Individual Report

Team Organization

Initially as a team we figured out our strengths and weaknesses just by discussing what we had previously worked on. We understood that the tasks given forward could not have been done by just splitting everything into equal parts. The smartest choice was to give each person a task that suited them and take advantage of each one's strengths. This would have meant for higher quality production of submissions.

By using Belbin's roles, we could figure out each member's role and find out what else we needed to make our work more efficient. It was all working out in the beginning, we had our Co-ordinator that was keeping everyone down on their feet and focused on their tasks, a few Teamworkers that kept discussing the tasks in hand finding exactly what had to be produced. We also had the Completer Finisher that could put everything together, polishing it, perfecting it and then submitting it. It seemed as great team structure and synergy. As time passed and all the courseworks from other modules started rolling in everyone's role changed. We had all succumbed to the additional work and just started slacking on our tasks. The team co-ordinator had other priorities, the Teamworkers just stopped caring about the subject and the Completer Finisher had nothing to perfect since we had all stopped working. It all started going downhill. As time passed and the deadlines were coming close we all started being interested in the subject again. After we had a meeting we had all started coming back with a few changed roles. One became the Shaper being driven by near submission date and motivating all of us into putting all our effort in making exceptional work. We though that we had overcome the dip, but as we figured out when more courseworks came in it would have been the same thing all over again.

When groups are formed randomly with people being strangers to each other in general its difficult for them to bind and work together. It can be relatively effortless when the work given is limited, but in extensive projects even a minority of people not communicating properly with the team brings great disadvantage to everyone. “Communication is key”. This was our Team’s barrier that we encountered and that we had to overcome. But also, this made working in a group more difficult than simple individual work. Overcoming this barrier cannot be easy since it depends on each person individually, even the lightest hesitance can disrupt the team’s communication and synergizing properly can be difficult. A way of overcoming this barrier is with discussions, several meetups and by being open to constructive criticism, normal conversation and by being motivated in learning and achieving greatly in the given work. Unfortunately, this couldn’t work out for everyone since one of our members hesitated to communicate with all of our team. After trying to get in touch with them several times the responses were minimal. Borderline submission they would present some effort in the courseworks adding to the final product, but the problem was not a subpar submission. It was the problem of communication, them not communicating with the rest of the team properly created that gap made everyone feel unsure and became a detriment in everyone’s work.

Software Development Process

Understanding the basics of Software engineering concepts and principles was key in making a more efficient software life cycle for our team. Splitting the whole project into modules and then completing each module one by one had been productive method. Perfecting that module and then moving on to the next was the best way in creating a high-quality software system.

Firstly, figuring out all the requirements our system needs, finding the most important ones and the least important, made the production of a prototype easier and more accurate to the Customer's needs. Secondly, designing the system before constructing it with the use of our previously made case descriptions and class diagrams helped us save time and effort and making the whole idea easier to grasp and apply in action. During the construction stage everything we had developed so far came together into making a sufficiently good prototype where we had the chance to improve into producing a high-quality system, implementing all the customer's requirements and needs. For monitoring version control of our whole system, we used GitHub, where everyone working on different classes could constantly upload their new versions in a shared repository making it easier to go back for any abrupt corruptions or data loses. Constantly updated code made testing a lot easier since pulling all the classes from the repository was one simple command.

"Programs are not only read by computers", this phrase was said during a lecture and it stuck as time passed. Keeping good programming style in the system meant for more readable code where bugs were found easier. Our thought coding this system was structure, simplicity, consistency and commenting. Everyone following the same standards saved us substantial time and made assembling everything together working smoothly from the first try.

During the code constructing stage as a team, we had good time manipulation. We had started early creating a simple GUI interface for our Menu, just so we had something as a base to work out of. By using our already made case description gave us a better understanding of the basic and alternative flows of each part of the system. Also, the additional pre-conditions were considered, creating a well-rounded and structured initial model. The UML class diagram was also used to better understand the layout and path our flows needed to take, and additionally to identify and implement the correlation between different classes, their attributed and methods.

