SEG Reflective Report

Matthew Dyson – SEG5

Contents

Introduction

Within this document I will be reflecting upon the work of Software Engineering Group 5, operating under the name of 'Team Typhos', a team of 5 students from Durham University comprising of 3 Computer Scientists (Dan Crook, Tom Oxenham & Jack Hopperton), one Software Engineer (myself – Matt Dyson) and a Natural Sciences student (James Walker – Physics & Computer Science). We were asked to undertake two projects during the course of our time together, one being the production of a wind farm planner as our major project, and the other being assigned by an external company as part of the Phoenix programme. For this we worked with East Durham Trust in order to write a new content management system to replace their old website.

Team Dynamics & Collaboration

In order to ensure that at least one member of the team had a clear picture of the current project status, we assigned a person to act as 'project leader', who would be responsible for the delegation of tasks amongst the rest of the team, and for tracking progress with relation to upcoming deadlines. The task of testing the final software was assigned to a lead tester, who would co-ordinate the implementation of whatever testing strategies they saw fit in order that the software would meet the required standards. During the final stages of coding and testing, a documentation co-ordinator was assigned to collate all technical and non-technical documentation into a readable format for delivery, including checking all source code comments were in-tact and the production of a user manual. Members of the team who did not have an active leading role would be delegated tasks in any area.

When we first met as a team, we were obviously unaware of each other's strengths and talents, leading to some tension amongst the team as people pushed for roles even though they were unaware of the responsibilities that would ultimately be involved, although this was quickly resolved.

Within the scope of the SEG project, I did not take one of the leading roles, instead concentrating on design and implementation of the low-end systems such as database and file integration, although within this I was expected to provide technical documentation as appropriate. Having fewer responsibilities within the SEG project meant that I could concentrate on leading the Phoenix project, which was commonly felt as being a more appropriate distribution of ability. Within this role, I was responsible for the majority of interactions with our client, and for translating these needs into specific requirements that the rest of the team could work with. I was also responsible for managing the specific tasks of the team, delegating roles as necessary in order to meet the deadlines. As I had prior experience working within web development, the majority of structural design work was left to me, and I distributed assignments out to the rest of the team according to existing skill sets and individual wishes, ensuring that no-one was left working in an area they felt overwhelmed with.

Communication within the group was mainly facilitated through weekly meetings, at which we would spend some time reviewing the work achieved since the previous meeting, discuss any issues arising from the project, and decide upon short-term targets and roles. Outside of this, details of the project status were kept on Google Wave, which acted as a central forum for discussions, as well as for tracking implementation progress and bugs. External communications (such as those from our Phoenix clients) were done through a group mailing-list to ensure that everyone was aware of the current status, however at any given time only a single person would be communicating, to ensure that external parties did not receive multiple communications.

Very early on in the project, we lost one of our team members (Jack) for unexplained reasons. This put considerable pressure on the team as a whole, as we were approaching deadlines for the hand-in of several documents. During the early weeks of his absence, we decided to carry on distributing roles to him via email, which were confirmed as having been received, and presumed that these would be delivered upon. As we moved towards the deadlines, we realised that the likelihood of us receiving said responsibilities was negligible, causing them to be re-delegated out.

Software Engineering

Whilst implementing the project, we used a variety of software engineering principles to aid the process. For instance, in order to achieve the separation of concerns, we chose to split apart the processing of data into a separate section of the program, so that the end user would not be aware of what was being done to their inputs and requests. We then separated out the low-level components that would be required to interact with the file system and database in order to create clearly defined modules that could be developed separately, defining interfaces in order that interaction could be calculated and programmed without the need for other components to be in place. In order for this to be tested at the time of programming, we created a series of mock classes that could be used before the appropriate component had been created.

This separation of different components proved extremely effective when building the program, as it allowed different members of the team to develop their own sections

independent of other peoples work, relieving the pressure on certain people to get their work done quickly so that others could continue. By defining strict interfaces early on in the development process we managed to avoid many issues that could have been caused by the refactoring of code, although in some respects this could prove to have caused bottlenecks in the system. For instance, the module involved with reading in data from files was defined to be able to carry out certain searching tasks. When it came to programming the module, it was proved to be fairly inefficient at performing searches as defined through the interface, and instead the search code would be better placed within the central processing module. However, as the relevant parts of the processing module had already been written, it was deemed to be an inefficient use of time, and hence was left as originally defined.

