Canadian Epidemic Tracker

E-Health Canada – COMP 3004 Deliverable 3

Leigh Pascoe - Shane Panke – Sebastian Schneider – John Vanden Heuvel

Table of Contents

1 Introduction

The C.E.T. is software designed for the tracking of disease cases, medical supplies across Canada as well as the manipulation of this data to build statistical models and generate reports to track epidemics. In phase 1 of the software development cycle the core functionality of the program will be provided. This mainly concerns the user controls of the clerk. This includes the ability to add new items to the database as well as edit existing items. The Medical Clerk will be able to create shipments of medical supplies from one location to another. This user will have the ability to view the disease case inside of the map screen at the desired level of granularity. If the Medical Clerk wishes, they may filter diseases and supplies displayed on the map in real time. The map will update as new data arrives from repository server the client is connected to.

1.1 Phase 2 Features

The second phase of the software contains new functionality and security measures. This phase will make the user log into the system before they can use it. The client will recognize the permissions of the user type and give them privileges base on that type. The two users that will be added to this phase of development will be the Medial Administrator and the Systems Administrator. The medical administrator will have the same functionality as the Medical Clerk but will be able to generate and view reports based on various kinds of criteria. They will be able to search by date range, quantity, region and/or types of diseases or supplies. The Medical Administrator will also be able to import predicted data models to view in reports.

The System Administrator will be responsible for creating and adjusting new types of supplies and disease. They will also have the ability to create and change user profiles.

1.2 Overview of Document

The following document contains the methodology and techniques used for testing the C.E.T. System software: Phase 1. The first section will contain the methodology used to determine which tests would be sufficient to ensure the software follows all of the functional and non-functional requirements to ensure the final product exceeds expectations in performance and reliability. This section will also entail the reasoning behind using these methods. Inside of the section the unit testing methods will give insight to how we went about testing each component and subsystem. The next subsection describes how we integrated our software to the other subsystems and components.

2. Test Methodology

2.1 Testing overview

During the unit-testing phase, we used the sandwich approach. This means that we can test the data objects, the GUI and Networking individually and integrate them as they become completed. We chose this method because it allowed segments to be built concurrently while ensuring the stability of the software. Testing is done in several stages. Concurrently we can test the network connections, the data objects, and the graphical user interface. The second stage of concurrent testing is data object manipulation: storage, transfer and display; this is also the beginning of our integration testing, we can test that conversion between XML to data object and XML to server

storage. This stage of testing will be the most complex and difficult to test; this is because we can run into data duplication, data loss, connection issues and synchronization issues. While the network protocol is being tested, the GUI can be tested and integrated to the client data store. The next stage of testing will involve the full integration of the GUI, the Network and the storage on both client and server side. The last stage of testing will involve primary user interactions with the system and swapping client and servers with other camp B protocol systems.

2.2 Unit Testing

The unit-testing phase gave us an opportunity to test the building blocks of the system. The building blocks of the system are the data in which the entire system is based upon. This phase of testing will have to be the most thorough to ensure the code does not have unexpected errors later in the systems development cycle. Unit testing makes testing easier, as it isolates components making it less difficult to track individual faults. For doing unit testing we will be using the sandwich testing method. This means that each of our team members will be able to test different layers without interfering with each other's code. The three layers will be the network layer, the database layer, and the graphical user interface layer. The network layer testing phase will involve the parsing and formatting of the message protocol as well as testing the established connection to verify that it is stable. The parsing subsystem will change the data in the databases into a format that will be understood over the network.

2.2.1 Network Testing

The goal of testing the XML parser is to see if the format maintained correctly throughout the lifetime of the string. To test the XML we have to create a series of the

XML files based on the format we have established and convert them back to code and test whether the object has kept the same data. This will have to be done several times on each of the data types being passed over the network. The next stage of testing will involve passing the tested XML file over the network and compare the XML file before it was sent and after it was sent to make sure that the data was not corrupted.

