Niall Curtis – C1623580

Team 11

INDIVIDUAL REPORT

CM1202 – Developing Quality Software

Table of Contents

[Introduction 3](#_Toc481161467)

[Team Organisation 3](#_Toc481161468)

[Team Development Process Model (Bruce W. Tuckman, 1965) 3](#_Toc481161469)

[Collection (Forming) 4](#_Toc481161470)

[Entrenchment (Storming) 4](#_Toc481161471)

[Resolution/Accommodation (Norming) 4](#_Toc481161472)

[Synergy (Performing) 4](#_Toc481161473)

[Dorming 5](#_Toc481161474)

[Decline/Breakup (Mourning) 5](#_Toc481161475)

[Belbin Team Role Theory 5](#_Toc481161476)

[Barriers to Team Working 6](#_Toc481161477)

[Leadership 6](#_Toc481161478)

[Time Management 7](#_Toc481161479)

[Teamwork Summary 7](#_Toc481161480)

[Managing the Software Development Process 7](#_Toc481161481)

[Project Management 7](#_Toc481161482)

[Planning 7](#_Toc481161483)

[Team Organisation 8](#_Toc481161484)

[Implementation 8](#_Toc481161485)

[Control 8](#_Toc481161486)

[Risk Management 8](#_Toc481161487)

[Version Control Strategies 8](#_Toc481161488)

[Techniques for Development 9](#_Toc481161489)

[My Role in the Team 9](#_Toc481161490)

[Technical 9](#_Toc481161491)

[Team Organisation 10](#_Toc481161492)

[Quality Criteria 10](#_Toc481161493)

# Introduction

This report will review and reflect the events and work that took place during our team project for Developing Quality software. The aim is to look at how we worked as a team, the techniques we used and what we learned from it. It will details the rights, the wrongs, and what can be improved in the future.

# Team Organisation

Team Organisation is the backbone behind completing a mutual task effectively. A team should be organised into having both individual and group work, as well as accountability for both. A strong team should show exemplary discussion and decision-making skills, being able to account for the ideas of all members and incorporate that into results that fit the criteria of the task. Through combined efforts of each member and their

output to the group, a team’s performance will be greater than the sum of its parts.

## Team Development Process Model (Bruce W. Tuckman, 1965)

In its simplest terms, it is a model of five distinct stages a group would experience on the path to becoming an effective team. I will evaluate our team’s performance to the stages involved. One of the only caveats in the comparison is that our team members were all good friends beforehand, which removed a lot of the awkwardness and bond-creating involved with a team of strangers.

### Collection (Forming)

The initial stage of team creating is forming the actual team from individuals brought together to complete the mutual task. It relies on some form of self-made authority, and establishing the expectation of team members. It’s not entirely relevant to our own group because we formed the teams of our own accord, with previous friends. In some respects, we actually did meet the idea that participants are initially enthusiastic, as the prospect of completing a task with friends was indeed good motivation to propel the progress of the task. It was effective in that we were completely comfortable discussing our ideas with each other, as well as what was effectively ‘arguing’ our points. The need for an authoritative figure was reduced when everyone was comfortable with each other.

### Entrenchment (Storming)

This phase is the idea that when teammates have settled down, tension starts to build because people all have their own preconceived ideas to solve the task which creates difficulty coming to a joint solution, thus a period of unproductivity. As previously mentioned in the forming stage, due to existing friendships we were generally more open towards others’ thoughts and ideas. There were occasionally underlying issues with disagreeing on things, as is natural in any team infrastructure, but was dealt with accordingly. We didn’t struggle with any kind of competition or confusion as everyone was happy in their immediate roles and adapting to the task at hand.

### Resolution/Accommodation (Norming)

This phase for us had more to do with settling into the task than coming together as a team generally, though the planning helped improve the team dynamic too. Some of the issues in the last phase were caused by lack of forward thinking during the early stages of the project, which meant people had slightly differing ideas on what was happening. Thanks to more careful use of team theory and methods of planning, we bounced ideas and built off each other as intended, and the uses of working as a team began to shine through. We settled into our roles and stuck to our plans which allowed the group to make solid progress. It wasn’t perfect, and there was some disagreements (naturally) but were easily solved by discussion and compromise.

