Closed-Source Development Deliverable 4: Software Process

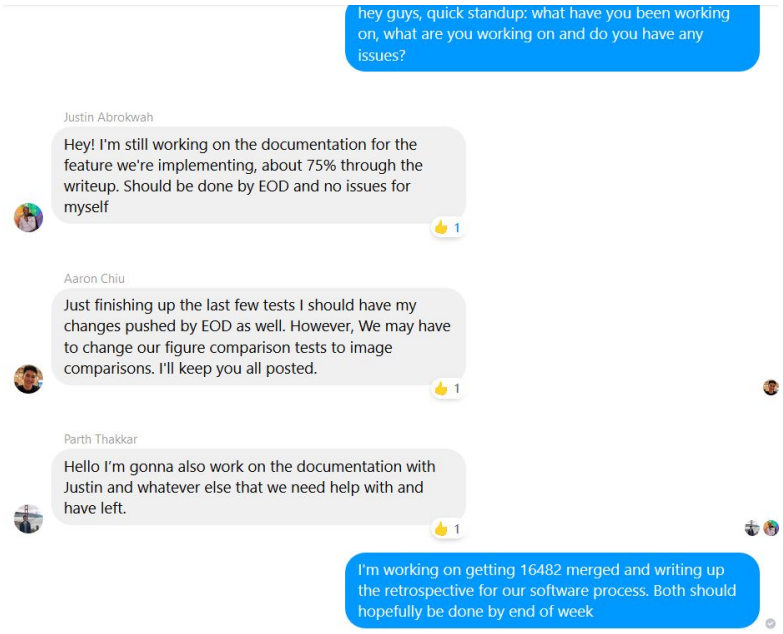
Justin Abrokwah
Yi Shiun Chiu
Spencer McCoubrey
Parth Thakkar

Overall Process Retrospective

For our project, we decided to use the Kanban software development process on top of agile due to its overall flexibility, enabling us to work productively. As university students, we have differing schedules and responsibilities both in and outside of school, and with the recent drastic changes in our lives, this flexibility and change tolerance still enabled us to work effectively as a team, maintaining our process. Kanban is designed to be flexible, to not completely tear down the schedule and plans that already exist; but instead is built to fit the already existing environment and team. Overall the Kanban flavoured agile process enabled us as a team to work effectively together, harnessing our already existing modes of communication and interpersonal relationships to create a streamlined process that never felt redundant, bulky or unnecessary.

Communications/Stand-Ups Retrospective

Our team decided to go with a looser form of standups, running them only every 3 days due to the fact that each deliverable did not have sufficient work to make daily standups make any sense. We used Facebook messenger and communicated what each member was actively doing and loose timelines for their work. We also asked each team member to communicate any issues they might be having. This allowed us to stay transparent and maintain open communication. The instant messaging also enabled us to get quick advice and feedback if needed, or set reminders for TA meetings. An example screenshot of one of our standups from D4 is below:



Kanban Board Retrospective

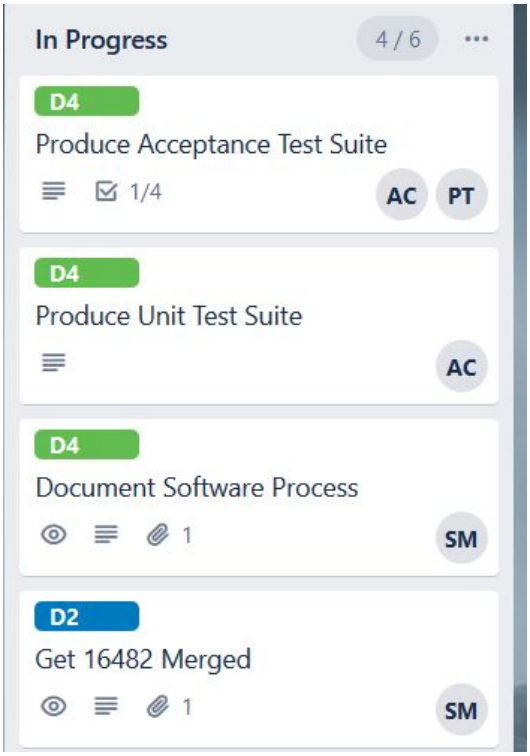
Our Kanban board, essential to the process, did not change much throughout the process, other than adding a better review process, which included a new column, after receiving feedback from our first instructor interview to make the process more transparent. Below we'll have a quick overview of each column and its purpose.