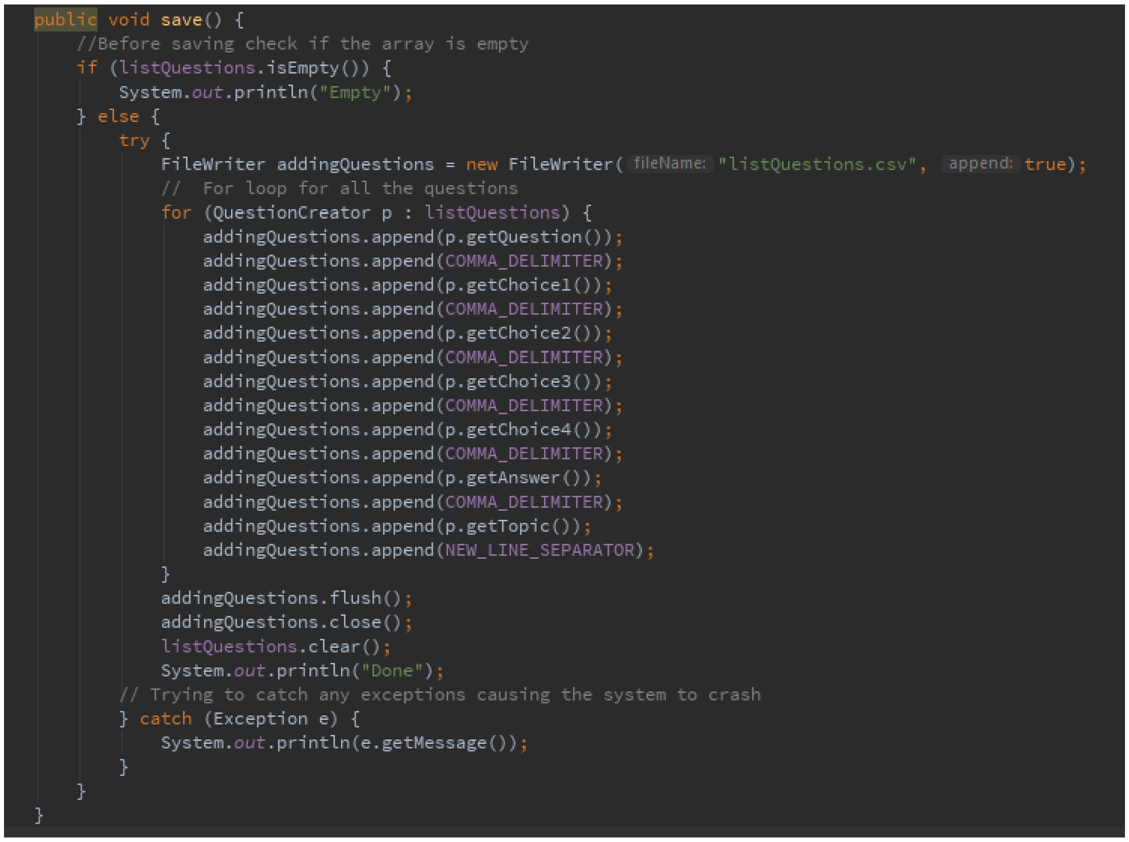
One of the techniques learned but wasn't implemented correctly was the Gantt Chart. Although we had created one and balanced the tasks into equally spread time slots, it simply wasn't executed correctly. We just didn't seem to follow the schedule. We had found ourselves being more productive in short bursts like 2-3 days and then stopping for a bit before proceeding again on to the construction of the system. Granted we could see our milestones and deadlines easier it was just that helpful for maintaining a more consistent workflow.

Our risk management considerably good, we never fell behind dude to unexpected events. Constantly updating the files in GitHub meant that we had several versions, so any corrupted files could have been downloaded back to their previous version and additionally data couldn't have been lost since they were all uploaded online. We had all used the same programming language (Java 8) and all used JavaFX for our system so everything was binding perfectly for all. Facebook group chat meant things that needed to be discussed where reviewed immediately, additionally meetings where easily scheduled making our workflow on the system as accurate and as satisfactory as we could. Frequently we would all refer to the initial stages of producing the system figuring out if the customer's needs and wants were met correctly but also if our time management would be accurate into finishing and perfecting the whole system on time.

