Team HuSCII – Group 2
TCSS 360A – Spring '15
Changes

Changes since Deliverable 2:

Changes in collaboration and responsibilities:

The way our group dynamics are at currently, each member of the team somewhat "specializes" in a class. Our division of labor followed this dynamic, which made it easy for us to delegate tasks and also avoid stepping on each other's "toes" while working on our code. Each member's responsibilities have not changed significantly since the last Deliverable but instead new responsibilities were added. For example, Ian and Thida specialized in Job.java and JobController.java. They both had to work more or less together on these classes because of dependencies in the code but with the addition of the console UI, their delegated responsibilities branched further apart. Ian's new responsibility was with the new BusinessRules class and Thida spent her time in the AdministratorConsole. As for Duy, he was delegated User and UserController and then had the ParksManager console and Main console added on to his responsibilities.

Another significant change to our group's workflow was the addition of using the "Issues" feature in GitHub. Utilizing the "Issues" feature made delegation of tasks extremely easy for the group and it was possible for us to get real-time progress checks of issues in our code. We were very happy with the "Issues" system and relied heavily on it to document bug fixes, ongoing tasks and also seeing what was resolved.

Our group meetings also saw changes since Deliverable 2. We noticed we were using the whiteboard more often during meetings. The whiteboard forced us to stand up, move around and interact with each other more. The whiteboard was home to our agenda now, the list of tasks to be delegated and our pseudocode when we group debugged. Another change in our group meetings was documenting the task matrix as a group, instead of individually as it was done in the past. After Josh brought to our attention ways to improve our task matrix, we decided delegating tasks during meetings resulted in a more detailed matrix, better accountability of responsibilities and an overall increase in group awareness of tasks assigned.

As a group, we were more involved with our code reviews than we had been in the past. We found putting up the code on a large screen in the meeting room helped direct our focus and effort in going line by line of each other's code and finding issues. In a sense, we were pair programming as a group and this made debugging effortless because of our collective minds being in one room. Prior to these changes, we identified issues but also assigned the solutions to be done individually which sometimes slowed us down.

Changes in code, classes and UML diagram, implementation

Team HuSCII – Group 2
TCSS 360A – Spring '15
Changes

Our class diagram has undergone slight modifications in terms of cutting code "fat" and rewiring how Business Rules are implemented. As of Deliverable 2, each user had its own class in our program; Volunteer.java, Administrator.java, ParkManager.java. We have all but cut out ParkManager.java, deleting Volunteer.java and Administrator.java. The reason for this change was because Administrator.java at present only had one function, searching for volunteers of a particular last name, that is now encapsulated in UserController.java. As for Volunteer.java, it was no longer needed because every single one of its methods, like signing up for a Job, were already implemented by JobController.java. ParkManager.java, however, survived the deletions because its function of finding a Park Manager's job seemed more personalized to a User, than say finding all Volunteers of a given last name.

Another major change to our class diagram was the addition of BusinessRules class. We felt it was better to encapsulate any Business Rule that cannot easily be handled by the console to be encapsulated in its own class. This way, our classes are able to go to one source to change and validate data before adding a Job. This change also made it easier for us to track down Business Rules as they are all conveniently located in one class. Along with creating a BusinessRule class, a test class was written with it.

As per project requirements, we moved from reading and writing our user and job data from text files to utilizing serialized objects. There were minimal changes done to our code since the relevant classes that needed to be modified were only UserController.java and JobController.java. All other classes were unaffected by the change to Serialized objects because we were simply serializing the Jobs collection object and the User collection object. The serialized objects when read are casted back to their collection types and used just as they had been prior to implementing Serializable.

Our classes for implementing the console and UI of the program undergone constant changes since the last iteration. The more significant changes done involved implementing menu selecting where they were not present before. For example, for Deliverable 2, Volunteers and Park Managers had to manually type in park names or park jobs in order to view or add them. This was error prone and relied heavily on the user for correctly formatted and valid input. Instead we implemented menu selection in these cases. What was also fixed was a bug in our system whenever the user had no jobs to choose from. Our buggy version resulted in the user getting stuck in these instances, unable to move forward in the console. The solution was a matter of checking for empty lists and alerting the user for when no jobs were found. Other changes involved in adding in Business Rules checks that can be implemented in the console, if they were not already in BusinessRules class. This

Changes to test

After Deliverable 2, our unit testing had major refactoring done to include more boundary cases as mentioned in class. In addition to the additional testing for boundary cases, we also had to modify the code to work with the new BusinessRules class we implemented. Along with that, we implemented a test class for BusinessRules. Now that we have so many tests running around, we decided to write an AllTests class that ran all of our tests. This based off a suggestion from in class and it made automating our tests far more convenient. Along with the refactoring of our tests, we had to create more tests input files to accompany the boundary cases, such as a maxed out list of jobs minus one job or an entirely maxed out jobs list. We had to carefully construct the job data and user data inputs to match the various boundary cases as well.