DQS: Individual Report

For this project I was part of team 23, a group of seven. In this report I will discuss how our team's experience compares and differs to Bruce Tuckman's Lifecycle of teams, the problems we encountered and how we overcame them. I will evaluate the effectiveness of our team's project management, risk management strategies and version control, as well as the various techniques we used to produce quality software. Finally, I will highlight how and where my own part of the final product took into consideration the different quality criteria.

Team Organisation:

Towards the start of the project, we started off quite well as a group that had just entered the forming stage of the lifecycle of teams (see ref 1 B Tuckman). We were a group of people, most of whom hadn't really spoken to each other much before – although I did know a few people beforehand which I think helped establish the group and bring us together. I think this helped because we were more confident in talking to each other, giving help and feedback on each other's work.

Coursework 1 – Requirements

Similarly, to the forming stage of the lifecycle, I think that most of our groups participants were initially very enthusiastic about the work. When completing the requirements and use cases we ran out of work to do, additionally we ended up including more detail than was required on the use cases. This was because everyone was motivated to be involved and complete their share of the work and hand in good quality work. We also worked well as a team, we constantly checked and reviewed the work completed by others. Additionally, when a team member was struggling with their assignment, we all helped them out.

On the other hand, at first we did not have a reliance on authority like in the lifecycle of teams. We mostly worked independently and did not start allocating work until around a week in. This did cause a few issues as we found people worked at different paces. Some members went and completed a lot of the basis of the requirements very soon after it was assigned meaning that the members who tended to leave their work closer to their deadline were left without a lot to do, which I think lead to them thinking they weren't being given the opportunity to take part and pull their weight.

Although, after a while I started to take on the role of leader – I started organising some meetings every week where we could meet up, discuss the work completed and start handing out assignments. I also kept a group log detailing who turned up to meetings, the work that had been completed and assignments. I think this was quite effective as it kept people updated on what work had been completed and what they had been assigned. Furthermore, it gave everyone something to work on at their own pace. So, in the end, I think we did manage to establish ourselves like in the forming stage of the lifecycle of teams. We all worked towards a common goal and helped each other out, although it did take us quite a few weeks.

Coursework2 – UML, Detailed Requirements and Risks

Following the lifecycle of teams, we did end up entering the storming phase (see ref 1 B Tuckman) around the start of the Christmas holidays. I think that the holidays caused some disillusionment with the project goals and a loss in enthusiasm because of a lack of contact with other team members and other priorities such as exam revision. I think that it was at this stage that the team split. Me and two other members continued to attend meetings and complete assigned tasks, however many of the other members decided to repeatedly not turn up to meetings or complete their assigned tasks before their deadlines. Furthermore, there was also a lack of communication between them and the team which caused conflict as the other members of the group had no idea what they were working on.

Me and the other two members were coming together as a team, I think we were entering the norming phase (see ref 1 B Tuckman). We kept each other updated with the work we were completing, attended all meetings and helped each other out by reviewing each other's work. We built up trust with each other and were reliable teammates, completing tasks before the deadlines. Me and another member also decided to share the leadership of the group which I think worked very well as we could support each other in managing the team.

Meanwhile the rest of the team members were still acting like a group. It was difficult to try and include them in the project, the reasons being that they did not commit themselves to the group, missing meetings and deadlines, and as the deadline was approaching we had to reassign some of their tasks. Furthermore, it was also difficult as a leader to keep the group working as a team as apart from me and the other two team members, the rest of the group rarely responded to our messages on our group chat.

Eventually we did reach the norming phase of the lifecycle with a few more members of the group. Me and the other leading team member decided to establish two days a week where we would have regular meetings at the same time and place. This was initially quite successful as we had higher attendance at meetings, this also lead to more established deadlines and common reviews of work completed. We also decided to keep a record of the work completed on the group log to motivate some of the less involved members to have something to fill in. This was quite successful as in the last week of the second assignment, we hit a more productive phase. All members (except one) worked together to complete the remaining work. Some members were still concerned that the other members left their work so close to the deadline which caused conflict and stress to the other team members, however I think that we had to accept that people work at different paces – which was fine as long as they completed their work before the deadline.

