SEM5640: Individual Report

Samuel Jackson
slj11@aber.ac.uk

January 4, 2016

Personal Evaluation

From a technical perspective, my contribution to the group project was primarily centred around developing the front end of both the .NET and Java EE versions of the project. Specifically, most of the tasks that were assigned to me where focused on activity data. This includes both the management of activity data, displaying the data to the user, and displaying data between users (especially leaderboards).

Excluding the creation of the database architecture for each of the projects and the corresponding entity models, I was responsible for almost all of the direct interactions with activity data. This meant I was responsible for the creating the views/JSPs, view models, and JavaScript behind these pages. I was the first team member to directly touch the scaffolded front end (followed closely by User management produced by Connor). Subsequently a lot of the specific interface design choices in this area were made by me and this led to similar patterns being adopted (for better or worse) by others elsewhere.

I feel that I made a good contribution to the project, but I also obviously believe that there are several areas in which I could have improved my performance in hindsight. The remainder of this section will discuss the strengths and weaknesses of my contribution, provide justification for my actions, and suggesting how I could improve in the future.

At the beginning of the project, I tried to encourage my fellow team members to set times which were suitable for project meetings as soon as possible. This was not difficult as the other team members were all of the same mind, but I felt it was important we try to push on with requirements analysis and produce a general sketch of the system early to give plenty of development time later. In those first few meetings I also tried to ensure that the team members stayed on track. This included things such as working through the requirements one by one rather than jumping around and going off on tangents. I was careful to give nudges to keep the conversation on track while not squashing enthusiasm. Looking back, I believe this did help to keep those meetings shorter and more focussed.

In the midst of the development process I was also keen to push for a clean wrap up of the project at the end of each sprint in order to make sure we could keep to the planned time allocation. I felt it was important to try and ensure we would start the next sprint with a clean, correctly building, master version of each project. This gave us a weekly checkpoint where we knew things worked and also provide a good point for us to bring our existing work together. I knew that this would be an easy area to let slip so I was regularly vocal in insisting that we finish up at then end of the week. Once or twice we overran a couple of times into the morning of the following day due to integration issues but the majority of the times we finished cleanly.

During the initial stages of the project I felt slightly under utilised. I think that considering my relative inexperience with the technologies I could have done more to prepare myself before implementation began. I did familiarise myself with Scrum based methodologies which was invaluable after the project started, but I feel that running through some additional .NET and Java EE tutorials would have prepared me better. One of my biggest weaknesses in the project stemmed from getting stuck on a problem where I knew what I wanted to achieve but did not know the correct way of writing it in the technology I was working with. This often caused me hours of lost time which was the biggest contributor to running over time at the end of the sprint.

One area where I feel I was a strong contributor to the team was my knowledge of Git. Other team members had varying degrees of competency with Git and I think it would be fair to say that I had the strongest experience. Throughout the course of the project I helped my team mates out of countless issues when switching between branches or fixing merge conflicts. I also played a large role in trying to ensure that that team members stuck to the group workflow, e.g. by not including changes for multiple issues on the same branch. I also found myself frequently giving advice on what was the "best thing" to do under certain circumstances (e.g. "should x be done on a new branch?"). I think this often saved my team mates time and pain when working together.

I think that towards the middle of the project, in contrast to the first few weeks of I became over committed to work. In the weeks following the first couple of iterations my assigned workload was well above what should have been my maximum for that week. I do not feel that I was exceptionally overcommitted as the other team members also had stretched workloads.

However, I do think that this led to a slip in the quality of the work I produced. I almost always felt like there was not enough time in the day to complete my assigned work. I believe this meant I would be more likely to take a shortcut and less likely to spend time reading around and finding the most optimal solution to the problem at hand. This effect was also compounded by my lack of knowledge of both frameworks. As mentioned I often got stuck in a rut over a simple problem which wasted hours (and occasionally days) of allocated time. I also often found myself knowing what the correct thing to do was, but choosing to use a different method because of a lack of time.

Another area where I feel I could have been more proactive was in helping to test and integrate the system. In our project a piece of work needed to be manually reviewed by another person in order to test the quality of what they had written. While I was very proactive at testing towards the end of each sprint, I often put off helping out midweek due to focussing on my own workload. This is actually like shooting yourself in the foot because it means that in practice more reviewing has to be done at the end of the week and subsequently this causes more end of sprint headaches.

