From: dcsdcr@nus.edu.sg
Subject: IVLE: CS2103/CS2103T: Common issues in V0.0
Date: 14 September, 2014 9:27 pm
To:

Common issues in V0.0

Here are some common issues we noticed in V0.0 submissions. While you wait for tutor feedback on V0.0 (to be released shortly) you should take some time to go through the below and see if they can be used to improve your app design.

- **Not enough 'thinking through':** This is the most common problem, present in almost all submissions to some extent. To put it bluntly, if your final products are going to be as they are proposed in V0.0, most of them have no hope :-p Although we gave mostly positive comments because you are new to product design and you did not have much time to work together as teams, the fact remains that more 'thinking through' (and a large dose of common sense) is required to be put in between now and the final submission.
- **Written as a specification, not a user guide**: Headings such as 'system overview' belong in a requirements/system specification, not in a user guide.
- Not addr
- **Too formal**: Given the nature of the product, the user guide can be quite informal. No need to use a very formal language such as used in contracts.
- **Not addressed to the user**: Some forgot to write the user guide to the user. You should have used 'you' to address the user, instead of 'the user'. e.g. 'You can edit the task ...' instead of 'The user can edit the task...'
- **Unnecessary features**: Reconsider features such as login, sort, hashtag, priority. They are OK to have if you have a strong justification. Always ask yourself, 'What if we don't have this feature? Is it going to be a problem for Jim?'. A feature common to other similar software may turn out to be useless for Jim.
- **Calendar-like UIs (i.e., grids)**: They are a lot of work to implement and hard to get right. Much easier to link up with existing calendar interfaces such as Google Calendar.
- **Unix-like commands**: Excessive use of symbols make the command hard to remember/type. We are targeting fast typists, not necessarily UNIX system admins. No matter how fast you can type, normal letters are faster to type than certain symbols such as -@#$^. But symbols such as . , / may be easier than other symbols.
- **Not allowing overlapping tasks**: People do schedule things in the same period when they are not sure which one they are going to do. Not allowing it can be a great inconvenience. Highlighting and warning about overlaps is a better approach.
- **No escape for keywords in description**: The software must cater for the possibility of task description containing keywords such as 'from' 'to'. e.g., If the user wants to add the task 'watch day after tomorrow' (i.e., watch the movie named 'day after tomorrow') to the list, there should be a way to enter it without the software interpreting 'tomorrow' as the day for the task.
- **Multi-step CLI**: The user should be able to accomplish most tasks using a single command. For example, the approach 'type add, press enter, then type description, press enter again etc. is unacceptable.
- **Using terms unfamiliar to users**: Using the names mentioned in the requirements (e.g. floating tasks) in the user guide might not make sense to users. If you want to stick with such terms, at least you should explain the terms to users.
- **Not enough examples**: You guys complain about 'not enough examples' in handouts, but some of your user guides had hardly any examples :-p
- **Not enough details on app behavior**: It is not enough to state what the user does. You should also explain how the app responds so that the user can verify things are going right.
- **Silly defaults**: Schedule the task NOW if no time is specified? really? Choose smart defaults when you can, but don't chose silly defaults.
- **Not enough UI prototypes**: Even text based UIs need prototypes. For example, the format in which you are going to show the results of a user action.
- **Not catering for advanced users**: It is good to make the app friendly for beginners, but do not forget that you user will become an 'advanced user' after some time. Consider providing 'advanced' ways to accomplish things with less effort. e.g. shorter command aliases.
- **Suboptimal display formats**:
    - For example, listing of tasks should be nicely laid out, sorted in a meaningful way, aligned properly, etc so that the user can read the information quickly. For example, the text UIs below have similar content but one is much more readable than the other.

```
1. Submit CS2101 assignment 2 2013 Nov 4 23.59 00
2. CS2020 project meeting 2013 Nov 4 16.00 00 - 2013 Nov 4 18.00 00
3. CS2101 lecture 2013 Nov 4 12.00 00 - 2013 Nov 4 14.00 00
4. Dinner with Ravi and James Nov 4 18.30 00 - 2013 Nov 4 20.00 00
5. Comp club meeting 2013 Nov 4 08.00 00 - 2013 Nov 4 11.30 00
6. CS2103 tutorial Nov 5 09.00 00 - 2013 Nov 4 10.00 00
```

```
[Today: Thu Nov 4]=================================================

1. [8am-11.30am] Comp club meeting
...[0.30       ] [][]
2. [12nn-2pm   ] CS2101 lecture 2013
3. [2pm-3pm    ] CS2020 project meeting
...[5.30       ] [][][][][] [][][][][] [][][][][] [][][][][] [][][][][] [][][][]
4. [8.30pm-10pm] Dinner with Ravi and James
...[1.59       ] [][][][][] [][][][][]
5. [by 23.59pm ] Submit CS2101 assignment 2

[Tomorrow: Fri Nov 5]==============================================

...[1.00       ] [][][][][]
6. [9am-10am   ] CS2103 tutorial
```

    - Consider how the display area will look like for scenarios such as when you have many tasks in a single day, when you have 100s of tasks in the system, when a task has a long description, etc. Will the display are look neat and readable in those situations too?
    - Because the display area is limited (the app is expected to run in a small window), you should optimize the display format for space. What format should you use to display

should optimize the display format for space. What format should you use to display date/times so that it takes less space but still easy to understand? Is there any point in displaying year, seconds?

- **Inappropriate saving strategy**: Saving things only at exit command is a bad idea; Data will be lost if the the program is closed without using the exit command.
- **Command format optimized for developers, not users**: Don't design the command format solely to make parsing easier. Primarily, it should be easy to remember and type. Ease of parsing is secondary.
- **Not enough feedback to the user**: When the user executes a command, the feedback given should be informative. e.g. When giving feedback after adding a task, 'Success' is not good enough. Jim should see the details of the task added so that he can verify if it is indeed the task that he meant to add.
- **Unutilized welcome screen**: The welcome screen (what you display at the very beginning) can be used to display something that is useful to the user, such as the tasks for the day.
- **Half-baked features**: When you implement a feature, consider all aspects of it. For example, if your extra feature is GCal sync, consider things such as how to store floating tasks in GCal, how to sync older items in GCal with your software, how to sync repeating events, and so on. It is not necessary that you implement all those things. However, you should have considered all those things and you should have good reasons why you chose what you implemented over what you did not implement. "Oh, we didn't think of that" is not a good answer.
- **Mismatch between user stories and the user guide**: There should be clear correspondence between user stories and the user guide. You were supposed to conceptualize the product based on user stories.
- **No value from product survey**: Most teams don't seem to have used any insights from the product survey.
- **Other things to ponder**:
    - What are the most common queries a user may want to ask the app?
    - Is there any situation where a user wants to see a list *all* tasks?
    - Are there other types of tasks that we should consider, in addition to the three types mentioned?
    - Should we force the user to consciously identify tasks types or can the task type be dealt with internally most of the time?
    - What about conversions from one task type to another?
    - Is it better to show deadlines and timed tasks together or separately? Do people process those two types of things separately when they think about things they have to do?
    - What is the best way to handle overdue tasks?
    - When searching for a task, what can we expect the user to type? Surely, we cannot expect them to type the original tasks name precisely?
    - Is there a need to ask for confirmation when an undo feature is available?
    - How useful is automatic scheduling? Would you follow a schedule automatically generated by an app?