While it is not a major issue, the MVS projects disappeared more than a few times, which could be considered a nuisance for customers.

**Issue 2: BMS Editor.**

- ⇨ *Stopper Fields* - The version of the BMS editor used for this study inserted "stopper" fields (3270 attribute with no data) at the end of Output-only fields selected from the pallet. These fields had to be deleted each time a new field was inserted because it made the position occupied by the "stopper" attribute unusable. When the "stopper" was not deleted, the length of the output field could not be set to any value beyond the initial default. The "stopper" field did not move if the output field was moved and it became an orphan. Other new fields could not be inserted if the new field spanned the invalid "stopper" position.
- ⇨ *BMS Pallet Properties* - The BMS pallet does not have an element for the Modified Data Tag (MDT) flag. When required, this property must be set each time a field is inserted.
- ⇨ *BMS Pallet missing Preferences* - The workspace preferences do not include all of the Pallet Icons. Defaults cannot be set for these Icons.
- ⇨ *BMS MACROs cannot generate OCCURRS for arrays (SDF2*)* - The BMS editor of WDz relies on the standard BMS MACROs supplied with CICS. These MACROs are not capable of generating OCCURS declarations for indexing arrays of panel fields.
- ⇨ *BMS MACROs fixed COBOL data levels (SDF2*)* - The standard BMS MACROs have fixed constants for the COBOL data nesting levels.

Correction of the BMS Editor problems and improvement of the BMS MACROs can significantly improve development productivity.

**Traditional mainframe development tools**

1. Overall, these were straightforward applications to build within a traditional mainframe development environment.
2. Syntax checking (compile/edit/debug) required the familiar cycle of switching between screens to hunt down, isolate, and correct errors.

**Bottom Line Points**

1. Even for simple applications, WDz has the potential to help seasoned mainframe developers be as productive through the new environment. Simple applications built using WDz sometimes took longer because of Project setup. For example, BMS project setup requires Filter setup for multiple maps in one mapset. Once the learning curve is dealt with and familiarity with the product increases so may the productivity.
2. For more complex applications, WDz helps increase productivity, particularly in the realm of compile, edit, and debug. ISPF syntax checking and error correction is definitely longer due to manual search for source segment related to error (i.e. which COPY book)? WDz has an advantage with multiple source files open for cross-reference. Syntax checking is much more efficient than ISPF mode of compile and visual scan. Error message to source file mapping saves time.
3. WDz would probably be even more effective for rework of existing code, change requests, etc. In theses cases, overhead for Project setup would be factored out and BMS maps will already exist. The syntax checking and debugging tools, which provide the largest productivity gain, would be the focus in these situations.
4. The BMS Editor and the LPEX editor are young compared to the other components of the WDz tool. As such, there are still a few small issues that are being ironed out.

## 2. Build Web Service from Existing CICS Code

**Brief Description:**

The web service generated for this phase used the existing quote application created in Phase 1. The quote application utilized a COMMAREA to provide a best case scenario for converting and exposing existing CICS applications as web services. Realistically, this could not be done proficiently through traditional mainframe development tools, and as a result, the focus is on the WDz development environment.

Table 12 summarizes the results of the web services generation.

**Table 12 - Summary of Productivity and Requirements**

| Description | WDz | Difference | Traditional |
|---|---|---|---|
| Average Time in minutes to build entire working application | **5** | **N/A** | **N/A** |
| Did it meet the application requirements | **yes** | **N/A** | **No** |

**How it was built with WDz tools**

(see Appendix 8.3 for the detailed steps)

Tools used included:

- IBM WebSphere Developer for System z

Developers created a new local project within WDz and copied the COBOL source (and any required COPY BOOKS) from the host to the local project. By right clicking on the local COBOL file, developers are able to select “Enable Enterprise Web Service” which initiates the web services generation wizard.

In most cases, the default options within the wizard were acceptable. Developers were only required to select the input and output parameters and specify the end point for the WSDL. The wizard then proceeds in generating the necessary driver, WSDL, XSD, and WSBIND files.

The driver file was then moved to the z/OS project and compiled. The load module was then copied to the load library and installed into CICS using CEDA. Finally, the WSBIND file was copied to USS on the host and CEMT PERFORM PIPELINE(testpipe) SCAN completes the implementation of the web service on the host.

For testing, the Web Services Explorer within WDz reads the WSDL and automatically generates the necessary structure for testing the web services to make sure it is working appropriately.

**Lessons learned and Issues Encountered**

**WDz**

1. WDz can dramatically help increase productivity for building and testing web services from existing code. WDz can also help build web services from scratch utilizing wsdl, dramatically decreasing setup time. Developers are left only to implement the necessary business logic.
2. WDz provides the necessary tools for testing web services, all dynamically generated based on the wsdl without the requirement to create an application calling the web service.
3. EXEC CICS and EXEC SQL do not have to be expanded for the Enable Web Service wizard to work.

   **Issue 1: Some COPYBOOK caveats**

   The Web Services wizards will generate the adapter programs necessary to run the application on the vast majority of user copybooks. There are however some limitations. For example, the Local COBOL

CICS option does not translate CICS DFHVALUE constants in COPY books, unlike the host COBOL 3.3. IBM has not had a request to work with CICS translator macro fed copybooks - which both Branham and IBM found interesting.

Some of suggested work around options included: produced post processed source on the mainframe, then import that; or comment out offending 88's. In this case the Branham developer ran the CICS preprocessor on the host for ODBMSUCM.cpy, ODBMSUWS.cpy, and ODCICSWS.cpy.

**Issue 2: CICS translation inconsistencies**

There are inconsistencies between CICS V3 translation and the CICS for Windows supplied translator. These are known, and is one of the reasons IBM supplies both local and remote syntax check capabilities in WDz. With the release of WDz version 7.0 due out next year, IBM will be supporting the translator for CICS V3, directly invokable on the workstation. This will address syntax check inconsistencies.

**Issue 3: Imbedded binary multipunch**

The Branham developer had to deal with correcting some IBM supplied CICS host code, which had imbedded binary multipunch (originally keypunch) codes. This is an ongoing problem for all CICS functions in WDz. In this case, DFHAID.cpy and DFHBMSCA.cpy were replaced by ODIAID.cpy and ODIBMSCA.cpy.

**Issue 4: CICS translation in COBOL V3.3**

The COBOL V3.3 integrated CICS translator could not translate the EXEC CICS INQUIRE PROGRAM command of the CICS Systems Programming subset. It always interpreted the "COPY" operand of this command as a "COPY" compiler directive, causing a syntax error.

**Issue 5: SYSLIB for Wizard**

The Enable Web Service Wizard did not honour the SYSLIB property for the COBOL program. All related COPY books (including IBM COBOL Library members) used by the program had to be copied into the same folder as the COBOL main program.

**Bottom Line Points**

1. In a best case scenario, existing CICS applications can be web services enabled in as little as five minutes. In some cases, slight code modification may be necessary.
2. One recommendation is that the Enable Web Service wizard should be upgraded to handle DFHVALUE constants.

## 3. Create a Service Flow

**Brief Description:**

It is not always the case that web services can easily be generated from existing CICS applications using the “Enable Enterprise Web Services” wizard, or that all CICS applications utilize a COMMAREA. In these cases the Service Flow Modeler can be used to web enable these applications. ODI would like to web services enable additional CICS applications such as its Customer Inquiry to view the details that ODI has on the partner brokers customers.

It should be noted that the Service Flow Modeler performs much more than simple web enablement existing CICS applications. This tool is capable of stringing together multiple CICS applications into a single process flow and web enabling the entire process. This allows applications to consume the services through a single step while all the incorporated sub steps are transparent to the calling application.

Table 13 summarizes the results of multiple timings by different developers.

| Table 13 - Summary of Productivity and Requirements | | | |
|---|---|---|---|
| **Description** | **WDz** | **Difference** | **Traditional** |
| Average time in minutes to build entire working application | **46** | **N/A** | **N/A** |
| Did it meet the application requirements | **Yes** | **N/A** | **No** |

**How it was built with WDz**

(see Appendix 8.3 for the detailed steps)

Tools used included:

- IBM WebSphere Developer for System z
- Service Flow Modeler

The Enable Enterprise Web Service wizard allows the easy web services enablement of existing CICS applications that utilize the COMMAREA. The Service Flow Modeler additionally provides support for web services enablement of existing CICS applications utilizing a terminal generated flow.

While the Service Flow Modeler is capable of generating more complex business workflows incorporating a multitude of applications, in this case Branham built a Customer Inquiry, which involves the customer browse screen followed by the customer details information panel.

Since we had access to the actual BMS files, once a new SFM project was created, developers simply added the host definition, imported the necessary BMS, and recorded the flow. Once the flow was built and recorded, the BMS fields were mapped to variables for the process flow. While our example was relatively easy in comparison to what is possible within the Service Flow Modeler, all operations are performed through a graphical user interface with drag and drop capabilities.

After all the necessary properties are set and the variables defined, the web services files were generated, deployed, and tested. Again, in the case where a web service is used as the end point, the Web Services Explorer can be used for testing purposes.

