Software Assistant for Management Tasks in a Fish Farm

A Manuscript
Submitted to
the Department of Computer Science
and the Faculty of the
University of Wisconsin-La Crosse
La Crosse, Wisconsin

by
Dean Robert Peterson
in Partial Fulfillment of the
Requirements for the Degree of
Master of Software Engineering
May 2010
Software Assistant for Management Tasks in a Fish Farm

By Dean Robert Peterson

We recommend acceptance of this manuscript in partial fulfillment of this candidate’s requirements for the degree of Master of Software Engineering in Computer Science. The candidate has completed the oral examination requirement of the capstone project for the degree.

______________________________
Dr. Kasi Periyasamy
Examination Committee Chairperson

______________________________
Dr. Kenny Hunt
Examination Committee Member

______________________________
Dr. Tom Gendreau
Examination Committee Member
ABSTRACT


Software is often created to automate some business processes a company develops and performs manually for a period of years. Developing such software products has many challenges. For one, the customer may be unaware of the software artifacts such as databases, requirements and design documents. In short, customers may only be able to use an executable given to them. Also, adoption of a new automated system can be hard for those customers who are accustomed to a routine process that they have used for several years. To make the transition from a manual process to using a software assistant easier and acceptable, the software system must be user friendly and must have close resemblance to the old processes as much as possible without sacrificing efficiency. This report discusses the design and development of a software package to automate many related business processes for a small fish hatchery and how each of the challenges mentioned above were successfully met. The report explains the details of the manual processes previously handled by the fish hatchery and what among those processes have been automated by the software package.
ACKNOWLEDGEMENTS

I would really like to thank Dr. Kasi Periyasamy for helping me a great deal throughout this project. His experience and expertise are a tremendous resource that I had needed often. From helping me gain access to the University server used to host my application to making excellent corrections on this document, he has played an instrumental role in helping me complete my work.
# TABLE OF CONTENTS

ABSTRACT ........................................................................................................ iii  
ACKNOWLEDGEMENTS .................................................................................... iv  
TABLE OF CONTENTS ..................................................................................... v  
List of Figures ................................................................................................... vii  
Glossary ............................................................................................................. viii  
1. Background Information .............................................................................. 1  
2. Getting Started ............................................................................................ 4  
3. Existing Processes ....................................................................................... 7  
   3.1 Fish Order Invoice/Worksheet ................................................................. 7  
   3.2 Cutting Report .......................................................................................... 7  
   3.3 Smoke Assessment .................................................................................. 7  
   3.4 E-mails .................................................................................................... 8  
4. Fish Processing Application ........................................................................ 9  
   4.1 Requirement Gathering .......................................................................... 9  
   4.2 Roles and Responsibilities ...................................................................... 11  
   4.3 Cross Cutting Concerns .......................................................................... 13  
   4.4 Benefits of the Software Assistant ......................................................... 13  
5. Application Architecture ............................................................................ 15  
   5.1 Spring Framework .................................................................................. 17  
   5.2 Hibernate Persistence .............................................................................. 18  
   5.3 Tiles View ................................................................................................ 19  
   5.4 Spring Webflow Controller ..................................................................... 20  
   5.5 AJAX, DWR, and DOJO .......................................................................... 21  
   5.6 Apache POI .............................................................................................. 23  
   5.7 Source Control ......................................................................................... 23  
6. Detailed Design ............................................................................................ 24  
   6.1 Entity Classes and Database Design ....................................................... 24  
   6.2 Business Logic and Persistence Classes ............................................... 28  
7. User Interface Design .................................................................................. 31  
8. Testing ........................................................................................................ 35  
   8.1 Project Challenges ................................................................................... 37  
9. Continuing Work ......................................................................................... 39  
10. Conclusion ................................................................................................ 40  
11. Bibliography .............................................................................................. 41  
12. Appendices ................................................................................................. 42  
   Appendix A. Sample Fish Order Screens .................................................... 42  
   Appendix B. Sample E-mail Screens ............................................................ 51  
   Appendix C. Sample Cutting Report Screens ............................................ 55  
   Appendix D. Sample Smoke Assessment Screens ...................................... 5960
Appendix E. Sample Administration Screens .................................................. 6263
List of Figures

Figure 1 - Chapter 2 Incremental Prototyping Model/“Staged Delivery” .................................. 5
Figure 2 - Chapter 4 Fish Processing System - Cross-functional Flowchart ......................... 12
Figure 3 – Chapter 5 Architecture of the Fish Farm Software Assistant ............................... 16
Figure 4-Chapter 5 Hibernate with a connection pool in a non-managed environment .......................... 19
Figure 5- Chapter 5 The "addOrder" state created for the ‘add order’ flow of the Fish Farm Software Assistant ........................................................................................................... 21
Figure 6- Chapter 6 Class Diagram for the Fish Farm Software Assistant .......................... 26
Figure 7-Chapter 6 Database Entity Relationship Diagram for the Fish Farm Software Assistant ......................................................................................................................... 27
Figure 8- Chapter 6 Business Logic and Persistence Classes ............................................. 29
Figure 9-Chapter 6 State Transition Diagram for Add Order Flow .................................. 30
Figure 10-Chapter 7 Main Menu Screen ............................................................................. 32
Figure 11-Chapter 7 First page when adding a new Fish Order ......................................... 33
Figure 12-Chapter 7 Generated Microsoft Word Document ........................................... 34
Figure 13- Chapter 8 Testing procedure ............................................................................. 36
Glossary

API
An Application Programming Interface is an interface implemented by a program to enable interaction with other programs, similar to the way a user interface facilitates interaction between humans and computers.

DWR
Direct Web Remoting is a Java library that enables Java code to be used on the server and JavaScript in a browser to interact and call each other.

EC2
Amazon Elastic Compute Cloud is a web service that provides resizable computing capacity in the cloud. It is designed to make web-scale computing easier for developers.

FO
FO is used as an acronym for Fish Order in the system being described.

IDE
An Integrated Development Environment is a tool that developers use to perform many tasks associated with writing software code.

IEEE
The Institute of Electrical and Electronics Engineers is an international non-profit, professional organization for the advancement of technology related to electricity. It has the most members of any technical professional organization in the world, with more than 395,000 members in around 150 countries.
**J2EE**
Java 2 Platform Enterprise Edition is a platform-independent, Java-centric environment from Sun for developing, building and deploying Web-based enterprise applications online.

**JSP**
A Java Server Page (JSP) is a Java technology that helps software developers serve dynamically generated web pages based on HTML, XML, or other document types.

**ORM**
An Object Relational Mapping (ORM) is a programming technique for converting data between incompatible type systems in relational databases and object-oriented programming languages.