### Synergy (Performing)

The optimal stage of teamwork, assuming the previous sections were successful, and the output of the group is greater than the sum of its parts. We all had the team’s combined vision in our heads and worked hard towards the end-goal. Everyone was at their upmost comfort in the team environment, and used this to convey ideas and theories which could be quickly adapted and implemented by the team. It was aided by everyone’s commitment to the task, and the small team size meant that synergising was easy.

### Dorming

This is a post-synergy phase between performance and decline, related to complacency from the performance phase. We never really experienced it because a lot of our work was completed in a small time-frame, so we powered through it; having the goal in sight was the cause for a lot of motivation, so we essentially experienced the synergy until just about the end. We tried to help each other forward when someone was demotivated, and the team ethic was a driving force for the project.

### Decline/Breakup (Mourning)

The period of teamwork where the work rate begins to decline as people grow bored/tired and drift away. As a whole we didn’t really get the chance to experience this due to the relatively short time frame of the project. By the time we got to implementation, we powered through until practically the end. It only reared itself towards the very end of our coding stage, when after long stretches of doing work people were becoming jaded and this was coming through in lazy code, more disagreements and less overall productivity. Our motivation to finished helped us overcome this, allowing us to finish the project more-or-less in the synergy phase.

## Belbin Team Role Theory

The Belbin theory is a study on how some teams can be more effective than others. It is the idea that a team should have a good balance of different team roles, each role with its own strengths and weaknesses. If you compare these roles with our team, you can draw comparisons and identify who had the characteristics of certain roles, and can also see what we lacked to evaluate what would’ve helped us. It’s not relevant to the fullest extent because we don’t have enough members to fill all the roles anyway, however having a good mix is important whatever the size.

I would consider myself as the Plant, and Completer-Finisher for the project. I came up with a lot of the original ideas for the implementation which my teammates liked, which was to do with my own creativity and enjoyment of the task at hand. I started a lot of it independently early in the process to create the baselines before involving the other team members to complete certain functions. I had created a good base for the project in a few days which gave us plenty of time to build on. It was the same attention to detail that allowed me to strive for perfection at the end of the task and complete it to a high standard before the deadline. This does, however, due to stress lead to anxiety and distrust of other teammates to complete the task how I want it; even though they can.

Another one of our members acted as a Shaper through the duration of the task. They had the specifics of the projects in mind and knew exactly what they wanted it to be, no matter how difficult. They used this motivation to manage the team assertively, thus completing the task to the necessary standard and detail. At the very end of our implementation, one of our features related to getting the quota wasn’t fully implemented, and the rest of us were prepared to sacrifice it due to difficulty. He yet persisted on the task, and after time using his Shaper characteristics finished the task to a good standard. One of the downsides of the Shaper attitude is sometimes being pushy as they see the task in their own specific way and are unwilling to budge on their own ‘vision’.

We also had someone who displayed characteristics of an Implementer. They approached the problem programmatically with practical, controlled approaches using their hard-work to complete the task in a very logical manner. They used a lot of their time to refactor code, create strong back-end functions and worked hard to create a very scalable program. The good part about it was that in the long term it was easier to program when we had a solid set of reusable functions, however their weakness was being unwilling to let other people change the backend when it was required.

Our other main identifiable Belbin role was the Resource Investigator. They were very outgoing with their communication and liked to explore new approaches and ideas wherever possible. He was heavily involved with collating people’s ideas and turning these into more usable implementations, specifically for our user interface. They wanted to make sure everyone had an input to try and best meet the specifications of the task. The problem they faced was staying motivated when everyone was ‘plugged in’ independently trying to finish the task in time for the deadline, because communication had slowed down.

It was quite difficult to completely categorise the team into Belbin roles, as in terms of our own members it wasn’t very black and white. It was our first team task, and a lot of the people exerted characteristics of many roles along the way as they settled in and learned their own skills. Towards the end it became more visible, but as everyone had their own input in every task it varied. To characterise people more specifically we looked at their input throughout all the tasks. I think what we lacked in the project was someone who was an outward team leader; or in Belbin terms, a Coordinator/Teamworker. We often felt the need for someone to take control, as usually team leading was shared between the five of us. Lack of centralised leadership meant disagreement which led to lack of productivity. In future, we should begin by allocating a leadership task to one of the members at the very start of the process. We struggled by having differing visions sometimes which made the elements of the software feel disjointed and independent, compared to a cohesive product.