2.2.2 Data Type Testing

Our testing process begins with testing all of our data objects that are stored in the databases. These classes have to be tested first because they are the foundation of which the entire system is based around. Refactoring of these classes in later iterations will be costly and time consuming if they have to be altered. The classes we test first are the *DiseaseCase*, *DiseaseType*, *Supply*, *SupplyType*, *Shipment* and *City*. To test these classes, a function is created to populate the local database with a couple thousand objects at once. All the data member values will be randomized to be between above the highest value allowed and below the lowest value allowed. This means that we created dates for objects that should not exist, i.e. -negative values, future dates as well as accepted values. If the data members work correctly then they will correct themselves. This will verify the integrity of the object data members to avoid future faults.

2.2.3 Graphical User Interface

The GUI can be tested while the other sections of the sections of the code are being built and tested. The GUI is the highest layer of the system and requires few new test drivers but many test stubs. The drivers will consist of the GUI components; they can be

interacted with through user input. The test stubs tests whether the interaction with GUI invoke the proper reaction in the system without the use of the database types. The GUI is built using the marble library and has to be tested on whether the library supports the goals that we want the system to achieve. Once the proof of concept has been established, the next stage is to test that the different components fit into the GUI in a format that we desire. This GUI component test phase requires the components to hold the data that is necessary for our system. For our tables, it has to be verified that the correct information is placed in the correct columns and the rows are displayed in the order that is necessary. In order to test the remainder of the components our team requires people who are not working on the GUI to test the components. This is because they are more likely to interact with the GUI in ways that were not intended by the developer. The data displayed will not be the data displayed in the system testing phase because it will not have the other parts of the system integrated into it. The main purpose of testing the GUI is to verify that the components act the way they are supposed to.

2.3 Integration Testing

After the unit tests stop revealing problems the next phase of testing begins. Integration testing involves taking the component and putting them together. The integration method that was used for this phase of testing is the sandwich testing strategy. This method allows us to test the interactions between multiple subsystems. The first testing sections in this phase are the adding, editing and querying of data in the databases. These sections have to be first because they must be correct before any further integration can occur. The sections that can be integrated and tested in parallel next are

the networking and database; this includes client and server side systems, the GUI subsystem and client side database.

2.3.1 Subsystem Integration

2.3.1.1 Database Integration

The Database will be responsible for storing all of the client side data. The tests needed for this subsystem will require that all data objects be stored correctly in a format that can be read for later. The first tests must require that the data is saved to the database. The tests for this will use a combination of the test drivers for the creation of data types and a driver to pass the objects to the database. Speed is taken into account for this section of testing because it may be required that thousands of objects be added to the database at any given point. To test run time, the time is taken before and after the addition has occurred; the time values are subtracted from each other and the run time is recorded. If the time is too high, then optimization may be required. To test that the database knows when to edit and when to add to the database text is outputted to the console describing what changes have happened to the database including the values it records.

2.3.1.2 Graphical User Interface

The GUI subsystem communicated with the client side database to retrieve data to be placed onto the display. To test the integration of these two subsystem it require that information is correctly extracted from the user is verified before placing inside of a new data object if it is correct and reject the data if it could not be verified. The data verifier will need to a test driver because entering data into the system through the GUI uses too much time and is less effective than automated testing. The test driver will create

many objects within the boundaries and some that are outside of the boundary through automated input. To verify that the test is successful text will be outputted to the console to display whether the content is correct or not and what information was places inside of the object through the objects print function. The GUI controller will have to be able to display data from the database. At this stage in testing the database should be in the testing stage and may not be stable. So test stubs will be used to test the GUI controller's ability to extract and display data from each of the objects on the display. To test this functionality the controller will need to have test stubs to extract the dummy data from. The data extracted will be displayed onto the GUI, and if the data is the same as the dummy data created then the test is successful.

