SE2832 Lab 3: The Golden Crust Pizza Company

Due: April 2, 2014 23:59

1 Introduction

Everyone loves pizza. It's the college dinner of champions. But did you ever think of what was involved in ordering your double cheese with bacon pizza and getting it to your door? After this lab is done, you'll have a better idea of this, as well as a better idea of how to develop tests from use cases.

Because of the increased scope of this project, you are permitted to work with a single lab partner. If you work with a lab partner, only one submission is required.

2 Lab Objectives

- Interpret use cases.
- Translate written use cases into activity diagrams.
- Generate test cases from activity diagrams.
- Write accurate and complete defect reports
- Enter defect reports into a defect tracking tool
- Track uncovered defects to closure using a defect tracking system.

3 Lab Overview

This lab commences a real world testing scenario. You will be writing test cases to verify the operation of the program. These test cases / test conditions should test, at least from the user's standpoint, the full functionality of the banking system as it is described to you. You may find the behavior of the system a bit quirky. That is by design; no system is ever perfect (or perfectly testable.)

You will also be provided with a completely developed version of the program as a Java jar file. This version has had defects injected and you are responsible for finding the injected defects within the system through your tests. The defects which you find are to be recorded in a defect tracking system.

4 Use Cases

The System Under Test (SUT) is best described using the use case scenarios described in this section. Pay attention to the sequences. These describe what the software should do. Any deviation from this description represents a defect within the software. Pay attention to the footnotes, as they may designate limitations in the system as developed thus far. This is, after all, cycle 1 software.

You will be randomly assigned by the instructor one or more use cases to develop tests for and to execute.