## Barriers to Team Working

### Leadership

I previously mentioned that the lack of leadership caused some difficulties to ripple throughout the project, which I will expand on here. Usually when you’re matched up with teammates randomly, one or two people initially take command to lead the project as people get to know each other. This leads to natural team leader roles forming, and the project continues as normal. The difference for us was that we were all friends, and were open with each other and nobody fell into a leadership role. This caused several issues throughout the project. Everyone had their own ideas and concepts for the implementation of the system, and everyone saw their own ideas as ‘correct’ so pushed for them. Compromising ideas was difficult when everyone had their own agenda and it would’ve benefited us to have someone to take command and make decisions for the good of the team. I think the team member that displayed characteristics of ‘Shaper’ would’ve been most fit to take charge of the team as they were driven and motivated to complete the task, with good communication skills.

### Time Management

A problem that effects many teams is managing time scale of the project, something that stems into the issue we had with leadership. We began the project quite efficiently, everyone completing the goal in good time to a good standard. This gave us a false sense of security with the timeframes given, and for the implementation section we became lax and had to rush a lot of work towards the end of the deadline. This of course compromised quality and caused tensions to be high. Underestimating this was due to lack of sticking to the proper planning that we made earlier, so the solution is quite clear. We had already expended the effort to make solid plans, but we just chose to ignore them later on; next time would be rectified by having a team leader who pushed for people to stick to individual deadlines.

## Teamwork Summary

Even though we did share a lot of similarities between ‘ideal’ teamwork models, both Belbin and Tuckman, it also made clear the issues that caused the most difficulty within our team. There are clear elements that we could’ve rectified that would’ve improved our end-product. The most prominent of which, that has already been heavily discussed, is leadership. The rest of our problems were clear stems from a lack of central vision, including communication, time management and implementation errors. Nonetheless I believe by the end of the process the team had gelled together, and we put our all into making the final implementation something we were proud of. Several members were heavily motivated, and in part due to enjoyment of programming, spent a lot of time striving for the end goal. Experience throughout the task also helped us learn efficient methods which can be continued in the future.

# Managing the Software Development Process

## Project Management

Referring to the Project Management Process from the Project Management slides

### Planning

We began the process by producing a Gantt chart which tried to set a time scale for each section of development. We also came up with goals to meet for each deadline that would in theory help us achieve our goal. In the end, sadly, motivation to do planning was much less than motivation to do the implementation, which resulted in a subpar schedule and vague goals. This meant we generally went in blind to most tasks, especially as we had to balance this work with other modules and other parts of life; but in those situations even a solid plan doesn’t work. A better strategy for us would’ve been to set more specific goals with more flexible time scales, as it’s difficult to motivate people to work towards a broad task due to the size of it. Less specifics on timing would make more it easier if people miss a goal.

### Team Organisation

Covered in the previous section – people were happy and flourished in their roles but we lacked any strong leadership.

### Implementation

We struggled to stick to any solid plan while doing the day-to-day implementation of the product. We generally created goals between us on the fly, for instance; “Today we need to finish the search function”. It’s far less efficient than having set guidelines and timeframes, but as we enjoyed the coding we got things done on time. We coded well as a team in general, getting ideas and theories from each other and building off each others’ code.

### Control

Control was one of the aspects that was lacking due to having no real leadership. We have nobody to oversee all aspects of the process, as everyone was always getting on with their own segment. This miscommunication meant that people perhaps went through whole stages of work without other people’s input, leading to inconsistency in our implementation. Any sort of management control would’ve helped, to check everyone’s progress on each task, set new smaller goals, and encourage the team to complete their tasks.

## Risk Management

Before the task, we looked at all the possible things that could go wrong with the project. As it’s quite a simple and contained task, this was mainly missing deadlines which can vary in its effect depending on how badly we miss it, or how important the task was. We generally just agreed to work together to help each other should someone be struggling to meet a specific deadline which did work. However, it would’ve been more thorough to assign a risk ‘rating’ to each deadline that would explain how crucial each deadline is and the flexibility. That way we know what to prioritise should things begin to go south.

