La Vité

Vilen Milner
Michael Rossinski
Qi Hang Yang
Tianyi Zeng
Arnob Talukder

Table of Contents

Code Review Summaries

Vilen Milner

Inspecting the project showed that just about every single aspect of the code review strategy was violated in some way.

- Documentation: there is none, or almost no documentation present anywhere in the code. This becomes a problem when there are things present that are not immediately obvious, such as when the same fields appear more than once inside a form, the solution (if any) has to be explained in some way and it is not.
- Naming Convention: We do not have a strict naming convention for fields, so for example the field “health_ref” in InformationOrientationServlet.java might mean “health referrals” but can mean something else too, only the person who wrote it knows and this can be a problem if there is a referral field in another form that uses a different convention.
- SOLID: most of them are violated, the most glaring one is the violation of the open/close principle, adding new fields and forms requires changing multiple files.
- Code Structure: While each form jsp, js and servlet has a general form it follows, there is a lot of it that can be written a single time in order to reduce lines, many fields are of the same type and thus the structure of said field can be defined once to be used by other fields.
- Understandability: Most of the code is understandable, but there can be some instances where this is not the case. This ties into the naming convention problem. Also, the code in the constructor of every servlet is not very readable in its current state.
- Proper Functionality and Testing: Since there is no testing suite at all, it makes it hard to catch bugs and ensure proper functionality of the code. The solution here is to obviously make a testing suite, that can be run automatically/mostly automatically and ensures coverage of the form’s functionality.

Qi Hang Yang

- Documentation: A very small part of the overall code is documented well. Some parts are not documented because the code is self explanatory. However, most of the written code that is not intuitive do not have proper commenting/comments as to what the code is doing. This is bad if the team that is working on the code changes or multiple people work on that certain file. Many of the files do not have a description as to what the purpose of the file/class is within the scope of the project.
- Naming Convention: There is no convention as to what the variables should be, so the result is that in the code, some variables are camel cased while others are spaced with underscores. There are also no patterns as to what concatenated or short form of words will be. An example of this is one of the fields in needsAndAssessment servlet where there is a clear acronym of "NOC". Some people may in the future using a different acronym for the same name/variable, which is not desirable.
- SOLID: It is very clear that SOLID principles were not used since the code do no follow any of the principles. There are no interfaces or any indications of the principles being applied.
- Code Structure: It is somewhat acceptable in terms of separating the classes. However, some of the classes or functions are unnecessarily large. These classes, like the servlet files, could be divided into more appropriate sized class/functions.
- Understandability: It is very hard to understand what the code is doing without documentation (which there is close to none). There are many components that are working together to create the working product and without one component, the code will not work. So if a new developer was to join the team, he/she will need to understand most of the overall code to be able to start developing.
- Proper Functionality and Testing: There are no testing suites in place for the time. So this part of the project must be improved on. Bugs and edge cases are solely dependent on limited manual testing with the team.

Tianyi Zeng

While inspecting our overall project, there were several problems that I believe needed to be addressed. Firstly, the overall design of our backend to front end structure requires several implementations every time a new field should be added. This includes setting the constant value for Field, as well as adding additional lines of code into the proper Servlet class and javascript file. This can pose a problem for the long run as there may be many additional templates, such that finding the servlet/js files might get tedious. Secondly, a lot of implement so far of code is copy and paste work. We can change that by perhaps creating a class for each template that holds an ArrayList to store all the templates fields, with a push/pop option to add/remove fields. Any future add-ons to the template can then be implemented as methods. Good things about the design so far is that we have a clean well documented front end that functions perfectly with our backend and database. This allows us to easily test the entire program itself, using mock classes and execute different test scenarios, instead of only being able to test individual components.

Michael Rossinski

During the process of reviewing code, many problems were prominent with the overall design. The biggest problem is the longevity of the code and how easy it is to upgrade it. In order to add additional forms to the system, 3 different files (jsp, js, and Servlet) have to be created along with a script and table scheme for the database. This design is problematic because the 3 files that have to be created follow the exact same template and convention as the files for every single other form. This results in many form files being completely identical. Another problem with this design is that some forms contain fields that are present in other forms, causing overlap in the implementation of individual forms. I believe that to fix this, field creation must be universal and not stored in individual form files. Implementing fields in this manner would result in code that is far less cluttered, as well as easier to read and interpret. This would also clear up any problems with naming conventions.

Arnob Talukder

It was clear during the code review that many problems were hindering future development; for both the short term and long term. In the short term, our messy code will overcomplicate a lot of things. For example, each templet needs 4 different files – servlet, js, jsp, and a sql – and three of the four files (excluding sql) are littered with duplicate code. This not only makes debugging and understanding the code harder, but also negatively affects integration of different forms with overlapping fields, as there are no standard naming conventions. Due to this, if a name must be changed in one file, attest three other files get affected. In the long term, due to the lack of organization and convention, adding more templates will also be very problematic. This is because more code will need to be duplicated, and all other templates will need to be parsed to use the correct names for pre-existing/overlapping fields.