While I feel my overcommitment was an issue, I struggle to suggest a cure in hindsight. The obvious solution would have been to spread the workload more evenly amongst my colleagues. However, as mentioned above, they each had workloads that often equalled or exceeded mine. Pushing the work forward into the next iteration (which commonly happened) only made more work inevitable in future weeks due to our fixed time schedule. Earlier negotiation of what could reasonably be achieved could have been one solution. This has issues because as this is an assignment, less work being completed will most likely lead to a lower mark. This is the solution we eventually settled on, but it felt by that time like too little too late.

Another issue I feel affected my performance was the technical difficulties I had. Owning a Macbook computer with only a relatively small hard drive meant I had to run a Windows Virtual Machine on an external HDD in order to develop for the Visual Studio application. This was both slow and prone to crashing. In hindsight I could have tried harder to find better access to a Windows machine or pushed for the Windows project to be developed by team mates who develop natively on Windows.

Group Evaluation

Generally speaking I felt that the group worked well together throughout the course of the project. In my own opinion, nearly all of the problems that we faced throughout the course of development did not stem from arguments or tensions between group members. I found that while some conflict is inevitable in a group project this was handled quite maturely and did not cause any major issues (this is in sharp contrast to the 2nd year group project).

I think our choice of development methodology was a good one, although I feel that our execution of the methodology could of been better. I am of the firm opinion that if we had chosen to follow a methodology at the other end of the spectrum we would not have achieved as much as we did. As it happens, using Scrum allowed us prioritise tasks by how much value they had to the customer. This proved to be very important because as we ran out of time we could shed less critical features. For example, we dropped email support, which while desirable, did not mean we had a non-functioning system.

Another strength that came from using the methodology we chose was that I felt we were able to adapt over the course of the project. Leaving a large part of the implementation specifics to individuals meant each developer could adapt how they implemented their work accordingly. While the requirements of the project were unlikely to change much, this point made a difference because we were unfamiliar with the technologies we were working so being flexible with the implementation was very useful. It meant each developer could reason about the best way to implement their part without being hindered by a rigid design spec. However this also had downsides in that there was occasionally duplicated functionality and inconsistencies in how things got implemented. Despite this I noticed that the inconsistencies were slowly reduced over time as “good” patterns were found, shared, and repeated by others.

This was also important because we were working with two distinct technologies. Different implementation dependant approaches were needed for each technology. I hypothesise that designing the whole of both systems up front without fully understanding the intricacies of the technology we were working with would have resulted in a worse result.

However, I would also like to counter this point with the fact that I believe we could have done with more initial design and/or prototyping to better understand some of the challenges we faced. I think that there is a critical balance that needs to be struck with starting a project (especially a short, time constrained project such as this) between too much and too little design initially. I also think this is highly dependant on both the team’s skills and the problem domain. In this project I felt that we got that balance wrong in the direction of too little design.

In hindsight, I think that some perhaps some early prototyping of parts of the system could have saved us time later in the project. I think this could have reduced some of the pain we experienced in areas such as the Jawbone API where we struggled to get a solution working and ran over the allocated time. We could have had a chance to experiment with the technology ahead of time and it might have given us some useful artefacts we could work from.

I believe that the issue with not performing enough design or prototyping up front stems from a large issue that we experienced as a group. We had a fair bit of difficulty with our inexperience with starting an agile project. While all of the team members had some experience with an agile methodology, none of us had any experience managing the start of a project. In all of our previous experience we had simply been dropped into established groups and projects where everything was already up and running.

We found starting a Scrum project was actually quite challenging. As I have already mentioned it is difficult to judge how much design work should be undertaken before development begins. An agile approach shouldn’t do too much or it defeats the point of agile, but likewise there should be some initial exploration in order to make sure all team members are on the same page as well as working out the hard aspects we don’t fully understand.

But there were some additional issues that we faced when starting the project such as how to initially create the product backlog items and how to initially assign these to team members. The latter point is worth underlining as I think it is something we had particular difficulty with early on. I think we found the initial sprint assignment hard because many tasks are dependant on each other. For example, almost everything was dependant on the creation of the database which made it difficult for everyone to try and begin work at the same time. In the first sprint we often found ourselves waiting on the work of other people.