2.3.1.3 Server Side Communications subsystem Integration

Testing for this subsystem requires that it has the ability to receive data from clients and send data to all of the clients. In order to test this without the client, test drivers will need to be created to act as the message senders and test stubs are needed to act as the clients to receive the new data. Having multiple test stubs in replace of clients ensures that if there are more than one client and the all receive that data that was pushed to the server database by the client test driver.

2.3.1.4 Client Side Communications subsystem Integration

Testing for this subsystem requires that it has the ability to receive data from clients and send data to all of the clients. In order to test this without the client, test drivers will need to be created to act as the message senders and test stubs are needed to act as the clients to receive the new data. Having multiple test stubs in replace of clients ensures

that if there are more than one client and the all receive that data that was pushed to the server database by the client test driver.

2.3.2 Inter-Subsystem integration testing

2.3.2.1 Graphical User Interface Subsystem and Client Database Integration

To test the interaction between the GUI and the database, the test stubs used by the GUI will be replaced by the functions inside of the client database. The same tests used for the testing of the GUI will be used excluding the functions that are involved in creating and updating data on the database. This is because functionality needed for this requires that the integration of the GUI and communications subsystem is required.

2.3.2.2 Graphical User Interface Subsystem and Communications Subsystem Integration

To test the interaction between the GUI and the communications subsystem, the test stubs used by the GUI will be replaced by the functions inside of the communications subsystem. To test the integration, you create an object using the GUI and submit a change. For the test to be successful the data should be added to the database and should reflect the changes made inside of the GUI. The tests need to be done on all of the different data types the GUI is responsible for changing.

2.3.2.3 Server Side Networking and Database Integration

This section describes the testing required to integrate the networking subsystem and the database subsystem for the client and the server. The test stubs used by the communications subsystem will be replaced by the data class. The data class is used to store all of the data server side but will also be used on the client side. The

communications subsystem will be responsible for converting data types to XML and back, as well as sending the XML file over the network to the client. The communications subsystem on the server side is also responsible for receiving new data, distributing new data, and adding it to the local database. One of the tests for these subsystems will involve the communications subsystem. Passing new data to the server database, the server will take the new data and add it to the tables. The database should be able to retrieve all the revisions requested by the other objects. The communications subsystem will request a range of revisions from the database to keep the other clients updated. This means the communications subsystem requires test drivers until the connection between server and client is established. The driver will pass XML data passes into the communications manager on the server, it is parsed and the resulted object is passed to the database where it is added/edited on the database. In order to confirm the test is successful the new object must exist within the database. The next section that needs to be tested is the data querying. Test drivers are needed to populate the database.

2.3.2.4 Client Side Networking and Database Integration

The Client communications subsystem is still responsible for converting XML to objects and passing them either to the client database or pushing over the network. For integration with the client side database test drivers are needed to generate data to be pushed to the client database. The data converted from XML to an object and is passed to the client database to be added or updated. The client database only has one function to add/edit data, so there only needs so be a series of repeated additions and

edits to test the database to be satisfactory. In earlier testing phases, the data conversion was verified to be correct so only passing of data objects needs to be tested.

2.3.2.5 Client and Server Side Communications integration

Testing of the client to the server is the last part to be tested in the integration phase. This is because it requires both the client and the server to be up and running. When both applications are running, the client tries to establish a connection with the server to start off. Once the connection has been verified, by popup message, the user will create a new piece of valid data and submit the new data. This data will be parsed and pushed to the server; if the data is sent back to the client then the data push was successful. The next step will be to attempt to have multiple clients accessing the server and have them push new data and see if each of the connecting clients receives all updated data. Once it is verified that each connected client has a copy of the pushed data.