Quality Criteria

Reliability

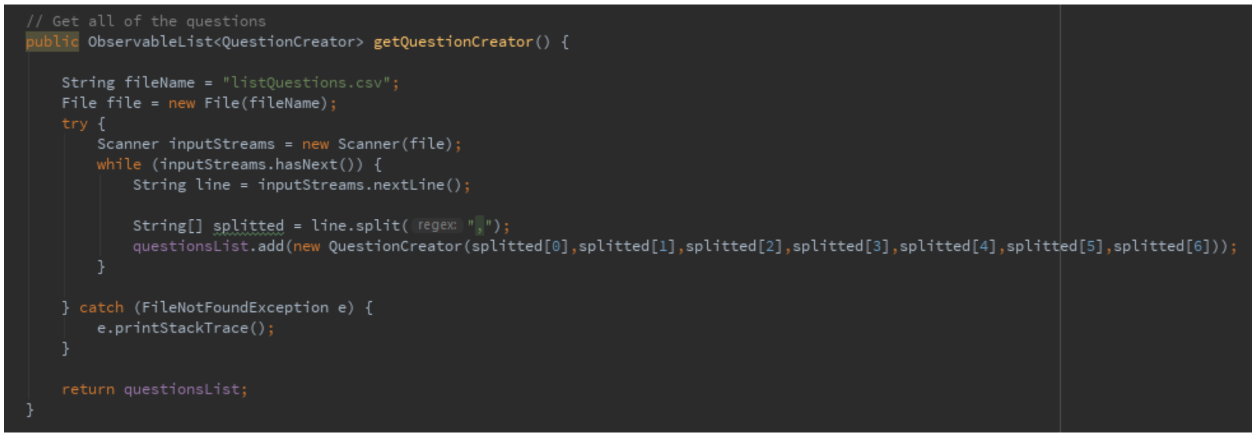
During the creating of the system, similar code had been used in creating all the classes needed to make up the completed system. When the tasks had been split up and everyone was working on their part any bugs and errors that appeared where instantly worked on and fixed before being implemented in the system. This made the system more tolerant and prevented it from having fatal errors that could have corrupted the accumulated work and data. At no point during the construction of the code we had any sort of failure, this saved us plenty of time in testing for any errors that may had appeared. We had all tested the system in its limits trying various inputs to see where any mishaps would show up. We had uncovered a few but quickly solved them, finally creating a reliable, error free system.



```
public void save() {
    //Before saving check if the array is empty
    if (listQuestions.isEmpty()) {
        System.out.println("Empty");
    } else {
        try {
            FileWriter addingQuestions = new FileWriter( fileName: "listQuestions.csv",  append: true);
            //  For loop for all the questions
            for (QuestionCreator p : listQuestions) {
                addingQuestions.append(p.getQuestion());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getChoice1());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getChoice2());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getChoice3());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getChoice4());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getAnswer());
                addingQuestions.append(COMMA_DELIMITER);
                addingQuestions.append(p.getTopic());
                addingQuestions.append(NEW_LINE_SEPARATOR);
            }
            addingQuestions.flush();
            addingQuestions.close();
            listQuestions.clear();
            System.out.println("Done");
        // Trying to catch any exceptions causing the system to crash
        } catch (Exception e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Maintainability

Several functions used inside the classes where self-contained doing 1 simple task needed making them easier to maintain and test. This break down of the complex code into smaller understandable methods made adding them in different classes much easier and efficient. It made error testing faster, more productive and we were able to create a good initial prototype. We could easily refine our initial code constantly improving it and creating improved versions.



```java
// Get all of the questions
public ObservableList<QuestionCreator> getQuestionCreator() {

    String fileName = "listQuestions.csv";
    File file = new File(fileName);
    try {
        Scanner inputStreams = new Scanner(file);
        while (inputStreams.hasNext()) {
            String line = inputStreams.nextLine();

            String[] splitted = line.split( regex: ",");
            questionsList.add(new QuestionCreator(splitted[0],splitted[1],splitted[2],splitted[3],splitted[4],splitted[5],splitted[6]));
        }

    } catch (FileNotFoundException e) {
        e.printStackTrace();
    }

    return questionsList;
}
```

Accessing the contents of the “listQuestions.csv” file and storing them as objects into observable list was undoubtedly important for several of our classes. The same method was used in the class important for Taking the Quiz and it was used in adding, amending and deleting the questions in the database. So, maintaining a method that perfect doing 1 simple task meant reusing it and implementing it in different ways much easier.

Overall this module has enhanced our overall skills in organizing tasks as a team and improved our efficiency in software development. After learning several useful techniques and models adding them in future work will improve production and develop the value of time.