1. Project Description: Line numbers should continue through the functions list section so we don't have to do math in our heads to get the line number references. MS Word has a line numbering feature in the tab "Layout/Line Numbers". Page numbering looks good. (Remember though, the final iteration 1 submittal is a single pdf file with all these things in that one file referenced in the table of contents.)

Line numbers, continue to functions. → Will be done
Add Requirements, Use case, High Level Use Case,

2. Requirements: R5.1 is an implementation approach. You can keep this requirement but you must use a checkbox in the design . . . even if during the design and programming the team finds a better implementation approach. This requirement is currently written as a constraint (which is fine but limits your design choices in the next iteration). Check spelling on constraint ID 3.

No changed. Just check Rquirements.

3. UCD: The names for each of the subsystem diagrams should be provided. For example, for the upper left diagram, use a name like "To Do Planner System: Task Management"; for the upper right diagram, "To Do Planner System: Display Management". Do the same for the bottom task set. The use case names should be "exactly" the same names as used on the HLUC. Check HLUC4, HLUC6, and HLUC7, for example.

Name for sub-system. To do Planner: Task Management and so on
UC name = High leve HLUC 4,6,7 should be checked

4. HLUC: Keep TUCBW and TUCEW short and simple. For example, HLUC1a: consider deleting the words "at the bottom right corner of the application". Similarly, for HLUC9 and 10. These words are not really needed here. Check HLUC7. In order for this to be a use case, it must start with an actor, end with an actor, and perform a useful function. This UC does not appear to meet these criteria as presented. If the user can set a notification time, then that might be different.

Notify user does not come under UC. ad extra matter is not required.

5.

Domain Model: There should be a user class in the diagram somewhere. For class names, these should all be nouns . . . 'things' that we might 'see' in the real world. So, for example check out this sentence, "The user creates a task". We might then expect to see a user class and a task class in the domain model. There would be an association between these two classes, direction . . . user-to-task, association name as "creates", with multiplicity on the task class side as '*'. The Task Class would have attributes as Task ID, Name, Description, Priority, due date, ... We might look at the project description some more to find other nouns/classes we might select to use in a similar fashion: Dashboard, Section . . . . "A user creates and names a section." "The dashboard displays tasks according to sections, priority, ... with a default display as . . . ." Then the dashboard and section classes would have associations with the user, the section class would have another association with the dashboard class, . . . , and so on. Suggest the team take another look at our class Canvas "Modules/Projects Information/Android Agile Project Iterations . . . .pdf" file page 18 for an example domain model. Notice that the class names are not functions or use case names; the names are 'things' that might be seen or used in 'the real world'. There is another example on Canvas Modules/'Class Notes'/'Car Rental System Domain Modeling example.pdf'. You can trace classes, attributes, and relationships from this domain model back through the schema to their project description. Back to chapter 5.4 in our textbook, checkout the examples provided through this process of creating a domain model. Compare that to your team's app project description; you may find that the "to-do planner" app has a much simpler domain model than the team is considering at the moment.