## Version Control Strategies

We used GitHub for our version control system. It allowed us to work on different branches when we needed to experiment, document our changes, share and test different things and generally collaborate without causing issues. One of the issues we met was not everyone being particularly versed in Git management, which caused problems like pushing dirty code to master, rebasing the entire project, and other crucial errors. It ended up fine because of local copies, but we certainly had teething issues with Git. This is another thing that could’ve done with a leader, someone who could manage the repository and deal with pull requests and such.

## Techniques for Development

The process had us use several typical team/software development tools to try and aid productivity. I believe the most useful of these were the Test Cases we made. They were documentation set that would be used upon completion of features in the implementation, to check they work correctly. In our experience, they felt natural to involve in development and genuinely helped us complete the task. It allowed us to delegate testing of personally made features to other members, as it would detail how to use the feature and what the desired output was. Multiple times it helped us identify a slight problem that needed to be fixed before release, that probably would’ve slipped through without the test cases. It was an extra development step that ensured we met the task’s standard and quality.

I think the least useful technique, at least personally, were the class diagrams. I think it may just be a character flaw as I’m new to programming, but I find it difficult to stick to very strict guidelines when going about a task. Even though I made sure to head for the same result, I ultimately found myself disregarding the class diagrams and heading my own way as I struggled to think ‘inside the box’. Even though I can respect the usefulness for other people, to me they weren’t at their full effectiveness.

# My Role in the Team

## Technical

I was heavily involved for the technical aspect of the project. As I previously mentioned, at the beginning I was the ‘Plant’ of the team. At the beginning of the process initial creativity and motivation helped me push the team to get tasks and done, and I was pleased with the amount of work I was getting done; I got into the task well and enjoyed it. The same thing happened when we got to implementation stage, and as we had recently been doing a lot of programming for other coursework I was at a good stage to start doing backend development, a lot of the low layer tools and functions. It allowed the team to more easily implement their areas of the software, and my codebase was well commented and simple to use. This did, however, come at a cost, as I had to spend a lot of time refactoring my original code a bit further on because of a couple of small niggling features that had to be addressed. Proper planning early on, and better communication with what the team needed would’ve helped me achieve originally what I had to do later. My next most involved production stage was when I acted as the ‘Completer-Finisher’ in the final stages of the implementation. I spent a lot of time cleaning up bugs, helping teammates theorise their functions and design GUI features. The motivation to finish the task helped me get my mind into the project, and I used this to push through to the end to a great standard.

## Team Organisation

My involvement in a lot of the programming meant I had to take a faux leadership role, in lieu of a real team leader. It wasn’t official in any manner, however I had to guide people through what I have done and tell them their tasks which kind of put me in that role. It’s not where I am most comfortable, and there were a few disagreements because I wasn’t always excellent at getting my points across. I don’t think I did a good job always of getting everyone on task in the right places, which meant conflicts of interest and some disjointedness in the program towards the end. I believe if I had a team leader to confer with, or I had put more thought into leading myself, I would’ve had more impact in organising the team. Nonetheless I believe that I did well enough that everything worked by the end, even if it took longer than expected.

# Quality Criteria

As I mentioned earlier, my main contributions to the project were at the beginning and end of the implementation. I worked heavily on the searching backend, and the GUI for the finished product. Here are three elements of quality criteria that I aimed to fulfil on these.

## Flexibility

Flexibility was the main concern from the start when developing the backend of the software. I needed it to be applicable to all the tasks that I and my teammates needed to program as well, so had to scale to these tasks. The main feature of the backend was reading, searching and writing to the CSV so I had to do all these in self-contained classes that could be adapted and reused easily, wherever necessary. I made expandability a huge design feature, and programmed with the idea that the features could be used however large the project got. This also spilled over into the generality; once again making everything basic enough that it could be reused basically anywhere. It took some time, and a fair amount of refactoring, but flexibility was a big deal to me.

