HCIN5200 UI Software Exploration Project

Direct Manipulation and Command Line Podcast Manager Interfaces

Joshua K Carr

October 20, 2018

1 Domain and Requirements

Podcasts are episodic audio (and sometimes video) series which are hosted online and can be downloaded by listeners on-demand. They can be viewed as a digital homologue to a traditional radio show, but their digital medium allows users to download and listen any time (rather than a fixed broadcast time, in the case of radio), and to play, pause, and skip back and forward through the recording. Some podcasts are initially broadcast live and then archived. Many software applications exist to support and simplify managing and listening to podcasts across various platforms (*e.g.* Google Podcasts, iTunes, Spotify). Generally, these applications simplify podcast management by automatically downloading new episodes to the local device and providing basic playback functionality, as well as the ability to search or browse through various podcasts and subscribe to them if desired.

The goal of this project is to explore and compare the advantages and disadvantages of two distinct interface styles in the domain of podcast management applications. To this end, I designed wireframe prototypes of a basic podcast management application using a command-line interface (CLI) and a direct manipulation interface (DMI [5]). The requirements of both interfaces are as

follows:

- *Library management:* Users can view a library of all the podcasts that they are subscribed to, unsubscribe from podcasts, and reorder and sort them. There is a visual indication of which podcasts have unplayed episodes, and which episodes are unplayed. Episodes can be played or added to the queue regardless of whether they are "unplayed" or "played". Podcasts are automatically marked as "played" once complete.

- *Queue and Playback:* Podcasts episodes are played by adding them to a queue, where the first element is the one currently being played. Items in the queue can be reordered or removed. When playback of the episode is complete, the episode is popped from the queue and the new first element is played, until the queue is exhausted or the user stops playback.

- *Browse and Search:* Users can browse available podcasts sorted by most popular, new, or trending, view a list of recommended podcasts based on their listening history, or search podcast titles. Selecting a podcast displays a description and allow users to subscribe to the podcast or add individual episodes to the queue (with or without subscribing).

2 Interface Styles

The first interface style to be explored was DMI [5]. According to Shneiderman [5], direct manipulation relies on continuous representation of interface objects and interactions centered around the manipulation of those objects. Actions in a DMI should be rapid, allowing the user to immediately see the results of their input, as well as incremental and reversible to provide users with a sense of control. An important concept in direct manipulation is that of **affordance**, whereby the design of the system indicates the way in which it is to be used (*i.e.*, a chair *affords* sitting, a button *affords* pushing). This style was chosen because many currently available podcast management apps implement direct manipulation to some extent in combination with other interactions styles (*e.g.* menus and form fill-in). Thus, an interface that uses direct manipulation exclusively (or to the greatest extent possible) should still be relatively familiar to users of other podcast apps, making

it relatively simple to evaluate the strengths and weaknesses of this interface style in relation to a more general mixed-style approach.

Conversely, a CLI was chosen because it starkly contrasts the design of typical podcast managers, which do not use command language at all. Briefly, CLIs do not use a graphical user interface (GUI), and function by users entering text commands into a command line terminal (*e.g.* UNIX terminal, Windows Command Prompt). CLI users can chain commands together and use various optional parameters to perform complex actions with a single text input. Components of CLI interaction are commands, arguments, and options. A command specifies a particular function which may or may not accept arguments specifying the elements that the function is to be performed on, and options can be added to the command in modular fashion to modify functionality. Consider the following example using bash: `rm -r example_directory`. In this example `rm` specifies the remove command, the option `-r` makes the function recursive, and the argument `example_directory` specifies the file/directory to apply the function to.

CLIs are generally considered to have a greater learning curve than GUI-based interfaces, as the user must memorize or refer to a list of commands in order to operate the interface. However, CLIs are generally powerful in the hands of an experienced user, *i.e.*, the user can complete actions faster and with greater flexibility, but only if they know the correct commands to execute. CLIs are also extensible by users via shell scripting for even greater efficiency and customizability. Therefore, designing a CLI for podcast management is rather unconventional and allows for a novel exploration of this interface style in a domain to which it is not typically applied.