**Lessons Learned and Issues Encountered**

The Service Flow Modeler (SFM) can be used to web-enable more complex applications. The SFM can generate interfaces to CICS applications either through the COMMAREA or through the Terminal Interface (BMS). The main feature of the SFM is to generate multiple message interactions, which may require a sequence of CICS applications with the potential for looping and conditional application selection. The Terminal Interface is useful for CICS applications, which have a COMMAREA that is too complex for practical conversion or where modification of the CICS application is not permissible. In this case, the 3270 Bridge is used to interface to the application through BMS. The SFM imports the BMS definitions and generates interfaces to the application. It then records interactions with the BMS panels to generate interface messages and protocols. Since the new web interface is invoked under a different transaction code than the original application, there is the ability to separate web and non-web workloads for management and security purposes. SFM provides graphical configuration of the application units and selection of data elements for interface between the selected applications. SFM also reduces the level of BMS knowledge required by using a recording function to capture and interpret control and input/output sequences between the 3270 interface and BMS. This information is automatically inserted in the messaging between applications.

The SFM approach to combining applications creates a communication workload because several network interactions are required to complete a transaction. It may be advisable to implement the SFM generated functions at an intermediate node that is co-located with the host so that remote network traffic is simplified.

Depending on the implementation requirements and an organization's SOA strategy, WDz could effectively eliminate the requirement to purchase SOA and web enablement products from third parties such as Seagull Software's Transidiom, AttachmateWRQ's Verastream, or Open Connect's xmlConnect. This would provide cost savings with respect to additional software costs, educational requirements to learn additional tools, and the need for additional support services.

1. SFM development using the Terminal Interface does not require analysis or modification of the program COMMAREA. The COMMAREA may be complex and may contain pseudo-conversation context information in addition to business data.
2. Importing of BMS maps or COBOL files makes use of the original field names when generating interface/intermediate messages. The fields are easily identified for message sequencing and value tests.
3. The 3270 control sequences are interpreted and inserted into the message while recording BMS interactions relieving the developer of intensive knowledge of BMS.
4. SFM has the flexibility of looping or conditional selection of applications, allowing the composite functionality to increase as required while demanding minimal knowledge of the host components.

**Bottom Line Points**

1. WDz can web services enable existing applications that are too complex for the simple Enable Enterprise Web Services wizard.
2. The Service Flow Modeler eliminates the requirement for extensive hand coding when stringing multiple applications together in a complex business process.
3. WDz's service flow capability could quite possible eliminate the need to purchase additional third party products.

## 4. Web Application for the homeowner policy quote

**Brief Description:**

Associate brokers and preferred customers have access to the ODI portal for resources such as form downloads, etc. ODI wants to help increase its response speed with respect to quote estimates without the need to download and mail in a form. Those with access to the portal will now have the ability to use a web based application form to submit and receive a new quote estimate instantly rather than the significantly longer paper based process. For each type of 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 user. This web-based form utilizes the web service built in Phase 2 to generate a quote based on the provided information.

The layout of the pages and the detailed requirements, including the look and feel prototypes, are available in section 8.2 of the Appendix.

Table 14 summarizes the results of multiple timings by different developers.

**Table 14 - Summary of Productivity and Requirements**

| Description | WDz | Difference | Traditional |
|---|---|---|---|
| Average time in minutes to build entire working application | **20*** | **N/A** | **N/A** |
| Did it meet the application requirements | **Yes** | **N/A** | **No** |

***** The bulk of the 20 minutes spent building the application was attributed to the building of the template (almost 14 minutes). For the actual inclusion of the web service and the page components was a brief three minutes (1 minute and 2 minutes respectively).

### How it was built with WDz

(see Appendix 8.3 for the detailed steps)

Tools used included:

- IBM WebSphere Developer for System z
- Page Designer and html
- JSF components
- Templates

**Note:** This application can also be built using the IBM Rational Web Developer with the same results. The IBM Rational Web Developer costs less than the IBM Rational Application Developer.

Developers accomplished this task by using the IBM WebSphere Developer for System z product. For this particular scenario, WDz utilizes the full web development capabilities of Rational Application Developer. Starting with the creation of a dynamic Web Project, the layout of the two pages (Quote Request and Quote Response) was completed with the Web Diagram Editor.

A page template file was created such that it could be re-used for multiple pages. This particular procedure took the bulk of the time for this entire scenario. WDz users also have the option of importing an existing web site, importing the necessary images, stylesheets, etc., which would have significantly reduced the development time. For illustration purposes, Branham built the template from scratch.

The two Faces JSP (Java Server Pages) pages were created, the web service was easily located and added to the project, and the necessary components where drag and dropped into the appropriate pages. When placing the web services components, developers are given the option to provide more appropriate names for the field labels. Only slight adjustment of the page was necessary afterwards given the extensive time taken to build the original template.

### Lessons Learned and Issues Encountered

WDz provides the ability to quickly incorporate web services into applications; web based or thicker java client applications. While there are numerous tools on the market for developing web applications, WDz includes all the necessary components without the need to purchase and utilize additional third party tools. Like many tools available, WDz eases creation through drag and drop WYSIWYG interfaces or allows traditional development through source code views.

1. WDz makes it very easy to locate and incorporate web services into Java Server Face pages.
2. Re-usable templates provide for a nice productivity feature. While Branham only created two pages, in much larger development projects, this would offer a significant time savings.
3. Web Services Explorer allows for testing located web services that are being incorporated into applications. This allows developer to assure that the web service selected performs as expected, reducing potential troubleshooting when trying to isolate the source of any issues.

### Bottom Line Points

1. The IBM Rational tools set has made major improvements in building simple web applications.
2. The incorporation of web services into a web-based application is extremely simplified.
3. Organizations that have web services enabled their existing applications can easily extend them to customers increasing self-service capabilities, potentially reducing customer service costs.

# 7.0 Conclusions

## 7.1 Productivity

The developer used for this productivity study was a seasoned mainframe developer with extensive experience using a traditional IBM mainframe development environment. While there was an initial learning curve related to the WDz development environment, it was minimal and quickly helped make the same developer as productive as using the traditional development environment. This of course is taking into consideration that the code was solid with no errors. In the event of an error, the WDz environment helped increase productivity when isolating and correcting syntactical errors.

This study was started with four goals in mind.

1. Define an objective methodology that would allow as close to an “apples to apples” comparison as possible between different mainframe development environments; traditional and WebSphere Developer for System z.
2. Build all the application components necessary to create a working application.
3. Provide an objective measurement methodology to measure the productivity differences between the traditional IBM and WDz tooling when building the application components.
4. Document the productivity measurement differences as well as any issues that were observed between the traditional IBM and WDz environments in the course of the study.

After 4 months of extensive work, the results of the study strongly indicate that WDz delivers a development environment that can enable developers to be more productive in building mainframe applications and extending them to the web. Out of the eleven components/applicaitons investigated, WDz was clearly more productive in more complex implementations. In cases where COBOL/CICS/DB2 applications where simple in nature, WDz made traditional mainframe developers similarly productive. Only in very simple applications were traditional mainframe developments tools able to take a slight lead.

Most of the productivity advantages seen within the WDz development environment can be summed up as follows:

1. Faster syntax checking through local preprocessor.
2. Visual interfaces that provided instant progress and feedback to the developer.
3. Test harnesses, forms and environments built directly into the development tools.
4. Debugging tools integrated into all of the IBM development tools.

In short, WDz demonstrated the ability to build and modernize mainframe applications, while also providing the facilities to create robust server side applications in an ever changing on demand environment.

## 7.2 Quality

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

1. The WDz development environment had the debugging and test harnesses embedded into the development tools. Even the application of constructing a web service provided a built in SOAP test environment. This was not the case for the traditional mainframe development environment.
2. In general, as the application requirements became more complex and mimicked more “real world”, the WDz development environment continued to surpass the traditional mainframe development environment. While users could purchase additional third party tools to meet their additional development needs, 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.

## 7.3 Requirements

The requirements for this application were developed before it was 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 8.1. These requirements outline a “realistic” insurance application as defined by the Insurance consultants and the researched on-line Insurance sites.

The WDz development environment was able to meet all of the realistic application requirements while the traditional green screen development environment fell very short once moving beyond more traditional COBOL/CICS/DB2 applications (9, 10, and 11).

This tends to highlight the fact that the traditional IBM mainframe development environment is very easy to use to build basic applications (application #1 through #7), but as the application demands require higher level functionality and the requirements become more demanding, the traditional mainframe development environment reaches its limit and requires the introduction of additional tools. Of course, any requirement can eventually be manually coded given an unlimited amount of time and assuming high-level skills are available.

This study indicates that the IBM WDz development environment has the ability to build demanding and robust mainframe applications – especially web based server side applications - while reducing the amount of manual coding required, ultimately reducing the requirement to engage highly experienced developers to perform all the tasks.

# 8.0 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:

## 8.1 Application Requirements

This section contains the detailed application requirements in order of the three phases.

