Second Iteration Progress Report:
mycheapfriend.com

COMS 4156: Advanced Software Engineering

Team: CheapSkates

Team Members

Double Trouble: Waseem Ilahi (wki2001@columbia.edu)

Shaoqing Niu (sn2385@columbia.edu)

Dynamic Duo: Michael Glass (mgg2102@columbia.edu)

Huning "David" Dai (hd2210@columbia.edu)

The **Cheapskates** from **MyCheapFriend.com** will lend you our **Progress Report,** if you give us back feedback soon.

2. Unit Testing Report

As documented in our 2nd iteration plan, we are performing our unit testing on our parser subsystem in EmailInfo.java. This class is organized into a set of rules for finding lexical and syntax errors, as well as parsing out relevant information from raw email data. The tests will make sure that any error flags are processed correctly, and any parsable items are parsed properly.

List of Interactive Elements:

1. Parse From: logic
2. Parse To: logic
3. Parse Body with To and From Logic
    a. With an open (not password protected) To: address
        i. With a body-agnostic To: address
        ii. With a body-sensitive To: address.
    b. With a password protected To address
        i. With a Report message
        ii. With a Bill Message
        iii. With a nickname message

Equivalence Classes and Boundary Values of each list item:

A note about our tests: For all of our tests, we assume Strings are case insensitive.

1. Parse From: logic

    By the nature of email, we can assume all messages have an '@', this logic only tests the prefix (before the @)

    a. Equivalence Classes
        i. Valid String Length of 10 with only digits as contents [0-9]
        ii. Invalid String length != 10 (including 0-length strings)
        iii. Invalid String contents of anything other than digits
    b. Boundary Classes
        i. "6462294051"
        ii. "64622940521"
        iii. "646229405a"
2. Parse To: logic

    By the nature of email, we can assume all messages have an '@', this logic only tests the prefix (before the '@').

    a. Equivalence Classes:
        i. Valid string of that case-insensitively exists in the following list: new_account, reset_pass, unsubscribe, resubscribe, robot

ii. Valid string that, is a case-insensitive alphanumeric six-digit string
iii. Any string less than six digits that is not in the list (including length 0, although that would be undeliverable)
iv. Any string greater than six digits not in the list
v. Any six-digit string containing non alphanumeric characters.

b. Boundary Cases
i. “New_account”
ii. “123abc”
iii. “123” (blank string)
iv. “123abcd”
v. “123ab_”

3. Parsing Body
a. With a non-password protected To: address
i. With a body agnostic To: address (everything except robot)
1. Equivalence Classes
a. Valid string with any contents
2. Boundary Cases
a. “13joifjl;;;;k”
ii. With a body sensitive To: address (robot)
1. Equivalence Classes
a. Valid string with any letters (a-z) contents beginning with a ‘y’
b. Valid empty string
c. invalid string with any contents not beginning with a ‘y’
2. Boundary Classes
a. “yasf”
b. “”
c. “jfla;skjf;”

b. With a password protected To: address

For simplicity, I will define here four terms we use in the application.

An “Amount” is a string of the following pattern:

"\$?\d{1,4}(\.\d{2})?"

(to clarify: a 1-4 digit number, with an optional prefix of “$” and an optional suffix of a period and a 2 digit number.

An “Identifier” is either a “nickname”,

"[a-zA-Z]{2,2}[a-zA-Z0-9_-]{1,8}"

(to clarify: a 3-10 digit string starting with two chars and ending with 1-8 alphanumerics, tabs(‘-‘), or underscores(‘_’))

or a “phone number”, a 10-digit numeric.

Furthermore, when talking about multiple items below, assume they are delimited by a “space”.

i. Equivalence Classes
    1. The valid string, “Report”
    2. A valid string containing one a single pair of nickname and phone number in either order
    3. A valid string containing one *amount* and one or more *identifiers* and one or no occurrences of the string “me”
    4. A valid string containing morethan one pair of identifier-amounts
    5. An invalid string containing more than one occurrence of “me”
    6. An invalid string containing any number of “me”s and more than one amount.
    7. An invalid string containing unmatched pairs of identifiers and amounts
    8. An invalid string containing no amount that isn’t ONLY a nickname-phone number pair.
    9. An invalid string containing anything that is not an identifier, an amount, the string, “me”, or a space.

ii. Boundary Cases
    1. “report”
    2. “jon 6462294051”
    3. “bob Susie me 660”
    4. “bob 60 60 susie 32 jon”
    5. “me me 60”
    6. “bob 60 65 bob me”
    7. “bob 60 60 susie jon dough 75”
    8. “me 6462294050 bob”
    9. “#$25 bob”

4. Code Inspection Meeting

We chose the class that implements most of the business logic for our system, to do the code inspection on it. The name of the class is Controller.java. This class has a method called Handle. This method is called by the poller, when the poller has some text messages that it grabbed from the inbox. The text message is given to the handle () method. The handle method then “handles” the message appropriately. The inspection took place in the Room 633 MUDD. The meeting started at around 11:45 am and finished around 12:20 pm. Since there was a lot of code in the “unit”, we could not go through all of the code in time; however, we got through the important functionality in time. However, the

auxiliary functions were looked over, because we ran out of time. But, the code was tested against all the points in the checklist, and it managed to satisfy almost all of those points.