3 Design

3.1 Direct Manipulation Interface

The DMI was designed as a web application. Upon opening the application, users are first presented with the Library interface (Figure 1), where they can view podcasts and episodes, reorganize and unsubscribe from podcasts in the library, or add episodes to the queue. Podcasts are represented by square tiles with images (generally the podcast logo), which can be expanded by clicking or

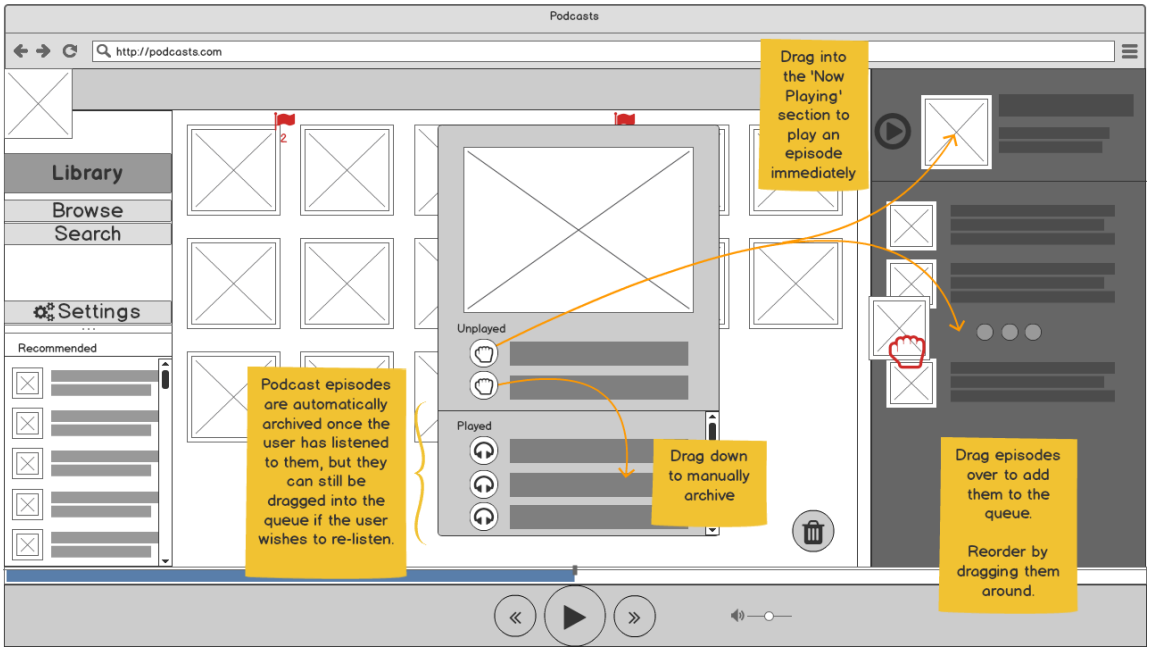

Figure 1: Schematic of the podcast Library interface in direct manipulation style.

pressing to show episodes within. Podcast icons and episodes are manipulable objects, and the core functionality of the interface revolves around the user dragging them to various parts of the screen. Podcasts can be reordered by dragging them around the main library window, or unsubscribed by dragging the podcast icon over to the **Trash** icon. If a podcast has unplayed episodes, a flag is displayed in the top right corner of its icon. A podcast icon can be dragged into the queue area on the right side to add an episode to the queue; if the podcast has more than one unplayed episode, the newest or oldest unplayed episode will be added to the queue, depending on the user's settings.

Clicking or pressing on a podcast icon will open the podcast menu. The podcast menu overlays the library screen, showing a larger version of the podcast image and a list of episodes separated into **Unplayed** and **Played** episodes. Episodes can be added to the queue by *grabbing* the associated icon (indicated with a hand) and dragging into the queue area, or marked as Played/Unplayed by dragging from the upper to the lower part of the menu. Once episodes have been played, they are automatically marked as Played and moved into the appropriate part of the menu.

Items in the queue can be reordered by dragging (shown in Figure 1). The top section of the

queue is reserved for an episode that is currently playing. Dragging an episode into this region will play it immediately; if an episode is already playing, it will be pushed back into the first position in the queue. Once playback of an episode is complete, the next item in the queue will begin playing and so on, until the queue is exhausted or the user manually stops playback.

Example Use Case: A user wants to listen to a podcast. They open the application to view their library and notices that there are a few podcasts with new episodes. The user selects each podcast in sequence to open the podcast menu and view the new episodes. There are a few new episodes that the user is interested in listening to, so they drag them into the queue. The user selects the play button and the first item from the queue begins playing. After some time, the user realizes that they are not enjoying the episode that is playing, so they drag it out of the queue area to stop it. In fact, the episode is so bad that the user no longer wants to be subscribed to that podcast; they drag the offending podcast over the trash can icon to remove it from the library. The next episode in the queue begins playing automatically once the previous one is removed.

The podcast Browse interface is shown in Figure 2. The queue and playback controls remain visible in the browse interface so that users can still listen and manage their queue while browsing podcasts. Trending podcasts (*i.e.*, those with high listenership over a specific timeframe) are shown at the top, and below is a menu that allows the user to view the most popular, newest, trending, or recommended podcasts (based on listening history).