There were still problems with one group member, even on the day of submission he did not attend the meeting or contact the team, even after we had tried contacting him for the three days beforehand. We ended up having to submit without him and then resubmit when he finally contacted the team.

Coursework 3 - Software Development

After entering the norming stage where the team starts to come together, in the lifecycle of teams, the performing stage usually comes next (see ref 1 B Tuckman). I thought that at the end of the second coursework we finally had reached the norming phase and were heading towards the performing stage. For about three weeks we had a few of the other members turning up to most meetings and completing test cases – we even managed to work together on reviewing them and starting parts of the code. It seemed like some of the members had found a new motivation in the team – like in the performing stage of the lifecycle of teams.

However, it was unfortunate that the Easter holidays started, as most of the team members lost their motivation again. We quickly left the performing stage and declined into the dorming stage for the remainder of the project. Our team effectiveness declined rapidly, most team members had left for the three-week holiday meaning we had to rely on the group chat and online meetings in order to communicate. However, this was not very effective as many members chose to ignore the chat – or just not respond. Furthermore, we had previously agreed to continue or weekly meetings throughout the holiday. However, only me and one other person attended the meetings. This meant that no-one was communicating with each other or updated on the work others were doing. Therefore, we became a separated group individually working on our separate parts of the project.

I tried to keep the team working together. I stayed in contact with the same two members from before and constantly messaged them about parts of the project. As I was working on the quiz, I was reliant on many of the other classes meaning that I needed to work together with the other members in order to integrate their parts of the code into my part. Me and the other two team members worked well together because we kept in contact with each other, additionally we listened to each other – we discussed changes and methods needed to integrate parts together. I also helped the motivated team members to write their code in an object-oriented manner and integrated their code together to check that it was compatible. I helped others with the themes and the timer and integrated them into the quiz. I think that as a team of three, we managed to stay within the performing stage of the lifecycle of teams.

However, apart from one member, we had no contact from the rest of the team for the whole of the holidays. Although two of the members did seem to upload some work, they rarely contacted the team which made it difficult to see what they were working on, additionally it made it hard to integrate their code as some changes needed to be made but they never responded meaning we would have to wait a long time for the necessary changes.

After the holidays had ended a few of the other team members seemed to become more productive, they submitted some code which we integrated into the project. Apart from one member who we had still heard nothing from and received no work from, we were starting to hit a more productive phase – similar to returning to the forming stage of the lifecycle.

A few days before the deadline, the active team members decided that we would limit changes to bug fixes. However, we had problems with the inactive team members ignoring this. We had spent a few days testing and were happy with the submission, however on the day of submission, the team member doing the stats kept changing my code to fix errors in changes he had made to his – which broke the encapsulation of my part and also did not fit in with quality criteria. So I spent hours helping him to integrate the stats properly to protect encapsulation with classes being responsible for their own functionality. I think this problem arose because not all team members were communicating with each other and because some team members were less organised and left their integration too late.

The team member who had had no contact with the group and had submitted no work turned up unexpectedly minutes before we submitted at our submission meeting. This also caused problems as we had no time to integrate the code. He had not been to a meeting for around 5 weeks, had not uploaded any of his code to Git, and had not contacted anyone explaining that he had code to integrate. Therefore we had to submit his code separately.

Overall, I think our greatest barrier to team working (ref Hans Tamhain & David Wilemon) was communication problems. Whilst some of the team members did consistently keep in contact on the group chat, many of the others had either limited or no input on the group chat. Additionally, their lack of attendance to meetings meant that they failed to keep the rest of the team updated on the project development. This caused many problems with integrating the work which happened later and took longer, leaving less time for bug testing and unnecessary stress with other team members. We tried to overcome this communication problem by having consistently timed meetings at the same place every week, meaning that everyone knew when and where they were meant to report progress. I also added a work completed area on the group log for each meeting where members could also view their assignments and update their work completed in order to keep the team informed even if they missed a meeting, however many team members chose not to fill this in. Finally, we had a group chat for the whole of the project on a platform we all agreed on and we checked at the start that everyone was added and could access the chat.

