CS-E5220 - User Interface Construction D

Jiri Simell

Assignment #3: Command line user interface (CLI) / Dialogue design / Conversational user interface

1. Introduction

This report presents the implementation of a Christmas present wish list application. More specifically, focus is put on explaining the design process and justifying the design choices that are made. The aim of the application is to provide users an easy and effortless way of creating and managing their wish lists through a command line interface (CLI). The primary target group of the application are users who have access to a command line, and know how to run the application through one.

Since the focus of this assignment is on interaction design, the developed application will not have a backend implementation. This means that once the application is closed, data will be lost, meaning that the items on the wish list will disappear. However, the interface design and the structure of the dialog is not affected by this, which means that focus can be put solely on the design of the CLI. Furthermore, implementing the CLI even without backend code provides a great foundation for possible backend development in the future.

However, there are several challenges that must be overcome in the design. For example, the dialog and structure of the application requires careful consideration. The goal is to design an application with a high usability and a good user experience (UX) for a specific user group. Key factors in achieving this are clear dialog messages and a logical application structure.

2. User Description

The target users of the application and their user needs can be defined with a detailed scenario description based on a specific persona. Using such a scenario is an effective way of describing the most important application requirements, because it concretely illustrates the user needs towards the application, and the key characteristics of the target users (Open Design Kit, 2024). The scenario is as follows:

> *Pekka is a software developer who is excited that Christmas is getting closer and closer. Each autumn, he writes a present wish list containing all the wish ideas he comes up with during the months leading up to Christmas. Typically, he manages his wish lists with a command line text editor, because he often gets his wish ideas while he is working on his computer. However, when getting a new idea, he sometimes feels like it is cumbersome that he always*

has to first locate the wish list file, then open it with a text editor, and finally write the new wish idea into the text file. Therefore, he would benefit from a quick and easy command line application for managing his wish lists, which would allow him to immediately add items to the list when he comes up with a new wish.

Since Pekka has easy access to a command line and knows how to use it, he is able to run the wish list application quickly and easily on his computer. If he gets a present idea during the day, all he needs to do is start the application through the command line, and add an item to his wish list with a single command. However, as already mentioned earlier, this type of application will not be useful for people who have no experience with command line applications. Therefore, the user group of the application consists solely of people who know how to use a command line, and have easy access to one.

3. Application Design

After defining the target users of the application, the design phase can be entered. Lewis and Rieman (1993) introduced a task-centered design process in their book, and its early stages are utilized in this design process. Furthermore, the eight golden rules of interface design by Shneiderman (2016), and the seven stages of user activities introduced by Norman (1986) are utilized in justifying the design choices.

The first stage of the task-centered design process, identifying the users and their needs, was in fact described already in the previous section. Continuing the design process, the second stage consists of identifying the main tasks that the system will be used for (Lewis & Rieman, 1993). In the case of our application, the tasks are:

1. adding an item to the wish list
2. removing an item from the wish list
3. listing the items in the wish list
4. listing available commands and their definitions
5. closing the application

From the above task definitions it can be observed that the application must have five different commands. According to Lewis and Rieman (1993), in the third stage of the task-centered design process, one should gather inspiration from other similar systems as the one that is being developed. In the case of this application, I used my previous experience with CLIs as the inspiration. In other words, I tried to come up with command structures that I have seen in other similar applications. Furthermore, the commands should strive for consistency (e.g. ‘add item’ and ‘remove item’ commands should have a similar structure) (Shneiderman, 2016). Considering these facts, I ended up choosing the following commands and command descriptions for the application:

1. add <item> adds an item to the wish list
2. remove <item> removes an item from the wish list
3. list lists the items in the wish list
4. help lists the available commands

5. quit closes the application

Next, the dialog used in the application must be defined. According to Norman (1986), the first activity that a user performs when interacting with a UI is perceiving the state of the system. Therefore, the application should print all the available commands when it is opened. The command descriptions should be clear and unambiguous, so that the result that the user expects from a specific command matches the actual result that occurs. Furthermore, the fifth golden rule of Shneiderman (2016) states that errors should be prevented by the UI. Printing all the available commands to the user when the application is opened guides the user towards correct commands, preventing errors.

In addition to this, the application should provide clear error messages to the user if invalid commands are inputted. In fact, the third golden rule of Shneiderman (2016) states that a UI should offer informative feedback to the user after all possible user actions. Therefore, the application must provide feedback to the user after all possible inputs (except an empty input). Based on these heuristics, a task diagram for the application can be constructed (Figure 1). Performing this type of command and task analysis follows the fourth and fifth steps in the task-centered design process (Lewis & Rieman, 1993).

