WOSTMANN & ASSOCIATES, INC.

Interagency Electronic Reporting Technology Demonstrator Analysis and Assessment

December 2003

Submitted by

Wostmann & Associates, Inc.

Table of Contents

1 Introduction

In 1999, the Alaska Department of Fish and Game (ADF&G) and the National Marine Fisheries Service (NMFS) began meeting to address fishery data acquisition issues facing both agencies. The International Pacific Halibut Commission (IPHC) joined their initiative to make comprehensive groundfish data available to all management agencies, and to provide seafood processors with consistent and non-redundant means of reporting commercial harvests. The initial effort under this initiative coordinated the coding schemes for reports made on each of the agencies' systems. As the effort progressed, it became apparent that a single electronic reporting system for commercial catch data might be feasible. The Pacific States Marine Fisheries Commission (PSMFC) received a $750,000 grant to fund development of an interagency electronic reporting system.

In late 2001 PSMFC engaged Wostmann & Associates, Inc. (WAI), a Juneau based Information Technology consulting firm, to assess the needs of ADF&G, IPHC, NMFS, and the processors who make landing and production reports, with regard to electronic reporting. The needs assessment covered data requirements, technological capabilities, regulatory implications, and procedural challenges that might affect the success of an electronic reporting system. WAI concluded that an integrated electronic reporting system would be feasible, and could provide significant benefits to both processors and the fishery management agencies.

Among the recommendations given in the needs assessment report was a staged development approach, where limited development would take place and the results considered before proceeding to the next development stage. The first stage development recommendation was a technology demonstrator. The technology demonstrator effort would build a system using the technologies anticipated for use in the electronic reporting system. It would simulate the communications and processing needed for the real electronic reporting system, thus allowing evaluation of the system components, development environment, and communications infrastructure in Alaska under actual use without committing to a major development project.

When the NMFS halibut and sablefish Individual Fishing Quota (IFQ) card-swipe reporting system was initially deployed in the mid-nineties it experienced significant performance problems due to communications lag times and failures, even though testing in Juneau had indicated that the system's data communication architecture was reliable. An important objective of the technology demonstrator effort was to prevent a repeat of that experience by

testing the system communications architecture against the communications infrastructure in Alaska before the project commits to building the production software. Testing with the limited bandwidth and satellite communications used in remote Alaskan fishing ports would provide important feasibility information for project decision making. Additionally, the knowledge gained would help software developers anticipate conditions the system will encounter in the field, and allow better design decisions to be made.

In the summer of 2003, PSMFC engaged WAI to develop the technology demonstrator and to report on the performance of Internet communications and systems infrastructures at seafood processor and agency locations around the State of Alaska. WAI also evaluated the software development tools and software components used during the project. This report provides the performance analysis and the assessment of the technologies used. It also includes lessons learned that will be beneficial in the electronic landing system development effort.

1.1 Objectives

The primary objective of the technology demonstrator project was to evaluate, under actual field conditions, a system built with technologies that may be used for integrated interagency electronic reporting system. A major emphasis of this project was the evaluation of the communications infrastructure and computerized reporting capabilities of seafood processors in Alaska. To support the evaluation the technology demonstrator had to generate quantitative data on performance of communications and application software.

Another objective of this project was to evaluate the system design elements intended to allow multiple agencies to view and update reports, and to provide a complete audit trail of all changes.

Additionally, an important objective of this project was to engage seafood processors in the testing effort and to collect feedback from them on their perception of system performance and on their needs. The processor personnel who are responsible for making required reports are in the best position to provide information on what is workable.

1.2 Scope

The project tested the technology that could be used for the interagency electronic reporting system. Testing was in terms of both measured performance and perceived performance on the part of users at seafood

processors and agency field locations. The project also tested the use of the database lookups for providing validation of data values, and for storing report version histories to provide a complete audit trail of changes to the data. The project was not intended to produce a prototype, and did not test the usability of any user interfaces.

The technology demonstrator was focused on web-based Internet reporting methods, considered the quickest and lowest cost avenue to electronic landing reporting. In addition, testing web protocols test the same communications channels that would be needed for desktop software using web services. Web services are the most attractive technology option for system-to-system communications from distant sites. However, full testing of desktop software technologies was not within the scope of this project.

1.3 Terms and Definitions

The following table defines the terms, abbreviations, and acronyms used in this document.

Term	Definition
ADF&G	Alaska Department of Fish and Game
ASP	Application Service Provider - a vendor that provides servers in a managed datacenter that customers can use to run their web applications
ASP.Net	Active Server Pages .Net - a web application approach that allows scripting to be embedded in HTML files on Microsoft servers. When a web browser requests the file the server runs the ASP script code to process or display desired data
Bean	A Java software component written according to the Sun component specifications
CFEC	Commercial Fisheries Entry Commission - the State of Alaska agency responsible for licensing commercial fishermen and fishing vessels
CGI	Common Gateway Interface - a method of running applications on web servers. CGI is the oldest system used for web applications
CMP	Container Managed Persistence - a part of the J2EE specification that defines how servers can manage access to databases on behalf of application programs, relieving them of the details of connecting to databases and storing and retrieving data
Cold Fusion	A web application approach that allows scripts to be embedded in HTML files on servers. When a web browser requests the file the

Term	Definition
	Cold Fusion server runs the script to process or display data
Communications Latency	See Latency
CVS	Concurrent Version Control System - a source code version control system that is commonly used for Internet project. CVS is supported as an open source project.
DBA	Database Administrator – a technical support specialist who maintains the DBMS
DBMS	Database Management System - software that stores and retrieves data from system files, isolating the programmer from the technical file system details. DBMS software allows data to be related, and provides methods for finding and filtering desired data
EJB	Enterprise Java Bean - a framework for software components that run on J2EE application servers
Entity Bean	A Java software component that holds database data, and that can be stored and retrieved from a database
HTML	Hypertext Markup Language - the language used to define web pages on the Internet
HTTP	Hypertext Transmission Protocol - the system protocol used to send web pages and forms between servers and web browsers on the Internet. HTTP is also used as the underlying protocol for web services
IDE	Integrated Development Environment - a software development tool that allows programmers to edit, compile, configure, and test programs all within a single interface on their workstation
IFQ	Individual Fishing Quota
IO	Input Output operation - the low level operation a computer must execute to read or write data from disk storage
IPHC	International Pacific Halibut Commission
ISP	Internet Service Provider - a telecommunications vendor that provides network connections to the Internet
IT	Information Technology
ITG	Information Technology Group - the internal statewide IT services provider for the State of Alaska
J2EE	Java 2 Enterprise Edition - a system architecture built around the Java language. It is designed to provide common features needed for applications distributed across networks and organizations

Term	Definition
Java	A leading object oriented computing language created and marketed by Sun Microsystems. Java is notable because it is designed to be platform independent, allowing programs written in Java to run on most types of computers without modification
JavaScript	A scripting language that allows program logic to be embedded in HTML pages. Web browsers such as Netscape and Internet Explorer interpret the embedded JavaScript and execute its commands.
JBoss	A leading J2EE application server. JBoss is a highly capable EJB container, and is supported as an open source project
JSP	Java Server Pages - a web application approach that allows Java programs to be embedded in HTML files on Java servers. When a web browser requests the file the server runs the Java code to process or display desired data
Latency	Lag time observed when sending electronic communications. The time to send electronic data across a network.
MB	Megabyte(s)
MS	Microsoft
MTBF	Mean Time Before Failure - the average time a system can run before a problem or fault is experienced
MTTR	Mean Time To Repair - the average time needed to correct a problem on a system
.Net	A systems architecture designed to provide common features and services needed for applications distributed across servers and organizations. .Net is marketed by Microsoft, and is generally limited to use on servers running Microsoft operating systems
Network Latency	See Latency
NMFS	National Marine Fisheries Service
Open Source	A licensing arrangement that avoids proprietary control of software source code. Open Source software is provided with source code, and allows the source code to be modified by organizations or individuals that use it
PDF	Portable Document Format - a file format created by Adobe Corporation that allows documents to be rendered and printed on many different computer systems and printers
PHP	PHP Hypertext Preprocessor - - a web application approach that allows script programs to be embedded in HTML files on servers. When a web browser requests the file the server runs the PHP script code to process or display desired data

Term	Definition
PSMFC	Pacific States Marine Fisheries Commission
Servlet	A Java web application that runs on a web application server. Servlets create their web pages programmatically, rather than processing script embedded in HTML files
Session Bean	A Java software component that handles business logic or other computations
SQL	Structured Query Language - the most common language for querying and manipulating data on DBMS databases
SSL	Secure Sockets Layer - a standardized method of encrypting communications between servers and web browsers on the Internet
WAI	Wostmann & Associates, Inc.
Web Service	A standardized method that computers can use to communicate with servers on the Internet
Xdoclet	A software utility that allows deployment data needed to configure components on a J2EE server to be embedded in the Java source code files, simplifying maintenance

2 Methodology

WAI followed a number of steps to build the technology demonstrator system, conduct the testing, and analyze the results. We identified the performance parameters that could be tested and that would provide insight to the aspects of the system that were of greatest interest. Two primary performance criteria were identified: the observed response time for web page transactions from the user's perspective, and the server processing time for those transactions. The response time was chosen because it is what the end user sees. Users would naturally compare their impression of the response time of the technology demonstrator to that of other web-based applications. Two components contribute to response time: the processing time for transactions on the server and the communications time for data to cross the Internet from the server to the web browser. The communications time lag is known as network latency, and is outside the agencies' control. The server processing time represents the time required for processing the data on the server, and storing it in the database. To some extent, the server processing performance can be improved by programming effort or procurement of faster server hardware.

We designed a web-based user interface that would allow users to interact with the system and would provide consistent data volumes that could be measured to acquire performance statistics. We designed an interface that minimized data input by the users, both to reduce variability in the tests due to user actions, and to reduce the effort users would have to put forth to assist us in the testing effort. The interagency steering team reviewed the interface, and their suggestions were incorporated into its final form.

We chose a systems architecture based on our professional experience working with government agencies in the Alaska region. The system included web application server software that we had previously used for the IFQ landing reporting system at NMFS and the ADF&G Southeast Region News Release system. The database management system (DBMS) selected for the project was Oracle, which is in use at both ADF&G and NMFS. The application server selected was JBoss, which is used by the State of Alaska Information Technology Group, whose data center was selected to host the test system.

WAI constructed the technology demonstrator software to implement the user interface as designed. The software included both client and server side performance measurement instrumentation, to collect actual performance data, store it in the database, and make it available for display.

We worked with the interagency steering team to identify agency and seafood processor personnel to act as system testers. We solicited personnel from agency

field offices around Alaska, and included testers in Seattle, to provide a broad basis for testing the communications infrastructure.

We conducted the system tests over a four-week period in September and October of 2003. Test time had to be limited because the testers were all volunteers with other job responsibilities. The initial testing was conducted with users at ADF&G and NMFS in Juneau, and IPHC in Seattle. The testing exercised the system and revealed bugs and procedural problems that were corrected before releasing the system into wider test. The initial testing also established baseline data for users with what is considered good communications infrastructure in major cities. The second phase of testing brought on additional agency users in remote locations. These users helped validate the changes made after initial testing, identified a small number of additional changes that were needed, and generated performance data for remote locations that have access to the State and Federal government IT infrastructure. The third phase of testing included users at selected processors around Alaska. The testing generated performance data for a significant number of locations that would be expected to use the electronic landing reporting system. The third phase of testing lasted for two weeks.

During testing we tracked problems and anomalies reported by users. We analyzed system logs to identify occurrences of problems that might not have been reported. This analysis also identified the occurrence and results of unauthorized access attempts from the Internet.

At the conclusion of testing, we solicited feedback from all test users. We focused our questions on the apparent performance of the system, whether its speed would be adequate for a landing reporting system, on the problems observed, and on user satisfaction with a web based system. We also inquired about the general availability of the system when tests were attempted.

Following testing we conducted a statistical analysis of the performance data collected during testing. This analysis focused on the transaction times observed, broken down by server processing time and observed client response time. The server processing time represented the time it took to process transactions on the server, and that would be susceptible to performance tuning on the server and the database if it was excessive. The observed client response time included that portion of the total response time that was attributable to communications network latency and user workstation processing time. The times were averaged by user, and variability was noted in order to determine both the anticipated and worst case performance that would be expected in a production landing reporting system.

Finally, we correlated user feedback with quantitative measurements to determine how user perceptions related to actual performance. The feedback

was combined with quantitative data to draw conclusions based on both user perception and actual performance for the system and infrastructure. This correlation provided a determination of the levels of performance at which users generally would find the system adequate or inadequate. The conclusions were used to make recommendations for proceeding with the project, and recommendations about changes to the technologies employed.

3 System Description

The technology demonstrator system was intended to acquire quantitative data on the performance of the Alaska Internet communications infrastructure, the extent to which end user browsers support web application features that would be useful in an electronic reporting system, and the performance of the particular server software components and architecture chosen for the project. The following sections provide descriptions of the major components of the system and note design elements that influence the performance observed.

3.1 Web Application Server

The web application was the primary user interface of the Technology Demonstrator system. The web application was developed in the Java language using the Enhydra web application framework. It connected to the application server using the Java 2 Enterprise Edition (J2EE) EJB protocol. The web application provided the focus for infrastructure performance testing since it explicitly measured both the application server response times and the Internet communications latency times.

The Technology Demonstrator web application provided a login page to allow users to authenticate to the system, and user maintenance pages to allow users to change their personal information such as the spelling of their names, their phone number, their email contact information, their login id, and their password. This feature proved useful when users were added with misspelled names and user login ids. The actual system userid was a numeric value that never changed. The technology demonstrator test pages provided a sequence of four pages that simulated the input and submittal of an electronic report.

A menu page allowed the user to select the size of report to submit. The menu provided two size options, a 12 line and a 50 line report.

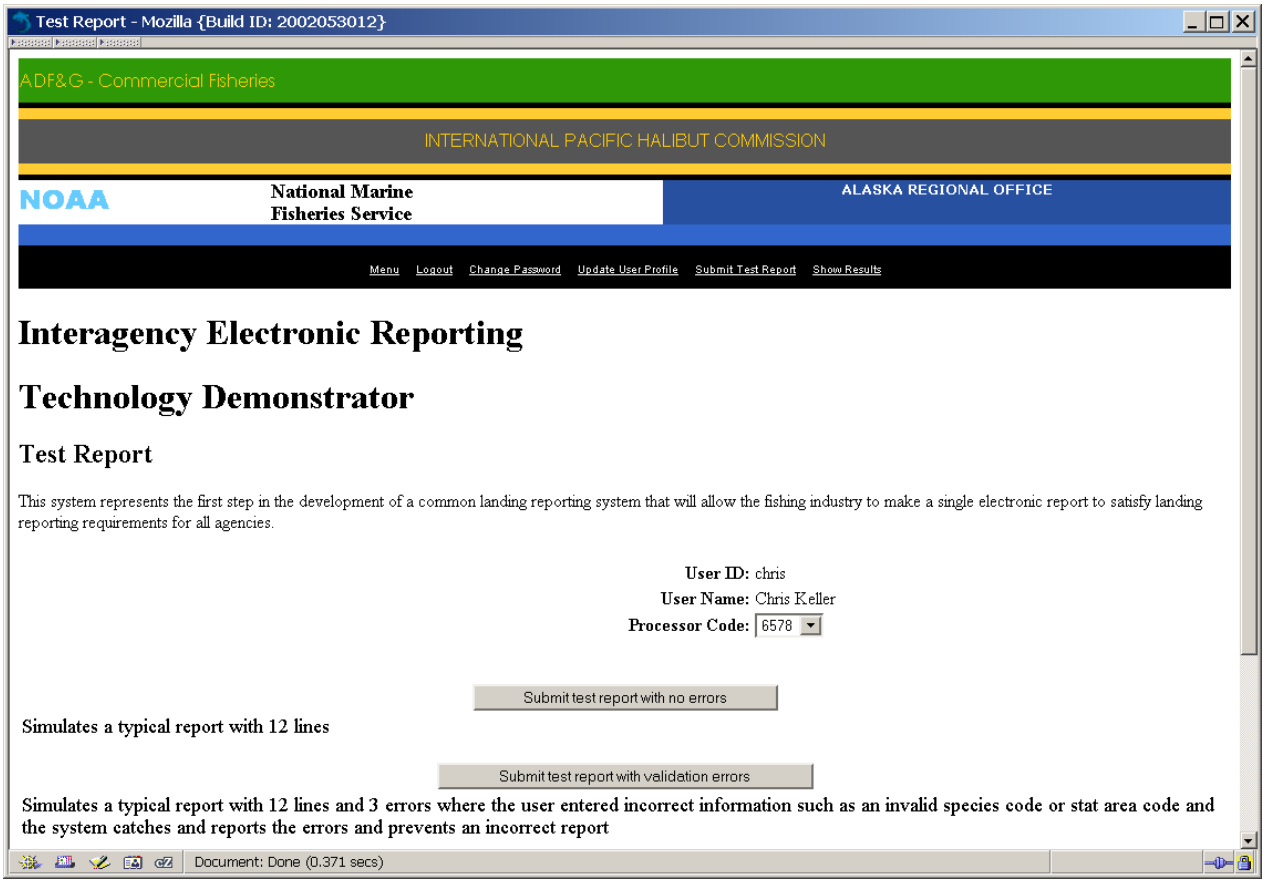


Figure 1

The test report page illustrated in Figure 1 allowed the user to submit a report with or without validation errors. Submitting a report with validation errors simulated the situation where the user entered the vessel ADF&G number, species code, or stat area incorrectly. Submitting a report with no errors simulated a report with valid data. In either case, the page actually submitted data for the number of lines specified and the server processed the data, validating the input against lookup tables in the database.