If I was to complete another team project, I would try to improve this barrier by having consistent meetings from the start, having multiple communication platforms set up, in case it was a problem with our method of communication. Finally, I would add in more serious repercussions for missing a meeting or failing to complete a report on what work had been done in order to try and motivate communication.

Managing Software Development Process:

Version Control and Git:

In our software development process, we decided to use Git as a way to maintain version control and share code. We used a shared maintenance model (see slide 12 from 'Git for teams' lecture slide) and trusted all members to check their code for errors before committing. This was successful as we were all able to work locally at our own pace and pull changes whenever someone made a change, though only five members out of seven used Git. We agreed to message the group when we had pushed a change into the master branch, although not done consistently which lead to other team members waiting for code which had already been pushed – we only found out after periodically checking GitLab for commits. Using Git was very effective as we could always access all changes our teammates had made, furthermore, previous versions of the project could be checked out if there was a major problem.

I decided to ask the group to use branching on our repository because this would reduce the risk of someone completely overriding someone else's changes. We followed the mainline branching model (slide 17 from 'Git for teams' lecture). This was a great help as it meant that even though our team had bad communication with some members, everyone's code was backed up and versioned on Git.

I wrote a step by step guide for everyone on how to use branching, detailing the commands needed to successfully create a branch, save changes, commit and merge. This was to help members of the team who were struggling with using branching. Branching was useful as it helped to reduce the errors put into the main branch which is the code that would be submitted. Furthermore, members could push their code up to branches on the remote repository where other members could see, and merge in working code. This helped to encourage regular integration and, in most cases, meant only working code was merged into master, although in some cases broken code was merged. Overall, the branching was effective in keeping a working copy of the project in the master branch ready to be submitted.

When Gitlab was down for the whole of the weekend before submission, I set up a GitHub mirror repository which cloned all the commits of my local repository and sent instructions on the group chat so that everyone working that weekend could connect to the new remote. This was very successful as we could carry on working without GitLab. However, we had to manage the risk that when GitLab came back up, we were all using the same remote repository and that GitLab was updated with the new commits.

Project and Risk Management:

To manage the project effectively, we decided to have regular team meetings every week and to keep a group log detailing all the reviews, decisions, assignments and work completed. At first, we were quite unorganised meaning that we did not have effective project management – and people missed meetings due to other priorities. However, as we completed the project, I think that our management improved with introduction of meetings at regular set times and a more detailed group log. These were effective as everyone always knew what they had been assigned, what was expected of them and when and where to turn up for meetings. Additionally, these also helped us to effectively assign deadlines and helped us to plan out the project and when tasks needed to be completed.

The meetings were also very useful and effective for sharing information between team members – particularly with team members who were quiet on the group chat. It also meant we could discuss assignments and share out the work fairly and get everyone involved. Most meetings had four or five members present. However, when only a small number of team members turned up they were less effective. The members who turned up frequently worked well together and ultimately produced work that was well integrated. Those that turned up infrequently missed some deadlines and the rest of the team had no way of knowing if they had been working on their assignment or if they needed any help, so in some cases other people had to complete the work.

The use cases and test cases were also very useful in helping to manage the project effectively. As they were both completed before the implementation stage, they helped us to see exactly what functionality needed to be included and in what way they needed to be implemented to carry out their tasks. This was useful as we then had a better understanding of what was required from each section of the project and could make a better estimate of how long each section would take to implement.

