CS 151: HW 2, Code Implementation

Summer 2014

Group Members:

Edmund Dao, Brian Lee, Dennis Hsu, Luca Severini

To start with our review of the assigned code we we're given we will cover the five code quality metrics that Horstmann describes: Clarity, Consistency, Completeness, Cohesion, Convenience. In our assimilation of this project, we came across multiple accounts of these coding quality metrics that we now realise are more difficult to see when drafting a project. The groups code specifications that was assigned to us had many inconsistent faults between the CRC cards, UML documentation, and the Use cases. We started by reading the CRC cards and were dumbfounded trying to make sense of the strange collaborations and responsibilities. Cautiously, we moved on to the UML diagram and, almost in tears, realized their UML was similar to our own, with no relationship to the CRC cards. The use cases didn't help us much, as they were generic and the UML was much more useful. We devised a plan quickly for implementation while noting some of the big coding metric issues such as, the lack of unique IDs for jobs, applicants, and interviewers, managed by the system (rather than the user), and the lack of cohesion in the "HireMe" class, which was assigned functions unnecessary to it's function.

First off, for the consistency metric, the management of the groups' classes, done by the "List" classes, are straightforward and consistent, allowing for easier management of the lists of job/interviewer/applicant objects. Furthermore, the naming of functions/classes and variables are consistent throughout their UML.

Secondly, the overall clarity of the design specifications was intuitive yet could have been more thoughtful. The naming of many methods such as the getter/setter methods were simple and clear. However, the naming of the classes “JobList”, “ApplicantList”, and “InterviewerList” were confusing because they claim to be lists in the title, but nowhere in the class are lists utilized. Furthermore, the naming of these classes are confusing and misleading; these classes are more than lists, they manage objects and operate with much more functionality. Instead, these classes act to store, add, and delete the objects. When we realized that the classes were managing the objects and storing the values into a HashMap, we decided that renaming the classes to names like “JobManager” would better match the role that these classes play in the program. Yet another clarity issue that we found was the naming of the class “HireMe.” At first glance, we thought that was a button that a user pressed to hire an applicant once they get a job. This takes us to the next quality metric, cohesion.

The cohesion within the HireMe class was difficult to follow because of its diverse role it played in the program. HireMe was documented as an overall manager class to handle the job/applicant/interviewer classes while also rating applicants and assigning interviewers to applicants. Despite its top hierarchical position as the primary use, it contained the functions rateApplicants and assignInterviewers. Also, the use of a HashMap for the data structure storing interviewers is not logical since interviewers contain only 1 attribute, so the hashmap will not work (since a hashmap need two values for indexing). On the other hand, the cohesion in the documentation was good because of the logical organization of their classes. We found the way they utilised their “List” classes to store and handle their respectable objects to be cohesive and well done.

The next quality metric Horstmann describes is Completeness, the overall ability for each method/class to handle all operations pertaining to it. As such, we found issues in the

documentation that were missing. For example, the documentation specifications do not cover assigning multiple interviewers to an applicant, so each applicant is assumed to be assigned one interviewer and receives only one set of 3 integer ratings, which is incorrect by the assignment standards.

Lastly, Horstmann talks about the convenience quality metric, which describes how easy it is to do common tasks in the program. Overall, the group did a good job with the addition of getter/setter methods, allowing for the easy access of different classes and methods to communicate with each other. Our ideas for implementing this program was very similar to this specification and relationships were clear within the final blueprint.

In the end, we made minimal changes to the design documentation to better suit the functionality of the program. We added iterators for each hashmap list so that the program could display the 3 lists: jobList, applicantList, and interviewerList. We modified the display methods to pass more parameters than just simply printing info to the console so that we could utilize the functions when assigning an interviewer to an applicant. For example, we made sure that the displayApplicantList provided applicant name, email, interviewer, and rating. Also, we made the job and applicant ID's auto-generated to allow for easier management and to prevent the user from accidentally creating jobs with the same ID. All in all, the implementation followed the given design, with only minor modifications.