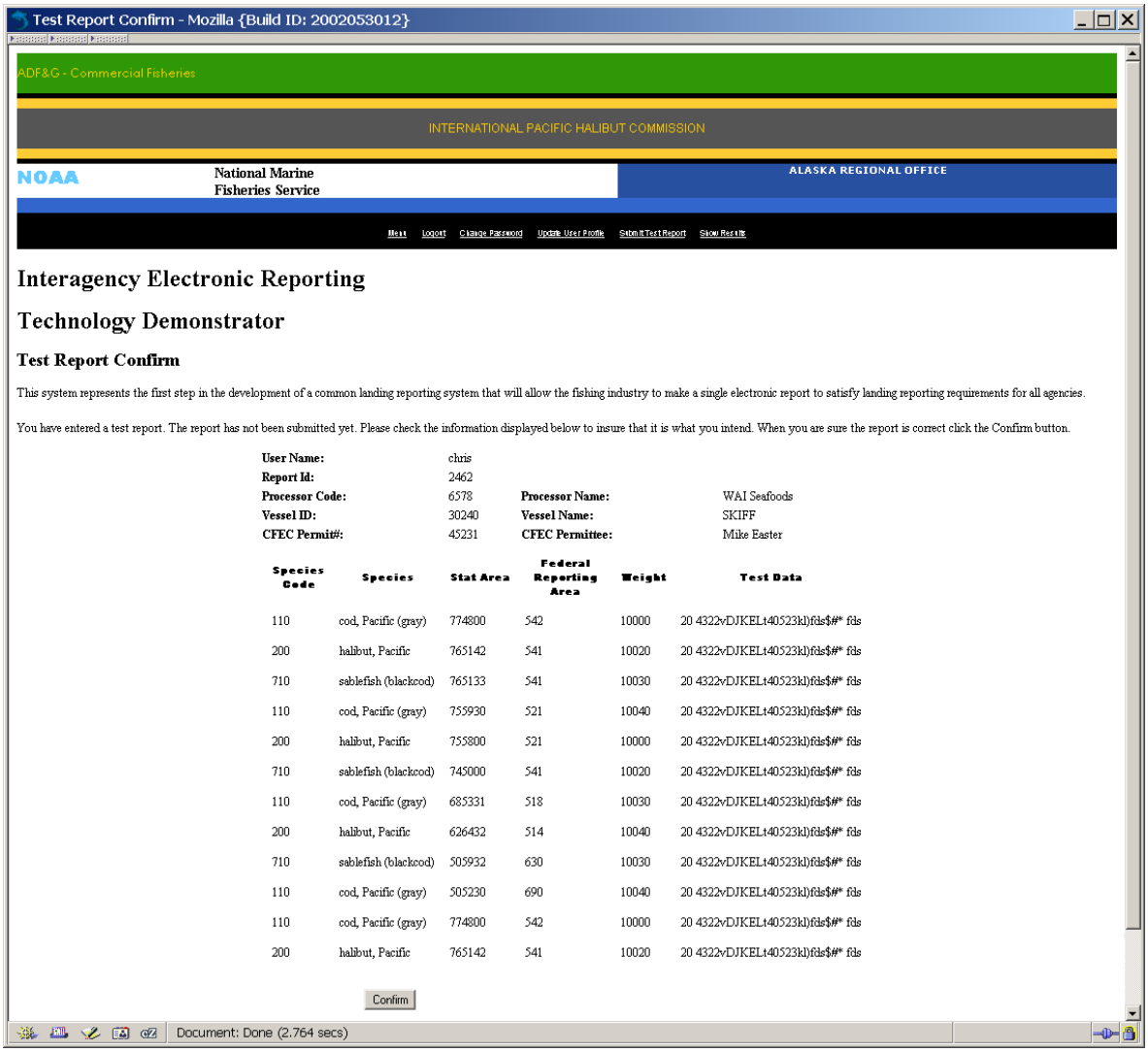


Figure 2

The test report confirmation page shown in Figure 2 displayed the submitted data, allowing the users to proofread their report before final submission. Existing web-based electronic reporting systems use confirmation pages to promote data quality. The confirm page showed the database lookup values for data such as species code, stat area, and vessel ADF&G number, which provided translations for the species name, federal reporting area, and vessel name. The test data column displayed additional data that was added to increase the size of the data streams to simulate reports with greater data amounts per line.

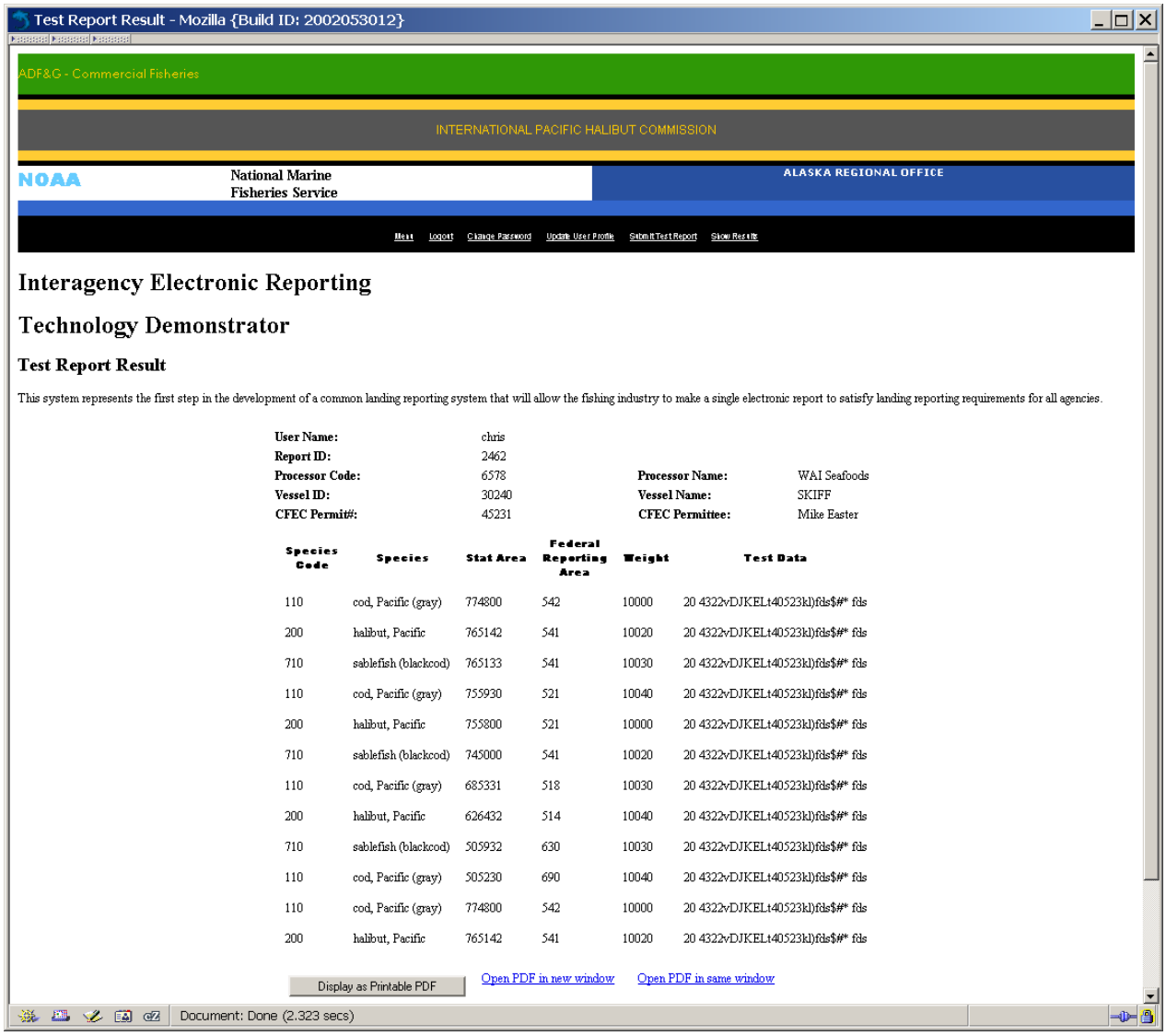
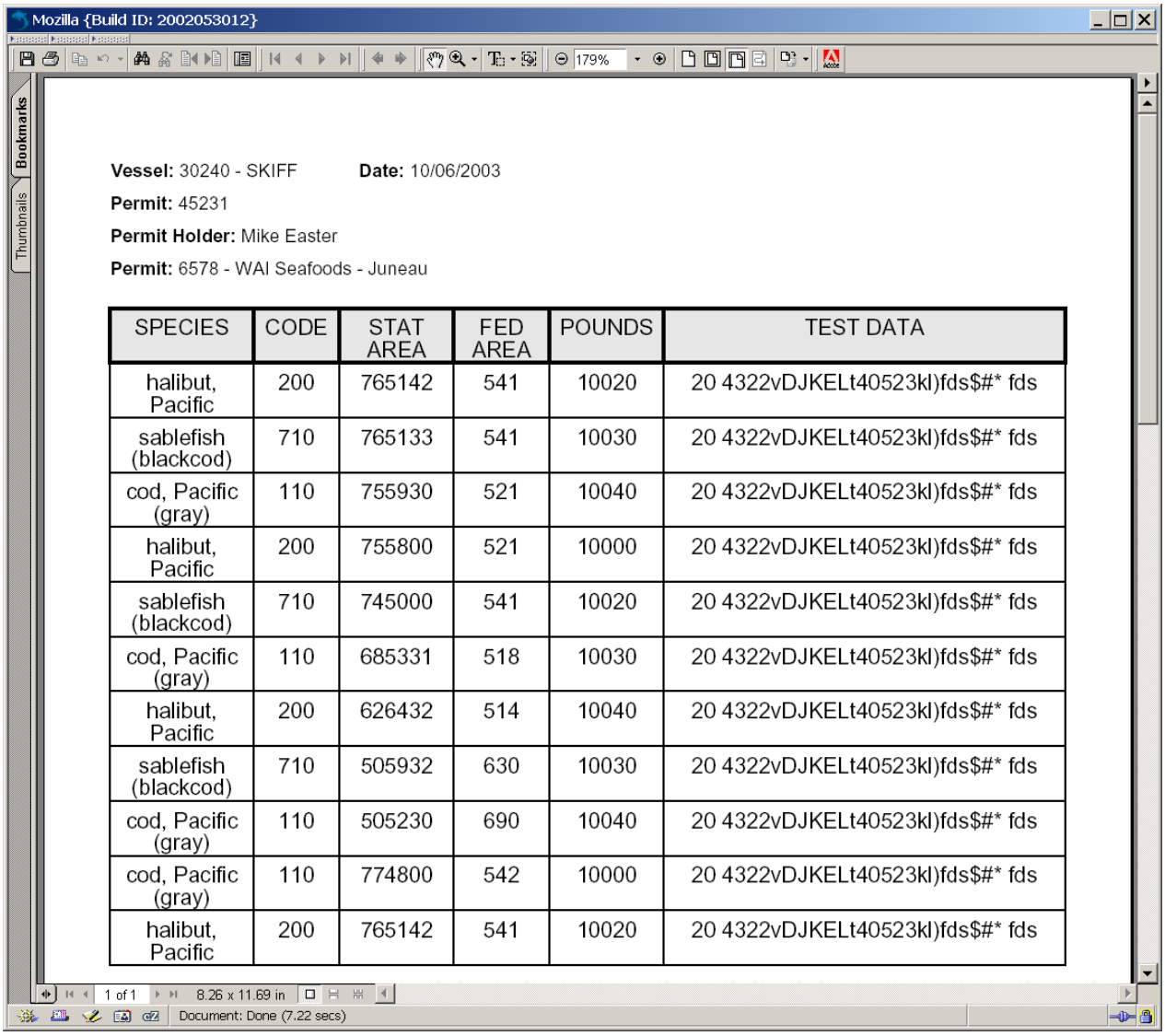


Figure 3

The Result page displayed a summary of the submitted report, similar to the confirm page. As Figure 3 shows, it also had links to allow the user to download the report as a PDF file suitable for printing. The initial version of the Result page had only a button for displaying the PDF file. During testing it was determined that some browser configurations would not display the PDF file when it was requested with the button. Hyperlinks for opening the PDF file in a new window and opening the PDF file in the same window were added to test alternate methods of requesting the PDF files. Figure 4 shows the PDF file as it displayed on the browsers.



SPECIES	CODE	STAT AREA	FED AREA	POUNDS	TEST DATA
halibut, Pacific	200	765142	541	10020	20 4322vDJKELt40523kl)fds$#* fds
sablefish (blackcod)	710	765133	541	10030	20 4322vDJKELt40523kl)fds$#* fds
cod, Pacific (gray)	110	755930	521	10040	20 4322vDJKELt40523kl)fds$#* fds
halibut, Pacific	200	755800	521	10000	20 4322vDJKELt40523kl)fds$#* fds
sablefish (blackcod)	710	745000	541	10020	20 4322vDJKELt40523kl)fds$#* fds
cod, Pacific (gray)	110	685331	518	10030	20 4322vDJKELt40523kl)fds$#* fds
halibut, Pacific	200	626432	514	10040	20 4322vDJKELt40523kl)fds$#* fds
sablefish (blackcod)	710	505932	630	10030	20 4322vDJKELt40523kl)fds$#* fds
cod, Pacific (gray)	110	505230	690	10040	20 4322vDJKELt40523kl)fds$#* fds
cod, Pacific (gray)	110	774800	542	10000	20 4322vDJKELt40523kl)fds$#* fds
halibut, Pacific	200	765142	541	10020	20 4322vDJKELt40523kl)fds$#* fds

Figure 4

The PDF file presented the data in a printable form, including the vessel id and name, fisherman's CFEC permit number and name, processor permit number and name, and line items. An extra test data field was included on all simulated reports to insure that the maximum expected data amount per line was processed.

3.2 Performance Instrumentation

In order to measure the actual performance of the web application, from the client workstation point of view, the web pages included client side scripting, written in JavaScript, to record and report transaction times. The JavaScript recorded the date and time from the client workstation's clock and passed

that information to the server when the user submitted the page for processing. The response page the server sent in reply included JavaScript to automatically send another message to the server when the page was loaded into the browser. The data in the message included the date and time from the workstation's clock, allowing the server to determine the response time of the original submission by subtracting the client time when the page was submitted from the client time when the response was loaded.

3.3 Application Server

The application server provided the database access, business rule logic, and security authentication and authorization services for the Technology Demonstrator system. The application server defined the functions of the system as they related to retrieving, storing, and updating report data. It provided functions to retrieve reports and line items, create reports, update report data, add line items to reports, change line item data, get reports as PDF documents, and save changes to the database. The application server required user authentication before retrieving or updating any data. The application server stored user information and a timestamp along with all changes to reports, to provide an audit trail.

In addition to functions for receiving and processing test reports, the application server also provided web service methods for user management. Web service interfaces are suitable for applications distributed across the Internet, and use the same HTTP protocol as the web application server used to communicate with web browsers.

The application server was implemented in the Java language as an Enterprise Java Bean application. It ran on the JBoss application server.

3.4 Database and Report Version Tracking

The database used on this project was stored on an Oracle DBMS. The database consisted of 20 tables. These included 10 lookup tables for coded data, both internal codes for data such as agency, client type, role, status, and transaction type, as well as user lookup data such as CFEC permit, species, statistical area, and vessel. The lookup tables also included a sequence block table that the application server used to store and maintain sequence numbers for assigning new keys.

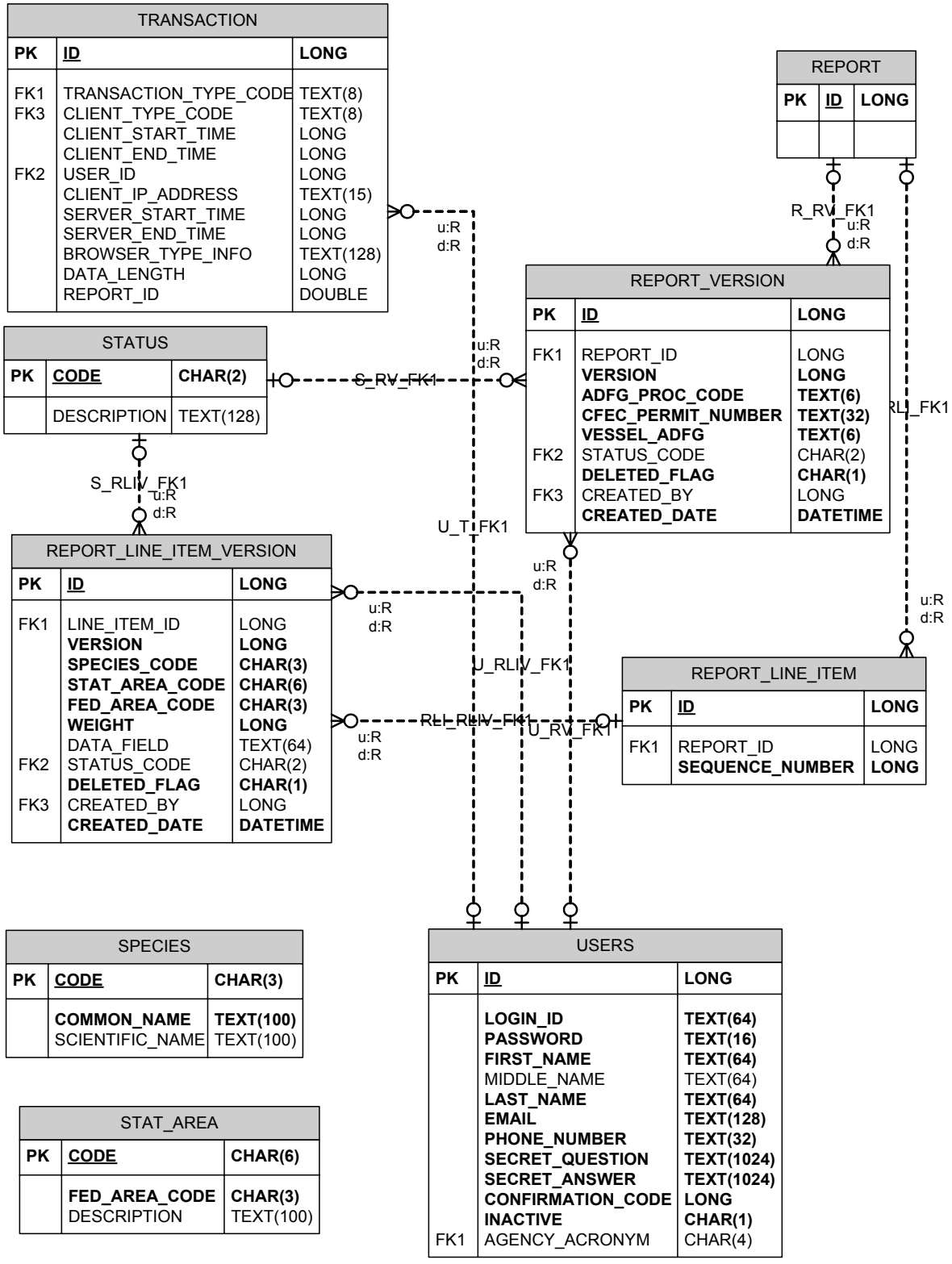


Figure 5

The database consisted of 5 tables for user management, two tables to store processors and processor locations, a users table, and two tables that

associated users with their roles and processor locations. The database had one transaction table to hold the transaction instrumentation data generated by the application.