## 8.2 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.

## 8.3 Detailed steps to build the required Applications

This section contains the step-by-step procedures that were timed for the development of the COBOL/CICS/DB2, Web Service, and web-based applications.

## 8.4 Glossary

## 8.1 Application Requirements

This section contains all the requirements information that was used to build the entire Insurance application. A fictitious company, On Demand Insurance (ODI), provides customers with homeowners insurance. The core applications that run ODI's business will be created and then extended to support their business growth. This will be completed over three different phases:

The requirements for the applications are divided up into three phases as shown in the table below:

- Phase 1: Build traditional COBOL/CICS/DB2 application with BMS screens
- Phase 2: Web Service enable core application, test and invoke web service. Create a reusable business process that consists of multiple CICS program interactions and web service calls. Test.
- Phase 3: Build a web application to access core CICS application using web service calls

| Phase | Name of Application |
|---|---|
| **1** | 1. CICS application for the homeowner policy quote |
| **2** | 2. Web Service built from existing Code |
| **2** | 3. Create a Service Flow using the Service Flow Modeler. |
| **3** | 4. Create a simple web application using JSF to call the web service built in 2 |

The requirements for all the applications are organized as follows:

**Phase 1, 2 and 3**

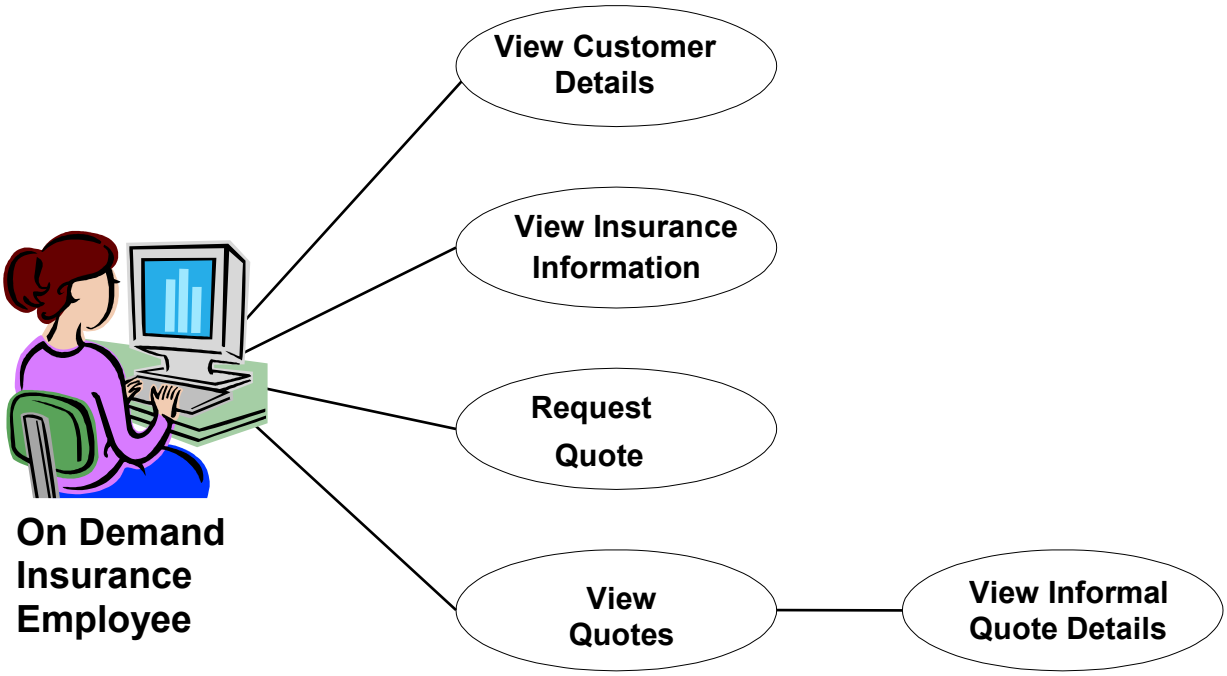
- Vision Statement
- Use Case Model
- Architecture Diagram
- Database Schema
- Documented Features

## Phase 1

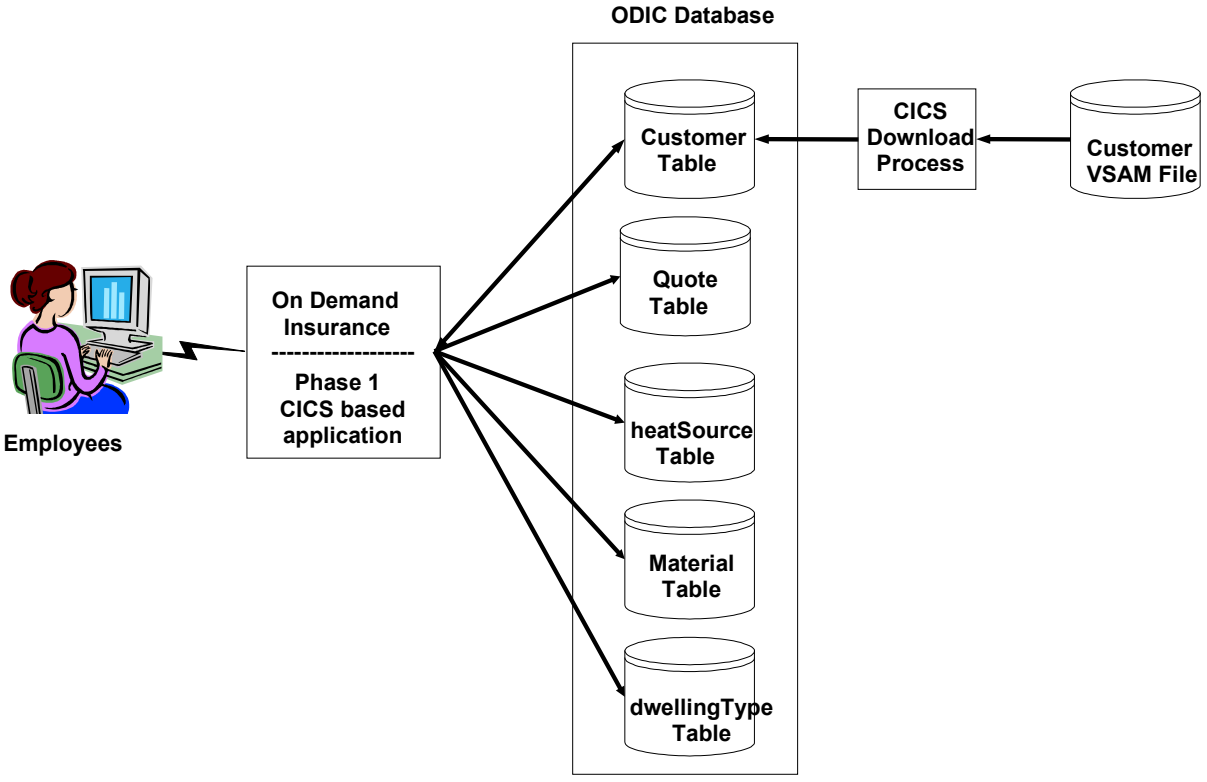
**Vision Statement**

Phase 1 of the ODIC application provides customer service representatives 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 have been requested previously as well as details about those previous quotes.