**SQL**
SQL stands for Structured Query Language. It is a database language used for managing data in a relational database management system.
1. Background Information

The Bullfrog Fish Farm is a company that provides fresh and smoked fish products to other businesses near Menomonie, Wisconsin and the surrounding area. The current manual tasks in the farm include taking orders, tracking fish counts, tracking fish cuts for an order, determining how efficient their cutting process is, their decision process for the number of fish to smoke, and how to generate business through reoccurring e-mails. The owner is bogged down in paper and computer work because he uses Microsoft documents and Excel spreadsheets to store all business rules, “cutting reports”, FO (Fish Order) invoices and “smoking assessments” that are used to run his business. The documents are stored in many different directories; he often has to track down previously created documents to make changes or to copy information from one document and paste it into another. There are also a lot of calculations on the cutting reports and smoking assessments that are done by hand. This makes his work time consuming and prone to error. The following departments of his business also share the same files: Product Control, Green Processes (fresh fish), Value Added (smoked fish), Retail, and Sales. They all share the same files and can change whatever they like because there are no security measures or user roles.

The business is growing and reaching out to even more markets. The farm is getting busier and they expect the trend to continue. The owner also wants to expand the types of products the farm offers. This growth has made it even more critical to automate much of the business processes. It will allow the company to focus on creating more value for its customers and open up new markets without slowing down. As the owner states, “If you pull in the reigns, the horse stops.”

This project aimed at developing a web-based software assistant that converts all Fish Order invoices, worksheets, cutting reports, smoking assessments, and e-mails into a cohesive data driven application.

1 Cutting Reports and Smoke Assessments are explained in chapter 3.
A Fish Order contains a list of all available products from which a user selects some fish. The order also includes prices but the prices depend on the customer information. Some customers also have automatic reoccurring orders that are transferred to the weekly cutting report and smoke assessment. Those rules are all associated with each order in the system. The Value Added department smokes a certain number of fish each week to have on hand some smoked fish in stock in order to fill their expected orders. This process can become very costly if they guess the number of fish incorrectly and have to set up equipment to smoke more fish when they run out. They currently use an excel spreadsheet called a “Smoking Assessment” that is created each week to help them decide how many fish to smoke. The information in the smoking assessment is generated using information from past sales, how much product is already sitting on shelves, the product’s expiration date, and special considerations such as holidays. The Green Processes department uses a “cutting report” to calculate the real cost of one serving of fish and to determine if they are cutting the right sizes of fish to fill an order. They use serving size, instead of pounds of fish, because serving size is more meaningful to their customers; the serving size of a fish is the usable product the customer can expect to receive. They calculate this information based on several parameters such as the number of fish to cut, product cost per lb., the number of lbs., live weight of fish, and the number of lbs. cut weight. All of this information is stored in a separate “cutting report” excel spreadsheet each week and it contains all of the information for each customer order. Instead of copying and pasting every order into a new cutting report, the cutting report can automatically be populated based on the actual order information already stored in the new system. The calculations are also performed automatically once the user enters the appropriate data. Once the calculations are complete in the cutting report, that information is transferred back to the Fish Order to calculate the total price of the order.

The owner also has a Microsoft document that contains the procedure for sending e-mails to each of his customers. Customers receive e-mails at different times based on the product and quantity they usually purchase. Different e-mails are also sent to more than one individual for a specific customer for different purposes. For example, chefs receive
e-mails that are different than the e-mails sent to owners of each customer establishment. The dates and times of those e-mails are manually recorded on the Fish Order for tracking purposes if there are discrepancies later on.
2. Getting Started

Because the company has been around since 1994, the sponsor has a very good idea of the business logic required to run his company. That meant a more formal requirement document could be written to capture the core functionality the new system is responsible for. During a couple of initial visits, the developer was able to learn a great deal about the major components the sponsor wished to automate and write a software requirement specification based on the IEEE Standard 830-1998 for writing software requirement specifications [4]. Even though a thorough requirement document was written and the owner reviewed and approved its content, a less formal approach to software development was needed going forward. To get an understanding of how to implement the requirements already gathered and create a useful application, the developer had to use an incremental prototyping approach to elicit real user requirements. Not only did that approach work well for the developer, it also gave the sponsor the ability to see the progress of the project before the final product was delivered. The sponsor was able to give immediate feedback and adjust the design to more accurately reflect their needs. Development was still a controlled process that required solid architecture, careful management, and detailed technical planning. The project was broken into a series of staged releases that grew the application from top down. Invoices were implemented first, and then the cutting reports, smoke assessments and finally the e-mails. Completing each component was set as a known milestone. Still, each milestone had multiple iterations (releases). A new release was given to the sponsor almost once in every three weeks, with additional features added to the interface. The sponsor gave a tremendous amount of feedback that was easily incorporated due to the design and nature of incremental prototyping. Figure 1, taken from Steve McConnel’s “Software Project Survival Guide”, gives an excellent overview of the form of incremental prototyping model used in this project. Steve applies the term, “Staged Delivery”, to describe the technique.
Design and implementation were repeated until the customer (sponsor in this case) agreed the implemented software fulfilled all requirements. Changes to requirements were designed for and expected. To facilitate the development of the Fish Farm Software Assistant, the initial prototype consisted of just the user interface. The first release is unique in that it required a fair amount of initial architecture building to take place. However, no more functionality was created than what was necessary to show the sponsor how the system would flow through the user interface. Subsequent implementations added more functionalities to the user interface and adjustments were made based on customer collaboration. Current development tools and frameworks make the process easier than in the past. However, it is much about increasing discipline rather than reducing it. Continuous testing, code reviews (paired programming) and other techniques are very important as well. Constant feedback from the customer is also
important to keep user expectations close to real functionality of the product. The initial requirement document or “plan” was created when the least amount was known about the system. Dwight D. Eisenhower said, “Plans are nothing; planning is everything.” The iterative process allows the developer to respond to change and to keep planning throughout the entire development process.

There are other added benefits to the iterative prototyping process as well. One benefit the developer was able to take advantage of while developing the Fish Farm Software Assistant was to create a better fitting architecture. By taking a few iterations to finalize the architecture, the developer had the option to evaluate ideas, let everyone understand trade-offs, and provide a platform to choose what is right based on growing understanding of the project needs [11]. Another benefit is that the very existence of a tangible object serves to contain and quantize user demand for changes [7]. Once the customer used parts of the application it seemed they would give more thought to requesting drastic changes to them. The sponsor automatically focused new requests based on what they could see on the screen.
3. Existing Processes

This chapter explains in detail the manual processes that are currently used in the fish farm.