Michael Glass was the appointed reader the rest of the group was acting as the recorders. And obviously the TA (Jonathan Bell) was the moderator; however professor Kaiser was also present at the meeting. Michael began with describing why the particular unit was chosen for inspection and then went ahead to go through the code. The main method that we started from has a nice flow of logic. The method starts with handling of errors and then checks for certain other conditions that can make the function exit immediately. And when everything is satisfied, it handles the actual commands from the user.

The logic of the code was all satisfactory and there was no problem with the code in that context. The TA went through the checklist to make sure each point was covered in the inspection. All the checkpoints were satisfied, except there was a little problem with the last two.

- Check for appropriate code comments (is code's behavior clear).
- Check for appropriate code factorization into atomic parts.

We found a little inconsistency in the code, in terms of the usage of "true" and false **(First Point in the non-logic coding problems in the defects log)**. Also there was one other rather large Boolean expression, that might confuse the reader (suggested the TA) **(Second Point in the non-logic coding problems in the defects log)**. There was also some concern with the comments and there placement **(Third Point in the non-logic coding problems in the defects log)**. There was a TODO comment left over from earlier development **(Fourth Point in the non-logic coding problems in the defects log)**. The last thing that the TA suggested was to divide the main "part/unit" of the inspected unit, into logical components **(Fifth Point in the non-logic coding problems in the defects log)**..

From the code files attached with folder containing this document, we can see that the modified "unit" implements the changes suggested. A "diff" of the two sources will show the exact changes. The main method (handle()) is left as a few calls to other methods, that act as a fork to take the control over to the appropriate method to handle the feature the user wants to use. This process simplifies the main method and divides the entire logic into smaller pieces. As mentioned earlier, the changes can be seen in the file named "code_inspection_unit_modified.java".

5. Defect Log

Non-logic Coding Problems:

Controller.java (MyCheapFriend\MyCheapFriend-ejb\src\java\mycheapfriend\Controller.java)

The following defects were found in the code during the code inspection. The file name is given and the line numbers represent where the defect was in that file. The modified file has different line

numbers for similar code, for there have been changes to fix the defects. So the reference is the original file, not the current file that is in the system. The original file has been attached with the submission folder with the name "code_inspection_unit_original.java".

- **Line 155, 157, 173, 184, 382, 404, 414, 490, 510, and 515:** Inconsistency in the use of true and false. Some places we use Boolean.TRUE/FALSE and others we use true/false.
    - Fixed in revision 219, committed 12/02/09 by waseemilahi
    - Everything changed to true/false. No more Bollean.TRUE/FALSE.
- **Line 73:** Very large logic expression. May be confusing to the readers.
    - Fixed in revision 219, committed by waseemilahi
    - A method was created that broke the expression into multiple if's and returned a Boolean value that was assigned to the variable. The method definition is at the line 199.
- **Line 75, 78, 83, 89, 92, 403, and 436:** Comments were placed to the right of the statements rather than above, like in other cases ion the source.
    - Fixed in revision 219, committed by waseemilahi
    - The entire inconsistent comment placement was removed and the comments were moved above each statement they document.
- **Line 612:** Left over TODO comment from earlier development.
    - Fixed in revision 219, committed by waseemilahi
    - The comment was removed.
- **Line 62 – 145:** The main method is a bunch of if – else and switch statements. TA suggested encapsulating this further by dividing those functionalities (Switch's) in to separate function calls, to avoid confusion.
    - Fixed in revision 219, committed by waseemilahi
    - Multiple methods were created, each handling a subset of features. The handle method calls these methods in succession. And the appropriate method handles the task, if it can't/doesn't the handle moves on to the next call. Thus, the control flows through the code.

Logic Errors (bugs)

EmailInfo.java (MyCheapFriend\MyCheapFriend-ejb\src\java\mycheapfriend\EmailInfo.java)

- **Line 55: Error type on success wasn't always set to "no error"**

6. Controversies

There are no controversies among the group.

7. Code

All the code and data is inside the zip file containing this document. The files are appropriately named and also contain documentation to state their purpose.

Files for Code Inspection unit:

1) code_inspection_unit_original.java
2) code_inspection_unit_modified.java
3) EmailInfoTest.java