The development cycle we agreed upon and adopted was similar to that of the 'waterfall' model, in that we had a single period defined for each area, such as the requirement specification, design and development. In practice, this model caused issues when we had no scope available for re-designing certain areas to better suit the specifications, and we would have been better off adopting a 'spiral' approach in order to allow time for adapting our design. However, the spiral approach does not lend itself to programming within a short time-frame, as it would have been unfeasible to develop the system and then return for further passes at programming due to the time constraints in place.

Very early on in the project, whilst working on the initial documentation, we realised the necessity to allow every member of the team access to the latest revisions of files instantly, and also to permit multiple people to work on documents at the same time, to avoid work being lost. For this reason, we chose to use Google Documents (a web-based office suite) to host the working copies of documents, allowing all members of the team to collaborate simultaneously. This benefited us hugely whilst producing the User Acceptance Testing documentation and also during the design stages, where updates could be made available to the rest of the team instantaneously. This also negated one of our key risks associated with this area of work – loss of data. The use of 'cloud storage' for our work guaranteed secure hands-off backups with automatic creation of revisions should the need arise to roll back to previous version.

For the sharing and distribution of other files, we chose to use Dropbox, a synchronisation service that allows any file to be moved between computers, whilst at the same time offering a backup solution. Using this service allowed us to move files between each member of the team quickly and efficiently, and ensured that any files that could not be used through Google Documents had an effective backup solution.

There are a variety of code revision control systems available that meet the needs of most software development teams (Git, CVS etc), however we chose to use Subversion as several members of the team had worked with it before. The use of a versioning system for the code we were producing allowed us to distribute changes to the entire team easily, the benefits of which have already been mentioned.

Successes

Overall, I would class the SEG Wind Farm Planner project as a success, as despite several issues we managed to deliver a feature-complete project on time. The final version of the product that was delivered did meet the all initial requirements dictated by the User Acceptance Testing and also the Requirements documents that we produced; with only minor changes to the initial criteria that were set, as described in the Design Documentation. The several non-functional requirements that we laid down within these documents were also met (system is fast/secure/easy to use/aesthetically pleasing, etc) to our internal satisfaction. We also exceeded the mandate in some areas, introducing new areas, such as visual representations of the data onto a map view that formed a major part of the user interface. This viewing pane allows the user to visualize the data contained within the database, such as the position of wind farms and substations, and also to overlay information gathered from the supplied wind data files in order to fully understand what the data is representing.

Critical Failures

We hit a major failure very early on in our time together as a team, when submitting the Requirements documentation. As discussed previously, this was written in separate sections through the use of Google Documents, and then was to be collated by the lead documenter before submission. During the process of putting the final document together, an entire section was missed out, which obviously had severe consequences. By the time this was realised, it was too late to re-submit the work, and in light of the situation we introduced better checking of submissions, and adjusted individual deadlines in order that the document would be completed several days in advance of the deadline in order to give more time for collation and checking, so that similar problems did not re-occur.

As previously mentioned, we lost a member of our team extremely early on in the project, which caused us noteworthy problems at the time. Within our requirements specification, the risk of certain failure points was analysed, with explanations given for how to negate the risk, and to minimize its effect. Although this list was fairly exhaustive, I believe we failed to properly implement the recovery strategies laid down within it when this issue occurred. At the time, the work that was due from said member was done by the rest of the team, and personally I assisted by adding some to my workload, as well as assisting with the redistribution of tasks. However, by the time this was completed, the workload of each team member had increased to a level whereby it was unfeasible to complete to a satisfactory standard within the given time-frame. The use of cascading deadlines, as discussed in the next section, would have prevented this issue from occurring.

We encountered a number of issues by using a central code repository, many of which stemmed from team members not having any previous experience of using similar systems. For instance, we would often find that members would treat the repository as a backup of their local work, causing multiple collisions and errors when they attempt to commit code a long time after their last update, ultimately leading to loss of work. Early in the development process, some members were unaware what these collisions actually meant, and would often overwrite other peoples work with their own, causing cascading issues with bugs re-appearing, through to other classes failing to work. We clarified this issue by having a team

meeting as soon as the problem was identified, in which we ran through the details of using Subversion, as well as the principles of revision control in general.