### Phase 1 - Use Case Model

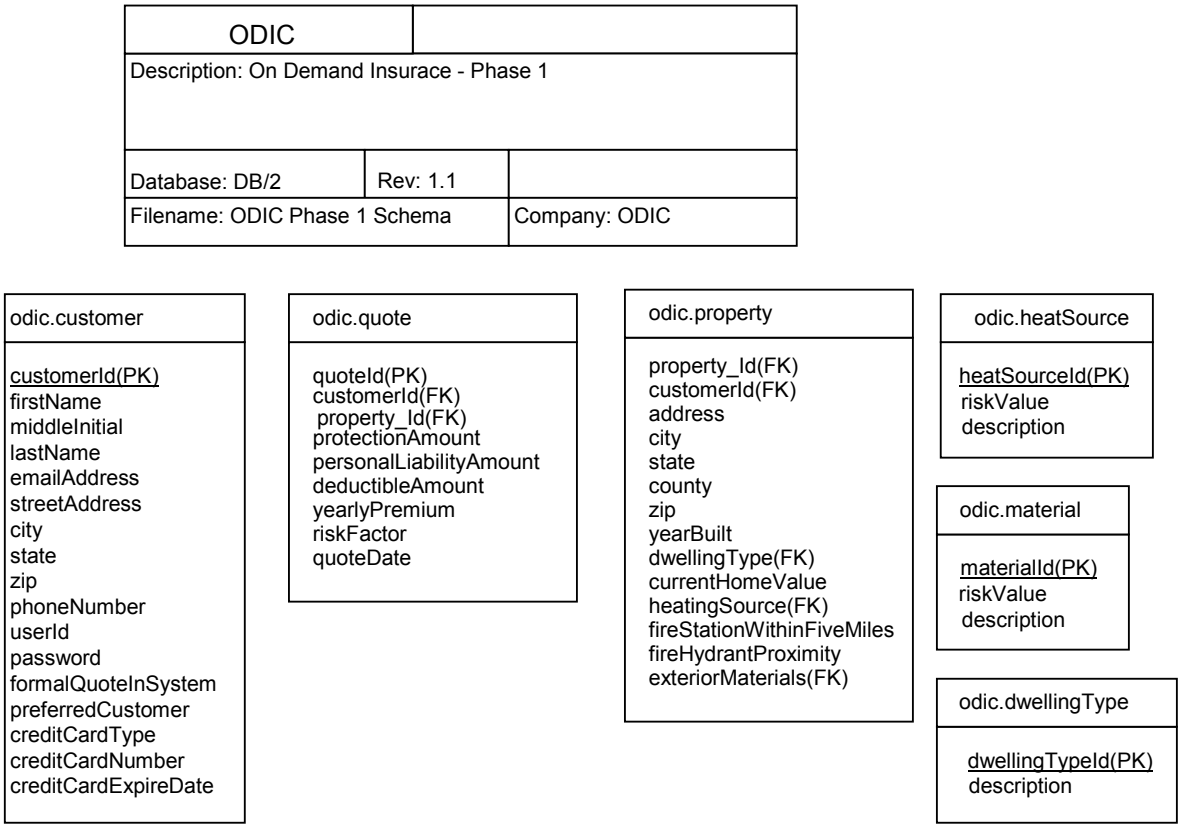



### Phase 1 - Architecture Diagram




**Note:** All the customer input is stored in the "Quote Table". ODIC DB is DB2 UDB for z/OS V8.1

**Phase 1 - Database Schema**




**Phase 1 - Features and Use Cases**

In Phase 1, the core insurance application for ODI will be built. This application is used by ODI's internal employees – their underwriters and customer service representatives – to provide customers with the following features: 1. Provide a homeowner insurance policy quote; and 2. view all of the quotes that a customer has requested previously as well as details about those previous quotes. The application is to be developed as a traditional COBOL/CICS/DB2 application that uses BMS screens for user interaction.

**Feature 1: Application startup**

Customer Service Representatives - henceforth referred to as CSR(s) - can access the application through their 3270 emulator.

Use Case 1.0 Basic Flow:

ODI employees start up the application by typing in the Transaction ID as the CICS terminal screen. Transaction Code PMNU provides the initial ODI Policy System Selection Menu (CICS and non-DB2). From this menu, various selections provide access to additional applications. The CSR may enter the system directly at any panel by entering the Transaction ID for that panel. If the panel was entered from the menu, F3 will return to the menu. If the panel was entered directly, F3 will exit from the application.

**Feature 2: Help Panels**

CSR(s) - can access Help for any panel.

Use Case 2.0 Basic Flow:

ODI employees can toggle to the Help panels at any time by using F1. F1 again or F3 will return to the application. Any partially entered input is preserved and restored while using the Help panels. While in Help, several Help panels can be accessed if available. F8 advances to the next Help panel and F7 returns to the previous Help panel.

**Feature 3: Application termination**

Users can exit the system.

Use Case 3.0 Basic Flow:

Users type CESF LOGOFF to exit the system.

**Feature 4: View Customer information**

Allow the CSR to view or update the Customer list and detailed customer information.

Use Case 4.0 Basic Flow:

From the main menu, users enter the transaction code PCUS to access the OSI Policy System Customer Browse application (using CICS and DB2). This provides a listing of all available customers. From here, users can access more detailed customer information. The CSR can browse through the Customer table and view an abbreviated one line summary of each Customer record. The CSR can then select the Customer record for detailed display or updating. The CSR can also create a new Customer record or delete an obsolete Customer record.

**Feature 5: View Dwelling Type information**

Allow the CSR to view or update the Dwelling Type list and detailed customer information.

Use Case 5.0 Basic Flow:

From the main menu, users enter the transaction code PDWL to access the OSI Policy System Dwelling Type Browse application (using CICS and DB2). The CSR can browse through the Dwelling Type table and view an abbreviated one line summary of each Dwelling Type record. The CSR can then select the Dwelling Type record for detailed display or updating. The CSR can also create a new Dwelling Type record or delete an obsolete Dwelling Type record.

**Feature 6: View Heat Source information**

Allow the CSR to view or update the Heat Source list and detailed customer information.

Use Case 6.0 Basic Flow:

From the main menu, users enter the transaction code PHSR to access the OSI Policy System Heat Source Browse application (using CICS and DB2). The CSR can browse through the Heat Source table and view an abbreviated one line summary of each Heat Source record. The CSR can then select the Heat Source record for detailed display or updating. The CSR can also create a new Heat Source record or delete an obsolete Heat Source record.

**Feature 7: View Materials information**

Allow the CSR to view or update the Materials list and detailed customer information.

Use Case 7.0 Basic Flow:

From the main menu, users enter the transaction code PMAT to access the OSI Policy System Materials Browse application (using CICS and DB2). The CSR can browse through the Materials table and view an abbreviated one line summary of each Materials record. The CSR can then select the Materials record for detailed display or updating. The CSR can also create a new Materials record or delete an obsolete Materials record.

**Feature 8: View Properties information**

Allow the CSR to view or update the Properties list and detailed customer information.

Use Case 8.0 Basic Flow:

From the main menu, users enter the transaction code PPRP to access the OSI Policy System Properties Browse application (using CICS and DB2). The CSR can browse through the Properties table and view an abbreviated one line summary of each Properties record. The CSR can then select the Properties record for detailed display or updating. The CSR can also create a new Properties record or delete an obsolete Properties record.

**Feature 9: View Quotation information**

Allow the CSR to view or update the Quotation list and detailed customer information.

Use Case 9.0 Basic Flow:

From the main menu, users enter the transaction code PQOT to access the OSI Policy System Quotation Browse application (using CICS and DB2). The CSR can browse through the Quotation table and view an abbreviated one line summary of each Quotation record. The CSR can then select the Quotation record for detailed display or updating. The CSR can also create a new Quotation record or delete an obsolete Quotation record.

**Feature 10: New Quotation**

An ODI CSR can provide a customer with an informal homeowner insurance policy quote. This feature will provide an immediate response to the customer based on information provided by the customer, insurance data, and a risk factor calculation.

Use Case 10.0 Basic Flow:

From the main menu screen, an ODI CSR selects the PNEW transaction to start a new policy. The customer then provides information regarding the home and insurance coverage needed. If the appropriate options are not known, users can browse the available options, selecting the appropriate one. Once complete, the CSR user can request a quote. After each step of the process the F4 key (CONTINUE) is used to continue to the next step.

Step 1 - Identify Customer

- F4 takes the CSR to the Customer Browse panel.
- The CSR may select an existing Customer or create a new Customer record.
- Once the Customer record has been selected or created F3 is used to return to the New Quotation panel.

Step 2 - Select Dwelling Type

- F4 takes the CSR to the Dwelling Type Browse panel.
- The CSR may select an existing Dwelling Type or create a new Dwelling Type record.
- Once the Dwelling Type record has been selected or created F3 is used to return to the New Quotation panel.

Step 3 - Select Construction Materials

- F4 takes the CSR to the Materials Browse panel.
- The CSR may select an existing Materials or create a new Materials record.
- Once the Materials record has been selected or created F3 is used to return to the New Quotation panel.

Step 4 - Select Heat Source

- F4 takes the CSR to the Heat Source Browse panel.
- The CSR may select an existing Heat Source or create a new Heat Source record.
- Once the Heat Source record has been selected or created F3 is used to return to the New Quotation panel.

Step 5 - Identify Property

- F4 takes the CSR to the Property Browse panel.
- The CSR may select an existing Property or create a new Property record.
- If a new Property record is created, the previously selected Dwelling Type, Construction Materials and Heat Source are automatically inserted.
- Once the Property record has been selected or created F3 is used to return to the New Quotation panel.

Step 6 - The CSR inputs the following information provided by the customer:

- 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)
- Fire station within five miles (yes, no)
- Fire hydrant proximity (In feet)

Use Case 10.0.1 Request Informal Quote:

Immediately following Use Case 10.0, the system calculates a risk factor (see algorithm below).

Use Case 10.0.2 Risk Factor

The system evaluates the risk factor calculated in Use Case 4.0.1. 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.
- 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 CSR may use F6 to store all of the data gathered and calculated as part of this quote in the quote table. The system provides the option for the CSR to request another quote.

**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.

| 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 5: View quote information**

An ODI CSR can view all of the quotes provided to a particular customer in a line item display and also view further details about each of the quotes.

Use Case 5.0 Basic Flow:

The CSR selects the View Quotes option and provides the Customer ID. The system displays a list of all of the quotes that the customer has previously requested. The CSR then selects one of the line items to see more details about that particular quote. The system displays a new screen with the rest of that quote's details. From this details screen, the CSR can navigate back to the Quotes list. The system re-displays the screen with the quote line items on them. This cycle can repeat any number of times.