2.4 System Testing

To test the whole system the application for the client has to be ran and the server has to be started. When the client GUI opens up the user presses connect to establish a connection with the server. At this point data that exists currently in the server should be pushed to the client in order to populate the client database with the most recent revision. All of the components inside of the GUI will be tested using the top down approach for initial testing. The main functions are tested, such as adding disease cases, adding supplies, and creating preparing, sending and receiving shipments. As new data is added to the database, the tester will make sure that the GUI updates the

map. System testing will also include the use of more than one client on the one server to make sure the data is being updated over the network. It may be worthwhile to recheck the entry of new data with invalid types to make sure that the invalid data is not pushed to the server.

3 Test cases for Creating, Sending, and Receiving Shipments

3.1 Unit Tests

Test	U1 – City Object Construction
Component Under Test	City Class (constructor)
Expected Input	1) String – *Province* 2) String – *City Name* 3) Float – *Latitude* 4) Float -- *Longitude*
Input for Tests	A. Valid Data B. Valid but large strings for 1,2 C. *Latitude* is greater than 90 or less than -90 D. *Longitude* is greater than 180 or less than -180 E. Integers for 3,4 F. NULL for 3,4 G. Characters for 3,4
Expected Output	A. Valid City object B. Valid City object C. – H. City object is not created appropriate notifications made.
Intercase Dependencies	None

Test	U2 – Shipment Object Construction
Component Under Test	Shipment Class (constructor)
Expected Input	5) Positive Integer or NULL – *Shipment ID* 6) Positive Integer – *Supply Type ID* 7) Positive Integer – *Quantity Shipped* 8) Enum – *Shipment Status* (PREPARED, SENT or RECEIVED) (cannot be SENT or RECEIVED if *Shipment ID* is NULL) 9) City* -- *Origin City* 10) City* -- *Destination City* 11) Date – *Created Date* 12) Date or NULL – *Sent Date* (must be NULL if not SENT, must have value if RECEIVED) 13) Date – *Expected Date* 14) Date or NULL – *Received Date* (must be NULL if not SENT or RECIEVED)
Input for Tests	H. Valid Data I. Negative integer values for 1,2,3 J. Character Values for 1,2,3 K. Invalid City pointers for 5 and / or 6 L. No date for *Created Date* / *Expected Date* M. *Created Date* in the future / *Expected Date* in the past N. Sent Date but *Shipment status* is not SENT or RECEIVED O. *Received Date* but *Shipment* status is not RECEIVED
Expected Output	D. Valid Shipment object E. – H. Shipment object is not created debug message output to console.
Intercase Dependencies	U1 – City Object Creation

Test	U3 – Shipment Object Conversion to XML
Component Under Test	Shipment Class (toXML())
Expected Input	1) Valid Shipment object
Input for Tests	A. New Shipment object with NULL *shipment ID* and PREPARED *Shipment_Status* B. Shipment object with PREPARED *Shipment_Status* and existing *Shipment ID* C. Shipment object with SENT *Shipment_Status* D. Shipment object with RECEIVED *Shipment_Status*
Expected Output	A. – D. QString consisting of properly formatted XML representation of the object (see below) <Sent> node should only appear for C and D, <Received> node should only appear for D,
Intercase Dependencies	U2 – Shipment Object Construction

```
<Shipment shipmentID="sh101">
      <ShipmentOrigin>
            <Location>
                  <longitude>10.5</longitude>
                  <latitude>10.0</latitude>
            </Location>
      </ShipmentOrigin>
      <ShipmentDestination>
            <Location>
                  <longitude>11.5</longitude>
                  <latitude>11.0</latitude> </Location>
      </ShipmentDestination>
      <SupplyTypeID>st123</SupplyTypeID>
      <quantity>50</quantity>
      <ExpectedDate>
            <Date>2010-11-15</Date>
      </ExpectedDate>
      <ShipmentStatus>
            <Created>
                  <Date>2010-10-11</Date>
            </Created>
            <Sent>
                  <Date>2010-10-12</Date>
            </Sent>
            <Received>
                  <Date>2010-10-15</Date>
            </Received>
      </ShipmentStatus>
</Shipment>
```