1. **Explore:**
    a. The explore column is for ideas and tasks that need flushing out. Any person can fill in details as needed until a task is sufficiently detailed enough to be placed into ready.
2. **Ready:**
    a. When a task is sufficiently researched and flushed out, it is placed in the ready column.
    b. Any team member can pull from this column and assign the task to themselves.
    c. A task in Ready can be pushed back into Explore if more research or planning is deemed to be necessary.
3. **In Progress:**
    a. When a member takes a task from Ready and assigns it to themselves, it should be moved into In Progress. This stage is for tasks that are actively assigned and being worked on.
    b. A task in In Progress can be moved back into Ready if something else needs priority, or even Explore if the task was incorrectly designed.
4. **Testing:**
    a. When a developer is done pure development, they move their task into testing to indicate they are currently self testing their unit of work.
    b. A task In Progress can be moved back to any column depending on the results of the testing.
5. **Waiting for Review:**
    a. When a task is completed from a work perspective, it is placed into this column to await other team members to approve the task.
    b. Tasks can move backwards from this column if requirements change or additional work is required.
6. **In Review:**
    a. When a task is Waiting for Review and a team member wishes to conduct the appropriate review for the task, it is moved into this column and assigned to the reviewer. Review can include code review, grammar review, test approval, etc...
    b. A task In Review can be moved essentially back into any column depending on the outcome of the review (i.e if tests fail, code review, fails to meet requirements, etc...).
    c. A task can be moved back into Waiting For Review if it is approved by one member, but requires another member's approval before moving on.

The member will attach their approval label, unassign themselves and move the task accordingly accordingly.

7. **Done:**
    a. This column is for any and all tasks that have been completed fully and to specification. Any member moving a task into Done should fully read the task description to validate correctness and doneness.
    b. A task should never move from Done to any other column; if more work is required after completion, a new task should be made.

We found the Kanban board to be extremely useful in quickly knowing which tasks were complete, in progress or hadn't been started yet. As we'll discuss, the review based columns were essential in being transparent about our review process and worked effectively to communicate the state of review of individual tasks. In majority, each column was essential to the process, not feeling redundant or "annoying" to move issues through them. The only exception was the Testing column; in practice we found that we often were testing our code as we program, and as such we rarely used the column. Not to mention that many tasks didn't have "testing" so to speak. Using the descriptions and comments section of each issue enabled us to flush out tasks and provide sufficient detail for any member to work on a task, making each issue more transparent and defined. As a team, we used labels and assignee to help communicate intention and state of tasks. An example below shows how we differentiated tasks between deliverables and assigned teammates to tasks. The labels allowed us to get quick glances at who is working on what the scope of the task.