The most notable database tables were those used to store report data. The database tables for the test reports where designed to provide full data version tracking. As shown in Figure 5, the report table contained only the report id number. All report header data was contained in the report version table. Updates to report header data did not update rows in this table. Instead, a new row, with an incremented version number, was created for each update. In this manner, older versions were never lost, but their data could be hidden from most users by displaying only the most recent version of a report. The report version has a created-by userid that associated the version data to the user who created it, regardless of whether they created a new report or updated an existing one. The report line item data was similarly stored in two tables. One was a line item table that recorded the existence of the line item, and provided its position in the sequence of line items. The other was a line item version table that stored all the changes ever made to each particular line item.

The version based table structure allows data views such as Figure 6 showing a report that was submitted on 10/8 at 10 minutes after 9. An ADF&G employee updated the vessel number at 1 PM on the same day. It was changed again by IPHC on the following day. The version number shows the sequence of updates. The replaced data is highlighted in orange, the current value in blue.

Rpt	Ver	Proc Code	CFEC Permit	ADF&G Number	Vessel Name	Update Date	User ID	Agency
2780	1	F0321	45231	30240	SKIFF	10/8/2003 9:10	klagasse	
2780	2	F0321	45231	346	ARTIC LOON	10/8/2003 13:02	mlambdin	ADFG
2780	3	F0321	45231	5106	SUSAN JOY	10/9/2003 10:49	lhutton	IPHC

Figure 6

Several line items on report number 2780 were also updated. The Seq number identifies each different line item in sequence, and the Ver number identifies the progression of updates within each line item. The species code was updated on line item sequences 2 and 4, the Stat Area was changed on line item sequences 3 and 5, and the weight was changed on line item sequences 6 and 8. Line items sequences 1, 7, 9, 10, 11, and 12 were not changed.

Rpt	Seq	Ver	Species	Species Name	Stat Area	Fed Area	Weight	Deleted	Update Date	User ID	Agency
2780	1	1	200	halibut, Pacific	765142	541	10020	N	10/8/2003 9:10	klagasse	
2780	2	1	710	sablefish (blackcod)	765133	541	10030	N	10/8/2003 9:10	klagasse	
2780	2	2	124	sole, dover	765133	541	10030	N	10/8/2003 13:02	mlambdin	ADFG
2780	2	3	212	hagfish, Pacific	765133	541	10030	N	10/9/2003 10:49	lhutton	IPHC
2780	3	1	110	cod, Pacific (gray)	755930	521	10040	N	10/8/2003 9:10	klagasse	
2780	3	2	110	cod, Pacific (gray)	305501	659	10040	N	10/8/2003 13:02	mlambdin	ADFG
2780	3	3	110	cod, Pacific (gray)	625404	610	10040	N	10/9/2003 10:49	lhutton	IPHC
2780	4	1	200	halibut, Pacific	755800	521	10000	N	10/8/2003 9:10	klagasse	
2780	4	2	250	tomcod, Pacific	755800	521	10000	N	10/8/2003 13:02	mlambdin	ADFG
2780	4	3	209	bristlemouth, lightfish, anglemouth	755800	521	10000	N	10/9/2003 10:49	lhutton	IPHC
2780	5	1	710	sablefish (blackcod)	745000	541	10020	N	10/8/2003 9:10	klagasse	
2780	5	2	710	sablefish (blackcod)	315330	690	10020	N	10/8/2003 13:02	mlambdin	ADFG
2780	5	3	710	sablefish (blackcod)	635100	610	10020	N	10/9/2003 10:49	lhutton	IPHC
2780	6	1	110	cod, Pacific (gray)	685331	518	10030	N	10/8/2003 9:10	klagasse	
2780	6	2	110	cod, Pacific (gray)	685331	518	18002	N	10/8/2003 13:02	mlambdin	ADFG
2780	6	3	110	cod, Pacific (gray)	685331	518	16999	N	10/9/2003 10:49	lhutton	IPHC
2780	7	1	200	halibut, Pacific	626432	514	10040	N	10/8/2003 9:10	klagasse	
2780	8	1	710	sablefish (blackcod)	505932	630	10030	N	10/8/2003 9:10	klagasse	
2780	8	2	710	sablefish (blackcod)	505932	630	9901	N	10/8/2003 13:02	mlambdin	ADFG
2780	8	3	710	sablefish (blackcod)	505932	630	8261	N	10/9/2003 10:49	lhutton	IPHC
2780	9	1	110	cod, Pacific (gray)	505230	690	10040	N	10/8/2003 9:10	klagasse	
2780	10	1	110	cod, Pacific (gray)	774800	542	10000	N	10/8/2003 9:10	klagasse	
2780	11	1	200	halibut, Pacific	765142	541	10020	N	10/8/2003 9:10	klagasse	
2780	12	1	110	cod, Pacific (gray)	774800	542	10000	N	10/8/2003 9:10	klagasse	

Figure 7

3.5 User Management Desktop Client

The user management desktop application allowed agency administrators to add and change processor and agency user setups. Deletes were not supported, since users are associated with reports, and must continue to exist in the database even if they are not longer able to enter new reports. All user identification data could be modified from the desktop client, including changing of spellings of names and userids. Users could also have their access disabled by setting a password that could not be entered on the web application.

The user management desktop application was developed in Microsoft Visual Basic.Net, and connected to the application server by calling its web services.

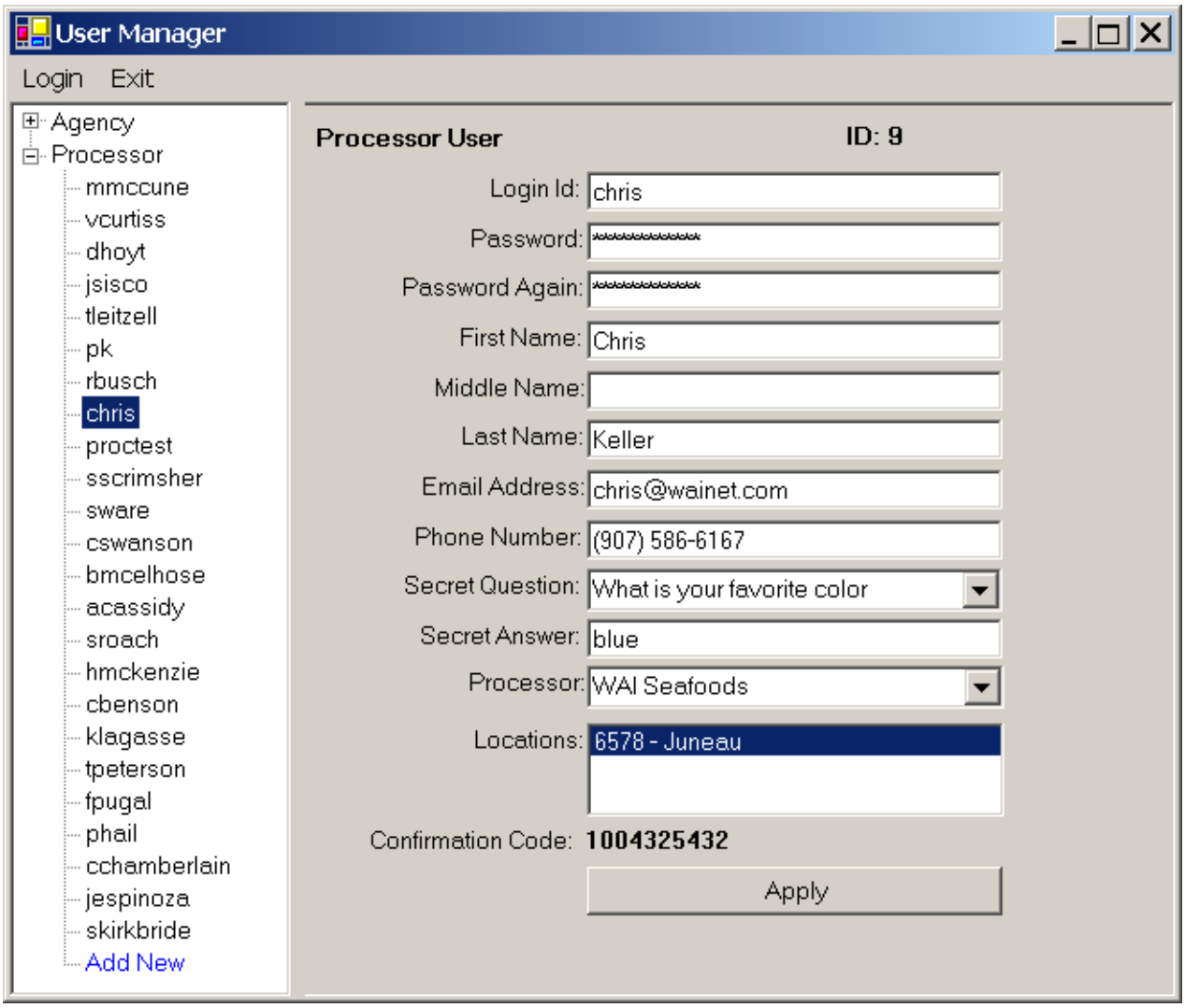


Figure 8

3.6 Security

The technology demonstrator system incorporated a number of security and access control features.

- Access to the database was controlled by a database userid and password. This userid had update privileges. The database userid and password was known only to system administrators and developers. If other users needed direct access to the database, they could have been given a database userid with read-only access. The database userid and password were embedded in the single deployment file that was loaded on the application server, which only system administrators could access.

- Application server client programs, whether the web client or other client software, had to use a system userid and password to access the application server. The system userid and password were embedded in the single deployment file that was loaded on the application server. The system userid and password were provided only to application developers who developed client applications. Additionally, the application server required that its client programs provide a valid end userid and password before it would retrieve or update data. The application server end user authentication provided the authentication services to client programs such as the web application.
- To connect to the server, web service client programs, such as the user management desktop application, required a userid and password. The web services checked both the authentication of the user via the password, and the user's role, which was setup in the database. For example, the user management desktop application used a web service that required a user with the administrator role.
- To use the web application to submit test reports the test users first had to login. This login provided application level user authentication, checking the provided userid and password against the user information setup in the database. Users also had to be setup for the data entry role, the lowest level of authorization in the system. The application associated the authenticated userid with all reports the user added or changed in the database
- The web application was tested with the industry standard Secure Sockets Layer (SSL), which encrypts the data stream being passed between the web browser on an end user's workstation and the web application server that processed the data. The SSL encryption protected the users' data from network traffic monitoring programs.

3.7 Test Data

The database was loaded with a test data set to insure consistency between tests, and to simulate the records that would be present in the production database for validation lookup tables.

3.7.1 Validation Test Data

The test system had four tables used to lookup and validate input data. The Table 1 lists the test data types and the numbers of records.

Data Type	Number of records
CFEC Permit	65531
Species code	295
ADF&G Statistical Area	3111
Vessel	36436

Table 1

The validation test data was extracted from the actual ADF&G production database. The full data sets were used, except in cases where ADF&G maintains multiple records for the same value, such as multiple years for the same CFEC permit.

3.7.2 User Test Data

Agency personnel could add user account records for actual users via the User management interface, but the processor and processor location data required to add a user were loaded directly into the database. This approach simulated the situation where an interface to the existing agency systems would provide the processor data.

3.7.3 Report Test Data

The report test data was hard–coded into the web application client. Two sets of report data were provided: one for a 12 line report and one for a 50 line report. Each set also had a version with validation errors, which was used to test the performance of report validation lookups. Hard-coding the report test data provided two advantages to the test system. It reduced the data entry burden on users for submitting tests, allowing them to submit more tests in the testing period for a given level of effort. Consistent data in the reports also eliminated a source of variability in performance data, providing a better sense of the effect of external factors on server processing and Internet communications times.

4 Performance Analysis

The technology demonstrator system captured performance data for a number of transaction types. The nature of certain failures revealed the performance and reliability of some aspects of the technologies employed. Analysis of the system logs showed the incidence of some system conditions and external factors such as malicious access attempts and their results. A survey of users produced feedback on the perceived performance of the system. From these data sources, we were able to determine how the system performed and the potential viability of Internet submission of landing reports.

4.1 Transaction Response Times

Performance response times were captured for four transaction types. Each transaction type had different characteristic amounts of data sent to and from the client browser and different server processing demands. Each transaction type could be run with 12 lines or 50 lines of data, simulating short and long reports. Within a transaction type and number of lines the controllable factors such as actual data stream length and number of validation errors introduced were similar for all users. The key variable measured was system response time at the web browser. The response time showed the capability of the Internet infrastructure in various locations. It is composed of server response time and network latency. The server response time was also measured directly. It is the time to process data on the server once a request is received from a web browser. Server processing includes data validation checks and database updates. Differences in server response times were expected due to varying server loads caused by activity on the database server, internal operating system processes, and workloads on the technology demonstrator application itself. Network latency is the delay in response time caused by the time required to transmit data packets across the communications network. Differences in network latency times were expected due to the use of satellite uplinks, differing ISP service levels, and the use of less capable networking equipment in some locations. The following charts show the average total response times at the web browser, with both the server and network performance components. In addition, the charts plot the standard deviation of the average total response time for each location.

The 12 and 50 line reports without validation errors tested the receipt and validation of report data. Both tests processed the same header data, validating the CFEC permit number and vessel ADF&G number by looking them up in the database. All the differences in the processing required for the

12 line and 50 line reports can be attributed to the additional 38 lines. The difference affected both server processing time and network communications time. The 12 line reports passed 1280 bytes of data across the network for input to the server application, while the 50 line reports passed 4967 bytes.

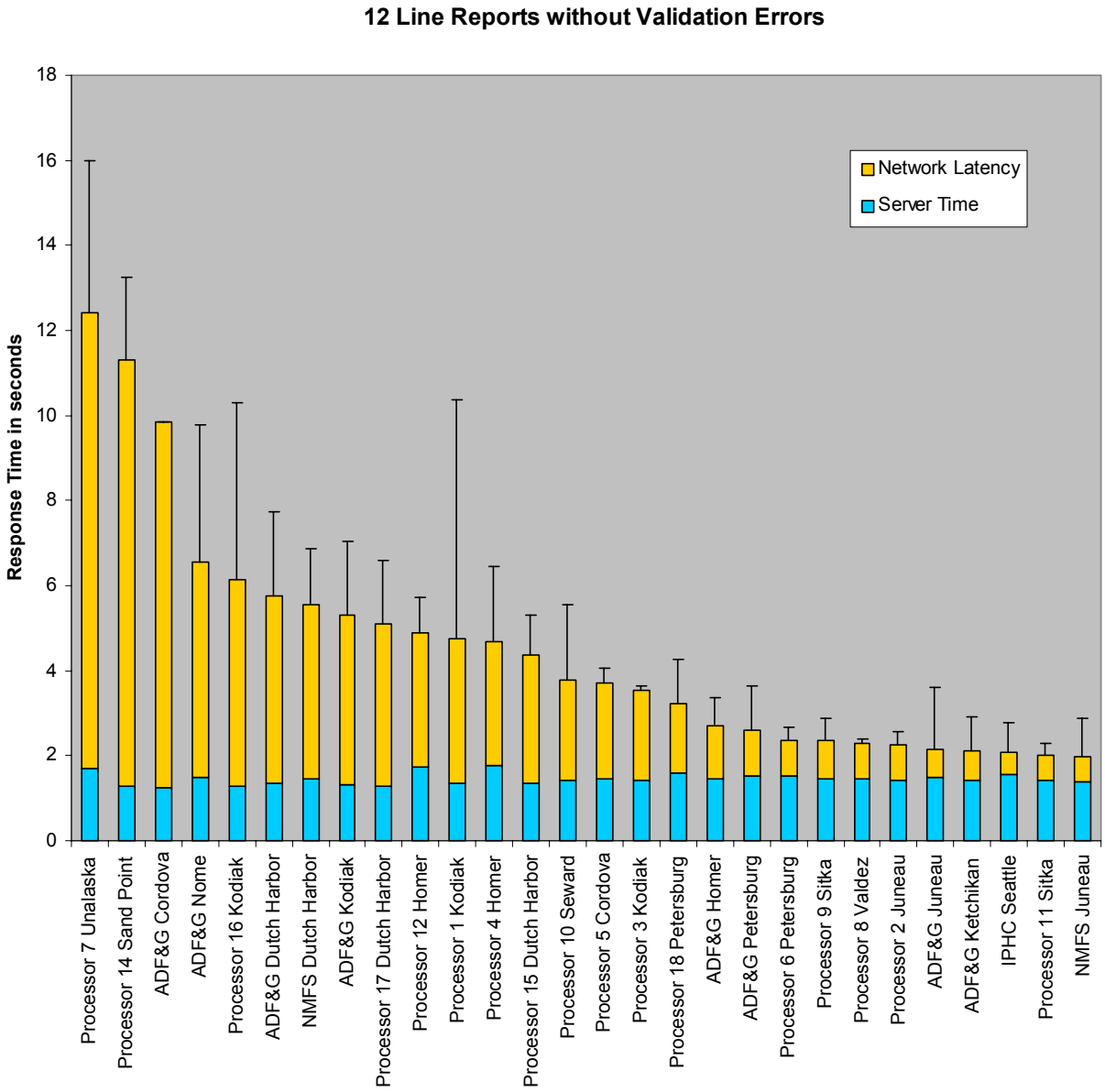


Figure 9

Figure 9 shows the response time averages for 12 line reports. Server processing times were uniformly a little under 2 seconds. Response time at the client browser was under 6 seconds for most locations; although, the worst

location was double that amount. The data for two of the Kodiak processors showed excessive variability, which could indicate that Internet communications to Kodiak experience intermittent slow-downs.

