DQS Individual Report

As part of this project I was in team 9, a group made up for 6 members. During this report I will run through how we worked as a team to manage the software development progress, evaluate how well we worked together as a team, and the quality criteria we used to develop our final product.

Part 1: Team Evaluation

A barrier that our team ran into was, unlike a real software development team, we didn’t have a normal work day/week. This means that we in any given week we would only meet for an hour or two, we would then go our separate ways, work on other coursework we had then possibly do more work on the software. We overcame this barrier by having regular check-ups on peoples work over our Facebook messenger chat. This worked quite well for everyone as it kept the current task in the forefront of people’s minds, it also meant that we could keep reminding people of the next time we were going to meet up to make sure no one forgot.

Out of the team dynamic theories that we covered during lectures I found that the Belbin’s team roles was the only theory the only one that I came back to thinking about. For this theory we had a team discussion at the beginning of the module where we placed ourselves into the roles that we thought most suited us. After looking back halfway through the development and at the end of the development I believe that we grown as people and developed our skills such that we have shifted roles.

There have been some big shifts in the positions that people have taken on, whereas some people haven’t shifted at all. As one example one person has shifted from being a shaper which is an action-oriented role (See slide 10 of “CM1202 Belbin’s team roles” lecture slides) to resource investigator which is a people-oriented role (See slide 10 of “CM1202 Belbin’s team roles” lecture slides). This shows a shift in his personality because originally, he wasn’t too comfortable talking with the team and would prefer to work alone, but now he’s become more confident and now prefers working with people and helping them rather than doing his own work.

Part 2: Managing the software development progress

Overall the team’s project management took a while to find it’s footing but was effective for what we needed. For example, with version control, we initially planned to use GitHub. As the member with the most experience using GitHub, I set it up and gave everyone full access to push and pull to the repository. Unfortunately, the plan to push to this repository soon fell down with the other 5 members of the team committing only 7 times between them. This was because people found GitHub too hard to use and often ran into errors with pushing and pulling. We sorted this issue with everyone the python files to our group chat on Facebook messenger. This still worked well as only once or twice would we have two people working on the same code and so having the collaborative functions of GitHub wasn’t a priority. I continued to upload to GitHub just for ease of use. For the limited time we used, we decided on using a Shared Maintenance Model (See slide 12 from “Git for Teams” lecture notes). We decided on this method because we our minimal knowledge of git it was the easiest way to allow everyone access the files remotely. Although Shared maintenance does have some drawbacks from the few times, we used it we didn’t run into any problems.

For team management we used a mixture of both scrum and agile. We took the best parts of both and combined them. This worked really well for our team and the limited time we could actually meet face to face. For example, from agile development we took the rapid delivery of useful and working software (See Slide 3 from “Intro to Scrum” lecture slides). We took these two from agile because we found that with the limited time frame we were given for our product, we really needed to make good headway on the product early to allow us to refine it later down the line. This meant that we could see progress in our product often.

From Scrum we took the ideas of daily sprints (See Slide 11 from “Intro to Scrum” lecture slides). This worked really well for our team because we would meet up once or twice a week for only an hour at a time. In this time, we would have a quick meeting to discuss the work we still have left to do, we would then assign the realistic task for the coming week. After this we would often do an all-out effort to get part of the assigned task done, this is very similar to the sprints in the way we only had a short amount of set time to get a useable piece of code.

The teams use of risk management was very lacking. We made one document towards the start of the project outlying the risk and their potential impacts. We have not visited the document since making it. This was a poor choice by the team because I feel that reminding ourselves of the possible risk’s in the development of our product could have saved us time by realising what we were trying to make was not actually worth it and would have too many risks attached to it.

Out of all the new techniques we were introduced to during the development of this software, I think that the test cases were the most useful for our team. This is because they were quite simple to understand. This helped our team because it meant that everyone was able to contribute equally to them which further developed our understanding of the task.

On the other hand, the UML diagram was the least effective for our team. I believe this because I found that not everyone in the team fully understood how to do the diagram. This resulted in other members of the team picking up for the people who didn’t understand. This resulted in most of the team not understanding the diagram, therefore when it came to implementing the software the diagram was practically useless.

Part 3: Quality Criteria

As part of the development process I was tasked with developing the formative test for the students to take.

For the relatability aspect of the software I made sure that I would limit the chances of the software having obscure answers which could break the software. I did this buy only allowing lecturers to enter in multiply choice answers and then students could only pick from the answers. This means that I can assure that all bases are covered with answers.

For the usability of the software it is extremely easy to use and requires zero knowledge of programming, the user will never see the command line or have to write any code. Lectures have the most control over everything with the ability to easily add, remove or edit questions. To do this they will never need to touch the code at any point. The software will do all the work for the lectures, they will only need to press buttons in the software to allow the students to get the new tests.

References:

* Helen Phillips, “CM1202 Belbin’s team roles”, Cardiff University Lecture slides
* Helen Phillips, “Into to Scrum”, Cardiff University Lecture slides
* Philipp Reinecke, “Git for Teams”, Cardiff University Lecture slides