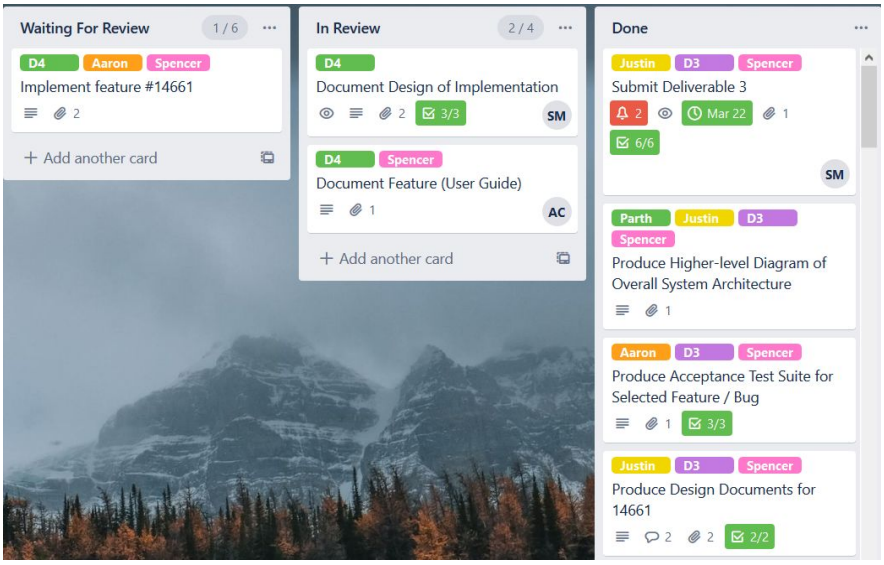
Review Process Retrospective

Most of the board is simplistic in nature, using the descriptions above and intuition it becomes apparent where along in the development life cycle each of our tasks is or was. The only more complicated aspect was the review process, as it necessitated a bit more thought for it to remain transparent and visual as Kanban requires. The process used 2 columns, 5 labels and the comments section of the issues themselves. When any teammate wanted something reviewed, it would be placed in the **waiting for review** column; then at some time another teammate would move the issues to **in review**, assign the issue to themselves and begin reviewing. The first column always communicated that the ticket required review, while the second indicated active review by a specific person. Then upon completion of the review, 1 of 2 things will happen:

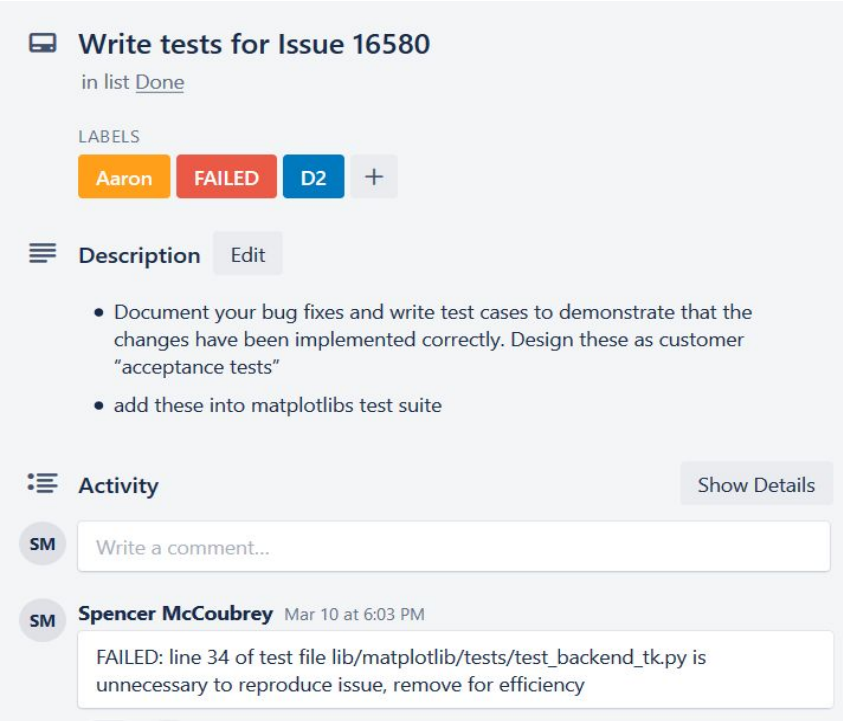
1. If the reviewer approves, they move the issue back into **waiting for review**, unassign themselves and attach their OWN label to the issue. This communicates that someone else can review and that it has been approved by a specific person
2. If it fails review, the reviewer moves back into **waiting for review**, attaches the **FAILED** label and leaves a comment in the issue describing why it failed. This clearly communicates a failure on the front of the card for the owner of the ticket to look at, and by "flipping over" the ticket they get more info. After this, the owner will move the issue accordingly, removing previous approval labels if required.

After a sufficient number of approvals, an issue is completed (by whatever means necessary such as merging) and moved into **done**.

The first image below shows the general review process at some point during D4. We can clearly see both Aaron and Spencer are actively reviewing separate issues, and that Spencer has approved 2 issues and Aaron 1. Looking at the **done** column, we can see a history of who has approved which issues, giving us a good snapshot into the history of our board and enabling transparency.



The next image is an example from D2 where during the review process, Spencer noticed an inefficiency in someone else's testing code. The failed tag can be seen from the front of the card, and when opened, you can see the detail provided to fix the issue.



We found our review process to be sufficient, being totally transparent and improving the look and state of our board at all times during the project.

Work in Progress Limit Retrospective

The WIP limit was set for nearly every column on the board (except **explore** and **done** as they don't have a theoretical limit). The WIP limit was most effective in the **in progress** and review columns, as they enabled good board flow and no backlog making sure teammates reviewed and worked at a steady pace, while not over-burdening the team. Overall, we didn't have a problem maintaining our WIP limits and they worked to our benefit.