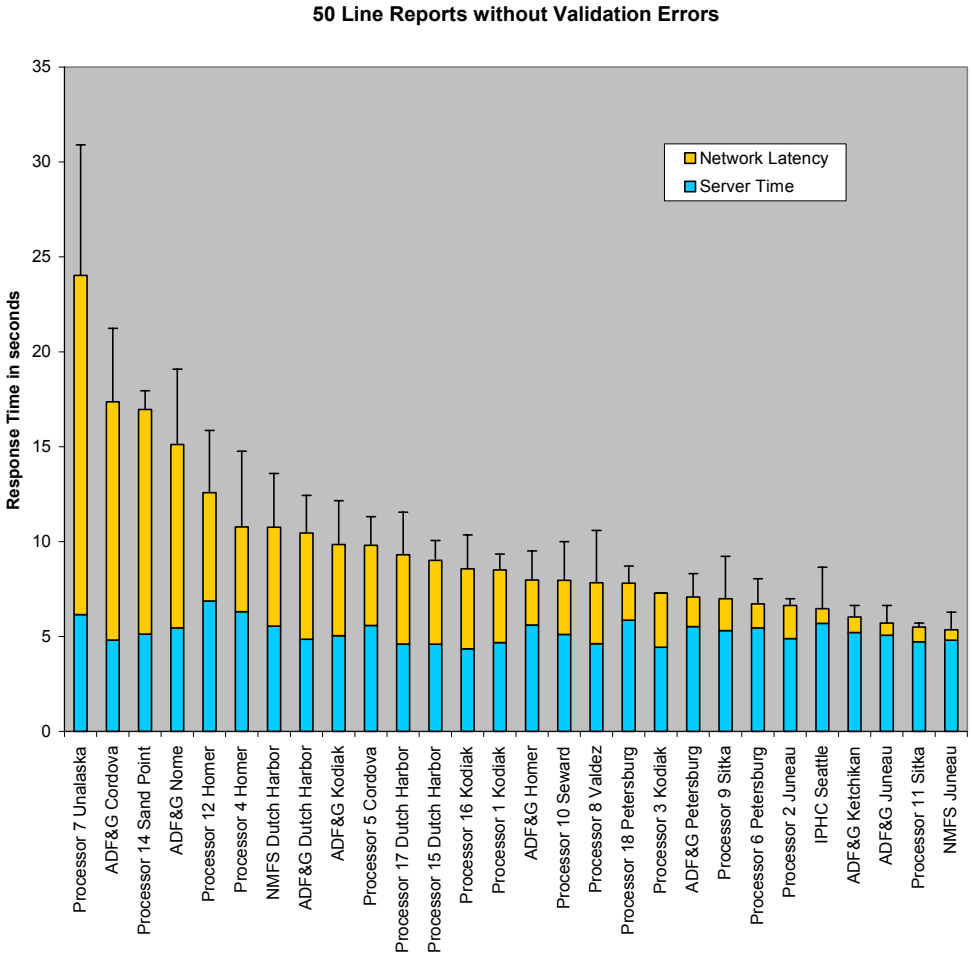


Figure 10

Figure 10 shows the response times for 50 line reports. As expected, the server processing time increased, from under 2 seconds for the 12 line reports to more than 5 seconds for the 50 line reports. Communications times also increased, but not proportionally. Most locations had average response times less than 12 seconds. Variability was fairly consistent, and the Kodiak locations did not exhibit excessive variability for these transactions.

The 12 and 50 line reports with validation errors tested the receipt and validation of report data containing validation errors. To insure consistent processing the system induced the same errors on all reports. It injected 3 validation errors on the 12 line reports and 10 validation errors on the 50 line reports. The system redisplayed the report submission page with error

messages after errors were detected. Some processors did not test reports with errors, so those processors are not shown on the charts.

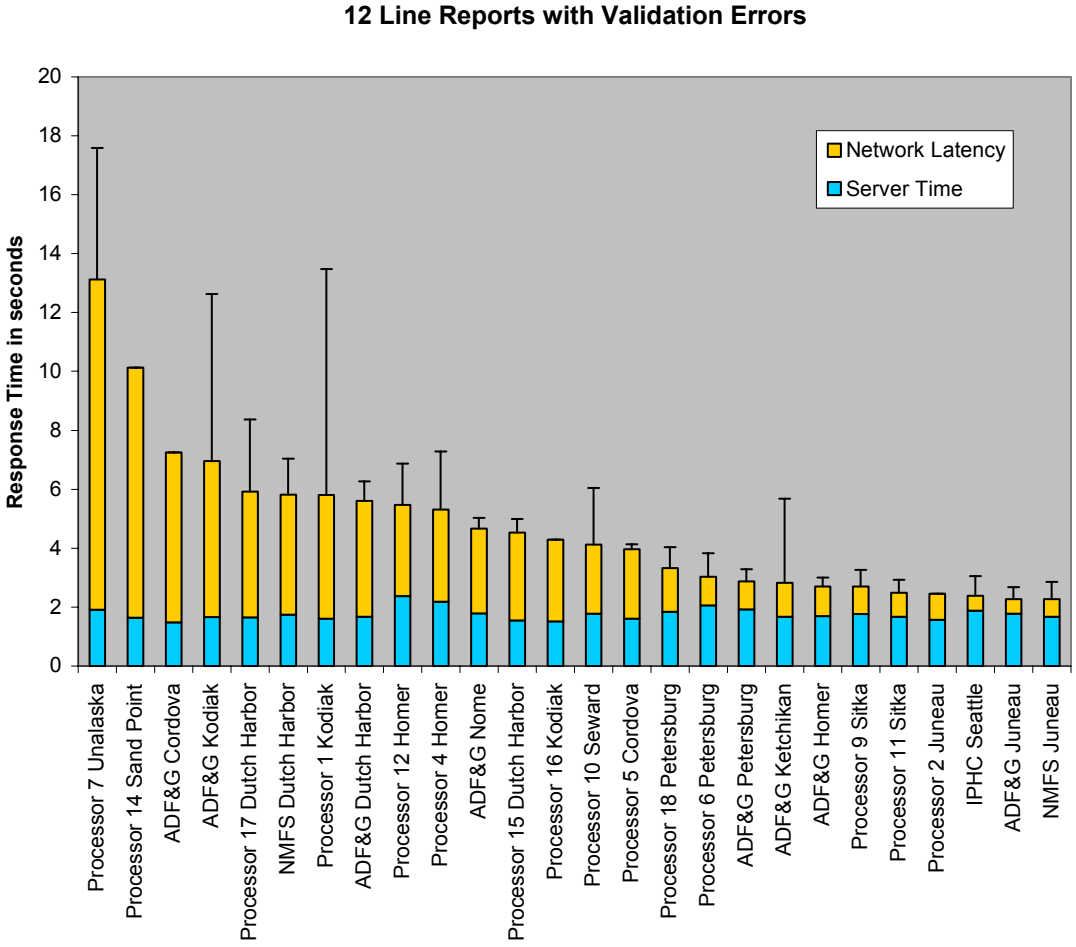


Figure 11

Figure 11 shows the response time averages for 12 line reports with errors. Server processing times were slightly longer than observed for 12 line reports without errors, but were fairly uniform and were still around 2 seconds. Response time at the client browser was still under 6 seconds for most locations. The two worst performing locations, in Unalaska and Sand Point, were the same as for the 12 line reports without errors. They had significantly worse performance than other locations, due entirely to network latency. Two of the Kodiak processors again showed excessive variability, which was attributable to network latency. The standard deviation of their server processing was similar to other processors, less than 1 second.

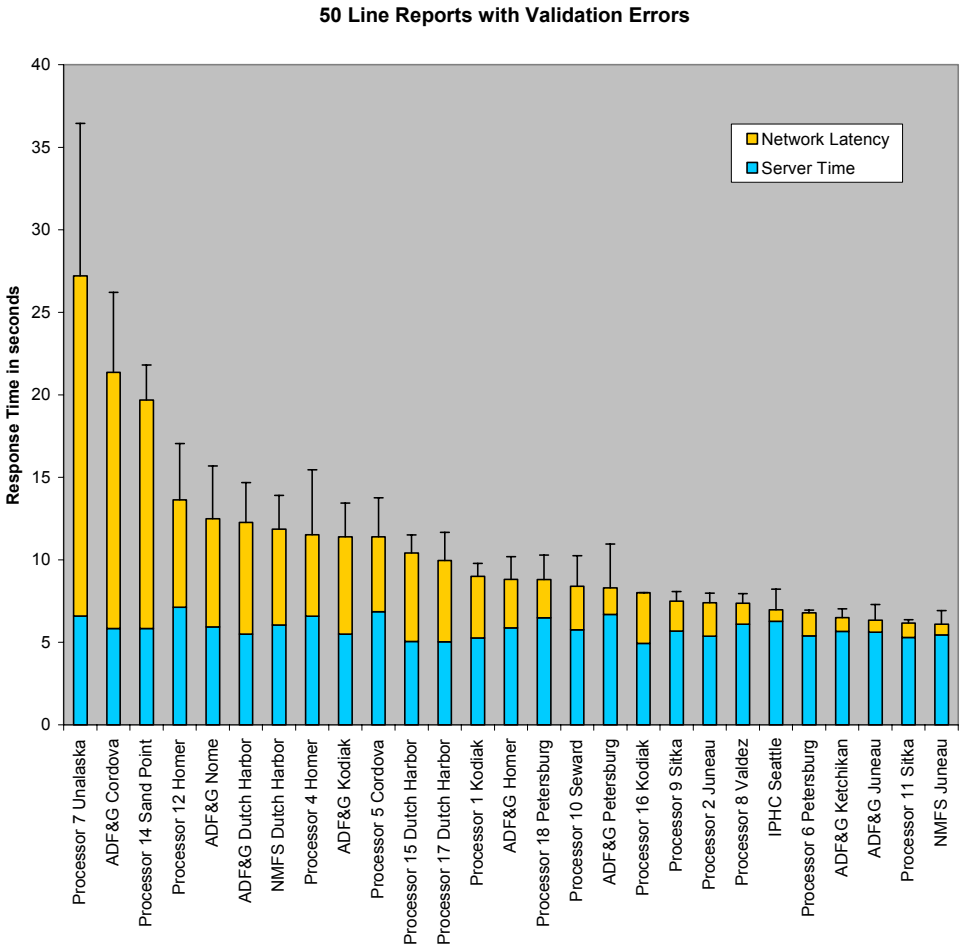


Figure 12

The performance observed for 50 line reports with validation errors shown in Figure 12 is similar to that observed for 50 line reports without errors shown in Figure 10. The server processing time increased slightly, as with the 12 line reports with validation errors. Communications times also increased, but total response times were very comparable. Variability was likewise consistent, with no locations showing excessive variability for these transactions.

The 12 and 50 line report confirmation transactions tested the commit of the report data into the database and the display of the report result page. The report confirmation transactions had considerably less data input to the server since the report data was previously submitted and was held in memory.

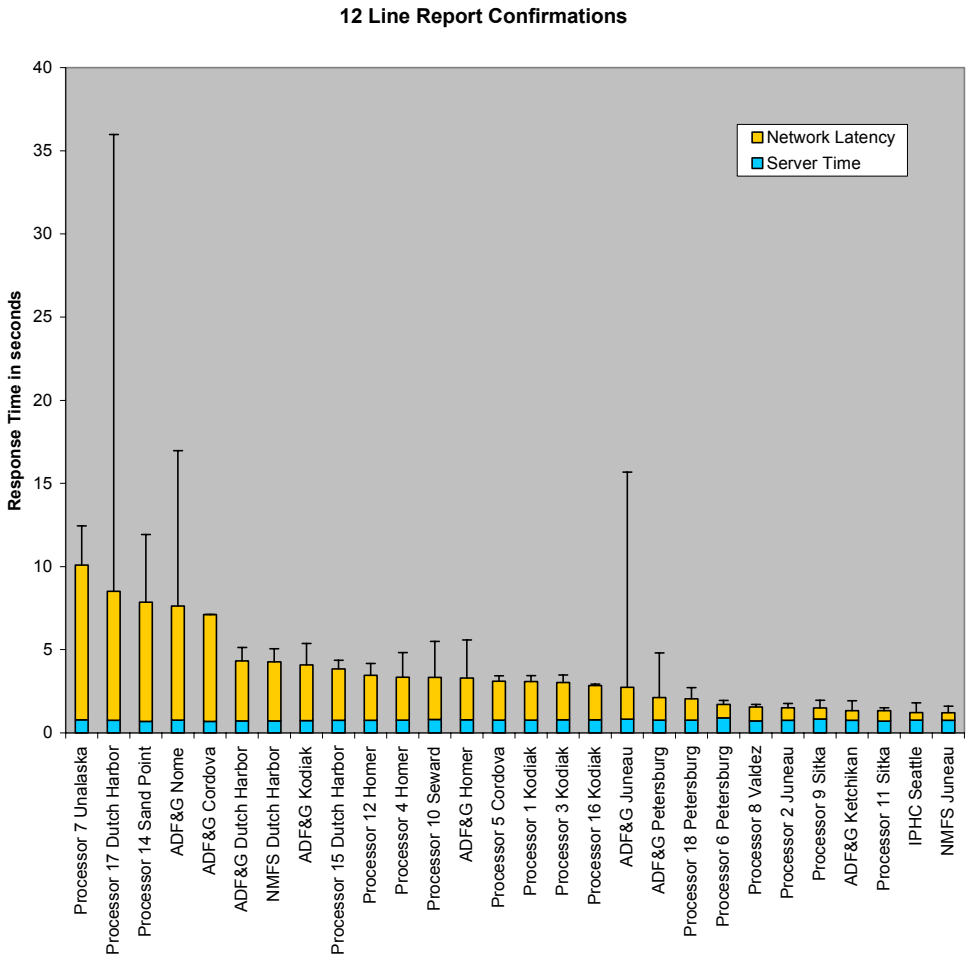


Figure 13

Figure 13 shows the response times for 12 line report confirmations. The server processing times are quite short. Short server processing times were expected since the in memory data objects were previously constructed and the data was already validated. The inserts of records into the database accounted for most of the processing. Variability was higher than other transaction types for some locations. It is possible that some variability was due to a transient slowdown at the ITG datacenter, since tests from the ADF&G offices in Juneau were affected. However, most locations did not experience excessive variability.

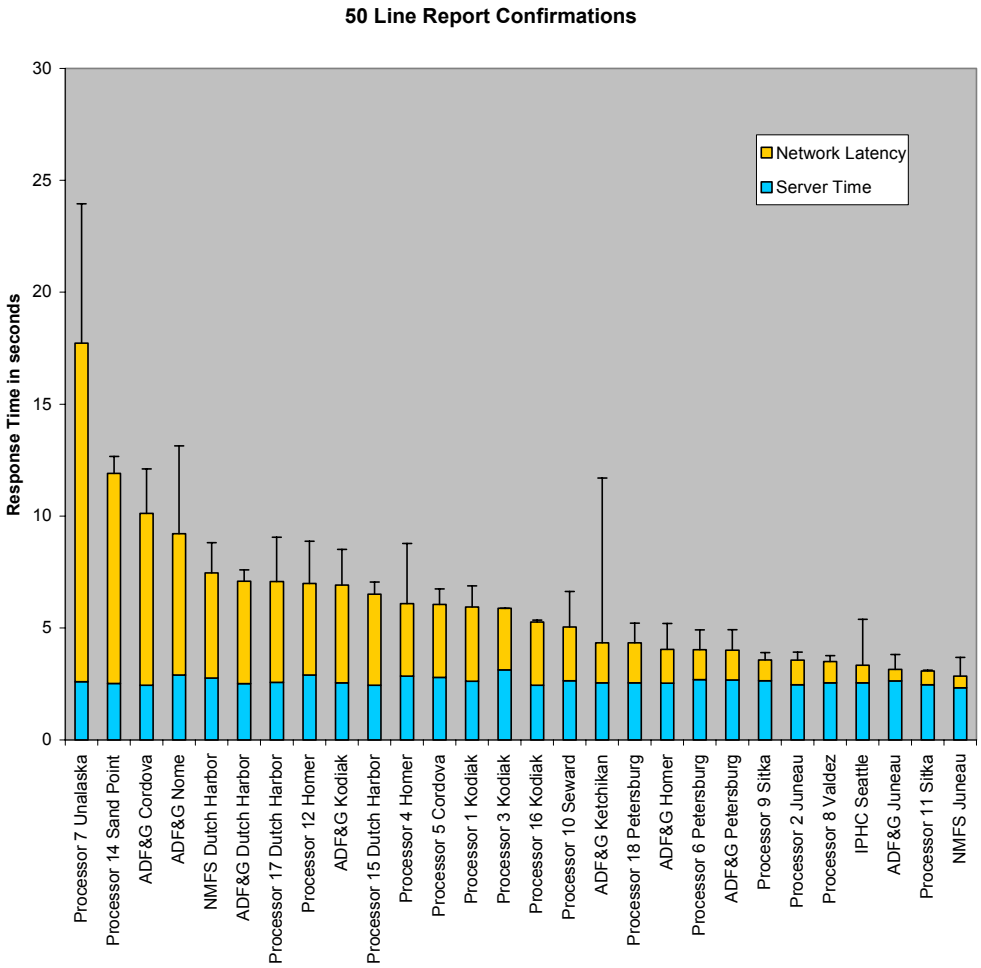


Figure 14

Figure 14 shows the response times for 50 line report confirmations. The server processing times increased over those for the 12 line reports. The total response times for these confirmation transactions remained below the times for initial data submission and validation. Variability was similar to that observed for the initial transaction. The ADF&G office in Ketchikan did show unusual variability just as it had for 12 line reports with validation errors. This variability may indicate intermittent communications slowdowns for that location, but overall the Ketchikan office's transaction performance was reasonably good.

