Agile methods

As a group we began this process at a disadvantage to other groups, as 3 of our members had an understanding of what Agile software development was but no real knowledge of how to implement it. We started the process by collecting ideas of how we should go about the project and entered them into a spreadsheet, as at this stage we had gained a little insight into how SCRUM worked.

We gathered our early drafts of how we wanted the website to work, what should go in it and how it should look and kept them in a centrally accessible point. At this point three of the team members had no knowledge of content management or version control systems, we used a shared Google drive to store all of our work. All of the code for the project was to be kept on the Raptor project drive. This was fine for collaborating work but we had no real way of discovering who had done what, when or why. This gave us a very simple repository for our growing body of work, but no real version control as such because work could easily be lost or written over without the ability to roll back or even spot a collision of work.

The process of creating the user stories saw our team using planning poker as it had recently been introduced to us in our Agile lectures. We wrote all of our stories on post it notes which we then wrote the knowledge acquisition and non-functional requirements of each story on the back. The team sat together and played planning poker for each story until we had estimated the points that we thought each story for the entire project was worth, then we decided on our first sprint goals according to importance and created our first backlog. We could then start to see some of these tasks were epic, allowing us to split these tasks into more manageable sub tasks. These metrics were then fed into our spreadsheet and we could plan our first SPRINT. To plan this first sprint we had a meeting to discuss what we could manage to complete in a period of 3 weeks (this was the period of all our sprints) to find out our initial velocity. As our project progresses we decided that we could use a hybrid of SCRUM and XP whereby we always had a sprint planned but if we finished tasks and found that we had time, or that there was a task that required urgent addition to the current sprint we added this job and included it in the workload for the sprint. This worked well for us as there were often times where we realised that a particular task required work on something else to be finished before it could be completed.

During our project we adopted the use of Taiga to keep track of sprint backlog tasks. We found the interface incredibly useful when planning our sprints and splitting our stories into tasks. It made allocation of tasks to individuals so easy as all we had to do was log in and pick ourselves something to work on, everyone could see what other team members were working on and any tasks that were unassigned. This made the picking up of tasks very easy as there was no need to ask if someone else was already working on something. The only downside to Taiga that we found was that it was hard to assign tasks if they were being done using pair programming, it meant the creation of two identical tasks and assigning both team members, this was not too much of an issue. We found that some tasks were unfinished at the end of some sprints, this might have been where we had underestimated the amount of effort required to complete the task and had assigned it a low number of story points, in these cases we ended up having to drag work across to the next sprint, Taiga allowed us to do this easily.

From this early and very rudimentary beginnings of a project we added to our knowledge base until we got to where we were comfortable and the process we used could easily replicated in our working lives after university.

Version control

It became apparent that a better solution to our Google drive was required when an incident occurred whereby a page of the code was accidently lost and no backup was present. The Agile module had shown us that GITLab was a useful tool for version control and the decision was taken to create a repository to keep the project code and all project related documents in. This provided a backup which could be used to track changes made and by whom so that if such an event occurred again a last working copy of the code could be reinstated. As a result of this project all of our team now appreciates the importance of having a content management and version control system, this has been an extremely valuable lesson learned. In order to use GITLab we required the installation of TortoiseGIT on the University library PC’s as this was where we were working from. We already had access to our own GITLab repositories via the University and had set up one for the project files, in order to make use of the repository we needed TortoiseGIT. Both our supervisor and our team members had sent emails to the Drill Hall library IT support team to request this software installation and eventually this was done for us.

Things we haven’t used:

Jenkins was explored as an option for automated building of the project files as it supports GIT. This option was discarded after research showed that Jenkins is more suited to the building of projects written in languages such as Java and would require the downloading of extra proprietary software onto the Universities raptor server, something that students do not have administrative privilege for. As it was not essential that we implemented Jenkins this was not pursued further.

Php unit testing was considered when performing the multitudes of tests and checks of our system. Further research into using PHPUnit revealed that, as with Jenkins, it required additional software to be installed on the Universities server which students are not permitted to do. In order to test the system, repeated entries into the database were made via the various forms created. Each entry made could not be a duplicate entry so writing unit tests would not have improved the testing process much as each test would have to be written with a unique entry, this was impractical and would not have saved much time.