Use Case 5.1 Alternate Flow:

If the Customer ID is not available, the CSR can do a search for the customer. Once the listing for that customer information is located, the CSR can directly access the list of quotes associated with that particular customer. The CSR then selects one of the line items to see more details about that particular quote. The system displays a new screen with the rest of that quote's details. From this details screen, the CSR can navigate back to the Quotes list. The system re-displays the screen with the quote line items on them. This cycle can repeat any number of times.

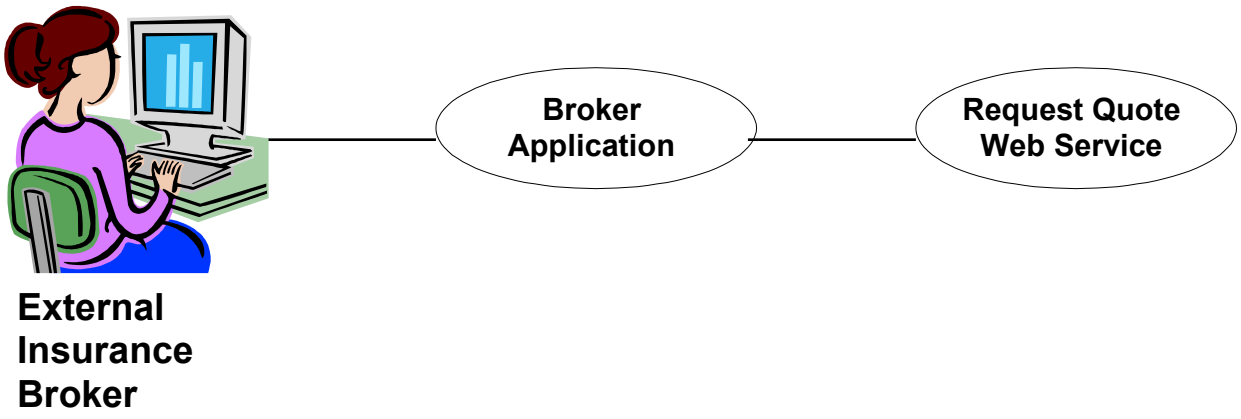
## Phase 2

ODI wants to increase revenue by bringing in more customers through external insurance brokers. In this phase ODI wants to take portions of their core CICS application and make it available to external brokers. ODI does this by enabling their existing CICS programs as web services.

**Vision Statement**

Phase 2 of the ODIC application introduces a new level of customer, introduced through external insurance brokers. External brokers will have the same functionality that ODI employees had in Phase 1. External brokers will be able to initiate a quote process, having the same business logic as the internal employee quotes, in that a risk factor will be calculated. However, the quote process of phase 2 is distinguished by the fact that it is not initiated directly through a BMS interface, but rather through web services. It is assumed that the External Broker will use a portlet that uses the ODI web service as the backend.

### Phase 2 - Use Case Model




### Phase 2 - Database Schema

Phase 2 database is the same as Phase 1. Please refer to the database diagram from Phase 1.

### Phase 2 - Features and Use Cases

External Insurance Brokers, herein referred to as Brokers, will have applications that are used to connect to the ODI web service for generating new quotes.

**Feature 1: Application startup**

Brokers invoke the quote web service through SOAP, similar to any standard web service.

Use Case 1.0 Basic Flow:

Following the basic service description (WSDL), the Brokers system invokes the ODIC web service, passing the necessary information with respect to dwelling type, proximity to a fire hydrant, etc., to generate a new quote. In return, the ODI web service will return the details of the quote to the Broker applications.

**Feature 2: Application end**

Broker application exits the system.

Use Case 2.0 Basic Flow:

After the ODI web service responds to the Broker request, the web service is automatically disconnected. It is assumed that the Broker application is finished when it displays the contents of the ODI web service response.

**Feature 3: View insurance information**

To retrieve insurance information, the external insurance broker application will contact the ODI web service, pass the appropriate information, and will receive the appropriate listing. How the information is displayed is dependent on the external broker application.

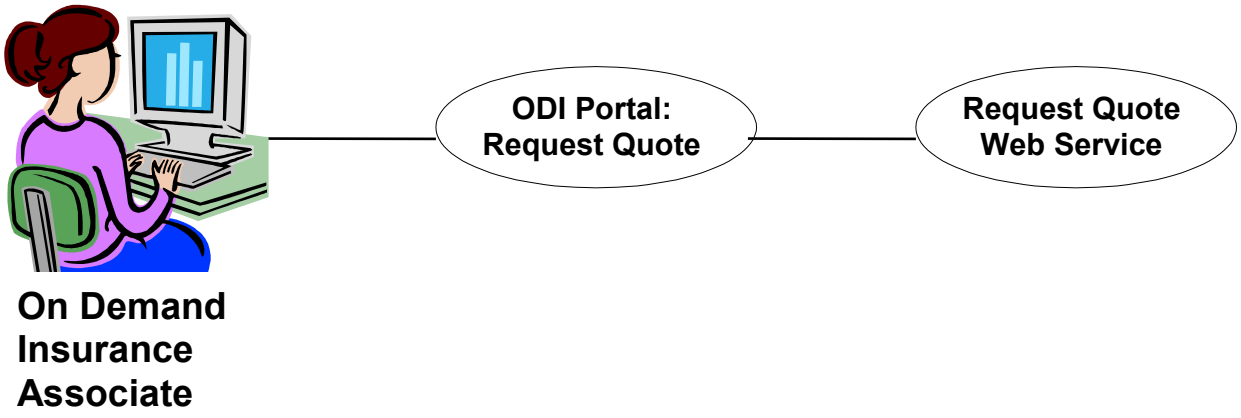
# Phase 3

### Vision Statement

Phase 3 of the ODIC web application is the “final” phase where ODIC will take all of the Phase 2 application and make it available on the existing ODIC main portal such that they can now offer the informal quote process to its associate brokers.

In Phase 3, ODIC will create a page that collects the necessary information, calls the web services, and then displays the results. Secure access to the page will be performed by the existing portal implementation controlling access to the page for those authorized to do so. The web services itself will not control secure access.

### Phase 3 - Use Case Model




### Phase 3 - Database Schema

Phase 3 DB is the same as Phase 1 and 2. For details, please see the database scheme details from Phase 1.

### Phase 3 - Features and Use Cases

Both customers and associate brokers, herein known as associates, will have access to the request for quote functionality through the corporate portal to request quotes. It assumed that the portal will control security requirements controlling access to the request for quotes page.

#### Feature 1: Application startup

Associates access the Request for Quote web application.

Use Case 1.0 Basic Flow:

Associates login to the ODI portal, providing the necessary username and password credentials. Once logged into the ODI portal, associates click on the Request for Quote link to access the Request for Quote page.

#### Feature 2: Application end

Associates leave the Request for Quote application.

Use Case 2.0 Basic Flow:

If the associate no longer wishes to request new quotes, they simply leave the page by accessing other pages within the ODI portal or logoff of the portal itself.

## 8.2 Application Prototypes

There were two application prototypes that were designed before development began. These two applications were end user applications and needed to be defined in detail, page by page. The two applications include:

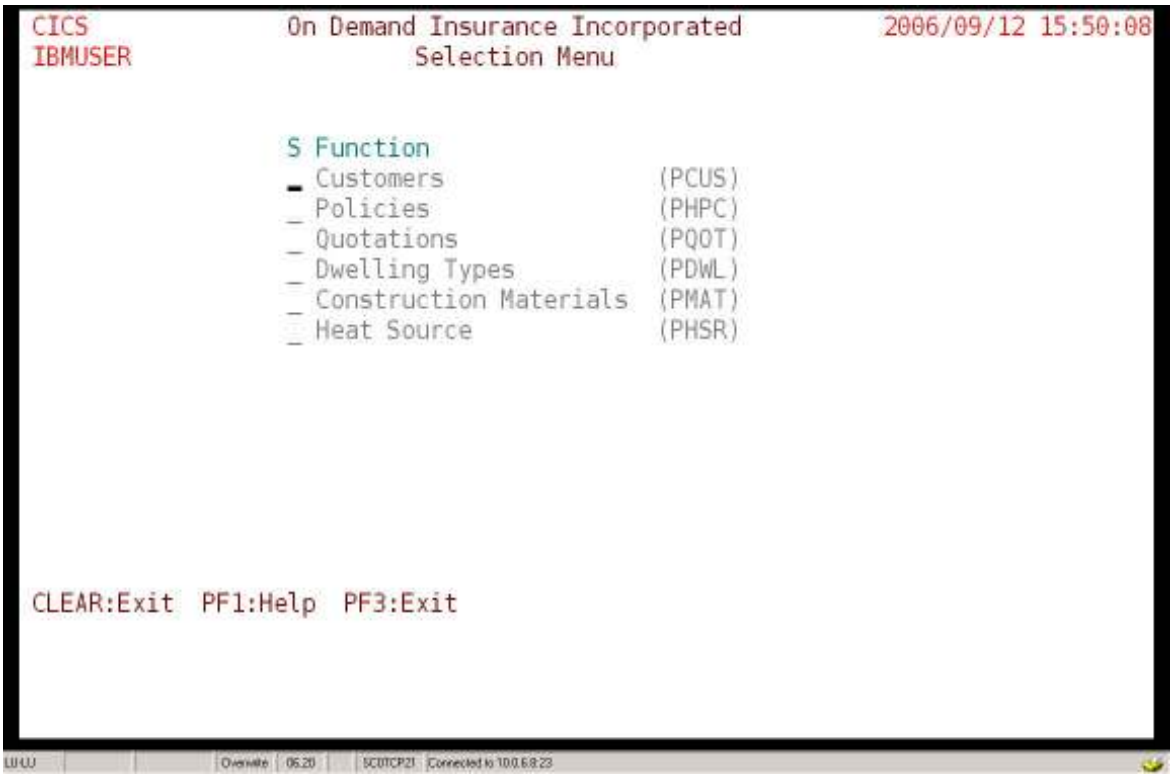
1. ODI CICS Application
2. Web Application

The web services application was generated based on the input and output of the CICS based application and did not have a prototype designed. The applications are described in detail in the "application requirements" section (8.1).