The main emphasis in the testing was initial report submissions. The system did provide for testing updates of existing reports. Not all testers executed update tests, and those that did executed many fewer updates than initial report submissions. This paralleled the expected usage of the electronic landing reporting system. As a result, the charts of report updates and update

confirmations display less data than the report submissions. Like the report submissions, the update process consisted of two transactions: one for data input, and one to confirm the user's inputs and actually update the database.

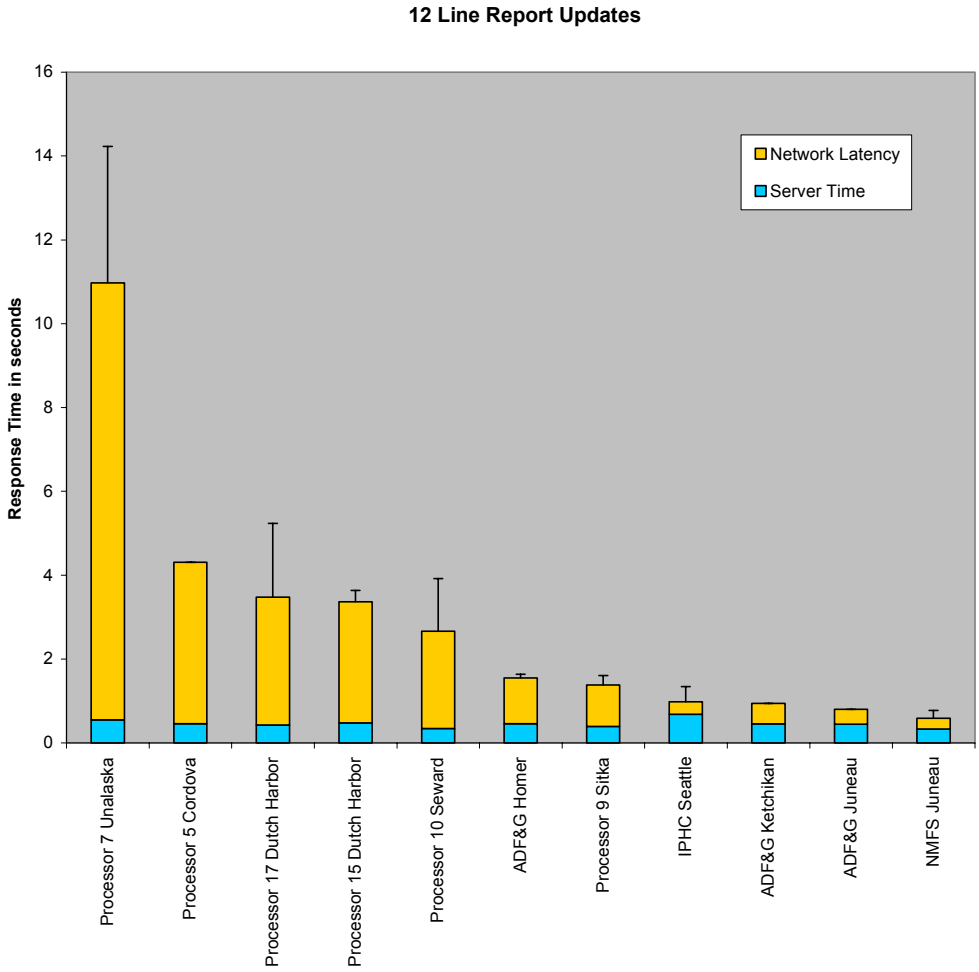


Figure 15

Figure 15 shows the response times for updates of 12 line reports. The server processing times were significantly less than those for initial report creation, and were similar for all locations. The network latency times were less than the initial report times.

The response times for 50 line report updates, as shown in Figure 16, were scarcely greater. This result is as expected because the database IO operations that retrieved the line items probably retrieved 50 line items with about the same number of IOs as for retrieving 12 line items.

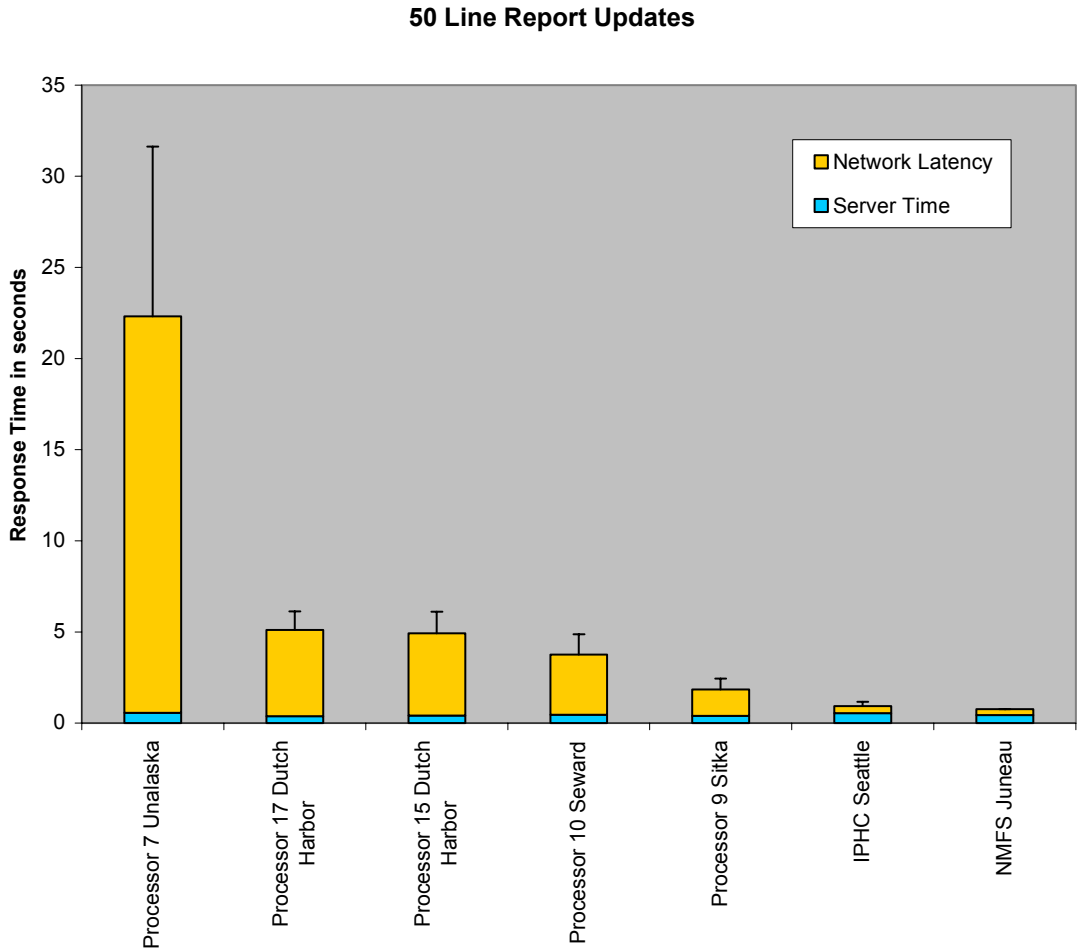


Figure 16

The report update confirmation transactions displayed in Figures 17 and 18 show similarly slight response time increases between 12 line and 50 line transactions. The response time of the transactions is 4 seconds or better for all but Processor 7 in Unalaska. This processor experienced the worst performance for all transaction types, and their data for updates is in line with their results for other transactions. In Figure 17, the ADF&G Ketchikan office shows an excessive amount of server processing time. This appears to be an anomaly. The zero for standard deviation indicates the aggregation of only one record. It is possible that the transaction executed at a time when a transient process on the server was consuming processing resources.

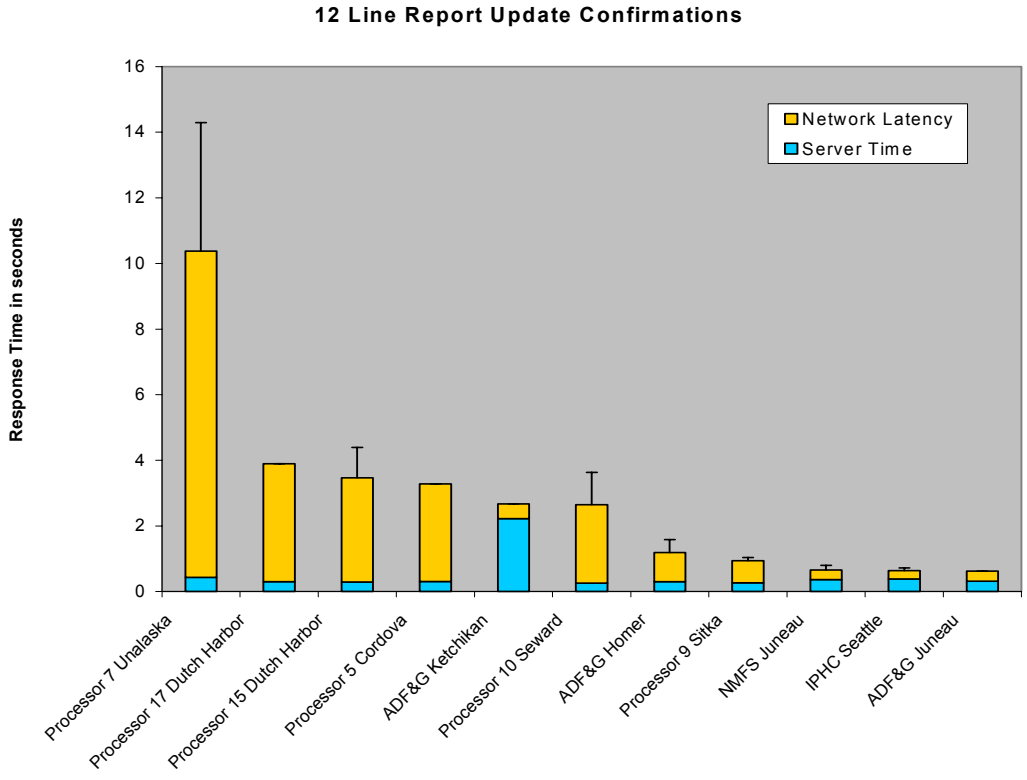


Figure 17

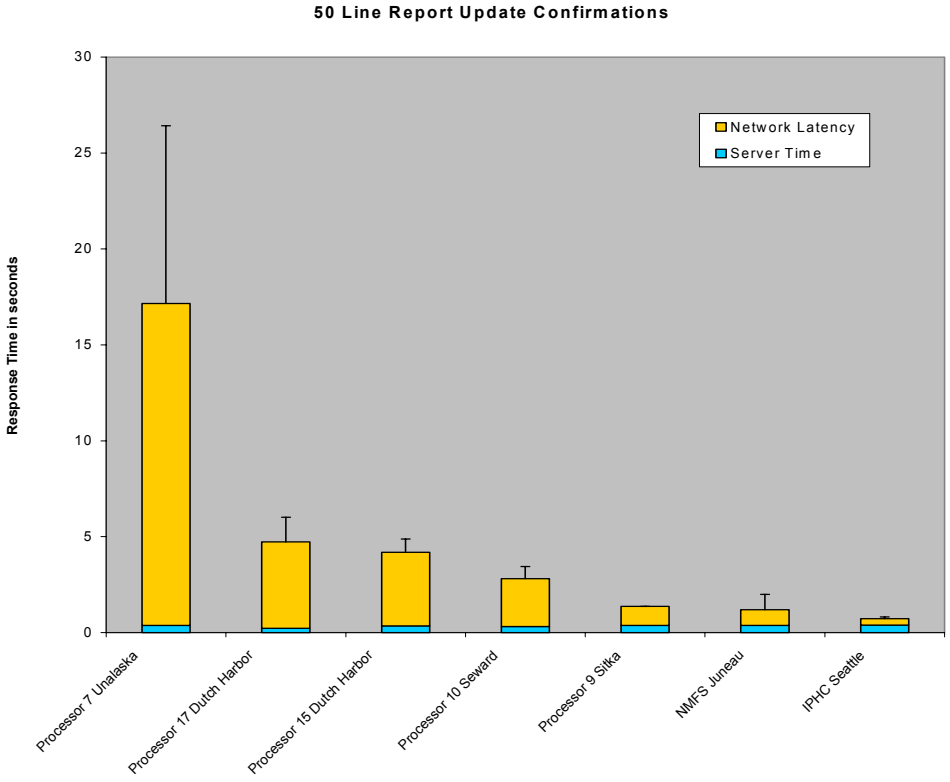


Figure 18

The performance data provides a coherent picture of the system performance at various locations. Server processing times are consistent, and increase as the amount of data to process increases. More remote locations had longer communications times, but the increase was modest for all but a few locations.

4.2 Script Execution Success

The performance instrumentation software, which produced the response time data presented in Section 4.1, depended on the successful execution of JavaScript embedded in the web pages. In addition to gauging the response time, the JavaScript also provided a measure of the extent to which the script execution encountered problems. Transaction records in the database that lacked response time data were marked as incomplete. We believe the primary cause of incomplete performance measurements on transactions was JavaScript failures on the client workstation. JavaScript is a known source of web application faults, and browser compatibility testing commonly identifies browser specific problems with configurations and JavaScript in new web applications. While other factors could account for transaction instrumentation failure, such as Internet connections failing between the time the transaction page was received by the browser and the JavaScript executed, these would be less likely. In addition, several of the system faults observed and documented in Section 4.4 can be traced to web browser configuration and scripting problems. Table 2 shows the percent of completed performance measurements on transactions broken out between processors and agency offices. The agency offices had a slightly higher rate of incomplete measurements on transactions.

Location	Incomplete performance measurements	Total Transactions	Percent of measurements completed
Agency Offices	122	2593	95.30%
Processors	44	2309	98.09%
Total	166	4902	96.61%

Table 2

Since browser versions and configurations might have varied between different computers in the same processor or agency office, Figure 19 shows the transaction measurement completion rates broken down by IP Address. While some workstations executed transactions using more than one IP Address, each column in the chart represents the results for a single workstation. One processor location and one agency location had

workstation configurations that appeared to be completely incapable of running the JavaScript. In each case, these sites were responsible for 10 instances of incompletion. Without them, the percent complete overall for both agency office and processors improved by about half a percent.

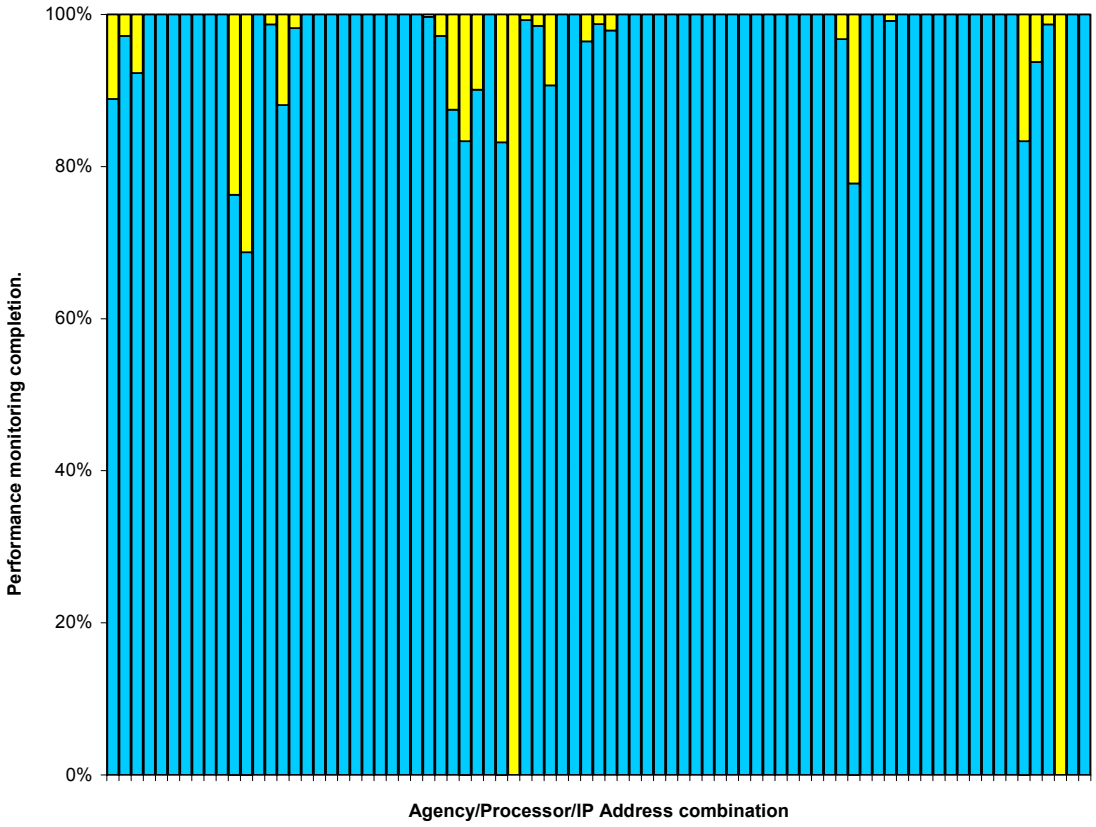


Figure 19

The results of measuring performance with 2 server requests per transaction are a reminder that web page transactions will occasionally fail. The Internet is well known to have a low mean time between failures (MTBF), but also a low mean time to repair (MTTR). Individual transaction failures should be expected, but retries will likely succeed. The design of the landing system should accommodate these occasional failures, and should tolerate them to the extent practical, with minimal inconvenience to the user.

