Code Review

CSCC01

Abraham Tsang, Chengli Yang, Vijanthan Thiruchelvarajah, William Rutherford

11/15/2017

Code Review Strategy

Who reviews what:

- Vijanthan: UI to view list of assigned questions, and a specific question's details.
- Abraham: The assign_questions UI as well as it's designated insert function.
- Chengli: Standardization of constants throughout the project, Insert question function in create MC.
- William: All database related classes and the create_mc UI.

What you need to talk about:

For each class we analyze in our code review we will discuss the code that we are reviewing in four separate categories in this order.

Style:

Talk about coding style, documentation, naming convention, copy and paste, constant usage. Will it be difficult to make changes in the future?

Design:

Can we structure the code better? Is anything redundant? Is it what we agreed to design? Can we make use of interfaces/abstraction to improve the structure?

Functionality

Is it working as we intended it to? Does it handle exceptions correctly? Are there inputs for which the code can give obscure results?

Testing

Is the test coverage good, and can you understand what the tests results indicate from the shown messages?

Code Review Summary

Vijanthan reviewing work of Abraham:

In general, the style of the code written in the View Assignment UI and the UI for viewing a question's details is pretty good. The naming convention for variables was consistently in CamelCase and your choice of helper functions to avoid excessive copy and pasting proved to be very useful. The UI for viewing the question's detail however had a bit of copy and pasted code from the create MC UI and what would've been better is to modify the create MC UI to change to the UI you need given a Boolean flag input. Another more organized way would be to use an abstract class in which both the create MC and the view question details UI could both inherit the shared features. The code amongst both classes also could've also used more commenting so other developers, and even you when you revisit the code after some time, could understand the code quicker.

The UI could have been designed better. There is excessive back and forth movement between two different JFrames to just view the details of one question. A better alternative is to enlarge the current frame and simply display the contents of the question to the right of the question's list itself. This way everything will display on one frame. The functionality of the classes is excellent, it works just as we'd described in our team discussions. The testing is also good, it is automated and the questions that were assigned appeared on the list as expected.

Abraham reviewing work of Chengli:

The style of the code for the assigning question UI and assigned question insertion function is great, there is plenty of commenting and CamelCase is followed on all variable names. Good usage of helper functions when the list needs to be refreshed. The choice of using a JList on a ScrollPane was an excellent choice as any other choice would have been limited in size. When inserting assigned questions however, a major design flaw exists when PreparedStatements are not used. By not using a PreparedStatement you put your database at risk of SQL Injection in which any user can perform malicious attacks when inputting a semicolon and a SQL query. Another design flaw is that you do not provide any more question details than the question ID and the question text. If multiple questions had a question text like "Please choose the best answer" then you would not able to distinguish between them. A quick fix to this would be to display the question's label rather than the question ID. Another fix would be to simply display the contents of the question on the same frame upon clicking. Regardless, the result of the code is working as intended in our tasks document and the tests clearly verify that the assignment of a question has its' insertions reflected in our database.

Chengli reviewing work of William:

Good style, the choice of variable names was carefully selected in such a way that commenting was not very necessary. Excellent constants used to standardize the colour scheme across all UIs in our project. Using such constants allow us to, in the future, make what would've seemed like a big change only require the change of one line. It would however be better if more parameters were standardized, such as button positions and button sizes. A clear usage of this is because the back button on all forms should generally be in one specific location throughout the project. The insert function makes use of PreparedStatements, great to prevent SQL injection. The insert function accurately handles exceptions well and denies any database insertions from occurring in such cases. Both the standardization and the insert function were implemented as discussed in our tasks document. Changing the colour of one of the constants changes the colour of all related colour parameters throughout the project as intended.

William reviewing work of Vijanthan:

Backend uses the standardized variable names like q, conn, and st. Anyone familiar with SQL queries would understand their meaning, especially in context. Some variable names are understandable, such as q_id for question ids, or ass_id for assignment ids. However, the input variables for "insert" includes an array called "options", which are actually the possible 'answers'. More care in the future should be made into keeping the terms use consistent to be clearer to new programmers reading the code base. The strangest design choice in the entire backend is that ids are sometimes strings and sometimes integers. This means the function "Integer.asString()" is used more often than required. Functions whose comments say they return or take in an id might take a String or might take an Integer. This is confusing, because it leads to more confusion between labels (always strings) and ids (sometimes strings) which are both used to identify a question/answer. It does seem strange to have two variables that are interchangeably used to uniquely identify a question. The id should be exclusively an integer for input and output of functions, and the text of the question should be used as the label for the user. "label" should be entirely removed. The backend code is extremely functional. Some return a boolean whether their operation was successful, and they throw an IllegalArguementException if they are passed incoherent data. The backend tests pass for all tests that would happen in normal use, and some that don't. However, this relies on the frontend classes doing proper validation of user inputs.

Create_MC's variables are mostly the names of the buttons, most of which include the term "btn" which is good for classifying them. This class is very well designed. It follows the same order as how a user would use it. First, the user inputs data, then it is validated by the application. If they ever do something invalid, they are taken back to the last step. This insures the data inputted into the database is legitimate data. The UI in this class is very functional. It leads the user along, and makes sure they never do something invalid. All validation tests pass for Create_mc.

All the buttons for the main menu start with "btn", and are initialized exactly the same way. They use the standardized colours, but not sizes. The design of the main menu is quite elegant. All buttons in this class work as expected. The validation testing for the main menu passes in flying colours.

Code Review Debriefing

Our style throughout the project is quite inconsistent in places, we are using camelCase naming convention sometimes and pothole other times, integers for ID sometimes and string other times. We need to be more inconsistent with this, as well as follow a specific tabbing convention on new lines. We also need to think more about how we expect to scale our current code to the future. An example is how we're creating things as if we only have one assignment, but with a simple change it should be functioning for multiple assignments. When describing our tasks for the week at the start of the sprint we should be explicit in saying how we're going to use this code now and in the future so the developer knows what to watch out for. One of our inconsistencies were increasingly occurring more often after one specific commit, so instead of following past code assuming its correct we need to identify if it is correct or not and if it is not correct then we should fix it before continuing. Below is a video showing our general summarized recommendations to the person who's code is being reviewed with all other members listening.

https://www.youtube.com/watch?v=f7gbe3WeuFU&feature=youtu.be