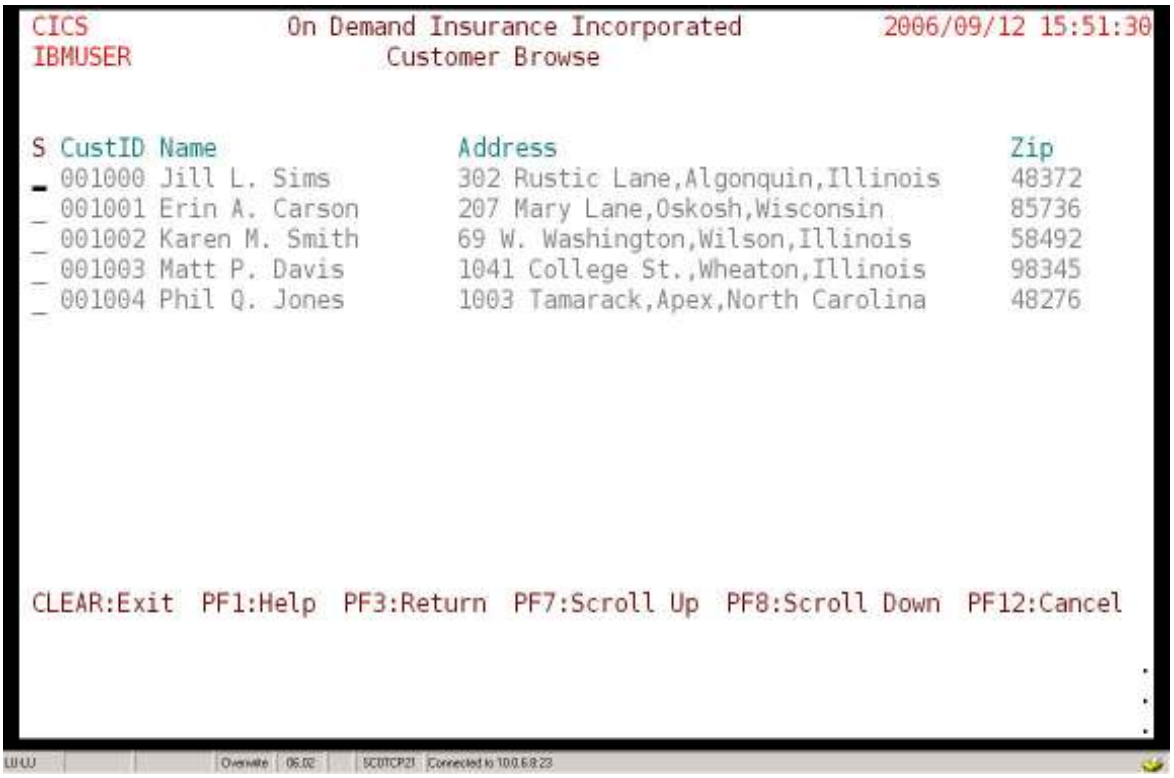
### Homeowner Policy Quote Application

The following screen shots provide the screens that users will see as they step through the portions of the applications. The prototype pages where created through both IBM WebSphere Developer for System z and traditional IBM mainframe development tools. The first is the menu select screen to access the various parts of the application, while the remaining provide views of the other input and browse screens. They detail the requirements as to what the employee will see as they step through each part of the application. The prototype pages were originally designed using the IBM Host ISPF interface. Both the IBM Host tools and IBM WebSphere Developer for z/OS had to build the corresponding applications to this prototype.

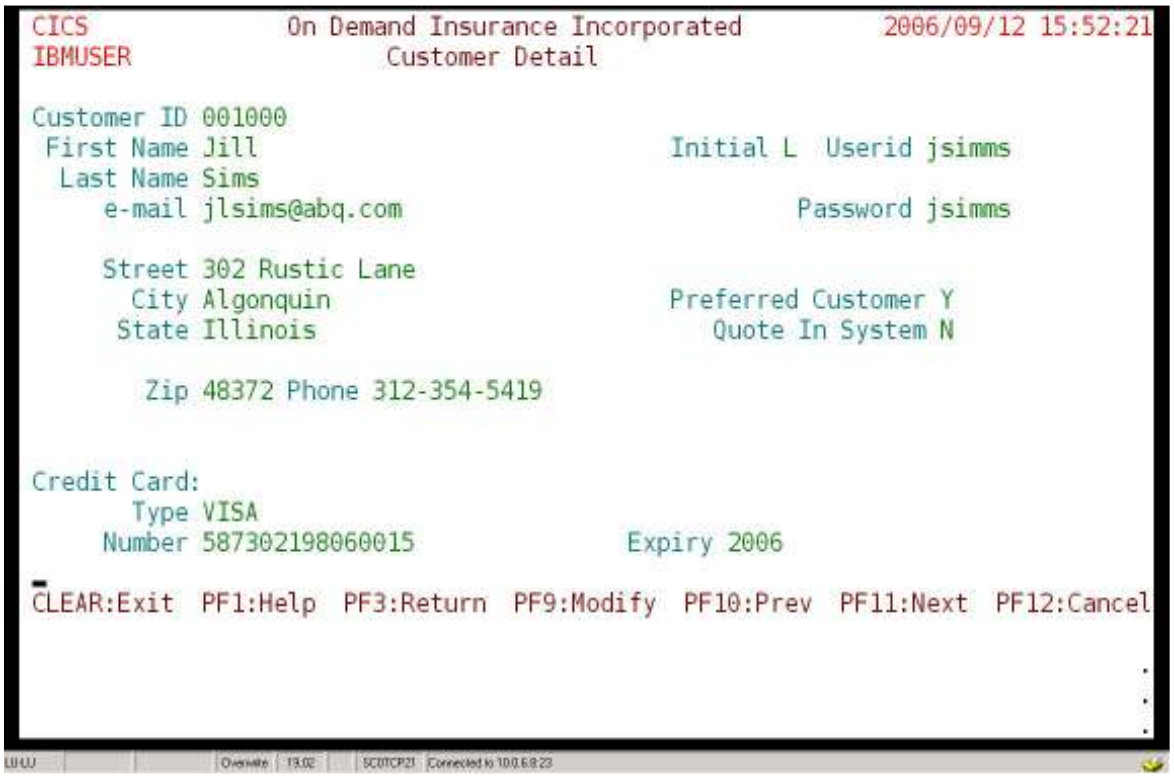
#### 1. Test Case 1B ODI Selection Menu - CICS Screen Shot