The risk assessments we completed were also very effective in helping us come up with risk management strategy for our project. They were useful because they helped us to think of many problems or risks that could occur and how to deal with them. A few examples were the use of version control to counter all project files being deleted, code being unreadable; so, we wrote comments and explained to each other, and finally the estimated time of development under estimated being countered by having regular team meetings in order to adjust plans.

Most and Least Useful Techniques:

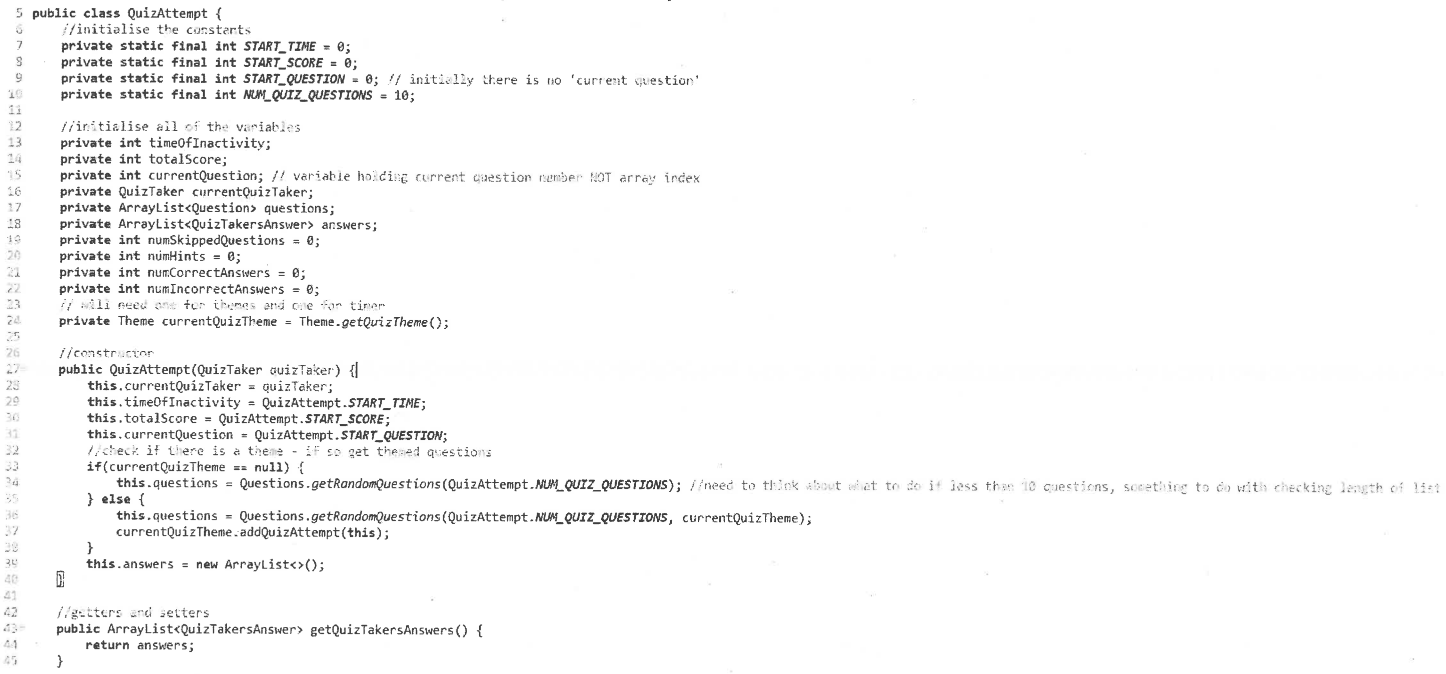
I think that the most beneficial technique for developing quality software was the UML diagram. It was effective because it helped us to think about how to split up the functionality into object-oriented classes and their relationships to each other. Additionally, it made it easier to split up the project among the group when it came to implementation as we assigned certain classes to team members which related to their task. I encouraged everyone to follow the diagram when coding. Although the diagram was missing a few relationships, variables and methods that we hadn't thought of until implementing the code, I still think it was beneficial as it gave the basic outline of the program and its classes. The diagram was very effective in helping to manage the project as we knew which classes needed to be implemented, how many there were and the basics of what they did.