## Maintainability

Leading on from flexibility, maintainability was the natural progression. If this backend was going to be flexible and general to other areas of the system, it had to be easily maintained. This generally came in the form of refactoring my original code to make it more concise, and more modular. I separated every feature I could into its own class or a function to bring redundancy to an absolute minimum, coupled with conciseness so that the other teammates could easily understand, and thus build upon the source code. For example, having a reading class, a writing class and a searching class; all which could be reused extensively. Another big element was commenting wherever possible to further reduce complexity and abstraction.

## Usability

As I also did the GUI for the implementation, which is the main user interaction with the software, usability was the main design tent pole for this part. Making an interface that required minimal effort to learn, with maximum productivity and feature set was not an easy task. I had to convert every possible input function into something that could be easily understood by anyone picking up the software, without any prior learning. I communicated what the software could do through a series of buttons and labels, but packed it all into unique sections to prevent overloading it. The general ethos was to split dealing with students and tutors into their own sections, and have another section again for loading files. I then reused features wherever possible in each section, and the end result being something that required little learning to use every part of it.

# Top Tips

This section is going to detail four top tips for undertaking a similar team project in the future. For sake of generality, I’m going to split it up into two teamwork specific tips, and two development specific tips.

## Teamwork

### Leadership

This is perhaps the thing I talked most about during the report, for good reason. I have identified many times that this was one of the primary reasons that things didn’t go as well as they could have, and a lot of other problems stemmed back to this. I cannot stress enough that having a good team leader would’ve been excellent for us. Even though we regularly settled in to leadership roles naturally, having someone to take charge and control the team is crucial. Having a democratic vote at the beginning to elect a team leader, even if they’re inexperienced, provides some form of structure to the project. It motivates people to work towards a specific goal if someone can delegate tasks well, and bounce ideas. It causes a lot more disagreement when everyone is considered an equal, as everyone believes they are correct all the time. A lot of unnecessary stress occurs when everyone is trying to manage other people all the time, as you have to put a lot of trust in your teammates to do work. In future, I will definitely try to match someone to one of the Belbin team theory characteristics for a leadership role.

### Planning

A solid plan is another thing that we had the means to use, but never followed through with. We even produced plans as part of the first section, but we were gung-ho during the implementation and generally just did whatever we wanted, whenever. This led to a lot of half-baked features, poor coding, disjointedness and redundancy. If we had more firmly stuck to a plan, which detailed who would do what and when, I believe we could’ve definitely had a lot of the task done with less stress than what we had this time. The lack of firm planning meant I had to spend time that could’ve been spent on other areas, refactoring my own code because it had to fit people’s features and functions they had designed of their own accord as well; something that could’ve been easily avoided would we have talked about it. I think another useful part about a plan is being flexible on the time scale, as timing caused us problems too. Planning to account for worst case scenario helps when things go wrong – we could’ve more heavily tied our planning into our risk assessment to help with deadlines and worst-case scenarios.

## Software Development

### Better Version Control

Version control is one of the unsung heroes in software development, and something that goes ignored too regularly. That’s not to say we didn’t use it, but for our project it ended up being a glorified file sharing system rather than doing its job. In the future, I would like, before ANY development occurs, setting up a full development environment. Have one person oversee the entire Git, then setup branches for everyone to test their own features on. Rather than let everyone commit directly to master, force people to make commented pull requests, detailing their changes and giving the Git leader time to check for conflicts and errors. A couple of times we ran into merge conflicts, people overwriting code, resetting the master branch, and other typical Git issues. They were easily sorted, but it wasted valuable time that could’ve been spent being productive. If necessary, giving people some time to get used to the version control system would prove valuable.

### Testing

One thing I found during this process was the usefulness of testing. It seems obvious in retrospect, but I never respected how good detailed, thorough testing was, both for individual features and production code tests. There’s nothing worse than finding substantial errors in the final product, so testing features as and when they’re complete, with very specific criteria, can make sure nothing slips past. I would want a wide feature test for every function that we plan, upon completion of its individual module and when its built into the main implementation. Each test should use both general, and fringe cases, so the program won’t run into any strange errors. In this process, we looked at class diagrams, use cases and Test Cases, which can all be intertwined to create a thorough and deep test system before deploying the program.