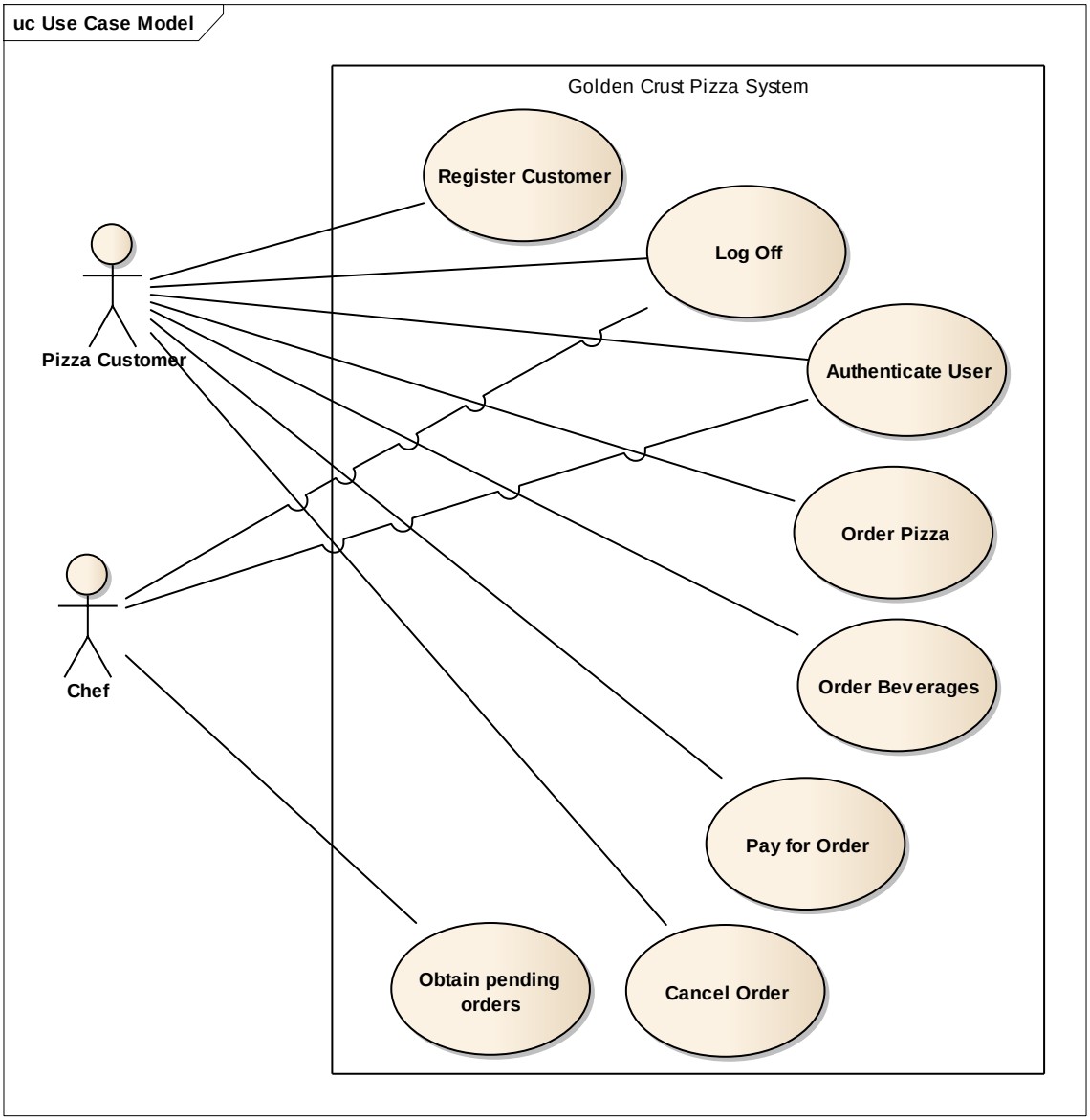

Figure 1 Golden Crust Pizza System Use Case diagram. Note the chef exists as a user with the username kitchen@goldenpizza.com and the password LetMeIn.

4.1 Register Customer

4.1.1 Actors

Pizza Customer

4.1.2 High Level Description

This use case will be invoked by a pizza customer who wishes to register for the Golden Crust pizza system. An account will be setup for the user, from which the user is able to access the online system.

4.1.3 Preconditions

None

4.1.4 Use Case Flow

1. User selects the register customer input.
2. System prompts the user to enter their name (first and last)
3. User enters their name.
4. System prompts the user to enter their date of birth.
5. User enters their DOB.
6. System prompts the user to enter their address and phone number.
7. User enters their address and phone number.
8. System prompts user to enter their e-mail address as an account name.
9. System verifies e-mail account.
10. System prompts user to enter a secure password.
11. User enters their proposed password.
12. System creates an account for the user and authenticates them with the system.

4.1.5 Alternate flows

5.a.1 The user is not 16 years old. System will abort registration, as they are not old enough to use the system.
7.a.1 Invalid address or phone number is entered. System will prompt the user to re-enter the information.
9.a.1 The e-mail address is already in use as an account. Prompt the user to enter a different e-mail address or allow the user to cancel account creation.
11.a.1 Password entered is insecure, in that it does not use at least three of the four forms of entry: capital letter, lowercase letter, digit, or symbol. Prompt the user to enter a new password or allow the user to abort registration.

4.3 Authenticate User

4.3.1 Actors

Pizza Customer

4.3.2 High Level Description

This use case will be use to determine if a given user is authorized to access the system. The user will be asked to provide appropriate credentials, and if these credentials are valid, they will be allowed to access the system.

4.3.3 Preconditions

None

4.3.4 Use Case Flow

1. User selects the log in functionality from the system.
2. User is prompted to enter their user id.
3. User is prompted to enter their password. (Note: Password should be fully case sensitive.)
4. User is granted access to the system.
5. Transaction log is updated with the operation.

4.3.5 Alternate flows

2.a User cancels operation. Use case aborted.
4.a Invalid username or password is provided. Message displayed to the user indicating that the user name and password combination was invalid. Transaction log is updated with an indication of the failed login attempt.

4.4 Order Pizza

4.4.1 Actors

Pizza Customer

4.4.2 High Level Description

This use case scenario will allow an authenticated user to place an order for a pizza. The user will be able to customize the pizza toppings as well as the size and style of the pizza.

4.4.3 Preconditions

1. User has previously been authenticated.

4.4.4 Use Case Flow

1. Customer selects the order pizza option.
2. System prompts the user to select either a Chicago deep dish pizza, a thin curst pizza, or a thick crust pizza.
3. Customer selects the pizza type.
4. Systems prompts the user to enter the number of cuts desired in the final pizza (4, 8, 16) which governs the size of the pizza.
5. Customer enters the number of necessary cuts.
6. System prompts the user to select toppings for the pizza. Selected toppings include mushrooms, sausage, peperoni, ham, and pineapple.
7. Customer enters their toppings.
8. System reports back to the user the ordered pizza for their confirmation.
9. User confirms their pizza selection.
10. Pizza is added to the customer's order.

4.4.5 Alternate flows

3.a User cancels operation. Use case aborted.
5.a User cancels operation. Use case aborted.
5.a User cancels operation. Use case aborted.
7.a User cancels operation. Use case aborted.
9.a User does not confirm their pizza order. Abort use case.

4.5 Order beverages

4.5.1 Actors

Pizza customer

4.5.2 High Level Description

This use case will allow a user to order beverages.