3.1 Fish Order Invoice/Worksheet

Fish Orders come in through phone or through e-mail. If the order is taken by phone, the products are jotted down on paper and saved for entering in a Microsoft Word document which becomes the initial Fish Order invoice. The Fish Order invoice doubles as a worksheet for keeping track of the current status of the order and for keeping track of other relevant information. If the order is received by e-mail, it is usually a reoccurring order. In that case, an old Fish Order invoice is searched for that is similar to the new request. In fact, that is how most invoices are created. To make entering new information easier, multiple options existed in text format on each invoice for various things such as market type. The correct choice could be highlighted rather than typed in. This created a lot of unnecessary text on the final invoice and made it harder to read. After all of the products are entered along with shipping type, shipping date, etc. the order information is transferred to a weekly cutting report for further processing.

3.2 Cutting Report

The cutting report is placed into a shared folder on the internal network. Employees cutting fish enter values into the document after they prepare an order and the total cost of the order is calculated. The total cost is based on how well employees are able to select and cut the right sized fish. After the cutting report is complete, the totals are manually transferred back to each invoice.

3.3 Smoke Assessment

The smoke assessment is also an excel spreadsheet. Its purpose is to give an estimate of how many lbs of spread to prepare and fillets to smoke. It is similar to the cutting report. However, it contains smoke product information from each order for a week
separated by requests for “better” and “best” ‘eat by’ dates. Customers can request smoked products with a varying amount of freshness. If they request all “best” dates, the amount of smoked fish that must be prepared is more. Products already on hand with “better” dates cannot be used to fill an order for customers requesting the “best” dates.

### 3.4 E-mails

E-mails are sent to customers for a number of reasons during each week. The sponsor has a Microsoft word document that contains templates for e-mails sent out on a regular basis. Each template has contact information, large order notes, a message, product information, quantities and price. The template is copied from the Microsoft word document and entered as a new e-mail. E-mails are also sent to customers for payment requests. When e-mails go out that are associated with an order, a note is made on the Fish Order’s invoice as a record.
4. Fish Processing Application

4.1 Requirement Gathering

An initial requirements document was written based on the developer’s understanding of the application domain. This requirements document had a high-level overview of the application and some GUI screens. Without something to see, it is hard to visualize everything the system must do to meet the business needs. Charts and graphs are often used to facilitate the process. However, for this application, the developer chose to implement and start with a non-functioning user interface. The interface was designed and implemented based on the initial requirement specification. Once complete, another meeting with the sponsor was scheduled.

During the first meeting, all of the component interfaces were demonstrated to the user. The sponsor gave feedback and the developer kept notes about new requirements that were missed in the initial specification. A lot of time during development was saved due to the first meeting alone.

After the first meeting, it took considerable time for the developer to re-design the interface and fully implement the invoice component. Then another meeting was arranged to view the progress and generate more feedback. There were a significant number of requests for changes at this point. The sponsor was finally starting to envision using the system and gave very detailed information about how the system should work. The invoice component was functioning and could be run by the user. This allowed the developer to set up a bug and enhancement tracking system. The developer used FogBugz (on-line bug tracking software) for the purpose of tracking bug and enhancement requests. FogBugz also gave the developer the advantage of getting constant feedback from the sponsor. The sponsor was asked to keep testing each prototype by using it to accomplish real work and was also instructed on how to use the basic functions of FogBugz. When bugs were discovered, the sponsor made an appropriate entry in the FogBugz system. In this way the developer automatically received an e-mail about each defect when it was entered. Change requests were also
made this way. One major problem was the difficulty of trying to get the sponsor to act as a tester. It really took a number of iterations for each component before the sponsor would be willing to use an already relatively refined piece of software. Instead, what more often happened was that requests were entered into FogBugz by the developer immediately after meetings. This still had the advantage of maintaining an on-line, central repository of bugs, change requests and notes. After a component was more refined, the user was then able and more willing to use FogBugz for entering minor bug and enhancement requests.

After the invoice section, a natural schedule of iterations developed. Every three weeks a meeting was arranged with the sponsor and functionalities for new components were demonstrated and discussed. Each component took a number of iterations before it was usable without the developer present even though the functionality described in the initial requirement document existed after the first iteration for each component. The full requirements were not revealed until the user sat down and tried to do real work.

The developer also observed the sponsor performing some of the tasks using their manual techniques. While focused on automating a specific task any time during a cycle, the developer asked to see how it was currently being done. For example, the earlier section about existing processes describes the observation that old orders were being searched for in directories, copied and used for new orders. This led to a system that focused on creating new orders from existing orders as the sponsor was accustomed to it. Now in the new system, creating an order from an existing client immediately brings the user to a search of old orders. When a similar order is found, the user presses the "Duplicate Order" button to create a new order with much of the information already filled out. The search is more efficient because the user can enter criteria to filter and perform a query on a database. Rather than search through directories in the computer's file system, the database returns the required information.
4.2 Roles and Responsibilities

The Fish Farm Software Assistant will be used by three distinct types of users. Each user has different responsibilities and can only perform certain tasks within the system. The cross-functional flow chart shown in Figure 2 illustrates their roles and responsibilities along with the business processes they access in the system. The Administrator performs operations in the Administration swim-lane of the flowchart. Customer service representatives perform operations in both Customer Service swim-lanes. Shop Workers have access to the Fresh Fish Department and Value Added Department swim-lanes.
Figure 2-Chapter 4 Fish Processing System - Cross-functional Flowchart
4.3 Cross Cutting Concerns

The main focus of the processes listed in Figure 2 is to sell the products the Bullfrog Fish Farm offers. These products and their business processes led to a kind of “cross-cutting” concern. Cross cutting concerns usually deal with generic issues that are common to many applications such as logging, transactions and persistence. However, in the Fish Farm Software Assistant, products span the entire system architecture. These concerns are specific requirements or considerations that must be addressed in order to satisfy the system goal. In that way, all concerns that affect multiple system functions and features as a cross-cutting concern can be addressed [10].

While creating an order in this application, the product belongs to one of the following types: fresh fish, smoked, pickled, miscellaneous, or sockeye salmon. Also, each product is packaged in one of the following ways: by the pound, by the tub, by the pack or by shaker. These two attributes drive much of the work within the application. For example, when associating a product with a client, fresh fish must have a price per pound and a target serving price. Custom smoked fish only require a price per pound, and a smoked pack product requires a price per package and package size in ounces. On the Fish Order, fresh fish are displayed differently than smoked fish. Calculations performed for the Cutting Report and transferred back to the Fish Order are also different based on product type and package. Smoked products are included on Smoking Assessments and the package size is associated with the columns. Sockeye salmon are included on a specific Sockeye Smoking Assessment.