4.3 Security Analysis

An important consideration for this project was the assessment of system security capabilities and options. A number of specific software development tasks addressed different security features for the system. The system provided

user accounts to secure report submissions. User accounts could only be added or modified by authorized administration users. Passwords were required for authentication before users could submit any data. The data stream was tested with encryption. Specific security areas analyzed were the performance implications of encryption, the resistance of the system to vulnerability probes from the Internet actually encountered during testing, and implications for security of relying on user passwords.

4.3.1 Affect of Encryption on Performance

In order to measure the effect on performance, if any, of encrypting the data being passed between the web server and browsers on user workstations we conducted testing before 9/30 without encryption. The browsers were required to connect with an encrypted data stream for the testing period from 10/3 until 10/8. During the period from 9/30 until 10/2 the system supported both encrypted and non-encrypted communications. The data stream was encrypted using the industry standard Secure Sockets Layer, which is built into most browsers.

Location	Latency w/o SSL	Latency w/SSL	Percent increase
ADF&G Cordova	14.6	10.47	-39%
ADF&G Dutch Harbor	5.77	5.46	-6%
ADF&G Ketchikan	0.77	0.92	16%
ADF&G Kodiak	4.17	8.02	48%
ADF&G Petersburg	1.71	1.37	-24%
Processor 1 Kodiak	3.97	3.9	-2%
Processor 2 Juneau	1.67	1.58	-5%
Processor 4 Homer	1.87	7.78	76%
IPHC Seattle	2.25	0.32	-602%
NMFS Dutch Harbor	5.57	4.89	-14%
Processor 5 Cordova	4.18	5.04	17%
Processor 6 Petersburg	1.41	1.17	-21%
Processor 7 Unalaska	12.04	23.87	50%
Processor 9 Sitka	1.68	1.57	-7%
Processor 10 Seward	1.73	3.22	46%
Processor 11 Sitka	0.82	0.71	-16%

Location	Latency w/o SSL	Latency w/SSL	Percent increase
Processor 12 Homer	4.16	7.16	42%
Processor 14 Sand Point	13.61	11	-24%
Processor 15 Dutch Harbor	4.46	4.38	-2%
Processor 17 Dutch Harbor	3.27	5.4	40%
Processor 18 Petersburg	1.12	2.66	58%

Table 3

Table 3 shows the average network latency for 50 line transactions without validation errors for the periods with and without SSL encryption. Although it was reasonable to expect that a slight increase in communications latency might result from the additional communications processing necessary to encrypt and decrypt the data stream it is evident from the data in Table 3 that other factors completely mask the contribution of the SSL processing. Many locations actually recorded decreased network latency. The locations that saw increases in latency had varying degrees of degradation. The affect of SSL encryption on network latency appeared to be very minor, and was completely overshadowed by other sources of communications lag times.

4.3.2 Application Server Security

The web application server experienced malicious probes on six different days. Each time the remote host probed for specific files. In four cases, the attacker was searching for the owssvr.dll and cltreq.dll files. In the other two cases, the attacker searched for the msadc.dll files. These files all contain known vulnerabilities. The vulnerabilities affect Microsoft web servers. Since the Technology Demonstrator web application server was Enhydra running on Linux, these probes posed no danger to the server. The attacks appear to be scripted and were not particularly sophisticated. They did not check that the server was running Microsoft software before searching for the vulnerable files. They might also have been automated probes of a worm attempting to propagate itself. The frequency of probes emphasizes the need to protect servers with good security practices.

4.3.3 User Security Awareness

Security is a complex, multi-faceted subject, and the human element is frequently the weakest link. For the Technology Demonstrator test, the users' passwords were all set to a default value. The system provided a

means for the users to change their passwords, and they were informed of this feature in the initial test startup email messages. The system did not require that users change their passwords, nor did it send reminder messages. Less than half of the users changed their passwords. Of those that did, some changed them to very short or otherwise relatively insecure passwords. This result is not unexpected, and should not be construed as an indication that the users involved in this test were unusually lax. However, it does indicate that the production landing reporting system should avoid setting and accepting default passwords and should have features that encourage users to select good passwords. Additionally, efforts to secure technical aspects of the system should be balanced by efforts to improve user security awareness.

4.4 System Fault Analysis

Unlike most system development efforts, where the goal is to build software with a minimum of faults, part of the Technology Demonstrator testing effort was oriented to discovering faults, and understanding not only how to suppress them, but also to characterize their nature. A key part of the evaluation of the development infrastructure, communications infrastructure, client environment, and design assumptions was to develop an understanding the faults and the design decisions that might make faults more or less likely to occur.

As can be expected with any software system fielded to a diverse user community and undergoing field testing, a number of software faults were identified. We identified types and occurrences of faults by two methods. One was direct user reporting of the fault and second was system log analysis that revealed occurrences whether reported or not. Both means were important in identifying the faults because the reliability of systems on the Internet is less than 100 percent. Many users will try a second time, and if the problem does not recur will fail to report it. This tends to mask intermittent problems. The use of log analysis helps reveal some of these cases, so long as the problem generates log records.

The following were the most significant software faults that persisted throughout testing or were difficult to characterize and solve. Other system faults that were identified in the first few days of agency user testing and were easily fixed are not included; the resolution of such bugs was a normal and expected part of any software development effort.

4.4.1 Server Object Passivate Problem

During the initial testing with agency users an intermittent problem was observed where data objects could not be accessed. The error message log indicated the cause of the problem was that the objects were not configured properly to allow them to be archived to disk. After reconfiguring the objects, this problem did not reoccur.

4.4.2 Missing Number of Lines Parameter Problem

The missing line numbers parameter was an intermittent problem with the data being passed to control automated test data generation. No cause of this problem was isolated during testing. It is believed to be related to JavaScript usage, and appeared to be browser dependent. During testing a change was made that moved the number of line parameter to a position higher on the page, based on the theory that a JavaScript bug might be cutting off the data stream before its end. After that change, no further incidents of this problem can be found in the log, supporting the theory that JavaScript was truncating the data stream.

4.4.3 PDF File Display Problem

About a third of the testers initially could not display and print PDF files. This condition appeared to be caused by browser configurations that did not allow the system to open a new window from a form button. Providing a link as an alternate means of requesting the PDF file resolved the problem for all but one user. For that user, the received file would not open properly. However, when the user attached the file to an email and returned to the developers it proved to be a correct and complete file. This result indicates a browser configuration problem regarding the display of PDF documents.

4.4.4 Object Reuse Problem

An intermittent problem occurred if a user attempted an unsupported operation, such as displaying a PDF file for a report that did not exist. The server object was marked as unusable after the failed operation, and the server would give system errors if any other operations were attempted. Refreshing the server object by logging out and then logging back in resolved this problem. Once it was understood, a software change silently performed the logout and login operations to hide this problem from users.

4.4.5 Space Quota Problem

One day before the end of testing, the application exceeded its space limit in the database. The space quota was originally set to 20MB, and the about 2600 reports had been stored when the quota was exceeded. The problem manifested itself as an application server Remote Exception error message on all data add attempts. When the space quota was expanded, the application server and web server resumed processing without either server requiring a restart.

4.4.6 Encryption Level Problem

About ten percent of agency users had older browsers that did not support the SSL encryption level used for the secure communications version of the system. This was a browser issue. The browsers that experienced this problem were at least two generations out of date, but some browsers older than that did not have the problem. No processor users reported this problem, indicating that among processors operating systems were more up to date than some agency workstations.

4.5 User Feedback

After the completion of system testing, we surveyed test users to collect direct user feedback on various aspects of the system. Table 4 summarizes the responses from processor users.

Test participants were asked to run tests several times per day over a 2-week period. Recognizing that the participants were volunteers and had other job responsibilities, we asked how many tests each participant recalled running. The Freq of tests line shows that data. We also asked how many times they tried to run tests and found the site unavailable. No user reported this condition. However, we know that Internet service was interrupted to at least one participant on several days, so this should be interpreted as the number of times they were unable to access the test site when they had connectivity to the Internet in general.

Testing included a number of transactions. We asked the participants to rate their perceived performance of each transaction on the following scale:

1 – surprisingly fast
2 – acceptable
3 – slow, but usable
4 – not acceptable
5 – could not determine/not applicable

The results of their ratings are shown in the *Transactions* section of Table 4.

The performance appraisal questions measured the impression of the participants on the question of whether an Internet based system would be acceptable for landing reporting from a performance perspective. The Performance line in Table 4 shows their impressions, using the following scale:

1 – agree that the performance would be adequate
2 – mildly agree that the performance would be adequate
3 – uncertain as to whether or not the performance would be adequate
4 – believe the performance would not be adequate

The performance appraisal also asked about performance improvement or degradation over time. The Performance Stability line gives their ratings, with three possible values:

-1 – performance degraded over time
0 – performance stayed the same
1 – performance improved over time

Location	Proc 1 Kodiak	Proc 9 Sitka	Proc 4 Homer	Proc 7 Unalaska	Proc 10 Seward	Proc 14 Sand Point	Proc 18 Petersburg	Proc 15 Dutch Harbor	Proc 17 Dutch Harbor	ADF&G Petersburg	IPHC Seattle	NMFS Dutch Harbor	ADF&G Kodiak
Freq of tests	>1/day	>1/day	>1/day	>1/day	>1/day	3-5/wk	3-5/wk	>1/day	>1/day	>1/day	>1/day	>1/day	>1/day
Site Unavailable	0	0	0	0	0	0	0	0	0	0	0	0	0
Transactions													
Login	1	1	2	1	2	1	2	1	1	1	1	2	1
Menu	1	1	2	2	2	1	2	1	1	1	1	2	1
12 No Error	1	1	2	2	1	1	2	1	1	1	1	1	2
50 No Error	1	1	2	3	1	1	2	1	1	2	2	1	2
12 Errors	1	1	2	2	1	1	2	1	1	1	1	1	2
50 Errors	1	1	2	3	2	1	2	1	1	2	2	1	2

Location	Proc 1 Kodiak	Proc 9 Sitka	Proc 4 Homer	Proc 7 Unalaska	Proc 10 Seward	Proc 14 Sand Point	Proc 18 Petersburg	Proc 15 Dutch Harbor	Proc 17 Dutch Harbor	ADF&G Petersburg	IPHC Seattle	NMFS Dutch Harbor	ADF&G Kodiak
12 Confirm	1	1	2	2	1	1	2	1	1	1	1	1	2
50 Confirm	1	1	2	3	3	1	2	1	1	2	1	1	2
12 PDF	1	1	2	5	2	1	3	1	1	2	1	1	2
50 PDF	1	1	2	5	2	1	3	1	1	2	1	1	2
Performance Appraisal													
Performance	1	1	1	3	1	1	3	1	2	1	1	3	1
Performance Stability	1	1	-1	-1	0	-1	0	0	-1	-1	0	0	-1
Problems Observed													
Remote Exception	1	0	0	1	0	0	0	0	0	0	>1	0	0
50/12 Problem	0	0	1	2	0	0	0	0	0	0	0	>6	0
50 confirm	1	0	3	1	1	0	0	4	0	0	0	>6	0
PDF Button Problem	N		N	Y		N	N	Y	Y		N	N	N
PDF New Link Problem	N		N	Y		N	N	N	N		N		N
PDF Same Link Problem	N		N	Y		N	N	N			N		N

Table 4

The questions about problems observed measured the extent to which individual participants encountered intermittent problems with the software. Some of these problems were bugs that were corrected during the test period; others were symptoms of workstation configuration issues. All participants did not answer all these questions. In some cases, it appears that they did not answer because they did not encounter the problem. The PDF problem lines represent a single problem: the inability to view reports as PDF files. Progressive changes were made to isolate and correct this problem, and

users were asked to test the changes. The questions about the PDF problem were intended to identify the results of the progressive changes. It appears that two of the participants stopped testing the changes once the problem was resolved for them.

4.5.1 Performance Assessment Correlation to Observed Data

Two of the processors appraised the overall system performance as "uncertain as to whether or not the performance would be adequate". One of these was Processor 7 in Unalaska. This processor recorded the worst performance on every transaction tested. Their communications capabilities were significantly worse than any other processor who participated in the test. It is somewhat surprising that they did not appraise the system as not adequate. The other processor who gave the lowest rating was Processor 18 in Petersburg. This processor did not execute many test transactions. Their actual system performance was generally better than average.

One agency respondent appraised the system performance as "uncertain as to whether or not the performance would be adequate". The measured performance for this office was among the worst for agency testers. In addition, this particular tester experienced an unusually high number of system problems, including the inability to handle SSL encryption from their default browser.

Only one respondent, Processor 17 in Dutch Harbor, appraised the system performance as "mildly agree that the performance would be adequate" for a landing reporting system. This processor had observed response times that were poorer than most.

All other processors who responded to the survey appraised the system performance as "agree that the performance would be adequate" for a landing reporting system, including Processor 14 in Sand Point. That processor's response times were consistently exceeded only by Processor 17. This rating could be an indication that in comparison to other Internet applications the Technology Demonstrator performed well in Sand Point, or that even with the response times observed in Sand Point electronic reporting using the Internet is desired.

The performance change over time reported by users in Table 4 correlated reasonably well with the observed response time changes between the testing without SSL in the initial test period and the testing with SSL encryption in the later time period, as shown in Table 3.

5 Assessment

The Technology Demonstrator project gathered quantitative data about the communications infrastructure, and state of computer and network technologies, at seafood processor locations and agency field offices. It also gave the development team firsthand experience with a number of technologies that could be used for the integrated landing reporting system. In this section, we assess the software developed and the technologies employed in terms of what worked, what did not work, what remains uncertain, and what we would consider changing in the next development phase.

5.1 System Assessment

The system architecture is shown in Figure 20. The system ran on two separate server computers, both of which were located in the State of Alaska datacenter in Juneau. One computer was a database server. It ran Oracle as the database management system for the Technology Demonstrator application. The database server was not dedicated to the Technology Demonstrator; it supported production ADF&G applications as well. The second computer was the application server. It was dedicated to the Technology Demonstrator system. The application server ran two software components, the JBoss application server that contained the primary business logic for the system, and the Enhydra Web Application server that provides the web page processing. The JBoss server also hosted the web services that supported the User Management desktop client, which ran on selected user workstations.

The application server architecture was implemented with a layered approach. All requests from client applications, whether the User Management desktop application or the web application, used the same interface to the business logic layer. This layer contained the primary processing for the application, and isolated all data modification code in the Container Managed Persistence (CMP) layer from the callers. The CMP layer used the JBoss container services to connect to and access the Oracle database. The business logic layer never accessed the database directly.

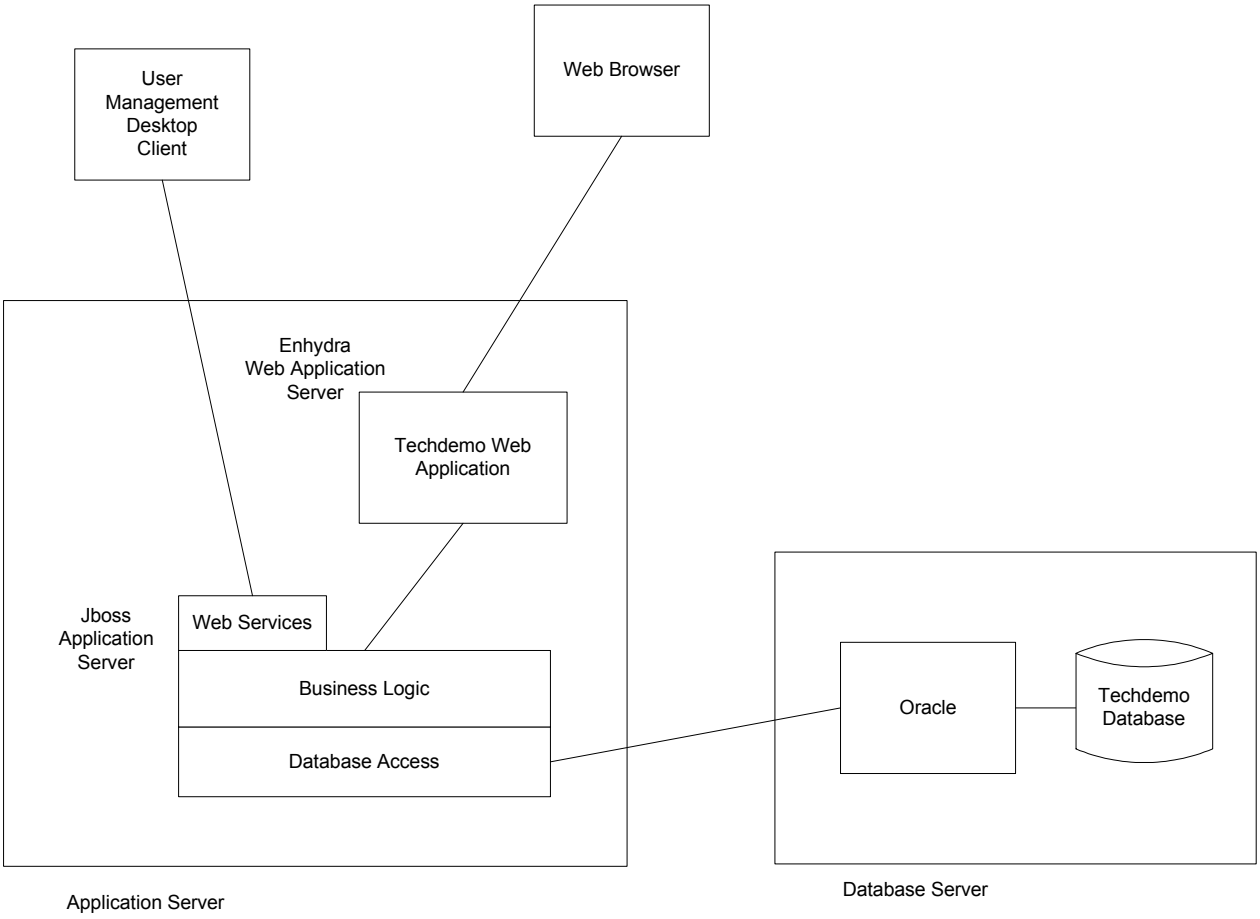


Figure 20

5.1.1 Server Application

The Technology Demonstrator server application components worked well. The application components for data storage in the database, and data retrieval performed to expectations. The validation logic worked correctly, and did not appear to extract an unacceptable performance penalty. No database errors were encountered. The initial version of the system had a coding bug that resulted in system errors when the server tried to save objects in memory to disk. This error, known as the "passivate problem" because it occurred when the server was saving objects in a passive form from which they could be reactivated, was corrected before testing with processor users commenced. Once the fault was corrected, the server did not experience a reoccurrence of this problem. After the problem was corrected the server application ran for the entire test period without being restarted, indicating that internal processing was well managed.