4.5.3 Preconditions

1. User is authenticated.

4.5.4 Use Case Flow

1. User indicates a desire to order beverages.
2. System displays the type of beverages available
3. User enters the type of beverage they would like to purchase.
4. System displays the beveridge choices for user confirmation.
5. User confirms choices.

4.5.5 Alternate flows

3.a.1 alcoholic beverage is selected. User is prompted to enter their birthdate.
3.a.2 System calculates the users age.
3.a.3 If the user is not old enough, the system rejects their order.
3.b.1 User cancels operation. Abort use case.
5.a.1 User does not confirm choices. Abort use case.

4.6 Pay for Order (Note: This use case is to be implemented in Sprint 2, and thus is not completed in the implementation available at the current time.)

4.6.1 Actors

Pizza Customer

4.6.2 High Level Description

This use case scenario will handle the user paying for their order. The order that is placed will be totaled and the user will provide a method for payment.

4.6.3 Preconditions

1. User must be authenticated
2. Order must be in the system which needs payment.

4.6.4 Use Case Flow

1. User has a desire to pay for their order.
2. System calculated the total for the order.
3. System displays the total. Prompts the user to confirm that this is their order.
4. User confirms that this is their order.
5. System calculated the sales tax for the order.
6. System displays total with sales tax.
7. System prompts user to enter their credit card number and expiration date.
8. User enters credit card number and expiration date.
9. System charges credit card for the dinner.
10. System displays receipt to the user.

4.6.5 Alternate flows

3.a.1 User cancels operation. Use case aborted.
8.a.1 User enters invalid credit card. System returns to the prompt.

8.b.1 Credit card is expired. System returns to the prompt.
8.c.1 User cancels operation. Use case aborted.
9.a.1 Credit card declined. Return to step 7.

4.7 Cancel Order

4.7.1 Actors

Pizza Customer

4.7.2 High Level Description

This use case scenario will cancel an order that has been placed.

4.7.3 Preconditions

1. User must be authenticated
2. Order must be in the system which needs payment.

4.7.4 Use Case Flow

1. User has a desire to cancel there order.
2. System confirms that the user wants to cancel their order.
3. User affirms that they desire to cancel their order.
4. Order is removed from the system.

4.8 Alternate Flows

3.a.1 User does not confirm their desire to cancel the order. Use case is aborted.

4.9 Obtain pending orders

4.9.1 Actors

Chef

4.9.2 High Level Description

This use case will provide the chef with the pending orders in order that the pizzas / beverages can be made. Once the pending order is retrieved, it can not be retrieved again

4.9.3 Preconditions

1. Chef is authenticated.

4.9.4 Use Case Flow

1. Chef has a desire to see the pending orders.
2. The system prints out in the order of submission the orders that have been received but not processed.
3. As each order is printed out, it is removed from the queue of orders to be processed.

4.9.5 Alternate flows

2.a.1 No pending orders. The system will indicate to the chef that there are no pending orders.

5 Assignment Specifics

5.1 Activity Diagram Creation

For the first part of lab, you are to take the use case scenario that has been defined and convert it into an activity diagram. The easiest way to do the activity diagrams is to draw them by hand on a clean sheet of paper. However, if you want to draw them on a computer, you may use Visio, or using enterprise architect.

After this has been done, use the activity diagrams to aid in the creation of a set of test cases that will fully test the operation of the system.

5.2 Test Case Creation

Since we now are dealing with a software package which retains data from one execution call to another, we now must initialize the system to a known state before we can perform certain tests. This initialization is the first step. The operations detail exactly what must be done as part of the test. Because this is becoming a larger system, it may be wise to group tests together in which you use a common set of instructions to initialize the system.

Because of the complexity of this program, you may need to create a lot more test cases than you did for previous problems. For each operation, you'll need to test with both valid and invalid combinations for all parameters, as well as testing that behaviors are correct given initial conditions and subsequent operations.

Since there will be more test cases, feel free to use an Excel Spreadsheet to document them if you so desire instead of a Word document; it may be easier to do this.

Test Case #	Description of Test	Initial Conditions	Operation(s)	Expected Result

In general, your testing should focus on the banking behavior of the software as expressed in the use case scenarios, not the specifics of the GUI. While you will create test cases for all use cases, you only will be responsible for testing those which are implemented in the current software, and due to the level of detail present in the requirements, it may not be possible to fully test all use cases to the precision that might be desirable.

5.3 Test Case Execution

Once you have created the test cases, download the jar file from the course website which represents the system under test. Execute the system under test, running the test cases you have created and log any defects that you find using the mantis system. You will find details about logging defects into mantis on the course website as well as in an addendum to this assignment.

6 Deliverables / Submission

Now that you have completed your lab assignment, submit the following lab report detailing your experiences. The lab report should be submitted electronically through the course upload page.