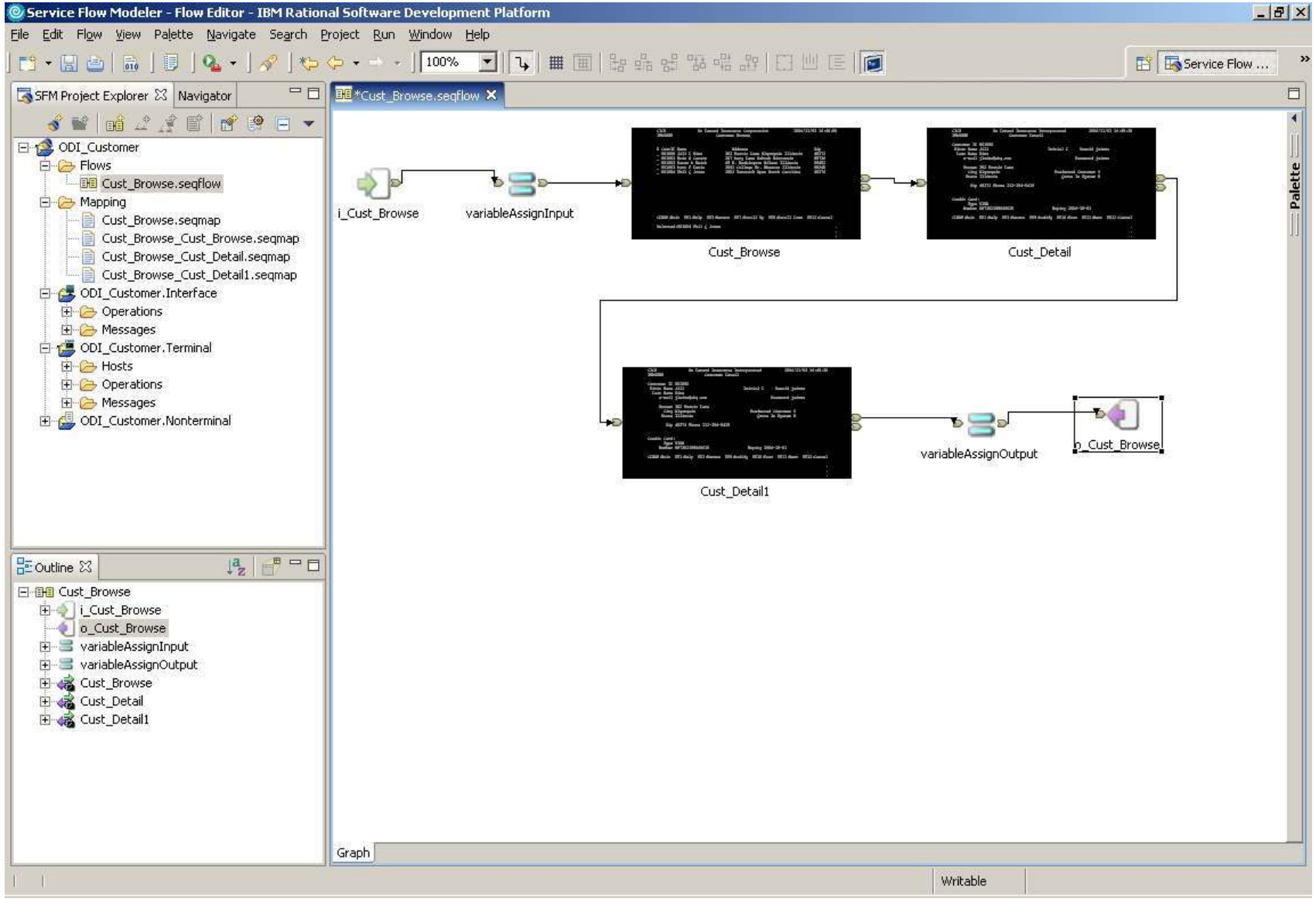
## 2. Test Case 1C ODI Customer Browse Panel - CICS Screen Shot



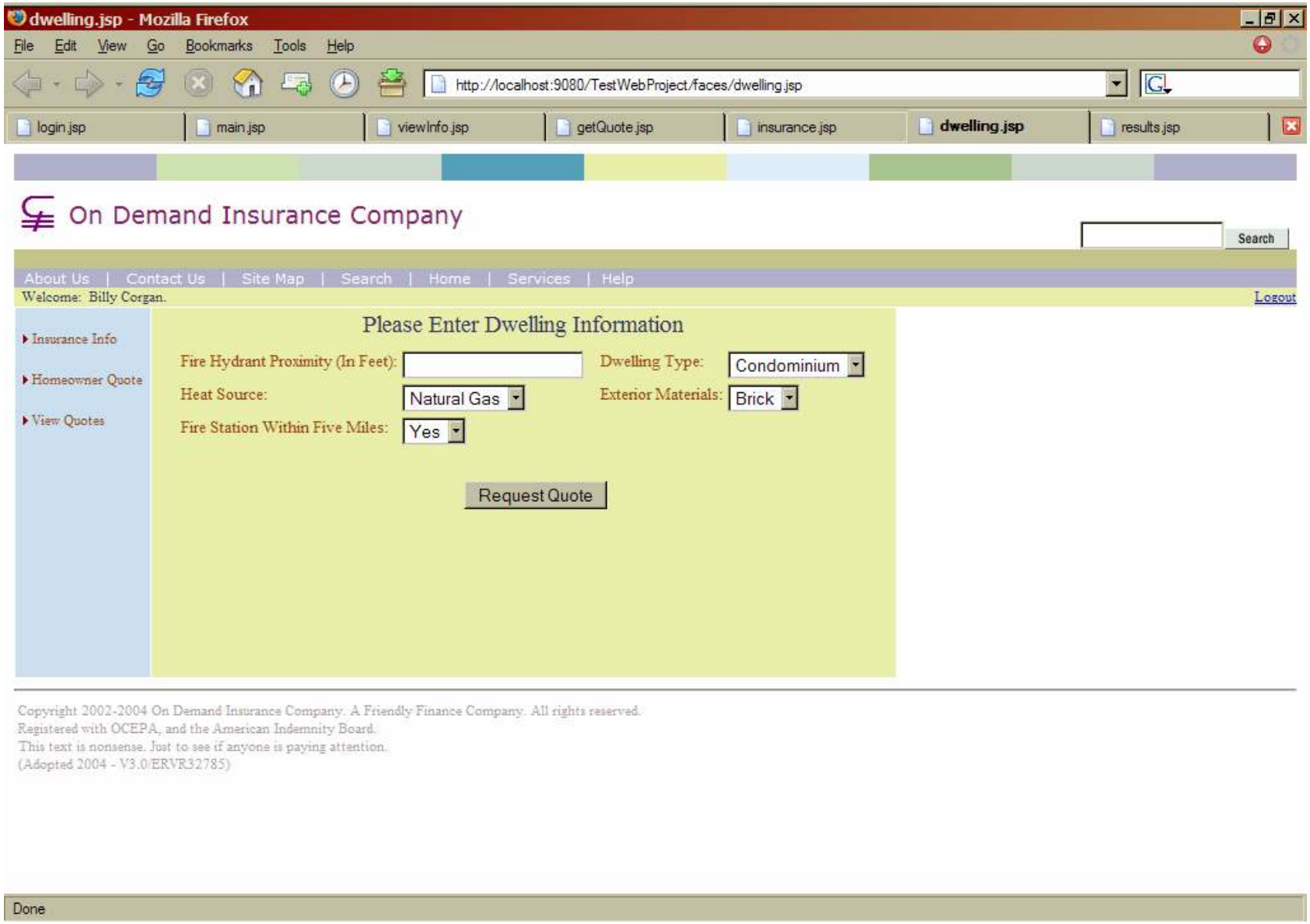

## 3. Test Case 1C ODI Customer Details Panel - CICS Screen Shot



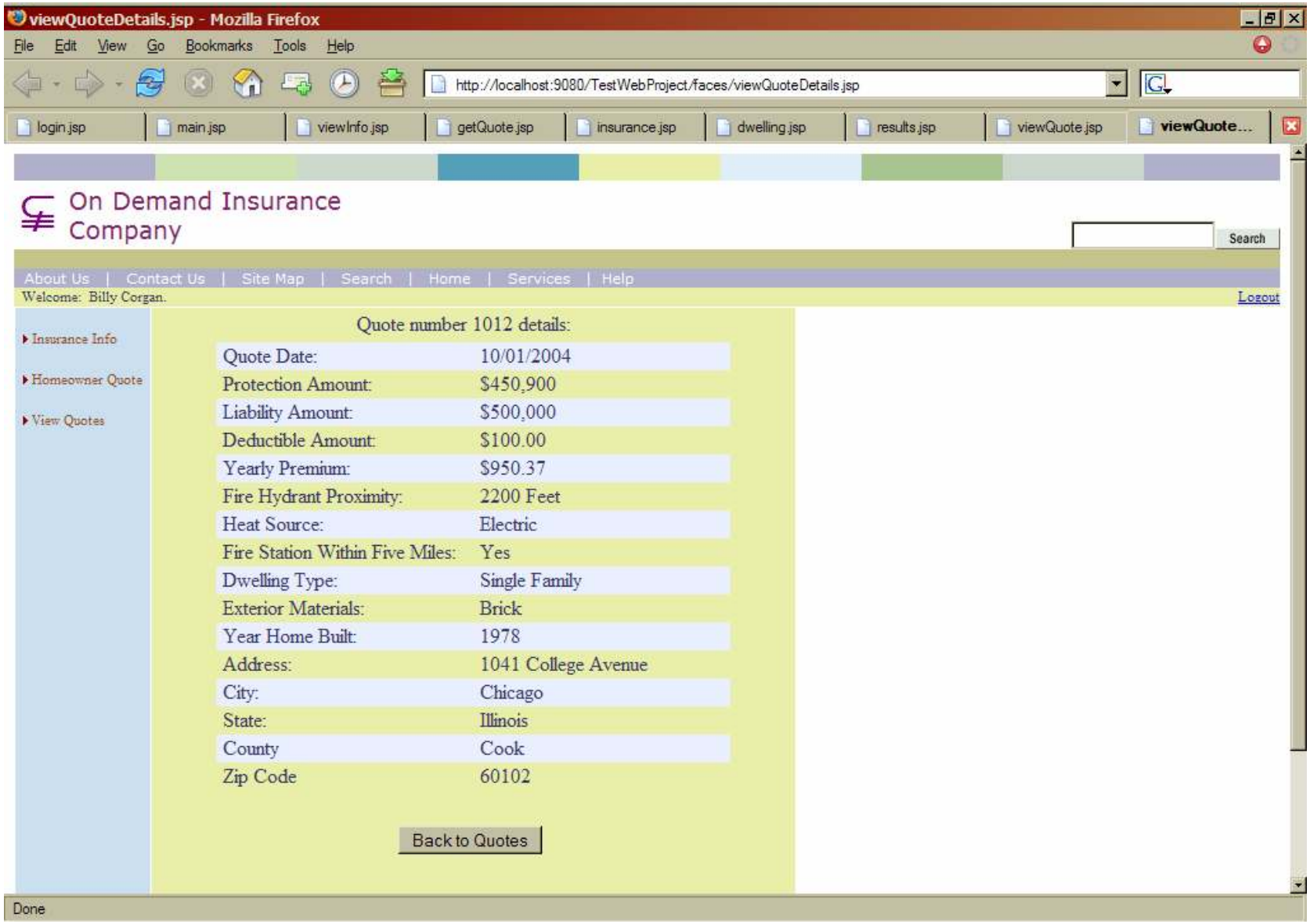
## 4. Service Flow for Application 3



## 5. Request Quote Page from Application 4



## 6. Quote Response Page from Application 4



## 8.3 Detailed Steps to build the IBM Applications

The following tables represent each of the applications that developers built within the traditional mainframe and WDz development environment. The individual line items within each table document the steps used for each of the applications and provide a reference for each timed step. Each of these steps were measured multiple times.