Much like other cross cutting concerns have specialized code decoupled from the rest of the software, Products have there own interfaces to perform functions specific to products. Chapter 7 contains the Class diagram that describes the separation of concerns throughout the application.

4.4 Benefits of the Software Assistant

The online Fish Orders have friendly tools to help enter information for dates and times to be shipped, billing arrangements, delivery options etc. The new system
automates creating smoke assessments and cutting reports as well. In the new system, every e-mail is associated with the appropriate Fish Order and saved as a template for future orders. There are also different types of users in the new system based on their department and role and they only have access to the functions they are responsible for. In summary, the software assistant will automate many of the manual processes that are currently managed by the fish farm and is expected to improve the productivity of the farm.
5. Application Architecture

The developer realized a web application would work well for this project because users at the farm access computers in different locations. The owner works primarily from his home on customer service tasks. His employees work exclusively in shops in the Fresh Fish and Value Added departments on the farm which is away from the owner’s home. An internal network between the farm and the house is used to share documents between computers. A web application would make sharing documents unnecessary. Furthermore, it would allow users to make modifications to data simultaneously. Figure 3 below gives an architectural overview of the Fish Farm Software Assistant.
Figure 3 – Chapter 5 Architecture of the Fish Farm Software Assistant
5.1 Spring Framework

The Spring framework was used to make the process of writing the Fish Processing Software more efficient. Spring provides a range of capabilities for creating enterprise Java, rich web, and enterprise integration applications that can be consumed in a lightweight, a-la-carte manner [3]. The Spring framework can be thought of as a toolbox for web application development. It is a sort of abstraction layer over many other frameworks that perform specific functions. One of the benefits of using the framework is that it enforces the MVC design pattern.

“Several problems can arise when applications contain a mixture of data access code, business logic code, and presentation code. Such applications are difficult to maintain, because interdependencies between all of the components cause strong ripple effects whenever a change is made anywhere. High coupling makes classes difficult or impossible to reuse because they depend on so many other classes. Adding new data views often requires reimplementing or cutting and pasting business logic code, which then requires maintenance in multiple places. Data access code suffers from the same problem, being cut and pasted among business logic methods.

The Model-View-Controller design pattern solves these problems by decoupling data access, business logic, and data presentation and user interaction.”


The MVC design pattern solves these problems by decoupling data access, business logic, and data presentation and user interaction [1].

Another benefit the Spring Framework provides is dependency injection. In object-oriented programming dependency injection is a mechanism for supplying a class an instance of a variable. The containing class does not need to instantiate its own instance variables. Instead, the Spring Configuration can describe the specific implementation its beans should use for instance variables. In this way Spring dependency injection acts as
a factory supplying implementation code to its beans. It is good that classes do not both create and use the same object. This allows the injected object to be any concrete type of an Interface or Abstract class. Its implementation details are decoupled from the class using it.

5.2 Hibernate Persistence

The Model code in the Fish Processing application consists of Java classes that reside on the server. The business logic is divided among classes based on concerns. The objects in the domain model are mapped to the database using Hibernate [2]. The same way model objects encapsulate their instance variables, classes that should only be concerned with Fish Orders or Smoking Assessments encapsulate those objects as well.

Hibernate is an Object Relational Mapping (ORM) framework that makes saving to and retrieving information from a database more programmatically intuitive. Instead of writing SQL statements and making persistence calls directly, objects in the domain model are mapped to tables in the database. The calls to the database are separated into persistence classes that simply save and retrieve information packaged into Java objects based on hibernate XML mappings. All of the relational integrity rules are contained within the mapping files as well as in the database (in case manual updates are performed). Figure 4 below describes the relationships between Hibernate, the database and the application.
Persistence-related code can be the most tedious in a Java application. Hibernate eliminates much of the grunt work and lets the developer concentrate on the business problem [6].

5.3 Tiles View

The Spring Framework has the ability to automatically decide which type of view to display based on a given URL. Different rules in the spring configuration file allow the application to automatically call code to generate Microsoft documents (Word and Excel) and display the results using appropriate types. Views can also be JSP pages that only contain display information. The JSP Views are constructed using another framework called Tiles. Tiles works when the developer creates display definitions that can be reused throughout the application. For example, the Fish Farm Software Assistant has the same look and feel throughout the pages. On each of those pages the header at the top of the page remains the same. It is not necessary to copy and paste the same information in multiple JSP pages to display the header on each page. Instead, a standard template definition is created and every page that extends the standard template automatically contains the default header.
5.4 Spring Webflow Controller

Without a framework there are many cumbersome tasks the developer would have performed manually while creating a web application. One of those tasks would be writing code to decide what business logic to execute or what page to transition after a user presses a button or performs some other action on the interface. Web applications often have many pages to display and numerous transitions between those pages. For a large application, such as the Fish Farm Software Assistant, it would be a lot of work just to write all of the code to decide how to handle transitions between pages. The developer would normally have to write a lot of supporting code to handle all scenarios of a user’s interaction with each page. Code would have to be written to decide if the user pressed the “previous” button or the “next” button on a page; what code should be called on previous and next for any given page? Trying to handle controlling logic without a framework can be quite tedious. Input validation is one of the major and important tasks to be implemented for any user interface. It may be tedious depending on the number of pages and the number of elements to be validated on each page. Luckily, Spring Webflow made validation and transitions easy in the Fish Farm Software Assistant application by automating much of the work.

Spring Webflow implements the model of a finite state machine. The states in the Fish Processor are defined in XML “flow” files. Each file refers to a specific “flow” in the application. A “flow” is a sequence of pages that are related to one another. For example, creating a new invoice requires transitioning through a number of pages and can be thought of as a “flow”. Flows can also call other flows called subflows. Transitions have a name such as “prev” or “next” and a “to” property that matches another state and a Tiles definition. The name of each state matches the name of a tiles definition. The match allows the framework to automatically display the right page when a transition is complete. Expressions that contain business logic to execute are also contained within each state and transition. Expressions can be entered at the start of a state, the start of a transition, the end of a state, the start of an entire flow, the end of a transition, etc. Figure 5 shows the XML code created for the “addOrder” state for creating a new fish order in
the application. At a glance, developers can see what is being executed between transitions.

```xml
<view-state id="addOrder" model="order">
  <on-entry>
    <set name="flowScope.modifyState" value="false" type="boolean" />
    <evaluate expression="clientService.getClientList()" result="flowScope.clients" />
  </on-entry>
  <transition on="open" to="open" validate="false" />
  <transition on="setClientToMenu" to="menu" validate="false" />
  <transition on="getLatestClientOrder" to="viewLatestClientOrder" validate="false" />
    <evaluate expression="orderService.getLatestClientOrder(order.client)" result="flowScope.order" />
    <evaluate expression="client = flowScope.order.client" />
  </transition>
  <transition on="updateOrder" to="viewOrder" validate="false" />
    <evaluate expression="orderService.createOrUpdateOrder(order)" result="flowScope.updatedOrder" />
    <evaluate expression="order = flowScope.updatedOrder" />
  </transition>
  <transition on="addToSmokeAssessment" to="addToSmokeAssessmentWhileViewingOrder" />
    <set name="flowScope.message" value="null" />
  </transition>
  <transition on="addToCuttingReport" to="addToCuttingReportWhileViewingOrder" />
    <set name="flowScope.message" value="null" />
  </transition>
  <transition on="save" to="addOrder" />
    <evaluate expression="orderService.createOrUpdateOrder(order)" result="flowScope.updatedOrder" />
    <evaluate expression="order = flowScope.updatedOrder" />
  </transition>
  <transition on="addClient" to="addClient" validate="false" />
</view-state>
```