Test	U4 – Shipment XML extraction
Component Under Test	Handler (handler class for SAX model XML parsing)
Expected Input	1) Valid XML message containing properly formatted Shipment message
Input for Tests	A. Valid XML message containing properly formatted Shipment message with PREPARED status and NULL Shipment ID B. Valid XML message containing properly formatted Shipment message with PREPARED status and valid integer Shipment ID C. Valid XML message containing properly formatted Shipment message with SENT status D. Valid XML message containing properly formatted Shipment message with RECEIVED status E. Valid XML message containing multiple properly formatted Shipment messages (mixture of A through D) F. Improperly formatted XML (improperly closed tags) G. Properly formatted XML Shipment message with invalid data
Expected Output	A. Through D. valid Shipment object via the Handler's replyObject list E. Qlist of valid Shipment objects via the Handler's replyObject list F. No Shipment object created and Handler reports parsing error G. No Shipment object created, error reported
Intercase Dependencies	U3 – Shipment Object Conversion to XML U2 – Shipment Object Construction

Test	U5 – Shipment Database Insertion
Component Under Test	Data (wrapper class for QSqlDatabase) (data::add(Shipment*))
Expected Input	1) Valid Shipment object
Input for Tests	A. Shipment object with NULL Shipment ID B. Shipment object with positive integer Shipment ID (Updating of existing record for Shipment) a. Changed from PREPARED to SENT b. Changed from SENT to RECEIVED
Expected Output	A. Shipment Table in database should contain an entry with an auto-generated shipment ID, Sent and Received Dates should contain NULL B. Entry in Shipment Table with Shipment ID should be updated to the values contained in the Input object a. Status should reflect SENT, Sent Date should no longer be null b. Status should reflect RECEIVED, Received Date should no longer be null
Intercase Dependencies	U2 – Shipment Object Construction U3 – Shipment Object Conversion to XML

Test	U6 – Shipment insertion into client Data list
Component Under Test	ListContainer (Client data storage unit)
Expected Input	1) Shipment from XML extraction
Input for Tests	A. Shipment object with NULL Shipment ID B. Shipment object with positive integer Shipment ID that matches an existing Shipment C. Shipment object with unique positive integer Shipment ID
Expected Output	A. Shipment should be rejected with appropriate error messaging B. Entry in Shipment Table with Shipment ID should be replaced with the input Shipment object C. Shipment object should be appended to the existing list
Intercase Dependencies	U2 – Shipment Object Construction U4 – Shipment XML extraction

Test	U7 – City extraction from Database
Component Under Test	Data (wrapper class for QSqlDatabase)
Expected Input	1) SQL Query ("SELECT * FROM Cities WHERE (Lat = :la) AND (long = :lng)") (where Lat and Long are the location co-ordinates in an object)
Input for Tests	A. Valid Latitude and Longitude for a known city in the database B. Latitude and Longitude that do not correspond to a city in the database
Expected Output	A. Valid City pointer to a new City object B. No City generated and appropriate error message returned
Intercase Dependencies	

Test	U8 – Shipment extraction from Database
Component Under Test	Data (wrapper class for QSqlDatabase)
Expected Input	1) SQL Query (“SELECT * FROM Shipments WHERE …)
Input for Tests	A. SQL Query (“SELECT * FROM Shipments WHERE …)
Expected Output	A. Shipment pointer to shipment extracted from database • Should not have NULL for ID as database should have assigned one • Sent and Received dates should properly reflect shipment status
Intercase Dependencies	U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database

Test	U9 – Shipment List extraction from Database
Component Under Test	Data (wrapper class for QSqlDatabase)
Expected Input	1) SQL Query (“SELECT * FROM Shipments WHERE …)
Input for Tests	A. SQL Query (“SELECT * FROM Shipments WHERE …) B. Improperly formatted Query • Wrong table • Wrong syntax
Expected Output	A. QList of Shipment pointers to shipments extracted from database • Should not have NULL for ID as database should have assigned one • Sent and Received dates should properly reflect shipment status B. Error should be raised at query level prior to execution
Intercase Dependencies	U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database

