What technical debt has been cleared up?

https://code.cs.umanitoba.ca/comp3350-winter2024/timeline-a02-9/-/commit/8fde091d25d65fa44bdcc95648241865338d5fba

This commit was a refactor that moved and changed how time was collected from the timer. The initial implementation of this feature caused 2 bugs where the timer would skip break sessions and the time collection wasn't 100% accurate after a few minutes and this commit fixed those 2 bugs. We would classify this as reckless and deliberate because the initial implementation worked well enough, but the bugs were only found just before the deadline after some extra testing that hadn't been done earlier meaning there wasn't enough time to fix them.

What technical debt did you leave?

The RecurringGoalManager class within our project serves as a pivotal component for managing goals that recur over time. It's tasked with the critical functionality of repeating goals and their associated tasks based on predefined recurrence frequencies. This functionality is crucial for users who rely on our application to maintain consistency and progress in their tasks and goals over time. About this Feature Technical debt includes the following:

Overly Complex Functions: The initial design and implementation featured complex functions that performed multiple responsibilities, making the code hard to read, maintain, and extend. This complexity hindered our ability to efficiently implement new features related to recurring goals.

Facade Design Pattern Overuse: While intending to simplify interactions with the goal and task management functionalities, the overuse of the facade design pattern ended up obscuring the underlying complexities. This made it difficult to extend or modify functionalities without risking the introduction of bugs or inconsistencies.

Open-Close Principal Violation: The design did not adhere to the open-close principle, as extending the functionality for recurring goals required modifications to existing code rather than extending the system through new code. This limitation was a direct result of the issues.

Code Duplication: The presence of duplicate code sections indicated a missed opportunity for abstraction and reuse, leading to increased maintenance overhead and potential for inconsistencies.

Discuss a Feature or User Story that was cut/re-prioritized

We decided to cut "Split tasks between goals" feature. This was because the feature did not fit our vision of the application after several iterations. Our workflow model involved the user setting a Goal and creating Tasks required to achieve that goal. Since the mental model bound Task tightly to a Goal, we did not find any situation where it was advantageous to allow movement of Tasks between Goals and decided to scrap the feature.

System test

For our timer screen tests, we tested:

- Goal name display
- Timer start/stop/skip
- Task addition, deletion
- Task editing

To make the test not flaky, we leveraged our dependency injection framework to inject an in-memory variant of our SQL database, allowing each test to start from the same baseline. This also meant that task-related tests are completely isolated from one another, removing any possibility of changes to the task list from a different test from affecting the current one.

System test, untestable

Task reordering

One challenge we faced while writing system tests was trying to test our feature of rearranging the order of tasks. To move a task, we had to tap and hold on to a task and then drag it, but simply tapping on a task would also edit it. Thus, the gesture for moving a task was 2 steps and overlapped with another gesture. This made it so that writing a system test that moved a task would end with the task edit box opening and the task not being moved.

Statistics screen

Another challenge was our statistics screen, for some reason the library used for displaying the graphs were not detectable by our assert statements. To somewhat counteract this, the system tests written for the statistics screen simply navigates to the screen and pauses for a few seconds so a tester can look manually at the graphs to see if they are correct.

Recurring goals

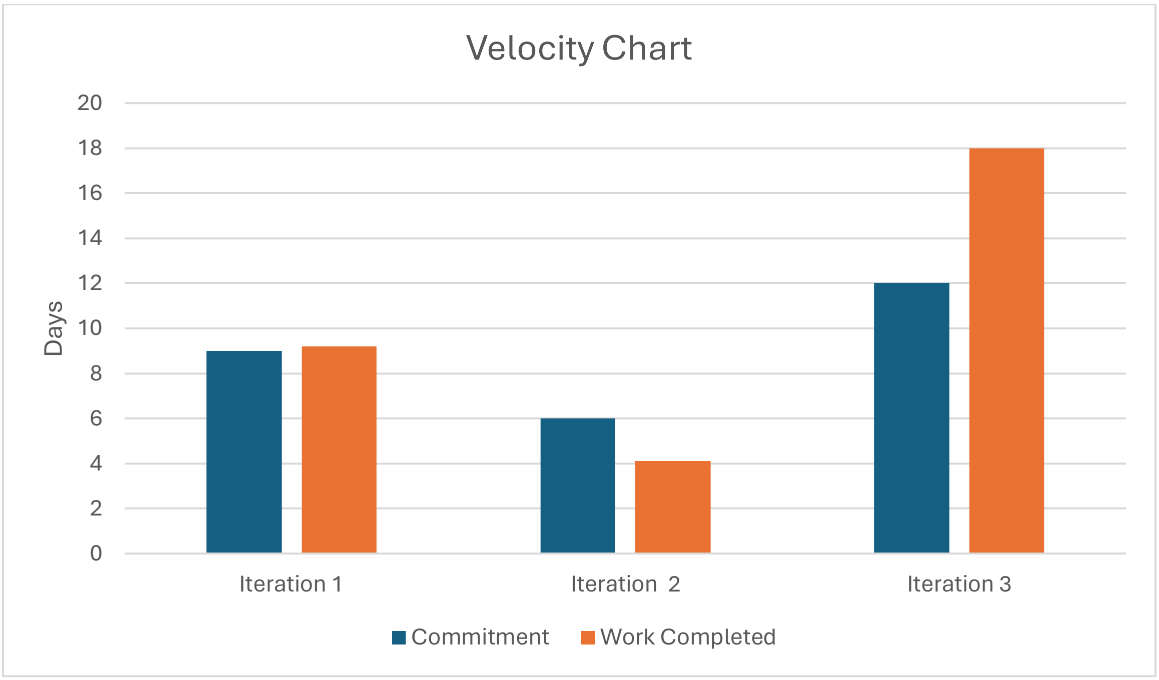
While writing system tests for the Recurring Goal feature, one significant challenge we encountered involved navigating between the task and goal pages. Initially, it wasn't clear how to switch back and forth between these pages effectively within the tests. After a deep dive into the documentation, we discovered the solution was to utilize navigation commands designed to facilitate transitions between the task and goal pages. This allowed me to simulate a user's journey more accurately, improving the robustness of our tests.

Another hurdle was encountered when testing the functionality that generates duplicate goals in the recurring goal test suite. Specifically, the system test was unable to assert the presence of a second, identically created goal. Despite the goal's successful creation within the application, the automated tests failed to detect and confirm its existence. This limitation necessitated a manual verification step, requiring human intervention to visually inspect and confirm that the duplicate goal was indeed generated as expected. This aspect of testing highlights the challenges of automating the validation of certain dynamic features within an application, especially when dealing with elements that the testing framework struggles to recognize.

SOLID

OCP Violation: "Back" destination is hard-coded: comp3350-winter2024/techtitans-a01-9#55

Velocity/teamwork



Initially our estimates were pretty good, but in iteration 2 we overestimated the amount of time required. In iteration 3 we overcorrected and significantly underestimated the amount of time required to complete our features. So as the project ran its course, we got worse in our estimations.

Learned

One unexpected thing we learned from this project was the Kotlin language. At the start of the project, one member made a good case for how Kotlin would make some things easier to accomplish for the project, so we all agreed to use it. However, most of us were new to using Kotlin and struggled a little to use it during the early stages of the project. As we worked more and more on the project, we collectively got a better grasp of how to use it. We didn't think we'd end up learning a new programming language but are happy to have gotten the chance to do so.