Selecting a podcast brings up a podcast menu that is slightly different from the Library podcast menu. This menu contains an enlarged image at the top, and a text description of the podcast directly below, and recent episodes at the bottom. To the left of this menu, there is an icon that can be used to subscribe to a podcast and send it to the library by dragging the podcast icon onto it. Episodes can be dragged into the queue regardless of whether the user has subscribed to the podcast, allowing them to try out a podcast before deciding to subscribe.

Example Use Case: A user is interested in finding new podcasts to listen to. They open the Browse interface and begins looking through the available podcasts and reading their descriptions. Not seeing anything interesting, they navigate to the "Recommended" section to see personalized

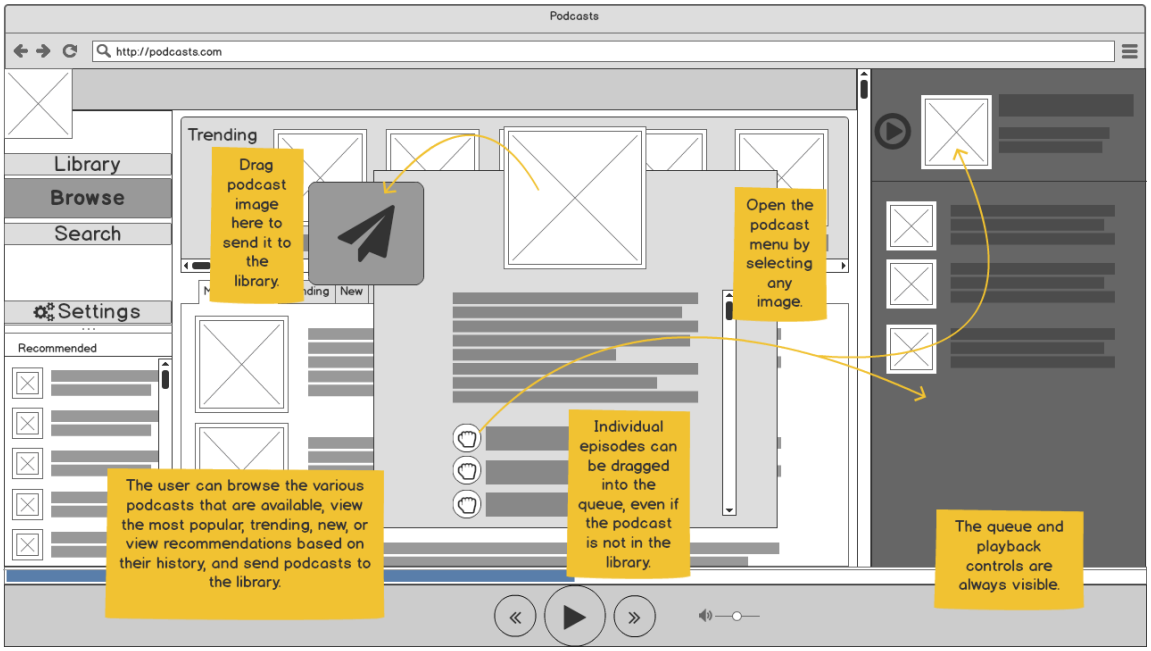

Figure 2: Schematic of the podcast Browse interface in direct manipulation style.

recommendations based on his listening history. The user opens a podcast and drags the most recent episode into the queue to see whether it is any good. After listening for a few minutes, the user decides that he wants to subscribe to the podcast, so they drag the podcast icon onto the "Send to Library" icon.

The search interface (not shown) allows users to browse the podcast catalogue with more control. Users can enter a search term to return results with the search term in the title or description. The user can further narrow results using filter options (*e.g.*, podcasts in English only), sorting (by popularity, by newest, alphabetical), and/or categories (*e.g.*, science or comedy). If the user selects a podcast, a podcast menu opens which is identical to that in the Browse interface, allowing the user to listen to episodes or subscribe to the podcast.

3.2 Command Line Interface

A CLI was designed for use with a simple UNIX/Linux terminal emulator. The application (called `pod` for simplicity) has four primary commands (`library, play, queue, browse`) for podcast library management. A schematic diagram of the commands and their usage can be found in Figure 3. None of the commands receive arguments, but they all have options that can be added to the command to modify their function. The basic structure of a command is `pod [COMMAND] --option [OPTION ARGS]` (*e.g.*, `pod library --sort newest` will return a list of subscribed podcasts sorted by newest episodes). Many of the commands open menus (demonstrated in Figure 3) wherein menu options are presented alongside numerical digits (*e.g.*, `\1) Add to Library"`) and the user can select an option by typing the corresponding digit into the terminal.