Test	U10 – GUI data submission
Component Under Test	GUI
Expected Input	1) SupplyType from drop-down list 2) Quantity from text field 3) Source City from drop-down list 4) Destination City from drop-down list 5) Created Date from currentDate()
Input for Tests	A. Valid Data B. Invalid Data • Negative Quantity • Quantity higher then source location has available
Expected Output	A. Correct Data to be fed to constructor • SupplyType ID corresponding to selected SupplyType • Quantity input into text field • Latitude and Longitude from Source City • Latitude and Longitude from Destination City • Correct current Date • SHIPMENT_STATUS - PREPARED B. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database

Test	U11 – Shipment List from Client Data List
Component Under Test	GUI
Expected Input	1) Search Criteria via GUI
Input for Tests	A. Valid Data • Selected to and from locations via drop-down list o Test with 1 item select, all selected and half selected • Selected SupplyTypes via drop-down list o Test with 1 item selected, all selected and half selected • Min and Max via text fields in GUI o Min of zero, Min of largest current quantity, max of 2, max of 1000 • Shipment status via drop-down list o Test each of Created, Sent and Received B. Invalid Data • Negative Quantity for Min or Max • Max is less than Min
Expected Output	A. QList of Shipments that meet the criteria B. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	

3.2 Integration Tests

Test	I1 – GUI Shipment Creation
Components Under Test	GUI and Shipment Class
Expected Input	1) SupplyType from drop-down list 2) Quantity from text field 3) Source City from drop-down list 4) Destination City from drop-down list 5) Created Date from currentDate()
Input for Tests	A. Valid Data B. Invalid Data • Negative Quantity • Quantity higher then source location has available
Expected Output	A. Shipment Object with the proper attributes • SupplyType ID corresponding to selected SupplyType • Quantity input into text field • Latitude and Longitude from Source City • Latitude and Longitude from Destination City • Correct current Date • SHIPMENT_STATUS - PREPARED B. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	U10 – Gui Data Submission U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database

Test	I02 – GUI and Communication Subsystem adding new shipment
Component Under Test	GUI Communications Subsystem
Expected Input	6) Shipment data gathered from the GUI entered by the user
Input for Tests	C. Data from the GUI
Flow of Events	C. A new Shipment object is created based on data extracted from the GUI D. The Shipment is passed to the XML parser. E. The parser passes the xml string to the Socket
Expected Output	A. The Shipment object is successfully converted into an XML string
Intercase Dependencies	U2 – Shipment Object Construction U3 – Shipment Object Conversion to XML

Test	I03 – Server Side Communication Subsystem and Database add shipment
Component Under Test	Server side Communications Subsystem Server side Database
Expected Input	1) Shipment XML string received over the socket
Input for Tests	1) Shipment with valid XML string 2) Shipment with invalid XML string format
Flow of Events	A. The XML string is received over the socket B. The String is then validated a. The string is discarded if the data is invalid. C. The XML string is converted to a Shipment Object D. The Shipment object is passed to the Server Database E. The Shipment object is stored on the tables.
Expected Output	A. The Shipment object is successfully added to the table
Intercase Dependencies	U2 – Shipment Object Construction U4 – Shipment XML extraction U5 – Shipment Database Insertion

Test	I04 – Client Side Communication Subsystem and Database add shipment
Component Under Test	Server side Communications Subsystem Server side Database
Expected Input	2) Shipment XML string received over the socket
Input for Tests	3) Shipment with valid XML string 4) Shipment with invalid XML string format
Flow of Events	F. The XML string is received over the socket G. The String is then validated and converted to a Shipment Object a. The string is discarded if the data is invalid. H. The Shipment object is passed to the Client Database I. The Shipment object is stored on the tables.
Expected Output	B. The Shipment object is successfully added to the table on the Client Database
Intercase Dependencies	U2 – Shipment Object Construction U4 – Shipment XML extraction U5 – Shipment Database Insertion