File and Image upload & checks

Security is a big concern when allowing others to upload files to any server, checks must be performed to ensure that users are not uploading anything malicious that could compromise the system. As such many steps have been taken to protect against such attacks. When allowing owners of items to add them to the catalogue, a picture of the item is required, the item is added to the database and this picture is saved into a file in our project. The image is uploaded via a html form and the image is checked using php. The code used was adapted from the w3Schools.com website to suit our purpose. The code checks to ensure the file being uploaded is actually an image, it checks to see if there is already an image of the same name in the folder the images are saved in, that the image is not larger than our specified image size and that it is in either JPG, JPEG or PNG format. If the upload passes all of these checks it is then uploaded into the image folder while at the same time an SQL query is adding the item that the image is related to into our database. This SQL query will only run if the upload passes all of the checks. If the file fails any of the checks the user is informed via the site on a landing page that tells them there is a problem with the file, they are then given a link to click to return to the main page of the site. The uploading of agreements works exactly the same way and both of these checks have been written as php functions so they can be called and used anywhere there is a need to run the checks, at present we only have the uploading of new item images and new agreement text files but the check functions could be used again if we had an inventory page that allowed the upload of documents and images relating to the condition of items on their return to the owner- this was a feature of the system that was not implemented due to lack of time.

SimpleSaml

To restrict the use of our booking system to University members only we employed the use of the login system provided by the University. This was relatively simple and required the addition of a small file to our project’s public\_html folder. The file is called .htaccess and contains the following code:

AuthType Mellon

MellonEnable auth

Once this file was saved in the public\_html folder it becomes hidden so that is not visible to anyone who may have accessed our folder with malicious intent. The file allows the use of SSO to authenticate users via the Universities own login system, and passes information about the user to our system. We were then able to access information about the user’s login info, email address and account type by examining the REMOTE\_USER variable of the person that had logged in. We discussed making a request to Information Services for extra information about the user – full name, campus etc but after asking Tim Bishop for assistance he suggested that we weigh up how important this was against the amount of work required to get this information, we discussed it further with our supervisor and decided that it might be less work to create a form for the user to fill in on initial login to request this extra information and store it in our own database.

Checks for eligibility of loan

In order to ensure that items are only borrowed by those who fall into the correct categories as stipulated by the restrictions set by the items owner, checks had to be performed when a request for a loan was made. At the same time the availability of the item for the date requested must be checked. Our initial plan was to only have a catalogue of items displayed to users based on their eligibility to borrow them, for example if there were items that were only available to third year and above students then these items would not appear in the catalogue displayed to a first year user. At the close of our project this was not implemented, as we had underestimated the amount of work required to implement other tasks we ran short of time, as an alternative we have a system whereby the eligibility is checked when the loan request is made. This is not our ideal situation as it would be something that might cause frustration form a user’s point of view, a user who has gone through all the stages to borrow an item only to find they are not eligible to borrow it might get annoyed, this is why we had planned to not show these items in the first instance. The bookings page displays the item restrictions on the page but unless the use checks it themselves and realises they are not able to loan the item there is nothing to stop them from going through the booking checks. The system currently allows the user to select an item to borrow, input the date they wish to borrow it and the length of time to borrow. At this point the date checks are performed via Javascript for the onchange function of the booking duration selector, the eligibility checks are made when the user clicks the BOOK button, if this check fails the user is told they are not eligible and to check the items restrictions.

The system checks the user’s information and compares this to the data held in our database about the item. Using SQL statements to return data relating to the item a selection of php variables is created and a loop performed to check that the user has agreed to the usage conditions, is not a banned user and if the restrictions of the item match the user type, if they do then the user is eligible and the availability for the selected date is checked, if not the loan is denied.

The date check takes values from the date selector on the item page, this system originally started with us using a date picker calendar from the JQuery libraries, for some reason that we have never been able to discover this would not work for us, we tried various versions of these calendars but had no luck, eventually we had to devise a system of our own to input dates. The result is less aesthetically pleasing but is functional. Our date picker displays 91 days from the present date, each date uses a value from 1 (today) upwards. The same applied to the booking duration, the duration values are 1-7 (up to a week). The process of checking an available date requires that the values of the dates selected are added to the current date and then using the strtotime php function are turned into a date variable that can be used in SQL queries. In our form we have ensured that a user cannot leave parts of the booking form un-selected by using the required attribute of the form element and using Javascript we have made sure that the booking duration is unavailable to select until the booking date has been chosen as the date check runs from the onchange function of the booking duration selector and requires both values in order to perform the check. If the date selected is unavailable the book button is disabled and the booking duration selector re-set, the user is informed the date is unavailable.

Length of loans + supervision

Deleting items from inventory – image stays