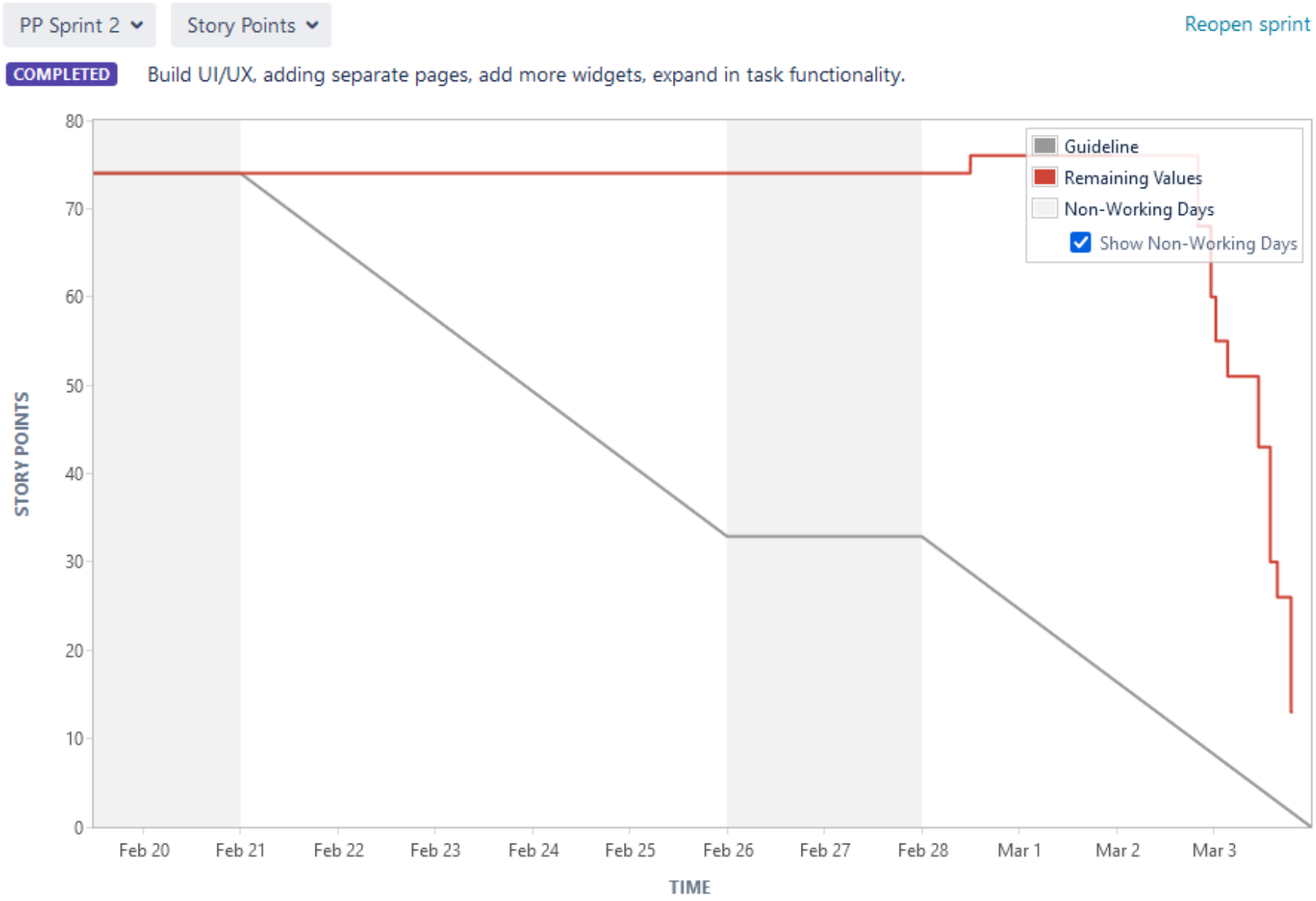
Sprint 2 Burndown Chart



Introduction

Our Sprint 2 velocity started off with 74 Story Points, up from 45 in Sprint 1, then changed to 76. This is because this time we had two Developer Stories, that were necessary to assign and complete in order for the rest of the team to code efficiently. These were configuring redux to store information in a global state (a substantial undertaking), as well as setting up TypeScript data models. Note that due to the large increase in User Stories, the redux was actually unable to be configured, and [PP-44] has therefore been pushed to Sprint 3. This User Story was weighted heavily, at 13 points. Another User Story was [PP-45] also at 13 points; this one was about reworking the entire UI and the current components into a different layout – the one we created using wireframes in Sprint 0. In addition, we also had adding additional functionality to existing features such as Task Lists, Do Not Disturb, and the Pomodoro Timer. The Music Player was an all-new feature. Ever User Story this sprint was however a distinct feature that our users required. We completed 8 of the 9 assigned Stories for Sprint 2.

Results

Our velocity changed significantly from last sprint because User Stories like UI layout or configuring redux after Auth0 were not prioritized when they should have been. Both of these we realized were essential to the app – yet neither were really demo-able without the corresponding components. As a result, we had to work on both levels in order to deliver for Sprint 2. In short, it was a result of inexperience with planning sprints, but an important lesson to learn now rather than later. In the future, we plan to reduce the sprint velocity to accommodate other workloads from this course and others as the semester enters its final stages. We feel a sprint velocity of 45-55 is sufficient for the team, in addition to the required documents we prepare each sprint. This will help us keep our sprint in schedule.

This was one major oversight of the team. We estimated too much work to get completed, and ended up having to push back a very central part of the app – one that affects the demo-ability of a feature like Notifications (PP-14) and Do Not Disturb mode (PP-21). Choosing to use Auth0 meant the user ID cannot be passed to the backend without redux, which meant a TaskList cannot be generated for a specific user, which in turn means the notification dates cannot be built. Although each of these User Stories have been successfully implemented, their demo-ability relies on having redux setup.

Looking at the burndown chart, every team member managed to complete their respective user stories in the final days. This is because most User Stories have a substantial amount of work and cannot be completed in only 1 – 2 days. This is why the burndown chart is not staggered in its completion, and will continue to not be for the most part. Most students have other time demands from courses; meaning progress on each User Story is gradually completed. In a real workplace however, the full attention would be on the assigned User Story, meaning the burndown chart would be more staggered as it would be possible to complete the stories earlier in the week.

Lessons Learned

The important take away here is to not schedule so much work even if it technically fits an “ideal” amount of time – things can come up, delays can happen, merge conflicts and issue blocks can crop up, and in general – there should always be room for some mobility, some buffer zone instead of being on a tight schedule. It is better to complete User Stories fully than to take on too much and being unable to finish important Developer Stories. Importantly note that this was a **team** issue, not a single developer issue. As a team, we should have prioritized this story to be completed before the others this Sprint, and **as a team** we should have identified an excessive workload, even if we did manage to complete almost all of it.

The other takeaway we learned is that any UI work to the general layout should be implemented first and merged to development **prior** to starting up additional components. This is to avoid having one gigantic merge at the end where the new UI has to merge itself with every other component; it is simpler to do the vice versa for everyone involved. That way, the components are created with the new UI in mind, and don’t raise a lot of merge conflicts.