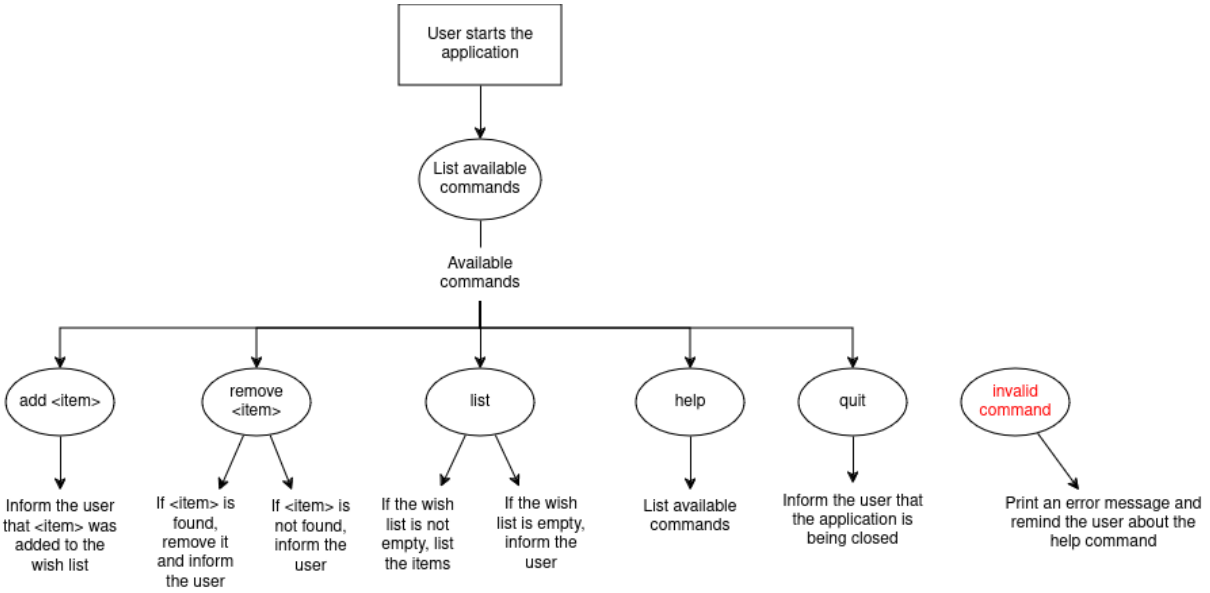

Figure 1. The task diagram of the application.

4. Implementation

Now that the design of the application has been specified, and the design choices have been justified, the only task left is to implement the application. In the task-centered design process, the next stages consist of multiple iterations of prototyping, building the design, testing the design with users, and improving the application in each iteration based on new ideas and testing results (Lewis & Rieman, 1993). However, due to the simplicity of the application at hand, and the absence of real users to test the application with, I decided to

move straight into implementing the final version of the application as a Python script. The script can be found in Appendix 1, and the opening view of the application can be observed in figure 2.

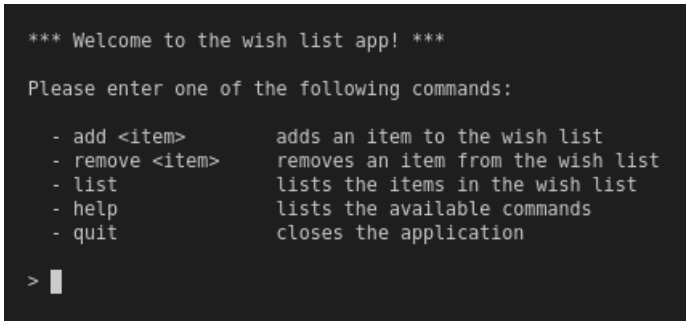

Figure 2. The opening view of the application.

5. Discussion and Conclusion

Obviously, the current version of the application is nowhere near finished. The absence of the backend implementation makes the application useless since the list will be lost when the application is closed. However, the dialog and command design of the application has at least been defined and justified based on valid heuristics. Although all the heuristics of Shneiderman (2016) could not be applied to the application design due to its simplicity, a good design was achieved nonetheless.

However, there are of course clear areas for improvement in the design. For example, the current version of the application is not usable for people who do not know how to use a command line. Moreover, using the command line to manage a wish list is probably not the fastest possible way of keeping track of Christmas present wishes. Therefore, implementing a graphical user interface (GUI) for the application would make the application more useful, while simultaneously expanding the user group as well. For example, a web based GUI would allow anyone who has access to a web browser to easily access the application.

Moreover, the current version of the application includes only the most basic functionalities (i.e. adding items, removing items, and listing items). Adding more functionality to the application would make it a lot more useful. For example, adding the ability to export a wish list or share a wish list to other users would increase the practicality of the application since wish lists are typically aimed to be shared to other people. However, sharing wish lists to other users would of course require user management, which would have to be a part of the backend implementation of the application, increasing the complexity of the design.

References

Lewis, C., & Rieman, J. (1993). Task-Centered User Interface Design: A Practical Introduction. http://hcibib.org/tcuid/

Norman. (1986). Cognitive Engineering. User Centered System Design. NJ,: Lawrence Erlbaum Associates, Inc.

Open Design Kit. (2024). Personas. http://opendesignkit.org/methods/personas/

Shneiderman, B. (2016). The Eight Golden Rules of Interface Design. University of Maryland. https://www.cs.umd.edu/users/ben/goldenrules.html

Appendix 1

```
# Prints a list of all the available commands
def help():
  print("""Please enter one of the following commands:\n
  - add <item>      adds an item to the wish list
  - remove <item>   removes an item from the wish list
  - list            lists the items in the wish list
  - help            lists the available commands
  - quit            closes the application""")


print("\n*** Welcome to the wish list app! ***\n")
help()
wishList = []
quit = False

### Main loop ###
while not quit:
  print()  # Print an empty line before each input
  cmd = input("> ")
  print("  ", end="")  # To align the output nicely with the input

  if not cmd.strip(): continue  # If command is empty, do nothing

  match cmd.split(maxsplit=1):  # Split the input into the command and the
possible item

      case ["add", item]:
      wishList.append(item)
      print(f"Item {item} added to wish list.")

      case ["remove", item]:
      if item in wishList:
      wishList.remove(item)
      print(f"Item {item} removed from wish list.")
      else:
      print(f"Can't remove {item}. No such item in the wish list.")

      case ["list"]:
      if len(wishList) == 0:
      print("Wish list is empty.")
      else:
      print("Items in wish list:")
      [print(f"  - {item}") for item in wishList]

      case ["quit"]:
      print("Closing the application...\n")
      quit = True
```

```
case ["help"]:
help()

case _:
print("Invalid command. Type 'help' to see the available commands.")
```