1. Introduction
   a. What are you trying to accomplish with this lab? This section shall be written IN YOUR OWN WORDS. DO NOT copy directly from the assignment.
2. Activity diagrams
   a. Submit your completed activity diagrams, one diagram for each use case.
3. Strategy
   a. How did you go about determining your test cases?
   b. How did you organize your test cases?
   c. Why did you choose the test cases and how did you approach organizing them in a logical fashion.
4. Test Cases
   a. What are the test cases which you developed? (Test cases should be formatted to be as legible as possible. It may be prudent to format the pages which contain test cases in landscape format.)
5. Bugs found
   a. Include a simple screen capture of the most basic Mantis report showing a listing of the bugs that you found in this program.
6. Things gone right / Things gone wrong
   a. This section shall discuss the things which went correctly with this experiment as well as the things which posed problems during this lab.
7. Conclusions
   a. What have you learned with this experience?
   b. How has this lab experience changed your attitude toward testing and when you should work on testing your projects?

If you have any questions, consult your instructor.

Appendix A. Writing Bug Reports

Bug reports are technical documents files by testers and other software users when a program appears to behave incorrectly. They provide tangible and traceable artifacts that document areas where software is performing in either an incorrect or unanticipated manner.

On any large scale project, thousands of bug reports may be received. Many will be of high quality and useful to the developers for quality improvement. But, many reports will also be bad. Instances of these types of reports may include:

- Reports that say nothing ("It doesn't work!")
- Reports that make no sense
- Reports that don't give enough information
- Reports that give wrong information.

Proper bug reporting requires practice, and it is a skill that is perfected not instantaneously obtained. While the specifics of bug reporting may vary given the environment and the uncovered bug, there are few things in common with all bug reports that can be used to make them better.

The goal of a bug report is to enable the development team to reproduce the failure which has been uncovered. Being concise yet detailed is important. It is important that the report clearly indicate the facts of the situation ("The output was 5 when it should have been 7"). Unless you are a member of the development team and have intimate knowledge about the system, do not speculate. Speculation simply increases the bug report and may send the developer down the wrong track. Bug reports should never be hostile or place blame on the development team, but rather should focus on conveying the information necessary to reproduce the bug in the development setting.

Each and every bug filed should have an accurate title which summarizes the bug in a few words. These words should completely and succinctly describe the issue. In some cases, a slightly longer title can add tremendously to the comprehension of the bug which needs to be fixed. For example, the title "Search Selected Text" conveys significantly less information than "Find and Replace dialog should default to last-searched term".

Once the title has been determined, a good bug report describes specifically what failed. Was an error code generated? If so, which one. Did the program crash? If it did, how did it crash, and did any diagnostic info get written to the screen. This can help tremendously when debugging a bug for the developer to ensure that what they are seeing matches what you saw during testing or program usage. If you can capture a screen shot, that is great, but do not simply file the screen shot without explanation. Same goes for a core dump on a UNIX system.

Next, a good bug report describes specifically what was occurring when the bug was found. This should be done in a clear yet minimalistic manner. Do not go back to turning on the PC and tell everything that was done, but highlight the key steps that you did to cause the problem to occur. If you can reproduce the bug, be sure that is mentioned as well. A bug that is extremely reproducible if one follows a set of steps is significantly easier to resolve than one which is not reproducible. When writing the steps, repeat them through after you have written to make certain you have not left out a step.

After documenting the process to create the bug, it is important to describe to the best of your ability the system configuration upon which the bug occurred. What version of software is being used and what OS?

What libraries are present that may affect the behavior? Have you installed add ins? Did you recompile the kernel with a patch from another vendor in order to add new functionality underneath the program?

A good bug report only addresses a single issue. If in testing you uncovered five problems, a total of five bug reports that should have been filed. Do not try to be efficient and combine bug reports, for this makes it difficult to assess and fix. When using a traceable process such as Unified Change Management (this is a topic for SE3800), each bug report will be linked with a bug fix and tracked to closure. If multiple bugs are submitted on a report, this important traceability step cannot be completed.

Appendix B: Bug Report Filing

To file bug reports in this lab, you will be working with the Mantis bug tracking system, available at http://emerald.msoe.edu/mantis/. Mantis is a complete bug tracking system, and it will be used again in the Software Development Laboratory.

The figure below shows a bug report template in Mantis. For this lab, you will be required to complete the summary and the description field. Additionally, if you are able to capture a screen shot showing the error, upload it as an attachment file. When submitting your issue, it should be marked with a private view status.

While you are working together as a team, each lab member must write up at least two defects and file them in Mantis. These will be graded individually.