On the other hand, I think that the Gantt chart was the least beneficial technique for our group in developing quality software. Although it was useful in planning the project, knowing when deadlines were so we could plan to finish beforehand, our plans kept changing and we couldn't stick to the project plan. This was mostly because of our communication again – members did not complete work by deadlines and did not contact or update the group on their progress, so we had to keep pushing back and extending deadlines, therefore making changes to our plan.

Quality Criteria:

I was responsible for developing the quiz taking and its functionality and integrating the other parts, such as schools, year groups, themes, timer and questions, into the quiz. I wrote Quiz.java, QuizTaker.java, QuizAttempt.java and QuizTakersAnswer.java. I also helped other team members with parts of theme.java and time.java in order to make them more object-oriented.

The following are two examples of where my part of the final product considered quality criteria, such as reliability, maintainability and usability:



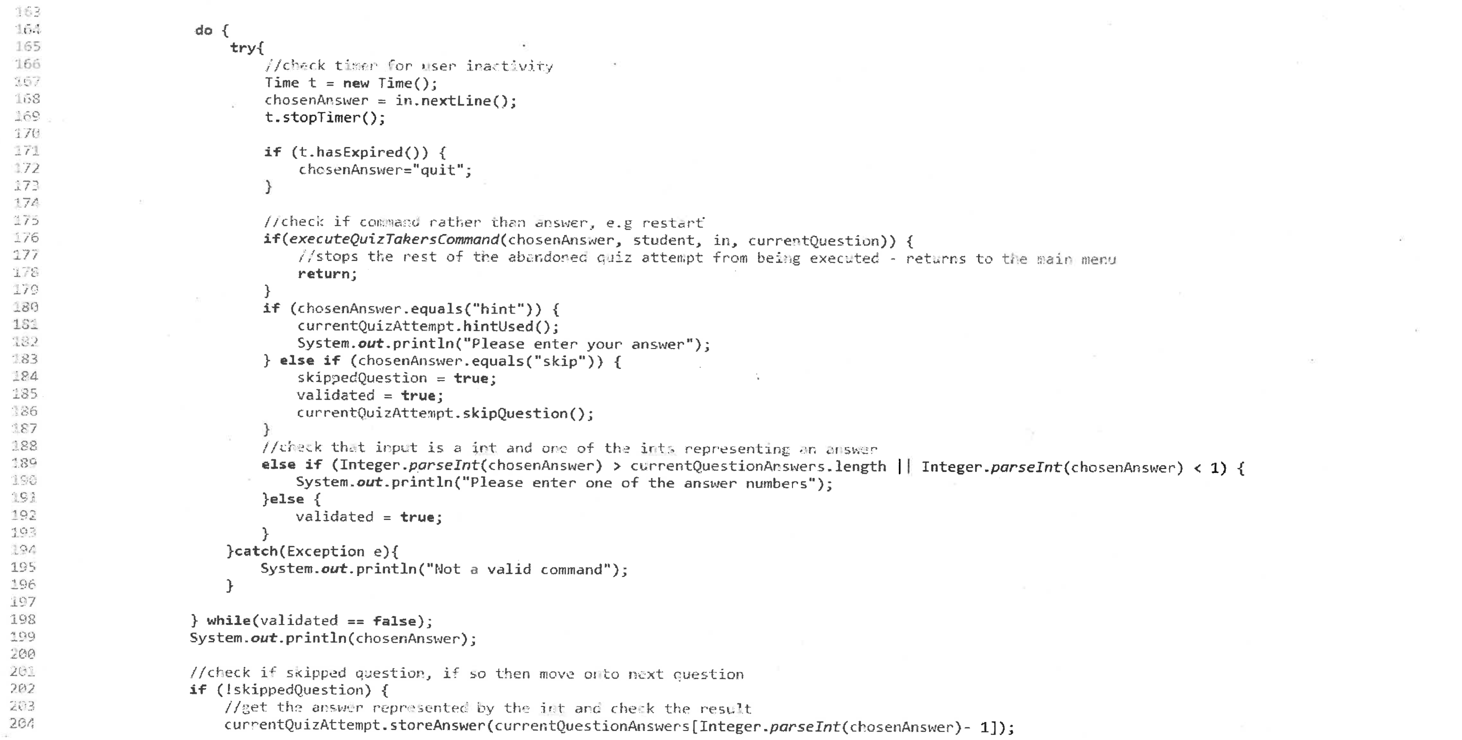
```java
public class QuizAttempt {
    //initialise the constants
    private static final int START_TIME = 0;
    private static final int START_SCORE = 0;
    private static final int START_QUESTION = 0; // initially there is no 'current question'
    private static final int NUM_QUIZ_QUESTIONS = 10;

    //initialise all of the variables
    private int timeOfInactivity;
    private int totalScore;
    private int currentQuestion; // variable holding current question number NOT array index
    private QuizTaker currentQuizTaker;
    private ArrayList<Question> questions;
    private ArrayList<QuizTakersAnswer> answers;
    private int numSkippedQuestions = 0;
    private int numHints = 0;
    private int numCorrectAnswers = 0;
    private int numIncorrectAnswers = 0;
    // will need one for themes and one for timer
    private Theme currentQuizTheme = Theme.getQuizTheme();

    //constructor
    public QuizAttempt(QuizTaker quizTaker) {
        this.currentQuizTaker = quizTaker;
        this.timeOfInactivity = QuizAttempt.START_TIME;
        this.totalScore = QuizAttempt.START_SCORE;
        this.currentQuestion = QuizAttempt.START_QUESTION;
        //check if there is a theme - if so get themed questions
        if(currentQuizTheme == null) {
            this.questions = Questions.getRandomQuestions(QuizAttempt.NUM_QUIZ_QUESTIONS); //need to think about what to do if less than 10 questions, something to do with checking length of list
        } else {
            this.questions = Questions.getRandomQuestions(QuizAttempt.NUM_QUIZ_QUESTIONS, currentQuizTheme);
            currentQuizTheme.addQuizAttempt(this);
        }
        this.answers = new ArrayList<>();
    }

    //getters and setters
    public ArrayList<QuizTakersAnswer> getQuizTakersAnswers() {
        return answers;
    }
```