Figure 5- Chapter 5 The "addOrder" state created for the 'add order' flow of the Fish Farm Software Assistant

### 5.5 AJAX, DWR, and DOJO

One downside to having a web interface is the lack of intuitive functionality like in desktop applications. Transitioning from page to page or refreshing a full HTML or JSP page each time can create a disjointed experience for the user. For someone inexperienced using web applications (there are such users at the Bullfrog Fish Farm), completing seemingly simple tasks can be difficult. To make the Fish Farm Software Assistant application more intuitive, a combination of DWR (Direct Web Remoting) and DOJO (client side JavaScript widget toolkit) was used to create a rich interactive interface when necessary. This combination is a form of what has been coined Ajax (Asynchronous Javascript).

21
Usually, in a web application, the user must click a button or link to perform a particular task. After the button or link is clicked, a synchronous call is made to server side code such as a Servlet. The Servlet calls business logic code and that can, in turn, retrieve information from the database. The information from the database is populated in objects and a new page is refreshed that displays the information in the objects after performing an entire roundtrip. The Ajax pattern on the other hand attempts to deliver a rich, responsive user experience.

Ajax is a high-level design pattern composed of many related technologies and ideas [9]. It can be implemented in a number of ways but it usually requires sending an asynchronous request to the server in the form of an XML document. After parsing the XML, the server side code performs some work and a small XML file with information is sent back to the client. A JavaScript callback method that has been waiting for a response is executed and parses the returned XML. The callback method has access to the browser’s DOM model and can update only a small portion of the screen in many different ways. One problem with AJAX is that writing JavaScript code to create and parse XML as well as update the screen can be a very difficult task. It also often has to be customized to work on each browser. DWR was used to eliminate the need to write Ajax script to communicate with the server from scratch. At runtime, DWR creates client side JavaScript based on server side code to give the client side developer access to the methods in plain old java objects (POJOs). In this way, the developer only needs to create a small amount of JavaScript to essentially call the methods on the server. The developer does not need to worry about browser compatibility because DWR takes care of that. There is still the problem of updating the screen with the information DWR returns. For that, the DOJO toolkit was used.

The DOJO toolkit comes prepackaged with many different types of widgets that can be accessed with a JavaScript API. For example, when DWR returns an object that contains dates, the dates can be set on a DOJO calendar widget and placed on the screen. Again, compatibility of the widgets for different browsers is taken care of by its developers and so the developer was able to focus on domain specific coding.
5.6 Apache POI

One of the central requirements agreed upon was to have the new application programmatically generate the same Microsoft Word documents and Excel spreadsheets the sponsor was already using. This would make the transition to the new system much easier. To accomplish the task, the open source Apache POI API was used. POI is written in Java and easy to include in Java Web applications simply by including a single jar file. Once the jar file was added, special View code was written to programmatically create the invoice, cutting reports and smoking assessments.

5.7 Source Control

Another tool used to track the progress of required tasks is a tool named Kiln. Kiln is source control software created by the same company that creates FogBugz. It is a companion to FogBugz and integrates with that system. Changes are committed to a clone created from a repository stored online. Branches are easy to keep up to date because committed changes from one clone can be pushed to the original repository and then pushed to other branches. This was one of the tools that made refactoring an easier task. If the developer was given a significant new requirement or redesign task, it was easy to create a new branch and start refactoring the code with little worry about irrevocably damaging the main code base. If the new development work did not perform as needed, the changes were simply not pushed to the repository stored on-line. Also, whenever a change is committed, a note is added that describes what was changed. It was helpful to be able to find a specific change-set and be able to immediately focus on the code pertaining to a past problem if more changes were required.
6. Detailed Design

6.1 Entity Classes and Database Design

The Fish Farm Software Assistant required designing tables in a database to represent the entities of the system. Because the developer was using the Object Relational Mapping tool, Hibernate, a number of design issues had to be taken into consideration when creating the tables. Each of the tables would have to have a corresponding Java class as well as a mapping file to automatically generate the state of the database in memory as the application runs. It is best to start with designing the classes that make up the entity objects of the application.

Before starting, the developer understood that more complex relationships such as polymorphism in Java classes are hard to map. That is why the developer used composition over inheritance. In the class diagram in Figure 6 the entity classes contain other objects or lists of other objects if there is a necessary relationship. Designing the classes this way makes it easier to ensure the database is normalized when the tables are finally created. There are five levels of normalization; the developer of the Fish Farm Software Assistant normalized the database to the third level.

The first level requires all attributes to be single-valued. By using composition in the class diagram, any multi-valued variables are represented by other classes in the diagram. The corresponding relationship in the database calls for any multi-valued variables (lists or other similar data structures) to have a separate table to represent them. For example, the FishOrder class contains a list of OrderProducts. Accordingly, order_product is a table in the database. Figure 7 contains the entity relationship diagram for the application. In the figure, foreign key relationships are quite simple. Each table has a primary key that is automatically generated by the database. When an entity object is composed of another entity object, the entity object’s primary key column is included as a column in the other entity object’s corresponding table. Primary keys were automatically generated by the database. Complex keys make relationships between the tables harder to map and were also avoided. The second normal form requires all non-
key attributes to be functionally dependent on the table’s primary key. That simply means that if the primary key of some row ceases to exist, there is no column in that row with data that continues to exist because it does not necessarily belong exclusively to that row. Again, thinking of tables and classes interchangeably, the developer only included columns in a table that were single-valued and belonged to the corresponding class. A table is in third normal form when it is in second normal form and no transitive dependencies exist. A transitive dependency occurs when a functional dependency is inherited through some other identifying attribute [8]. The database was automatically in third normal form because the class design does not have transitive dependencies.
Figure 6- Chapter 6 Class Diagram for the Fish Farm Software Assistant
Figure 7 - Chapter 6 Database Entity Relationship Diagram for the Fish Farm Software Assistant
6.2 Business Logic and Persistence Classes