3.3 System Tests

Test	S1 – Full Shipment Creation
Components Under Test	Full System
Expected Input	1) SupplyType from drop-down list 2) Quantity from text field 3) Source City from drop-down list 4) Destination City from drop-down list 5) Created Date from currentDate()
Input for Tests	A. Valid Data B. Invalid Data • Negative Quantity • Quantity higher then source location has available
Flow of Events	1) GUI invokes the shipment constructor with information from User 2) Shipment is sent to parser for packaging as message 3) Parser packages message and hands it to Socket for transmission to server 4) Server socket passes message to Parser for extraction 5) Parser passes extracted shipment to database for insertion 6) Parser passes "new revision" message to server 7) Client receives "new Revision" message and requests update 8) Server parser responds by requesting all new updates including new Shipment from the database and packages it into a message 9) Client parser takes message from socket, decodes it and inserts shipment into local data storage 10) GUI receives updated shipment information and displays it
Expected Output	A. Database, and Client possess the correct shipment information and GUI now displays the just created shipment B. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	U10 – Gui Data Submission U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database U11 Shipment List from Client Data List I1 Gui Shipment Creation

Test	S2 – Updating shipment to SENT
Components Under Test	Full System
Expected Input	1) Status change from drop-down list
Input for Tests	A. Status Change From PREPARED to SENT B. Status Change from PREPARED to RECEIVED
Flow of Events	1) GUI invokes the shipment constructor with information from User • Constructor receives previously created unique id 2) GUI invokes the Supply constructor with update on the inventory subtracting the quantity in the shipment from the on-hand quantity 3) Shipment and Supply are sent to parser for packaging as message 4) Parser packages message and hands it to Socket for transmission to server 5) Server socket passes message to Parser for extraction 6) Parser passes extracted shipment to database for insertion 7) Parser passes "new revision" message to server 8) Client receives "new Revision" message and requests update 9) Server parser responds by requesting all new updates including new Shipment from the database and packages it into a message 10) Client parser takes message from socket, decodes it and inserts shipment into local data storage 11) GUI receives updated shipment and inventory information and displays it
Expected Output	A. Database, and Client possess the correct shipment and Inventory information and GUI now displays the just created shipment B. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	U10 – Gui Data Submission U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database U11 Shipment List from Client Data List I1 Gui Shipment Creation

Test	S3 – Updating shipment from SENT to RECEIVED
Components Under Test	Full System
Expected Input	A. Status change from drop-down list
Input for Tests	A. Status Change From SENT to RECEIVED B. Status Change from SENT to PREPARED
Flow of Events	1) GUI invokes the shipment constructor with information from User • Constructor receives previously created unique id 2) GUI invokes the Supply constructor with update on the inventory adding the quantity in the shipment from the on-hand quantity 3) Shipment and Supply are sent to parser for packaging as message 4) Parser packages message and hands it to Socket for transmission to server 5) Server socket passes message to Parser for extraction 6) Parser passes extracted shipment to database for insertion 7) Parser passes "new revision" message to server 8) Client receives "new Revision" message and requests update 9) Server parser responds by requesting all new updates including new Shipment from the database and packages it into a message 10) Client parser takes message from socket, decodes it and inserts shipment into local data storage 11) GUI receives updated shipment and Inventory information and displays it
Expected Output	B. Database, and Client possess the correct shipment and Inventory information and GUI now displays the just created shipment C. GUI should reject attempt at submission and report reason for rejection
Intercase Dependencies	U10 – GUI Data Submission U2 – Shipment Object Construction U5 – Shipment Database Insertion U7 – City extraction from Database U8 – Shipment extraction from Database U11 Shipment List from Client Data List I1 GUI Shipment Creation