The application server environment provided a highly capable logging mechanism. The server application did not exploit the logging mechanism as fully as possible. In part, this was due to the complexity of the mechanism. Simply add more log points could overwhelm support personnel with data. Logging design that considered in advance what log data would be useful, and how to organize it to provide the clearest picture of server activity, would allow superior insight into the internal processing of the application.

The validation processing employed in the application demonstrated the capability for using database lookups to prevent bad data from being stored in the database. However, the actual validations needed have to be identified from requirements that fully address the detailed business needs for a landing system.

5.1.2 Web Application

The Technology Demonstrator web application worked well. In most cases, the pages functioned properly and allowed the users to control the system as desired. The application was easy to change, and several changes were made during the course of testing to determine the cause and effects of unexpected conditions, such as the inability to open PDF documents. The ability to submit test reports with and without errors illustrated how validation error messages and sanity check warning messages could be handled in the landing reporting system. The web application would report the errors, and the users would have a button that could be used to resubmit a report ignoring the warnings, just as was done in the Technology Demonstrator. In the case of the test report submission, the button to submit with errors disappeared. For the landing reporting system the button to submit ignoring warnings could appear only after the warnings where displayed. The warnings would need to be precisely specified in the requirements, perhaps by analyzing the most common errors currently found.

The use of the PDF format for downloading report documents worked reasonably well. However, the sensitivity of browsers to the linking mechanism was greater than expected. Once plain hyperlinks to the PDF documents were provided on the pages, almost all users were able to download them without incident. The only case where the user was not able to view the PDF document was apparently caused by a browser or workstation configuration problem because they were able to send the downloaded file as an email attachment, and the attachment could be opened normally.

The JavaScript code that collected performance data worked in most cases, but experienced a significant intermittent failure rate. JavaScript compatibility with different web browser and operating system combinations is a known problem for web applications. The extent to which the system had problems with browser interactions was surprising, and existing web applications in production at NMFS that do not use JavaScript have not experienced such problems. The use of JavaScript to submit a second message to the server after a successful transaction was acceptable for a test system, but should not be used for a production system that depends on receiving the messages. In general, JavaScript should probably not be employed in the production landing reporting system.

Like the application server, the web application environment provided a capable logging mechanism. The web application would have benefited from more extensive log messages. While a number were added during the test effort as some of the identified problems were researched, a detailed definition of what logging is required and how it should be organized would be useful. It would ensure that logging is used consistently across the application to provide support personnel with the information needed to address issues.

5.1.3 User Management Desktop Application

The user management desktop application worked reasonably well. The web service interface to the application server was effective and performance was good. The design of the user management application assumed that processor data would be preloaded to the database. That assumption seemed reasonable because the data is available in existing agency systems and could be uploaded on a production system. However, some of the processor data is in the ADF&G database and some is in the NMFS database. Combining the data automatically might not always be effective. The user management application would have been more usable if processor and location data could be entered if it became available, without waiting for a database upload. This should be considered when defining the detailed requirements for the landing reporting system. Additionally, the user management application assumed that different locations for the same processor would have different processor codes. It appeared that this is not the case. While the application handled the data without problems, this indicates that a more extensive definition of the business requirements will be needed before the production system is developed. The Technology Demonstrator system design was based on capabilities envisioned for a landing reporting, but it was vastly simplified in order to control costs at the current stage of

development. All aspects of the production landing reporting system will need to be addressed during the requirements definition effort, and most will be more complex than corresponding functions in the test system.

5.2 Development Environment Assessment

Two enterprise application architectures dominate the IT literature and marketing space. One is Sun Microsystems' Java 2 Enterprise Edition (J2EE) and the other is the .Net architecture from Microsoft. The Technology demonstrator project used tools from both architectures. J2EE was chosen for the primary server components. Java is its development language. Java has a great deal of support in the Open Source community. Open Source software typically has very liberal licensing terms, and usually no initial cost for the software. Support varies with some commercial firms providing support and some projects supported without cost by their developers. One of the tenets of Open Source is that source code is provided and can be changed, allowing programmers to correct problems if support is unavailable. The primary advantages of Open Source are no cost to procure, no management effort required to insure license terms are not violated, internal workings are not hidden so problems are more obvious and solutions are more evident to programmers, and interoperability between products is generally good.

The user management desktop application was implemented using the .Net architecture. The programming language for this development was Visual Basic. The .Net architecture is mainly limited to Microsoft operating systems. Its interoperability is generally good, but in some cases is intended to interoperate better with other Microsoft products than with those of competing firms or architectures.

5.2.1 J2EE Application Architecture

The J2EE application architecture generally worked well. The architectural design supports a layered or component based development style that is intended to promote code reuse. The J2EE container servers provide many services to applications written to the J2EE standard, saving effort on the part of development programmers. Many components are available for J2EE, many of them open source. J2EE and Java have no licensing costs. The J2EE architecture demands greater skills and knowledge on the part of the developer than simple web applications or desktop applications written in Java. This is due partly to constraints on the Java programmer when writing for the J2EE model, partly to knowing when and how to user existing components instead of writing new code, and partly to

complexities in configuring the development and runtime environments correctly.

An alternative to the J2EE for the server application would be the Microsoft .Net architecture. The Microsoft development tools are more integrated that those the Technology Demonstrator team used because they come from a single vendor. They have features that allow inexperienced developers to build applications very quickly. However, many of these features generate code from user input on graphical design tools. The generated code is often not suited to enterprise applications, since it does not reuse application code already written, and is itself difficult to generalize and make reusable. The rapid development features frequently embed configuration information in the generated code, making production configuration changes difficult. The rapid application development features are most appropriately used for prototype development.

The Microsoft toolset has a significant licensing cost, although this might not be an issue if the organizations doing the development have existing licenses in place. Add-on components are less likely to be open source, and so have greater license costs. In addition, the .Net architecture requires a Microsoft operating system on the servers where it runs.

5.2.1.1 Business Logic Components

The primary unit of business logic in J2EE is known as the Session Bean. These software objects are fairly straightforward to write. They provide a consistent interface pattern, allowing them to be called from other software in a fairly consistent manner.

The J2EE architecture has no alternative to using Session Beans. However, the business logic could be written as stored procedures in the database, providing a much thinner Session Bean layer. Oracle allows stored procedures to be written in Java, so some level of Java code reuse could be achieved with this approach.

5.2.1.2 Database Access Components

The Technology Demonstrator used the J2EE concept of Entity Beans to handle interactions with the Oracle database. Entity Beans are software components that isolate the details of storing and retrieving data from other objects, returning the required data as normal Java objects. Several approaches to Entity Beans have been implemented. The Technology Demonstrator used the Container Managed Persistence approach.

CMP attempts to hide the complexity of mapping Java data objects to database tables, committing, and other database processing from programmers. It does so by having the programmer declare the database and tables that are needed, and how they will need to relate in the Java code. These declarations can be quite intricate and complex, and do not take advantage of programmer knowledge of SQL techniques for particular ends. Until a programmer gains experience in how to configure the CMP to handle the database calls, it is in many ways less flexible than traditional database calls.

CMP is intended to provide intelligent use of database resources and good scalability. For example, if one user retrieves a particular record and another user needs it, the CMP mechanism will reuse the existing data object containing the record, but will make a copy if updates are made to the data. The CMP mechanism appears to be robust and flexible in handling database problems. When the Oracle database exceeded its quota and generated error messages, the CMP reported them back through the application to the end user. After the quota was increased the application server and CMP resumed processing database access and update requests without having to be restarted.

The alternative to CMP in the J2EE architecture is to handle all database interactions with application code. This approach is known in the J2EE architecture as bean managed persistence. In bean managed persistence, custom Entity Beans procedurally retrieve data from the database and update it through traditional database calls. The use of bean managed persistence would result in more programming of the database interface. It would allow more flexibility for application programmers with limited CMP experience, allowing such techniques as using Oracle stored procedures.

5.2.1.3 Xdoclet Deployment Descriptor Generation

The J2EE architecture allows single file deployment of applications to servers. While convenient, this facility requires that deployment descriptor files containing configuration information be embedded in the deployment file. Coding the deployment descriptors by hand is labor intensive and error prone. The Xdoclet initiative provides a set of tools that allows deployment descriptor information to be coded as tags in the comments of the Java programs that will be deployed. This approach improves the situation, because the tag language has some structure, and the necessary commands are embedded in the programs that need them. However, the process is still error prone because the correctness of the tag commands cannot be verified until runtime. Xdoclet tags improve

the situation, but deployment descriptors are still problematic. The alternative to Xdoclet use is to code the deployment descriptors by hand. Even with its limitations, Xdoclet is the better method to create these files.

5.2.1.4 Web Services Implementation

The requirement for desktop applications on remote sites, and the requirement to allow processor's systems to interface directly with the landing reporting system demands a system interface that can be accessed from across the Internet. Web services are a standard interface method for meeting this type of requirement. The Xdoclet utility provides tags that allow particular Session Beans to be declared as web services. The server automatically publishes the designated methods, and processes input from client web service calls so that the Session Bean can process them like any other method call. This approach is a very effective way of implementing web services.

The alternative to Xdoclet generated web services would be to create a web services server that calls the application server in the same manner as the web application server. This would require considerable programming and systems administration effort for no particular gain.

5.2.2 JBoss Application Server

The Technology Demonstrator project used JBoss as its J2EE container server. JBoss is a very successful open source project. Its no cost, liberal license allows many permutations of development, test, and production servers with no concerns about licensing. JBoss servers are in use in many organizations with processing loads greater than those anticipated for the landing reporting system. The State of Alaska runs JBoss servers in its Juneau and Anchorage data centers, providing agencies with application hosting and failover capabilities. The Technology Demonstrator project JBoss server ran for the full processor test period without restart. It tolerated the database exceeding its space quota without requiring application restart.

An alternative J2EE server would be the Oracle Application Server. Oracle's server is tightly integrated with the Oracle development environment, so to gain its maximum advantages those components should be used. The full Oracle suite is somewhat costly unless arrangements could be made to utilize existing licenses.

5.2.3 Enhydra Web Application Architecture

The Technology Demonstrator project used the Enhydra web application architecture and server for its web application component. Enhydra is an open source framework for developing Java Servlet web applications. It includes a lightweight, but capable application server.

Web application server tools are commonplace, and many alternative servers could have been selected for this project. The JBoss application server can run Java Server Page (JSP) applications. JSPs are somewhat less convenient for the developer than Enhydra applications. ASP.Net could have been used for the web application, but this would have required a Microsoft Windows server. Cold Fusion and PHP are other web application tools that could have been employed. Traditional CGI programs could also have been used. With the exception of JSPs and possibly Cold Fusion, none of these options have J2EE integration equal to Enhydra's. Additionally, excepting CGI, these tools all embed script code in HTML pages. Enhydra takes a different approach, using HTML pages as templates to compile into Java objects that render the pages at runtime. The application's Java code uses these objects to create the pages, but the code is isolated from the HTML, for better maintainability.

5.2.4 JavaScript Browser Client Scripting

The Technology Demonstrator project used JavaScript embedded in web pages to implement its performance instrumentation functions. JavaScript appeared to cause problems for some client browsers. In some cases the browsers would not run the scripts, resulting in no performance data being recorded. In other cases the scripts were run but appeared to occasionally truncate the data being sent from the page, resulting in a number of intermittent and difficult to reproduce problems.

The alternative would be to use HTML pages without JavaScript, and do all formatting and validation functions on the web application server. This technique has been used in other electronic reporting systems, most notably the IFQ Web reporting system, and has resulted in many fewer browser interface problems than were experienced during the Technology Demonstrator projects.

5.2.5 Eclipse Programmer Integrated Development Environment

Eclipse was used as the programmer's Integrated Development Environment for the Java development on the Technology Demonstrator

project. Eclipse is an open source project supported by an industry consortium led by IBM. It provides a high quality IDE that is easily extensible with third party plug-ins. Eclipse worked very well on this project. It had very good integration with the project's CVS source code version control system. Open source plug-ins allowed programmers to run the JBoss application server, as well as other tools, from within the programmer's environment, enhancing productivity. Integrating plug-ins from multiple sources required effort on the part of the programmers, but not excessively so. The lack of licensing restrictions allowed the developers to install on one of the ADF&G programmer's workstation to support test effort with no licensing concerns or costs.

Potential alternatives to Eclipse would be Oracle's JDeveloper, or Borland's JBuilder. Both are capable IDE's but both have significant software licensing costs with no better, and in some cases worse performance.

5.2.6 MS Visual Studio/Visual Basic Integrated Development Environment

Microsoft Visual Studio was used as the programmers IDE for the User Management desktop interface. Visual Studio is an extremely well integrated IDE. It provided a very productive environment for developing the Visual Basic desktop application, and provided all the tools necessary for calling the Technology Demonstrator server's web services.

Visual Studio has significant licensing costs, which were nominal in the case of the Technology Demonstrator project since the developers already subscribe to the Microsoft Developers Network.

5.2.7 Database Tools

The Microsoft Visio diagramming tool has a data modeling component that was used to create the data model for the Technology Demonstrator database. It proved to be a reasonable tool to eliminate much of the drudgery of coding database specification files by hand, but was somewhat inflexible. Alternative data modeling tools would be other commercial products such as Visible Analyst, ERWin, or Oracle Designer. Oracle Designer is a very comprehensive tool for data modeling, but has significant licensing fees. Each of the tools has areas where they are inflexible. None of the other tools have any compelling advantage over Visio.

The development team used SQL Navigator from Quest Software as its general database query tool. In addition, one of the ADF&G programmers

used an evaluation copy of SQL Navigator while supporting the project. All programmers found SQL Navigator to be an effective tool. In addition, it is in widespread use at NMFS. A lower cost alternative would be SQuirreL, an open source project. However, the capabilities of SQL Navigator far exceed those of SQuirreL, and of most other commercial tools.

5.2.8 Linux Server

The Linux server that hosted the Technology Demonstrator application performed very well. The server was physically isolated in the ITG data center. Developers did not have access to the server console, but this did not cause any inconvenience. All configuration and maintenance activity on the part of developers could be done using the Linux command line interface.

The primary alternative would have been to run JBoss and Enhydra on a Microsoft Windows server. Lack of physical access to the console would have required a terminal server, which is a more complex solution for remote access than a command line session.

5.2.9 Oracle Database

The Oracle database performed very well in this application. Reasonable performance was achieved without requiring any tuning. The database was subjected to an out of quota condition, and recovered without any problems when more space was added to the quota.

Both ADF&G and NMFS use Oracle for production database. The Technology Demonstrator database was hosted on an existing ADF&G Oracle server. A number of other SQL databases could have been used in place of Oracle. The most common alternative is Microsoft SQL Server, but IBM's DB2, and the open source databases Postgress and MySQL could have been considered. Support staff experience with the database product is a key consideration when choosing a database management system. The assistance of ADF&G data base administration personnel was invaluable on the Technology Demonstrator project.

5.2.10 ITG Datacenter Test Environment

The Technology Demonstrator software ran on a leased server in the State of Alaska Information Technology Group's Juneau datacenter. Running the system at an Application Service Provider (ASP) was an important undertaking for the project. The use of an ASP is expected to result in

superior support for the system infrastructure and network services. The use of the ITG datacenter provided an opportunity to assess the extent to which it met those goals, and imposed constraints.

The datacenter provides 24 hour infrastructure monitoring and physical system security. The arrangement worked very well. No outages were noted, although we believe the State infrastructure had some instability during the testing period that was dealt with promptly. The data center provided a firewall that blocked access to the server on all but needed ports. ITG was accommodating on the terms of the lease, and provided reasonably fast turnaround times for service and configuration change requests.

The alternative would have been to host the application in a commercial ASP, such as Rackspace, although this would have resulted in a higher lease cost. Another alternative have been to run the server in one of the sponsoring agencies' computer rooms, at the cost of decreased support since they are not staffed 24 hours a day.

5.3 Security Assessment

The Technology Demonstrator project gave the development team the opportunity to assess security features of the selected software components both in implementation of the system and in execution during the test phase of the project. The J2EE architecture and the JBoss server provide a good user/role/function security model. However, like all such models it does not provide conditional security within a function. For example, it can grant or deny the right of a particular user to update reports, but the model cannot conditionally allow the user to update one report, but not others. This conditional, or entity level, security must be provided by custom code. The Technology Demonstrator application used a layered security approach. The application interface required a system password to access the functions, and a userid and password to actually read or update the database. Userids and passwords were case sensitive, which contributed to good security, but which caused irritation to some users.