Example 1 QuizAttempt.java



```java

            do {
                try{
                    //check timer for user inactivity
                    Time t = new Time();
                    chosenAnswer = in.nextLine();
                    t.stopTimer();

                    if (t.hasExpired()) {
                        chosenAnswer="quit";
                    }

                    //check if command rather than answer, e.g restart
                    if(executeQuizTakersCommand(chosenAnswer, student, in, currentQuestion)) {
                        //stops the rest of the abandoned quiz attempt from being executed - returns to the main menu
                        return;
                    }
                    if (chosenAnswer.equals("hint")) {
                        currentQuizAttempt.hintUsed();
                        System.out.println("Please enter your answer");
                    } else if (chosenAnswer.equals("skip")) {
                        skippedQuestion = true;
                        validated = true;
                        currentQuizAttempt.skipQuestion();
                    }
                    //check that input is a int and one of the ints representing an answer
                    else if (Integer.parseInt(chosenAnswer) > currentQuestionAnswers.length || Integer.parseInt(chosenAnswer) < 1) {
                        System.out.println("Please enter one of the answer numbers");
                    }else {
                        validated = true;
                    }
                }catch(Exception e){
                    System.out.println("Not a valid command");
                }

            } while(validated == false);
            System.out.println(chosenAnswer);

            //check if skipped question, if so then move onto next question
            if (!skippedQuestion) {
                //get the answer represented by the int and check the result
                currentQuizAttempt.storeAnswer(currentQuestionAnswers[Integer.parseInt(chosenAnswer)- 1]);
```