The only other significant failure we encountered as a team was with the delivery of the Phoenix project final version. A series of cascading deadlines had been set and achieved; however the client requested a number of last-minute changes to the design and core functionality of the content management system, including the addition of some new features. Because of the quick turnaround required to try and deliver on time, we reverted back to our original style of a single bulk deadline (discussed in later sections). Most of the requirements were met in the final version; however we failed to negate the risk of team member loss, which unfortunately occurred in the final few days before the scheduled delivery date. As we were not informed about this situation we failed to redistribute the work accordingly, leading to the deadline having to be pushed back in order to deliver a feature-complete product.

Lessons Learnt

One major point I will take away from working on this project as part of a team is the need for clearly defined internal deadlines. As noted in previous sections, many of the issues that we came up against as a team stemmed from the fact that our internal deadlines lay too close to the externally set deadlines, which left little to no time for recovery when problems arose. For future projects, I will set deadlines giving several days slack-time which, in the worst case, can be eaten into in order to perfect the hand-in. The team adopted a similar strategy for deadlines later on in the project, although instead choosing a more cascading approach, where certain sections held their own deadline, so it was clearly defined who would be working on what section at any given point. This helped immensely in keeping to deadlines, however I did not fully agree with this approach, as discussed in the next section.

The allocation of tasks and roles within the team was also a major point of contention, and in future I would treat the initial allocation of roles a lot more carefully than we did initially. The roles that we defined as necessary at the start of our time together appeared to work well, however the balance of work at different points in the project was, at times, extremely unfair. For instance, during the run-up to documentation hand-ins, the documentation manager would be putting in considerably more time than other members of the team in order to collate and deliver on time, whereas people involved in the programming side of the project would not have as many tasks assigned to them. I believe that a better approach to this situation is required, and would work to improve this in future.

Whilst I personally was not involved heavily in the documentation process at the end of implementing the projects, I am aware that it was made extremely difficult by the lack of code commenting throughout certain areas of the program. I ensured that all forward-facing elements of my code were fully commented, as did the rest of the team, however internal documentation was not always provided, which lead to some problems if a member was absent, and also when the system was being documented as part of the design document. In future, I would ensure that all areas of software are fully commented both as standard block comments (Javadoc), and also explanation within methods to describe their operations, which would make it a lot easier to document the internal workings of the program.

As discussed previously, we as a team put together a fairly comprehensive risk analysis covering the majority of issues that we foresaw could occur within the scope of the project. For each of these areas, we decided upon the likelihood of the risk becoming reality, and the effect this would have upon our productivity and output if it were to happen, using these figures to calculate which risks we should take further steps to prevent from happening, and where we needed more detailed contingencies. I believe that this process was taken too lightly, and as a result we were ill-prepared for the issues we came up against during the course of our time together as a team. For instance, the permanent loss of one team member, and the absence of another at a critical time greatly affected the team dynamics, and ultimately had a substantial effect on the deliverables we produced. Had we planned better for this risk, the course of action would have been more clearly defined and could have been put into effect sooner, whereas when it did occur we wasted valuable time in debating our next actions.

Personal Reflection

The main thing I personally will take away from this project is the importance of inter-personal relationships both within the team and when interacting with clients. I felt that as a team we began to work better once we had a clear understanding of each others strengths and weaknesses, which obviously lead to a much better working atmosphere, and better results. As this did not happen for some time into the project, the first set of deadlines were not up to the standard we would have liked. During the course of the Phoenix project, we spent a considerable amount of time ‘on-site’ with our clients, the majority of which was spent observing them at work with their existing system. Whilst this was useful in order for us to ascertain what the client really wanted, it may not have been the most efficient way of completing this. As we were approaching this with zero experience, we did not see any other way of gaining this knowledge, however in future I would approach this in a different manner, instead choosing to get a better background view of the clients operations and needs before conducting an on-site meeting.

As I have previously discussed, we came across a number of issues regarding the deadlines we had been externally set. The ‘cascading’ style of soft-deadlines that was eventually introduced alleviated the initial problems we had with sub-standard submissions; however this was implemented by assigning tasks to team members with set deadlines for each. This meant that, especially with regards to documentation, sections were completed in parts, and not always by the same person, which lead to inconsistencies in style and format. In future, I would consider other options for internal scheduling, and develop a more efficient way of managing team members and their time to ensure that these problems did not occur.

Overall, I have enjoyed working closely with peers on this assignment, and it has given an amazing experience despite the issues we have faced. Working on a group software project is a valuable skill, and I believe the experience was extremely worthwhile, and will prove very useful in future.