11/12/2018

Verification and Code Review

Project Deliverable 4

Andrew Yue-Keung Leung, Anthony Leung, Rundong Liu, Sherry Hoi Yan Ma, Sean Hung Nguyen
MACROHARD RAINFOREST INC.

Table of Contents

Testing

Unit/Integration Testing

Our unit tests can be found in the **Team7** repository in the following directory:

Application/src/test/java/com/teqlip

In that directory, folders to the **database** unit test files and **gui** unit test files can be found.

To run all unit tests, use the **mvn test** command while in the **Application** directory.

Acceptance Testing

U1: Create Account

1. Login as a TEQLIP user
2. Click Create/Manage Account
3. Enter a new username, role, and email for the new account
4. Click submit
5. Check email for credentials. Password is automatically generated and sent there.
6. Login with those credentials

U2: Upload Template

1. Login as a TEQLIP user
2. Click Upload/Manage Files
3. Enter a name for the template, and either type in a path or browse for a path to the .xls file
4. Click submit
5. Login as organization user and click Download Template to see if the name of the added template is displayed there

U6: Query

1. Login as a TEQLIP user
2. Click Query
3. Enter a valid SELECT statement on our database
4. Click Execute Query
5. A table with the requested data should pop up

Code Review

Strategy

While performing the code review, each team member should answer these main questions:

1. Can I understand what the code does just by reading it?
2. Are proper coding style and conventions being followed?
3. Is the code well-documented?
4. Are there multiple if/switch-case statements?
5. Is the code easily testable?

If a team member has done something particularly well that stands out, it should be commented upon, so all team members will know what direction to follow. If improvements can be made to a team member's code, constructive criticism should be provided so that all team members can benefit.

The following are some optional questions to consider during the code review:

1. Could we reuse code through generic functions and classes rather than copy and paste?
2. Are there places where we can use constants/enumerations instead of hard coding?
3. Does the code perform according to the design?
4. Does the code follow SOLID design principles?
5. Can we use design patterns to structure the code better?
6. Do I understand what the test cases are testing?
7. Do we have good test coverage?

Summary

Anthony

For the actionPerformed() function in LoginMenu.java, I can understand that the code is trying to process the username information and checking that it's correct. However, this function is doing too much at once. The use of design functions can greatly reduce the responsibility and complexity of the code. Here it is managing the try/catches. logging the errors, checking the login information, checking the roles, and creating the GUI all in one function. There's nothing wrong with the function executing these events, but it should not be responsible for all of it. If something goes wrong, we won't know exactly where it fails. It is nicely documented, which is how I know how the function works.

Andrew

I will be reviewing TEQMainMenuPanel.java and BodyPanel.java. First of all, the code is really easy to understand since the main chunks are separated into their own methods. The code is also simple enough to not require much documentation. However, when trying to navigate to the next panel, the code uses multiple if statements and switch-case statements. Also, if new panels were to be added, this code would need to be modified as well. So, not only does this code violate several SOLID design principles, but it could also be improved upon by following some design patterns. The code can also be easily tested by navigating through the GUI. Overall, I think the code is well-done in terms of simplicity and understandability, but we should refactor it to follow SOLID and some design patterns so that we do not have trouble in the future when we add more features.

Sherry

I am reviewing the get template features getTemplateFile() and getTemplatesName() in databaseSelectHelper.java. The functions are used for retrieving the Excel file from the database. There are appropriate comments which give clear descriptions. It also handles unexpected cases such as displaying an error message when a given template name is not found in the database. However, an exception should be thrown instead of just printing out an error message. The functions are following proper coding styles and conventions. The if statements are necessary since they are only used for logic checking. The getTemplateFile() method needs to check if the given template name exists in the list of template names. getTemplatesName() can call getTemplateFile() to retrieve the list of template names instead of calling the function in DatabaseSelector.java. The function only has a single responsibility which follows the SOLID design principle. In general, these functions are well coded.

Sean

I am reviewing the functions related to user account such as login and create account. These features involve functions such as insertNewUserAccount(), insertNewUserAccount(), etc. These functions are easy to understand by just reading it. The code is well documented, with internal comments to explain the steps when necessary. Most functions have a detailed docstring which give an idea of what the function does, however, a few do not such as insertUserLogin() and insertPassword(). All the functions only have a single responsibility, so it is easy to test. All code reviewed here does not use unnecessary if/switch-case statements. For this feature, I think Sherry did well in taking the time to refactor the code to follow the SOLID design principles. Many functions are created with the intent to be reused rather than copy and paste.

Rundong

I am reviewing the GUI functions for an organization user. The code is easy to read and is properly formatted. I personally prefer to limit each line to 90 characters and some lines of code exceed this limit. All classes are structured well in their packages and it is very easy to figure out the functionality of each class. Also, all the classes and packages have an appropriate name. However, there is no Javadoc written for the function, but some of the important functions are documented. There are some functions that are commented out and there are some unused variables and imports which may need to be modified later. No extremely long functions are found, and all functions are performing a single responsibility. Finally, the code may not be easily tested since it is a GUI, but since all the functions are separated, they can be tested properly.

Debriefing Meeting

The video recording of our code review debriefing meeting, can be found at:

https://youtu.be/lXp6KYYPDYY