Example 2 Quiz.java

Reliability

According to McCall's quality model (McCall, J. A), a programs **reliability** (ability not to fail) depends on its accuracy, error tolerance and simplicity. In example one, I have kept my code well-structured as an object-oriented class. I first define constants, then static variables, then class variables. After that I have my constructor before my methods that manipulate the data in the class. I have no global/public variables; all of the data is accessed using methods in the class. This way I used encapsulation to ensure that each class is responsible for its own data. Furthermore, I have used the same structure in my other classes and helped to implement this style in themes and time.

In addition to this, in example two, I surround many pieces of my code with try catch blocks to catch errors in order to ensure continuity of the program when an exception occurs. I catch all exceptions and then give alternative code to keep the application running. In most cases I print my own messages explaining what the problem is, such as 'please set up themes, school details and questions' and return to the main menu or wait for a correct user input.

Maintainability

McCall also claims that programs are **maintainable** (ability to locate and fix faults) if they are concise, use modularity and are self-documented. In both examples, I comment my code and use self-explanatory names to help others to understand what my code is doing.

Furthermore, I used modularity when writing my classes – each class is responsible for its own data. In example one, the QuizAttempt class contains all the attributes required in a quiz attempt, which can only be modified or retrieved by methods in the QuizAttempt class. I also separated the logic from the UML diagram from the user interface by separating them into different classes. The QuizAttempt class contains all of the logic for each quiz attempt and the Quiz class is the user interface. The user interface (part of it shown in example two) calls methods from each of my classes such as currentQuizAttempt.storeAnswer() in order to save user input and get data to print. The use of modularity also made it easier to test each of my classes and made it easier to link into the user interface as I only call the methods. This means if I wanted to fix a bug or change the way a method worked, I would not need to change the user interface.

Additionally, in example one. I have kept my code compact by splitting the code up into appropriate methods which can be used to avoid code duplication.

Usability

Finally, McCall's quality model also says that a programs **usability** depends on its communicativeness, operability and ease with which new users can use the system. In example two, I surround my code with try catch blocks when asking for user input. This is to increase operability by anticipating errors that the user may enter and returning helpful error messages to help them enter valid input.

Furthermore, in example two, my user interface applies some of Ben Shneiderman's golden rules of interface design (Ben Shneiderman). All of the required user inputs are consistent within the quiz – asking for the number relating to the answer or school. This helps make the quiz easier for quiz takers to use. In addition to this, I give the users feedback on their actions – either an incorrect input, or confirming the answer that they entered.

Conclusion:

Overall, this project was a very useful experience in group software development. The fact that my group was split up into a team and a group (and thus differed from the usual lifecycle of teams) gave me a good insight into the problems of managing a project – especially where not everyone is motivated to the same degree, and the impact this can have. I learnt how to use many different techniques in order to create quality software and was able to evaluate which work better in certain circumstances, thus helping to solve problems and mitigate any risks. Finally, I learnt about the about the different quality criteria and how to apply them to my part of the final product.

References:

- Bruce W. Tuckman, "Development Sequence in Small Groups", Psychological Bulletin. 1965. In 1977 Tuckman (in collaboration with Mary Ann Jensen) updated the model to include the fifth stage – adjourning
- Thamhain, H.J. & Wilemon, D. (1979) Team Building in Project.Management in Proceedings of the Annual Seminar/Symposium of the Project Management Institute. Atlanta, Georgia.
- Helen Phillips, "Team Theory", Cardiff University Lecture Slides
- Philipp Reinecke, "Git for Teams", Cardiff University Lecture Slides
- McCall, J. A., Richards, P. K., and Walters, G. F., "Factors in Software Quality", Nat'l Tech.Information Service, no. Vol. 1, 2 and 3, 1977.
- Ben Shneiderman*"Designing the User Interface -Strategies for effective Human-Computer Interaction"* Addison-Wesley