I do not think there is a simple solution to this problem, but I have a suggestion in hindsight. Linking in with my suggestion of doing more prototyping, perhaps we could of had someone experiment with setting up a database and a basic project scaffold before the initial sprint. The prototype artefact could have then been used in the first sprint instead. But perhaps this is slightly circular as all you’ve really done is move the initial setup outside of the first sprint. However perhaps this could have been a way to better utilise the time in the early part of the project before full on development began. This also suggests that we may have needed to have made a better distinction between what goes into the initial sprint and what to prioritise.

This leads into my next point which is that I strongly felt we needed to spend more time setting up the project to make our workflow more streamlined. While this would have sacrificed more time at the start of the project, I feel that the savings generated later on would have outweighed the initial cost. For example, correctly setting up the git repositories to sensibly ignore some configuration files and not others. In both projects we had issues where the Git would include configuration files that would often get modified when people committed changes leading to conflicts that were not easy to resolve.

One specific example problem with the Netbeans project was that it included a configuration file listing how third party JARs were linked to the project. Depending on how the the had been linked this could lead to them being missing or broken in the repository. Issues such as these should have been addressed much earlier in the project.

The solution to issues like this would have been twofold. Firstly, we could have better defined the conventions used by the team. In the previous example this could have been solved by adding the JAR dependancy as a relative path from within the netbeans project, rather than an absolute path to a specific place on a specific machine. Alternatively an even better solution to that specific problem might have been to setup a dependancy manager such as Maven. Secondly we could have used more automation to avoid repeated mistakes where possible. One obvious example of where automation could have been used better within the project was testing which I will discuss in detail next. Another place ripe for automation was the Git workflow. Common operations such as creating a new branch and merging work could have used some custom scripts to ensure common “gotchas” (e.g. not cleaning out the database old migration files) were avoided.

I feel that the only reason why automated solutions such as this were not implemented in our project was that we simply didn’t have the time to write them. We had to pour so much time into completing the work we had that there was ironically no time to setup or write automated tools after we started development. Better conventions and automation should have been setup much earlier in the project and I think we didn’t realise this until it was too late.

This brings me nicely into the discussion on testing and integration. Our project possibly suffered the most from the lack of automated testing on the system. I feel that this problem stemmed from a combination of three issues. The first is that we did not set up an automated build system anywhere near fast enough. Secondly tests were not written early on because people were under pressure to complete features quickly and so often decided to let this slip. Thirdly the nature of the project meant that it was often difficult to create unit tests for the code, especially in the early stages when much of the code was scaffolded. This meant that we were effectively flying blind during the week until a piece of work got reviewed.

Each one of those points taken on there own would be a challenge but the combination of the three lead to more headaches than I care to mention. I think the solution to these issues would be twofold. Initially we should have setup proper testing environments on the build servers and we should have written very basic tests for the skeleton code generated by the scaffolding and then ensured the code coverage was to an acceptable level. Secondly we should have had a stricter acceptance policy that required tests to be written before the work would be accepted as done. In justification to the first point above I believe the scaffolded code should have been tested so that the developer would be forced to update the test when something was rewritten or changed. Regarding the second point this was the official project rule anyway, but in practice this was often wavered due to time commitments.

One aspect which I feel did work well was the reviewing of the features before they became fully integrated into the project. While it is painfully obvious that this needed to be supplemented by a better automated testing and integration strategy it did seem to work fairly well. Due to time constraints and the logistical constraints of getting people into the same room I feel that the general consensus was that it would have been impractical to have a full team wide code review for every feature. We settled on requiring two developers inspecting a feature for quality assurance before it was approved. This was later dropped to one due to time pressure. I feel that this system worked well because it provided a balance between providing quality assurance responsibility distributed over the group and the practicalities of having the workload, time and logistical constraints we were facing.

Conclusion

In conclusion to this project I would state that I am largely dissatisfied with how we worked as a team. We frequently found ourselves in positions where we knew what we should do but due to time pressure found ourselves cutting corners. I think a large part of our problems stemmed from initial teething difficulties during the project setup that snowballed into causing us to run over in the early part of the project. This meant we subsequently had to play catchup in successive sprints. Racing against time forced us into a circular problem of cutting corners (often with automation and testing) to save time leading to the introduction of more bugs and issues which sucked up more development time. In hindsight more time should have been devoted to project setup, design, and testing. I think this would have caught more issues earlier and saved us time in the long run.