Figure 8 consists of the classes used to perform the business logic and persistence associated with each entity. Many of the objects in the domain model affect multiple parts of the application. That is why each model class has its own service implementation class to perform operations specific to it; much like persistence is separated into classes specific to saving and retrieving information from the database. That way behavior performed on specific entity objects only has to be modified in one location if changes become necessary. Notice that there are no class diagrams for user interface or controlling logic. That part of the application is written outside of Java classes. The controlling logic and calls to business logic are performed within the Spring Framework and are written within XML flow files. Specific implementations of the business logic are injected into the flows based on a configuration file. As described in section 5.5, the XML flows are written based on a state transition machine. Before each flow file was created, a state transition diagram was developed to visualize the states and transitions needed to perform the work of specific tasks. Figure 9 is the state transition diagram the developer created to describe creating a new Fish Order in the application.
Figure 8 - Chapter 6 Business Logic and Persistence Classes
Figure 9: Chapter 6 State Transition Diagram for Add Order Flow
7. User Interface Design

Because the Fish Processing Application is a web application, the implementation of the graphical user interface depends on internet browsers such as Internet Explorer and Mozilla Firefox. Most screens are divided into three sections: Header, Menu, and Content. Spring Webflow is used to transition from page to page and Ajax is used to perform actions on a single page. The online Fish Orders have friendly tools to help enter information for dates and times to be shipped, billing arrangements, delivery options etc.

When the user arrives at the first page, he/she must login. The login page uses a small amount of JavaScript built into the Spring Framework to supply hints to the user they should enter their username and password. If they leave anything blank, the input boxes will caution the user there is something incorrect. After logging in they are taken to the main menu screen. The user clicks on a menu button and the application transitions to a new page built using Tiles.
The main menu has buttons for each of the major components in the application. By clicking on a button the user transitions into a specific state associated with one of the many flows in the system. As mentioned earlier, a flow is a group of related states the application can be in and transition between to perform a specific task.

Figure 11 is a page that has typical attributes found throughout the application. The menu is still on the left side, the header is on top and it has a section for content. The page is bound to an object in the domain model with the help of the Spring framework. In this case, the page is bound to a FishOrder object.
On the left side is the option to “View Invoice”. By clicking on the link, a unique flow is invoked that programmatically creates a Microsoft Word document with the current Fish Order information (Figure 12). The content type of the browser is changed and the Spring Configuration knows to build the view using Microsoft Word and custom code rather than Tiles and JSP pages. The new system automates creating smoke assessments and cutting reports as excel documents as well.

Figure 11-Chapter 7 First page when adding a new Fish Order
Figure 12-Chapter 7 Generated Microsoft Word Document
8. Testing

Black-box test cases were created by the developer to ensure the application performs correctly. The test cases were recorded in an Excel spreadsheet.

The Fish Farm Software Assistant generates a number of documents. Each document consists of several sections. Many of the test cases set up scenarios to test the correctness of creating sections individually. The sponsor has many old documents the developer was able to derive test cases from. Each test case consisted of a column in the Excel document for Test Scenario, Expected Results, pass indicator, fail indicator, and notes. Keeping a record of each test allowed the same test cases to be used during regression testing after each iteration.

Testing was continuous throughout the development process. Testing and test case generation were largely done by the developer and that was not ideal. On a project of any magnitude, it is difficult, if not impossible, to do justice to some tasks involved in the software engineering process without separate people for different responsibilities. The test cases were not as complete or thorough as they would have been had a separate individual or individuals, more knowledgeable of the Bullfrog Fish Farm’s business, created them. Instead, it was a collaborative effort between the developer and a contact from the sponsor that has no business analysis training or knowledge of software testing. The sponsor would use the system during the arranged meetings and then reveal another version that must be handled. At that time, the new scenarios would be added as test cases to the Excel document. Once the test cases were known they served well for regression testing. However, a lot of time could have been saved if the organization was larger and had more resources for creating thorough test cases early on.

Usability testing was also extensively performed and that is where the iterative approach to software development shines. During each meeting with the sponsor, the actual user of the application would sit down and try to do real work. Often the user would replicate an old Fish Order, Cutting Report, or Smoke Assessment while testing the application’s usability. Usability problems are not revealed during unit testing or
when the developer performs test case scenarios. Before moving on to the development of a new component in the system, the user was known to be able to perform the desired task of the previous component. Also, the testing of later components such as Cutting Reports and Smoke Assessments depended upon the usability and refinement of the creation of Fish Orders. Having the Fish Order section near perfect along with having its test cases recorded made the development of the other components easier.

After the Fish Order component was complete, integration test cases were created for the Cutting Report, Smoke Assessment and E-mail sections of the application. For example, to create a Cutting Report, the user potentially creates many Fish Orders that appear on a single Cutting Report. Variations in Fish Orders affect the behavior of the Cutting Report. Test Cases for creating a Cutting Report involved creating a variety of Fish Orders.

The Figure 13 below portrays the iterative approach to test case creation used during the project.

![Figure 13](image)

Figure 13- Chapter 8 Testing procedure


Debugging and testing code was also made easier by the use of a commercial IDE plug-in for Eclipse (open source IDE created by IBM) called MyEclipse. MyEclipse
allows the developer to automatically deploy web application code to the application server of their choice with the click of a button. The developer can also start and stop the server from inside the Eclipse IDE with MyEclipse installed and even run in debug mode to attach breakpoints to the server-side code. Another helpful tool that was used to debug the client-side JavaScript code is the Firebug plug-in for the Mozilla FireFox browser. Firebug runs inside the browser and just like Eclipse is used to set breakpoints in server-side code, Firebug is used to set breakpoints in the JavaScript running on the browser.

8.1 Project Challenges

At first the developer took a different approach while designing the initial demo screens. The developer spent too much time turning the initial concepts from the requirement document into actual JSP pages and even using Tiles. The screens were somewhat distractive and caused the sponsor to focus on different things than the actual functionalities; for example, font size, color and spacing. The developer was trying to impress the sponsor with screens that made the application look almost done. Also, a demo was given that did not challenge the sponsor to think and so it only generated positive feedback. Assumptions were made from the initial requirements about how tasks could be accomplished on the screen and the developer narrated through the steps for each task. Instead, the developer should have created very rough sketches and asked very specific questions. For example, the screens for creating a new Fish Order could have been drawn by hand and the developer could have asked: “How would you enter the products for an order?” Any changes required at that point would only involve modifying the sketch. After the demo, a lot of time was spent developing functionality into the screens that were not thoroughly analyzed. A report published in 1988 indicated that requirements that change late in the development cycle cost 50-200 times as much to fix as those caught early in the development cycle [5]. Tools and frameworks today have made refactoring a much simpler task but it is still costly. Spending more time with sketches and screen design early on would have saved a considerable amount of time.