| Application 1 – Building the CICS App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the Object** |
| Test Case 1A | Application Setup | 1. Allocate ISPF project files for COBOL |
| | | 2. Allocate ISPF project files for DB2 |
| | | 3. Allocate ISPF project files for BMS, SDF2 |
| | | 4. Create CICS system definitions for Programs, Transactions and BMS MAPSETs. |
| Test Case 1B | Menu Panel | 5. Create BMS Selection Menu panel in SDF2 |
| | | 6. Create BMS Help panel in SDF2 |
| | | 7. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 8. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 9. Compile MAPSET load module using ASMH. |
| | | 10. Test MAPSET in CICS using CECI. |
| | | 11. Develop COBOL application to display Menu and switch to application transactions selected on the menu. |
| | | 12. Compile CICS program and install on CICS system. |
| | | 13. Test Menu transaction, Help panels and Selections. |

| Application 1 – Building the CICS App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the Object** |
| Test Case 1C | Customer Browse Panel | 14. Create BMS Browse panel in SDF2 |
| | | 15. Create BMS Detail panel in SDF2 |
| | | 16. Create BMS Help panel in SDF2 |
| | | 17. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 18. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 19. Compile MAPSET load module using ASMH. |
| | | 20. Test MAPSET in CICS using CECI. |
| | | 21. Develop COBOL application to reformat table columns into browse line for display. |
| | | 22. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 23. Compile CICS program and install on CICS system. |
| | | 24. Test Browse panel, Detail panel and Help panels. |
| Test Case 1D | Policy Browse Panel | 25. Create BMS Browse panel in SDF2 |
| | | 26. Create BMS Detail panel in SDF2 |
| | | 27. Create BMS Help panel in SDF2 |
| | | 28. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 29. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 30. Compile MAPSET load module using ASMH. |
| | | 31. Test MAPSET in CICS using CECI. |
| | | 32. Develop COBOL application to reformat table columns into browse line for display. |
| | | 33. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 34. Compile CICS program and install on CICS system. |
| | | 35. Test Browse panel, Detail panel and Help panels. |

| Application 1 – Building the CICS App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the Object** |
| Test Case 1E | Quotation Browse Panel | 36. Create BMS Browse panel in SDF2 |
| | | 37. Create BMS Detail panel in SDF2 |
| | | 38. Create BMS Help panel in SDF2 |
| | | 39. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 40. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 41. Compile MAPSET load module using ASMH. |
| | | 42. Test MAPSET in CICS using CECI. |
| | | 43. Develop COBOL application to reformat table columns into browse line for display. |
| | | 44. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 45. Compile CICS program and install on CICS system. |
| | | 46. Test Browse panel, Detail panel and Help panels. |
| Test Case 1F | Dwelling Type Browse Panel | 47. Create BMS Browse panel in SDF2 |
| | | 48. Create BMS Detail panel in SDF2 |
| | | 49. Create BMS Help panel in SDF2 |
| | | 50. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 51. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 52. Compile MAPSET load module using ASMH. |
| | | 53. Test MAPSET in CICS using CECI. |
| | | 54. Develop COBOL application to reformat table columns into browse line for display. |
| | | 55. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 56. Compile CICS program and install on CICS system. |
| | | 57. Test Browse panel, Detail panel and Help panels. |

| Application 1 – Building the CICS App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the Object** |
| Test Case 1G | Materials Browse Panel | 58. Create BMS Browse panel in SDF2 |
| | | 59. Create BMS Detail panel in SDF2 |
| | | 60. Create BMS Help panel in SDF2 |
| | | 61. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 62. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 63. Compile MAPSET load module using ASMH. |
| | | 64. Test MAPSET in CICS using CECI. |
| | | 65. Develop COBOL application to reformat table columns into browse line for display. |
| | | 66. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 67. Compile CICS program and install on CICS system. |
| | | 68. Test Browse panel, Detail panel and Help panels. |
| Test Case 1H | Heat Source Browse Panel | 69. Create BMS Browse panel in SDF2 |
| | | 70. Create BMS Detail panel in SDF2 |
| | | 71. Create BMS Help panel in SDF2 |
| | | 72. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 73. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 74. Compile MAPSET load module using ASMH. |
| | | 75. Test MAPSET in CICS using CECI. |
| | | 76. Develop COBOL application to reformat table columns into browse line for display. |
| | | 77. Develop COBOL application to display detail fields for table and edit input for updates. |
| | | 78. Compile CICS program and install on CICS system. |
| | | 79. Test Browse panel, Detail panel and Help panels. |

| Application 1 – Building the CICS App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Test Case** | **Test Case** |
| Test Case 1I | New Quotation Panels | 80. Create BMS Help panel(s) in SDF2 |
| | | 81. Create BMS Quote Progress panel in SDF2 |
| | | 82. Create BMS MAPSET (Panel Group) in SDF2 |
| | | 83. SFD2 Generate: MAPSET COPY books and BMS source. |
| | | 84. Compile MAPSET load module using ASMH. |
| | | 85. Test MAPSET in CICS using CECI. |
| | | 86. Develop COBOL application for New Quotation panel input.<br>87. Compile.<br>88. Install on CICS.<br>89. Test panel. |
| Test Case 1K | End to end test | 90. Verify menu works |
| | | 91. Verify all Help panels |
| | | 92. Verify all navigation works |
| | | 93. Verify exit works |
| | | 94. Select Customer for Quotation |
| | | 95. Request quote with the following, and see note<br>- Lowest RF<br>- Highest RF (No quote given)<br>- Medium RF<br>- Very High RF (Quote given) |

| Application 2 - Building a Web Service from existing CICS code | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the object** |
| Test Case 2A | Web Service Generation | 96. Workspace configuration |
| | | 97. Import necessary COBOL and COPYBOOK files into the local project |
| | | 98. Enable Enterprise Web Service |
| Test Case 2B | Host Installation | 99. Copy driver file to host |
| | | 100. Install load library into CICS with CEDA |
| | | 101. Install WSBIND file |
| Test Case 2C | Testing | 102. Use Web Services Explorer to test application. |

| Application 3 – Create a Service Flow | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the object** |
| Test Case 3A | SFM Model Definition | 103. Create blank SFM |
| | | 104. Add host definition |
| | | 105. Import BMS |
| | | 106. Record flow |
| Test Case 3B | SFM Field Mapping | 107. Map BMS fields to variables |
| Test Case 3C | Generate and Deploy | 108. Generation of Properties |
| | | 109. Generate Web Service and Deploy |
| | | 110. Verify Host deployment |
| | | 111. End to end testing |

| Application 4 – Building the Web App for the homeowner policy quote | | |
|---|---|---|
| **Test Case** | **Object being Created** | **Detailed Steps to Create the Object** |
| Test Case 4A | Application Setup | 112. Create Dynamic Web Project |
| | | 113. Layout application with Web Diagram Editor<br>114. Layout JSPs and name them<br>115. Connect JSPs together and make them realized |
| | | 116. Create images folder<br>117. Copy in images |
| | | 118. Create page template file |
| | | 119. Update stylesheet<br>120. Output text color<br>121. font size: 8pt; color:#98450E |
| Test Case 4B | Web Services Client | 122. Create Faces JSP files from template |
| | | 123. Add Web Service |
| | | 124. Add input fields to request page<br>125. Add output fields to quote page<br>126. Drop in components<br>127. Manipulate components |
| | | 128. Test page and adjust as necessary |

## 8.4 Glossary

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

**HO Policy** – Homeowner Policy.

**Partner Broker** – Constitutes an On Demand Insurance Company partner with a significant number of brokers that resell ODIC homeowner insurance policies.

**Associate Broker** – Constitutes individuals who make their living on the reselling of ODIC homeowner insurance policies.

**Informal Quote** – This type of quote provides an automated response to 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:

**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.