The Technology Demonstrator survived probing attacks from the Internet during the test phase. The probes appeared to be scripted attacks targeting any web server found on the Internet, rather than directed attacks targeting the Technology Demonstrator application. The vulnerabilities that the attacks attempted to exploit were not present, and the application continued processing without interruption. The underlying Linux server operating system was not affected. Linux is somewhat less threatened by viruses and Internet

worms than are Microsoft operating systems, but care should be taken with any server, and multiple layers of security are always prudent.

The web application was tested with and without encrypting the data stream. While intercepting a data stream between a browser and server is not particularly easy, the performance testing indicated no particular penalty for using SSL encryption, and the system configuration effort needed to use it is modest.

While the Technology Demonstrator exercised some of the security capabilities of JBoss, J2EE, Enhydra, and Linux, it highlighted the need for a detailed security model in the system requirements in order to restrict user access to only appropriate records in specified states.

Finally, the level of concern about security for the application on the part of users was low. While this might be natural for users of a test system, it seems to indicate that an effort to remind users of security concerns would be worthwhile. Such an effort might include system elements such as password audits when users set or change their password, to prevent them from using passwords that are too short or too easy to guess. It also should include education and reminders to keep passwords secret, not to disclose them to anyone, and to change them periodically.

5.4 Communications Infrastructure and User Site Capabilities

The Technology Demonstrator performance testing provided good data on the Internet and computer capabilities at various processor and agency locations.

5.4.1 Internet Communications

As expected, the Technology Demonstrator project showed that both agency and processor users in different locations have varying levels of Internet service. Research has shown that in general, users give decreased quality ratings to websites when response times exceed about 10 seconds.[1] The response time measurements discussed in Section 4.1 show

[1] Anna Bouch, Allan Kuchinsky, and Nina Bhatti, "Quality is in the Eye of the Beholder: Meeting Users' Requirements for Internet Quality of Service", Proceedings of CHI2000 Conference on Human Factors in Computing Systems, ACM Press, 2000, pp. 297-304.

average response times less than 10 seconds for all but the worst locations, for all of the 12 line transactions, and some of the 50 line transactions. Even the 50 line transactions have 10 second response times for many users.

We believe that several factors would tend to increase the level of acceptable response time for the landing reporting system users. Since the remote locations have Internet service that has lower performance than the typical user on the general Internet, users in those remote locations probably have lower expectations of Internet performance that found in general surveys. In addition, users at seafood processors will be more highly motivated to use an Internet landing reporting system than typical Internet users using eCommerce sites. Finally, users will probably be tolerant of slower performance for large sized transactions if performance for normal sized transactions is good.

In most cases the level of service appears to be adequate to support a web based landing reporting system. However, in some locations, the service level is poor enough that using the web based reporting might be inadvisable. A desktop application might be usable in such locations, since it could capture the data from the user and then transmit it to the server in the background, retrying as necessary until the transmission succeeded.

5.4.2 Browser Configurations

The test users had various browser types and versions. Although some users had problems with JavaScript, most of the browsers were able to handle the basic functions of the system. Some agency users had old versions of Netscape that could not handle the encryption used for secure communications, but none of the processors had this problem.

Beside JavaScript problems, the most common browser trouble for users during the test effort was receiving and opening PDF files. These problems were resolved by providing access to the PDF files through a hyperlink rather than a button. Almost all browsers were able to download the PDF files from the links.

The minimum recommended browser versions for the landing reporting system should be defined. A common practice is to recommend no more than two releases back. However, this might require a number of users, particularly agency users, to have to upgrade. In some cases upgrades will be required. Some agency users had Netscape version 4.77, which was not adequate due to SSL processing limitations.

5.5 User Management

The Technology Demonstrator project made several assumptions about user management that may need to be reconsidered during detailed requirements definition for the landing reporting system. The system expected processors and processor locations to be in the database. In the landing reporting system this requirement could be implemented with interfaces to the ADF&G and NMFS systems. However, this solution might not provide the data in as timely a manner as might be desired, so the ability to add processors and locations on the user management interface may be needed. This should be considered in the requirements specification, along with the security needed for this function if it is provided.

Additionally, the user management interface assumes that agency users will do all user management. This assumption is probably correct; none of the users in the test indicated any interest in adding other users to the system. However, some processors may want the ability to add their own new users or to disable their own users themselves. This option should be considered during the requirements definition phase.

5.6 Database Audit Trail Design

The Technology Demonstrator included database features to provide a full audit trail of all changes to report data. The system never actually updated report records. Instead, new versions of modified records were inserted in the database, along with data identifying the user who made the modification and the time at which it was made. As a result, the database contained the complete modification history of every record. When displaying the data, the system would show only the most recent version of each record, but all versions were maintained in the database.

The use of a version table for each data table increased the complexity of programming the database interactions. It also consumed database space at a greater rate than a system that allowed updates in place.

The version table approach provides excellent visibility of database changes and the state of each report over time. Multi-agency updates are easy to identify and track. An alternative to the version table approach is a change history table that stores a change record for each field that is modified. A change history table allows users to research who changed what fields at what time, but is far less capable in terms of showing the state of the record at any point in time.

Working with the version tables in the Technology Demonstrator application revealed some considerations that should be addressed during requirements definition. Only data that can be changed should be stored in the version tables. Some data is never changed, if it is wrong the record is deleted and a new record is created. The version table approach by its nature stores duplicative data. Efforts in requirements definition can help minimize unnecessary duplicative storage.

6 Conclusions

The technology demonstrator succeeded in generating unambiguous performance statistics for server response times and communications latency. The nature and types of system faults encountered provide significant information about the problems that can be anticipated in the production system deployment effort.

The performance and stability of the server architecture and components appears capable of supporting a web based landing reporting system. Although the system had some problems, these must be expected with any system development project with a short testing period. The ability of the application server to run for the entire processor test period without requiring a restart indicates a level of stability in the application infrastructure that can support the desired system.

The Oracle database performance was acceptable without tuning. This indicates that performance improvements could be made for a production system. Additionally, the recovery from the problem of exceeding the space quota on the database was easy on the part of both the database and the server application.

The user management subsystem was adequate for the Technology Demonstrator, but its use highlighted some areas that will need more development for the electronic landing system. Specifically, the Technology Demonstrator design assumed that in the electronic landing system processor information would be supplied by an interface from existing agency systems. Since no one agency system has all the processor data that might be required, this design might not be the best solution. A user interface that allows administrative users to setup and maintain processor data may be needed and should be considered.

The performance data demonstrates that network latency is the leading contribution to response times on Internet applications for many users. Some locations have considerable variability in the quality of their Internet service. However, the Internet infrastructure and observed performance for most locations tested appears to be adequate for a web based reporting. Since web services use the same protocols they would work as well. However, a pure web application may not be able to support all users. Performance for some sites is questionable for reporting via web pages, but is probably sufficient for a desktop application designed to be tolerant of intermittent communications and long network latency. Web services such as those tested in the user management desktop application could be designed to function under these conditions.

The variety of browser configurations encountered and the sensitivity of JavaScript to them were greater than anticipated. JavaScript is widely used on the Internet, and can be made to work in most cases. However, its use requires very careful design and programming, and extensive testing with a wide range of browser configurations. The problems it manifests are frequently intermittent, as the Technology Demonstrator system experienced, and so are difficult to reproduce and resolve. Encountering browsers that did not support the current level of SSL encryption was surprising. Minimum browser versions for the electronic landing system will need to be specified.

The Adobe PDF format and Acrobat reader are generally considered the most reliable way to print documents on the Internet, so the problems encountered with displaying PDF files were also a surprise. However, the PDF experimental fixes introduced during the testing period indicate these problems can be overcome.

The impact of SSL encryption on performance was negligible. While unencrypted data streams are not a major security concern, the effort to use SSL was minor and it works well. Its use would force users to upgrade very old browsers, which is probably desirable. Other security measures such as required login and password authentication worked well. While the Technology Demonstrator was not subjected to determined attack, it handled typical malicious probing from the Internet without difficulty.

User feedback was generally positive, with neutral responses from some of the sites with the worst observed performance. The feedback indicates a relatively favorable level of acceptance on the part of the seafood processors toward electronic reporting and web based applications.

7 Recommendations

Based on the conclusion that system performance of a landing reporting system similar to the Technology Demonstrator would be acceptable to users, WAI recommends that the development of the integrated landing reporting system proceed. The following sections provide detailed recommendations on various approaches and tools used on the Technology Demonstrator that should be retained or changed.

7.1 Technologies and Tools

The layered architecture used for the Technology Demonstrator project, with both a web application front end and web services to support desktop applications, worked well. While a simpler approach could be used for a web only application, the complexity is justified for the overall application envisioned. WAI recommends that a layered architecture, with a common layer of business logic used by all presentation interfaces, be employed for the landing reporting system.

WAI recommends that the database approach used for the Technology Demonstrator be retained for the landing reporting system. Since the agencies make significant use of Oracle as a DBMS for other applications, Oracle is recommended as the database platform. However, the licensing cost of Oracle could be significant. Licensing is subject to many factors, and Oracle changes them frequently as it maneuvers for market position. It can be expected to be in the neighborhood of $10,000, unless existing licensing at NMFS or ADF&G can be used. If the licensing cost is found to be excessive then Microsoft SQL Server or one of the open source databases, Postgress, MySQL, or SAP DB, could be substituted. The versioned approach to landing report database tables appears to be effective, and is recommended for the landing reporting system. Visio and SQL Navigator are recommended as database tools; however, either could be replaced as development programmers prefer.

The technical architecture based on Java and J2EE worked well for the Technology Demonstrator. WAI recommends that it be used for the landing reporting system. Although its complexities were challenging to the development team, any enterprise level application development framework will be complex. Alternative architectures such as Microsoft's .Net will experience development problems due to complexity; they will simply be different problems. The J2EE architecture appears fully capable of supporting

the needs of the landing reporting system. The Session Beans in particular provided a strong structure for implementing business logic. The JBoss server ran the software effectively, and exhibited no problems during the test program. The extensions that implemented web services using Session Beans were easy to use.

WAI found the container managed persistence approach somewhat more difficult to use than other aspects of the J2EE architecture and JBoss. While it is still recommended, alternatives could be considered on the part of the developers. A thin Entity Bean layer, with some logic encoded in database stored procedures, perhaps written in Java, might offer greater ease of maintainability, at the price of possible scaling problems. However, the volume of transactions anticipated for the landing reporting system may never reach the level at which scaling problems will be experienced.

WAI recommends Linux as the operating system for the production landing reporting system servers. It is somewhat less prone to attack from the Internet, and lends itself to remote access by developers. Microsoft Windows should be used as the server operating system only if adequate support resources are planned for monitoring, installing, and testing security patches. That level will be higher than the level required for Linux.

WAI generally found the development tools used on the Technology Demonstrator project to be effective. Eclipse is recommended as the programmers' IDE, and the plug-in tools for JBoss and J2EE that can be integrated with it are also recommended. The Xdoclet tools are recommended, but could be replaced according to developer preferences. While the tools provided some advantages, they were themselves complex to use.

Visual Basic was used as the development language for the User Management desktop application. An alternative would have been to develop the desktop application in Java. While this approach would have provided a common language across all system components, Java desktop application development is more difficult than Visual Basic desktop application development. WAI recommends that developers select whichever desktop development tool they prefer, with input from the agency support programmers.

WAI recommends that Enhydra be used for the web application framework and server. It has been used effectively on other projects for NMFS, and worked well for the Technology Demonstrator. SSL encryption should be used as a security measure. Although unauthorized reading of the data stream transmission is not a significant risk, the effort and cost of SSL should be

relatively modest, particularly if one of the agencies' existing certificates can be used.

WAI recommends that JavaScript not be employed in the landing reporting system. While JavaScript can be crafted and tested to the extent that it will work for most users, the experience with JavaScript problems on the Technology Demonstrator indicates that an excessive amount of developer effort would be required to eliminate all JavaScript induced faults. The intermittent nature of the JavaScript faults makes them particularly difficult to resolve. While JavaScript is capable of implementing desirable client side features, WAI believes that the developers' efforts can be more effectively focused on the capabilities of server software in order to provide the best system for users.

The State of Alaska ITG datacenter provided good service hosting the Technology Demonstrator. WAI recommends that it continue as the site for the landing reporting system. Plans for the system should specify the number of servers needed, and should explore alternatives for database hosting and DBA services that ITG might be able to provide.

7.2 Usability

The extent to which usability features can be built into applications is greatly affected by requirements. However, the Technology Demonstrator project identified a number of overall usability factors that should be considered in the landing reporting system development effort. WAI recommends that a web based application be built as the initial production landing reporting system. A web application may not be suitable for all processors; the testing showed that some processors might not have Internet communications that are adequate to support it. However, most processors would be able to use a web based system, and it would allow for the easiest deployment and greatest tolerance for change as the system evolves.

WAI recommends that a desktop application be considered for development for agency users. While an Internet application may be adequate for some agency users, those that will continue doing data entry of landing reports submitted on paper will probably need the flexibility provided by a desktop application. The design of the desktop application should consider the network latency and variability of network service observed at some agency locations, and should allow for semi-detached operation. The needs of different types of agency users should be considered during the requirements definition effort, and used to make the final decision on developing the desktop application.

A desktop application for processors is probably not needed for the first release of the landing reporting system, but will eventually be needed for processors with slower, less reliable Internet connections. As for agency users, the ability to cache data to isolate the users from slow response times would improve the usability of the system.

Several things could be done that would potentially improve the usability of the landing reporting web application. A minimum supported browser list, and preferred configuration values, should be developed and published to users. The minimum browser level is important because of the level of SSL security that will be used for encryption. Additionally, the system could be developed with a browser test page that would evaluate the user's browser and configuration.

System usability and good security are frequently conflicting goals. Security measures do not generally improve usability, but some security related steps could be taken to encourage good security practices on the part of users. The system could be made to audit passwords when the users set them, requiring passwords that are not easily cracked by dictionary search. Additionally, the system could send periodic email reminders to users to encourage them to change their passwords, and to keep them secure.

7.3 Design and Development

WAI recommends that the landing reporting system project continue with staged development, so that each stage can build on the successes of the previous stages and correct deficiencies encountered. The next step of the landing reporting system development should be the specification of detailed requirements. The Technology Demonstrator project identified three areas where attention to detail in the requirements specification process will yield benefits in the production system.

The Technology Demonstrator system used data from existing agency systems to validate inputs such as CFEC permits and vessel ADF&G numbers. Acquiring that data and loading it into the database was more difficult than expected. The requirements definition process should consider all aspects of interfacing with existing agency systems to both received validation data and provide landing report data. Particular attention should be paid to potential situations where needed data is not on the interfacing systems and needs to be entered manually.

User management requirements should be considered in detail, particularly requirements for setting up and maintaining user accounts. Input from processors and agency personnel should be considered for these

requirements because the burden of maintaining user accounts will fall on these two groups. The extent to which processors can setup and change their own user accounts must be balanced with the needs of agency users to control access to the system and provide needed security checks.

The application server tools used to run the Technology Demonstrator provided extensive activity logging capabilities. The detailed requirements for the landing reporting system should specify what information should be logged using the logging capabilities. Agency system support personnel who will manage the system once it is in production should be consulted to identify what data they will need to diagnose problems and maintain a clear picture of system activity.

Appendix A Acknowledgements

The success Technology Demonstrator project would not have been possible without the many individuals at seafood processors and agency offices who participated in testing the software and provided feedback. We would like to express our appreciation for their time and efforts.

Judy Sisco	Northern Victor	Unalaska
Aregash Tesfatsion	IPHC	Seattle
Peter Hail	Unisea	Dutch Harbor
Chris Chamberlain	Westward Seafoods	Dutch Harbor
Jennifer Starr	ADF&G	Ketchikan
Bev McElhose	Seafood Producers Co-op	Sitka
Tom Kong	IPHC	Seattle
Susheela Roach	Seward Fish	Seward
Kamala Carroll	ADF&G	Juneau
Kara Lagasse	Alaska Fresh	Kodiak
Rance Morrison	NMFS	Dutch Harbor
Duff Hoyt	Icicle Seafoods	Homer
Teresa Stolpe	ADF&G	Petersburg
Kim Phillips	ADF&G	Kodiak
Lara Hutton	IPHC	Seattle
Gail Smith	ADF&G	Juneau
Barbi Failor	ADF&G	Dutch Harbor
Heather Gilroy	IPHC	Seattle
Jerry Smetzer	ADF&G	Juneau
Scott Johnson	ADF&G	Juneau
Larry Talley	State of Alaska ITG	Juneau
Phil Witt	ADF&G	Juneau
Josh Keaton	NMFS	Juneau
Galen Tromble	NMFS	Juneau
Vicki Curtiss	Icicle Seafoods	Petersburg
Mike McCune	The Fish Factory	Homer

Sabrina Ware	Alaska Glacier Seafoods	Juneau
Bernard Vienneau	IPHC	Seattle
Cathy Benson	Norquest	Cordova
Steven Whitney	NMFS	Juneau
Mo Lambdin	ADF&G	Homer
Scott Kent	ADF&G	Nome
Seth Scrimsher	Norquest	Petersburg
David Ackley	NMFS	Juneau
Patty Britza	NMFS	Juneau
Chris Swanson	Sitka Sound Seafoods	Sitka
Shelly Kirkbride	Trident	Sand Point
Carmine Dicostanzo	ADF&G	Juneau
Denise Branshaw	ADF&G	Cordova
Terry Leitzell	Icicle Seafoods	Seattle
Dave Colpo	PSMFC	Portland
Afshin Taheri	IPHC	Seattle
Angelyn Cassidy	Peter Pan	Valdez
Troy Peterson	Western Alaska	Kodiak
Fredelyn Pugal	Alaska Pacific Seafoods	Kodiak
Tom Jarvis	ADF&G	Juneau

226 Seward St.

Suite 210

Juneau, AK 99801

Phone: (907) 586-6167