2 “Tiles” is a framework used to organize the view layer of a web application. It is described in more detail in section 5.3.
Another challenge was finding a host for the application. During testing, the developer wanted to allow the sponsor to have access to the application 24 hours a day as well as be able to modify the same version the sponsor had access to from a distance. To solve the problem, the developer had to find a host capable of serving a J2EE application on-line. There are not many hosting companies with servers capable of running J2EE applications. The developer first tried a hosting company called Kattare but was not satisfied with the processing power and bandwidth they provide. The developer then hosted the Fish Farm Software Assistant on Amazon Web Services EC2 (Elastic Compute Cloud). The performance Amazon provides was good and having the ability to add server instances for more processing power made the application run well on-line. However, the costs were prohibitive for this project. The developer then researched Google’s Application Engine to see if that could be used for hosting the application. Unfortunately, the developer found that hosting the Fish Farm Software Assistant on Google’s Application Engine would require a significant amount of redesign. Finally, the developer was able to use a server at the University of Wisconsin La Crosse at no cost to host the application during the testing phase. A permanent host has yet to be found.
9. Continuing Work

Clients could be added as another type of user to the Fish Processing Application. It would be easy to modify the system to allow clients to log in and order their products online. By removing some of the display information, the sponsor uses on the client product screen, a client could generate a new order for themselves by entering the quantities of products that are specific to them. In the past, the Bullfrog Fish Farm tried to offer this ability to their clients using static HTML web pages. However, the products and their prices are specific to each customer and the process quickly became too complex to continue on-line orders. With the current application in place all of the complexity is built into the system and can be hidden from the client.
10. Conclusion

The Fish Processing Application automates many of the tasks the Bullfrog Fish Farm originally had to perform to prepare and track the progress of an order. The application was designed with maintainability in mind. Many tools and frameworks were used to create a stable, solid and easy to use application. The iterative prototyping model of software development was implemented to keep real functionality as close to the sponsor’s expectations as possible. The process was not ad hoc as often occurs while using a more agile approach to software design. Many people embrace the idea of having to create less documentation and other aspects of an approach they view as being less formal. However, the iterative process requires more discipline than even the waterfall approach. It is about taking what works in other development techniques and pushing them even further. Instead of gathering requirements in the beginning and expecting little or no change, the developer designs for change and expects it. New requirements are gathered much earlier in the process than if a waterfall approach is used. Therefore change costs much less to implement.

Other aspects of an iterative approach are frequent automated tests. A unit test framework was not used for this application but it is another example of pushing what works in other techniques to higher levels. Again, instead of testing only at the end of a development cycle, unit tests can always be run to keep the code clean. Also, if this had been an application with more than one developer, pair programming could be used to make reviewing code frequently as well. This application was really a learning process for how to successfully create software using the iterative prototyping approach. Not everything was done right the first time and there is plenty of room for improvement. However, the application was a success; mistakes were caught early and expectations were kept very close to the delivered product.
11. Bibliography

12. Appendices

Appendix A. Sample Fish Order Screens

Search Fish Order Screen
### Search Fish Order Results Screen

<table>
<thead>
<tr>
<th>Fish Order Id</th>
<th>Client Name</th>
<th>Date Created</th>
<th>View and Modify</th>
<th>Add To Cutting Report</th>
<th>Add To Smoking Assessment</th>
<th>E-mail</th>
</tr>
</thead>
<tbody>
<tr>
<td>3016</td>
<td>Francesca's Cafe &amp; Catering</td>
<td>2009-11-29 14:48:55.0</td>
<td>View Fish Order</td>
<td>Add To Cutting Report</td>
<td>Add To Smoke Assessment</td>
<td>Send E-mail</td>
</tr>
<tr>
<td>3015</td>
<td>Francesca's Cafe &amp; Catering</td>
<td>2009-11-29 15:42:24.0</td>
<td>View Fish Order</td>
<td>Add To Cutting Report</td>
<td>Add To Smoke Assessment</td>
<td>Send E-mail</td>
</tr>
<tr>
<td>3014/2217</td>
<td>Birchwood Cafe</td>
<td>2009-11-29 15:52:33.0</td>
<td>View Fish Order</td>
<td>Add To Cutting Report</td>
<td>Add To Smoke Assessment</td>
<td>Send E-mail</td>
</tr>
<tr>
<td>3013/2219</td>
<td>Heartland</td>
<td>2009-11-29 15:05:26.0</td>
<td>View Fish Order</td>
<td>Add To Cutting Report</td>
<td>Add To Smoke Assessment</td>
<td>Send E-mail</td>
</tr>
<tr>
<td>3012/2222</td>
<td>The Good Apple</td>
<td>2009-11-29 14:36:36.0</td>
<td>View Fish Order</td>
<td>Add To Cutting Report</td>
<td>Add To Smoke Assessment</td>
<td>Send E-mail</td>
</tr>
</tbody>
</table>

Previous Results  More Results
Modify Fish Order top

|--------------|----------------------|-----------------------------|--------------------------|

- **System Order #:** 3020
- **Fish Order #:** 2194
- **Client:** CABLE NATURE
- **Sold To/Core of:**
  - Chef Mike
  - Bill Brakken Proprietor
- **Shipping/Handling Type:** Shipped
- **Shipped or Ready By:** 5/26/2009
- **Delivery or Picked Up:** 5/27/2009
- **“Shipping” NOTES:** Depending - could be delivered next day.
View/Modify Payments

Payments made for fish order: 2194

<table>
<thead>
<tr>
<th>Payment Date</th>
<th>Amount</th>
<th>Modify</th>
<th>Delete</th>
</tr>
</thead>
<tbody>
<tr>
<td>2019-02-27 14:07:27</td>
<td>12.0</td>
<td>Modify</td>
<td>Delete</td>
</tr>
</tbody>
</table>
Make Payment Screen

Payment for Orders: 2194

Amount

Check Number (optional)

Remaining Balance: 91,449

Previous  Save  Finish
### Select Products Screen

**View Invoice**

**Add Product**

<table>
<thead>
<tr>
<th>Product</th>
<th>Price</th>
<th>Size</th>
<th># of fish</th>
<th># to include</th>
<th># to include</th>
<th># to include</th>
</tr>
</thead>
<tbody>
<tr>
<td>Delicious Hickory Smoked Fillet-Rainbow Trout</td>
<td>$12.00</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sell By for product above:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Famous Smoked Trout Spread</td>
<td>$6.50</td>
<td>8.00 oz.</td>
<td>20</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Sell By for product above:</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Fresh Pn Baked Fillets-Rainbow Trout</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>HOBO Fish Spice</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**View Cutting Report**

**View Smoke Assessment**

**Previous** | **Save** | **Finish**
Add Fish Order To Cutting Report

Add FishOrder 2021 to cutting report:
Cutting Report Week Begin Date: 

[Buttons: Previous, Save]
Add To Smoke Assessment
Appendix B. Sample E-mail Screens

New E-mail Screen

Dear Mike,

...... Attention ........
Search Sent Emails

Client: CABLE NATURE      System Order: 3020

Create New Message  Search/Resend Emails  Options

Search

Message:

Subject:

Date Sent Start

Date Sent End

Search

New Message  Search

<table>
<thead>
<tr>
<th>Date Sent</th>
<th>Address</th>
<th>Subject</th>
</tr>
</thead>
<tbody>
<tr>
<td>2009-12-25</td>
<td><a href="mailto:peterson.dean@gmail.com">peterson.dean@gmail.com</a></td>
<td>Link text</td>
</tr>
<tr>
<td>2010-01-12</td>
<td><a href="mailto:peterson.dean@gmail.com">peterson.dean@gmail.com</a></td>
<td>Hi</td>
</tr>
<tr>
<td>2010-02-27</td>
<td><a href="mailto:peterson.dean@gmail.com">peterson.dean@gmail.com</a></td>
<td>Test</td>
</tr>
</tbody>
</table>
Modify Email Settings

Client: CABLE NATURE  System Order: 3020

Create New Message  Search/Resend Emails  Options

New Message  Search  View Email

Send To: petroncon.d chemicals@gmail.com
Subject: Hide text
Date Sent: 2009-12-26

Options:

Transport Type: POP3
Host: mail.chemicals.com
From: bulletin@chemicals.com
Username: bulletin
Password: ********

Previous  Finish
Appendix C. Sample Cutting Report Screens

Search Cutting Reports

Search Cutting Report Results
Modify Orders on Cutting Report

<table>
<thead>
<tr>
<th>Include</th>
<th>Fish Order Id</th>
<th>Client Name</th>
<th>Action</th>
</tr>
</thead>
<tbody>
<tr>
<td>✓</td>
<td>3814</td>
<td>Birchwood Cafe</td>
<td>Included Products</td>
</tr>
<tr>
<td>✓</td>
<td>3517</td>
<td>Just Local Foods</td>
<td>Included Products</td>
</tr>
<tr>
<td>✓</td>
<td>3818</td>
<td>Frankow's Cafe &amp; Catering</td>
<td>Included Products</td>
</tr>
</tbody>
</table>

Previous Results  More Results

View Products on Cutting Report for Order

| Cub Foods
Product Name                     | Cost  | Action                   |
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Fresh Pin Bone Fillet (L2)</td>
<td>$24.0</td>
<td>Modify Cutting Report Values</td>
</tr>
<tr>
<td>Fresh Head-On Hemmingway Rainbow Trout</td>
<td>$23.0</td>
<td>Modify Cutting Report Values</td>
</tr>
</tbody>
</table>

Previous Results  More Results
Modify Cutting Report Values using System

<table>
<thead>
<tr>
<th>Fresh Pin Boned Fillet (Lb)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Order # of UNITS:</td>
</tr>
<tr>
<td># of FISH CUT:</td>
</tr>
<tr>
<td>Fresh # of LBS LIVE-WT:</td>
</tr>
<tr>
<td>Fresh # of LBS CUT-WT:</td>
</tr>
<tr>
<td>COST PER SERVING RESTAURANT:</td>
</tr>
<tr>
<td>SMOKER # of LBS LIVE-WT:</td>
</tr>
<tr>
<td>SMOKER # of LBS CUT-WT:</td>
</tr>
</tbody>
</table>

[Buttons: Previous, Save, Finish]
Sample Excel Cutting Report

Upload Cutting Report Screen
Appendix D. Sample Smoke Assessment Screens

Search Smoke Assessments

View Smoke Assessment Search Results
## View/Modify Orders on Smoke Assessment

### Smoking Assessments

<table>
<thead>
<tr>
<th>Week</th>
<th>Add/Remove Orders</th>
<th>View Regular Excel Smoke Assessment</th>
<th>View Sockeyu Smoke Assessment</th>
<th>Upload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Wednesday, March 17, 2020</td>
<td>View and Modify</td>
<td>View</td>
<td>View</td>
<td>Upload Excel File</td>
</tr>
<tr>
<td>Wednesday, February 17, 2020</td>
<td>View and Modify</td>
<td>View</td>
<td>View</td>
<td>Upload Excel File</td>
</tr>
<tr>
<td>Wednesday, December 16, 2009</td>
<td>View and Modify</td>
<td>View</td>
<td>View</td>
<td>Upload Excel File</td>
</tr>
<tr>
<td>Wednesday, December 9, 2009</td>
<td>View and Modify</td>
<td>View</td>
<td>View</td>
<td>Upload Excel File</td>
</tr>
</tbody>
</table>

### Results

<table>
<thead>
<tr>
<th>Include</th>
<th>Fish Order Id</th>
<th>Client Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>✔️</td>
<td>3017</td>
<td>Just Local Foods</td>
</tr>
<tr>
<td></td>
<td>3016</td>
<td>Franzoi’s Cafe &amp; Catering</td>
</tr>
</tbody>
</table>
Sample Smoke Assessment

Upload Smoke Assessment Screen
Appendix E. Sample Administration Screens

Add Client Screen
## Add Contact Screen

**Contact Name:** Chef Mike  
**Same as address for:**  
**Address 1:** 201 60 County Hwy M  
**Address 2:**  
**City:** Cable  
**State:** Wisconsin  
**Zip Code:** 64621  
**Phone:** 715-794-2061  
**E-mail:** mjgbaloy@charneet.net  
**Websites:**  
- www.cablelodge.com  
- Recovery Pub & Cafe  
- www.roverpub.com

---

63
Add Client Product Screen

Client Id: CABLE NATURE
Add product to Select Product dropdown

Select Product: Fresh Hemingway Hemingway (Pounds)

Price per lb./item
Target Serving Price

Previous Add
Add Product Screen