The podcast library interface is accessed by the command `pod library`, which can be given the options `--sort` to organize the list, or `--pod` to open the menu for a specific podcast. When the `library` command is given, the applications displays a list of podcasts and prompts the user to make a selection by entering a corresponding numerical digit. This opens the podcast menu, which displays the latest episodes for that podcast, with unplayed episodes marked with an asterisk (*). The user can select a podcast episode in a similar manner, which then opens a menu for that episode which includes options to play the episode, add it to the queue, or mark it as played/unplayed.

The queue is accessed by the command `pod queue`, which has one option (`--play`) that takes no arguments. If the `--play` option is given, the application will play the first item in the queue and exit. Otherwise, a menu will be displayed which allows users to select and manipulate items in the queue ("Play now", "Move", "Remove").

The play command (`pod play`) with no options will simply play the queue from the first item. The option `--pod` will play an episode from a specific podcast (it is is not in the queue, it will be added to the front). If the podcast specified by `--pod` has more than one unplayed episode, the episodes will be displayed and the user will be prompted to select one. The option `--index` accepts an integer argument, and will play the *nth* item from the queue.

Finally, the browse command (`pod browse`) is used to browse and search the catalogue of

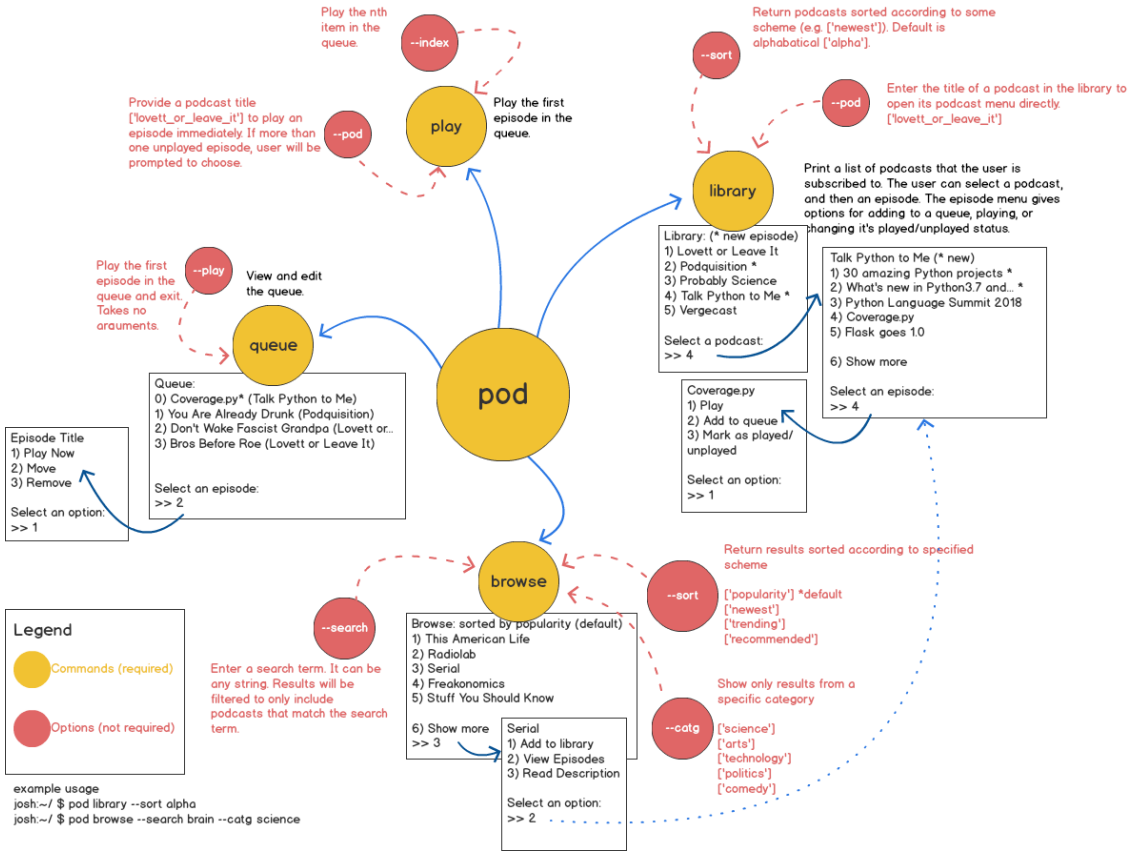

Figure 3: A schematic diagram of the basic command structure of the CLI podcast manager.

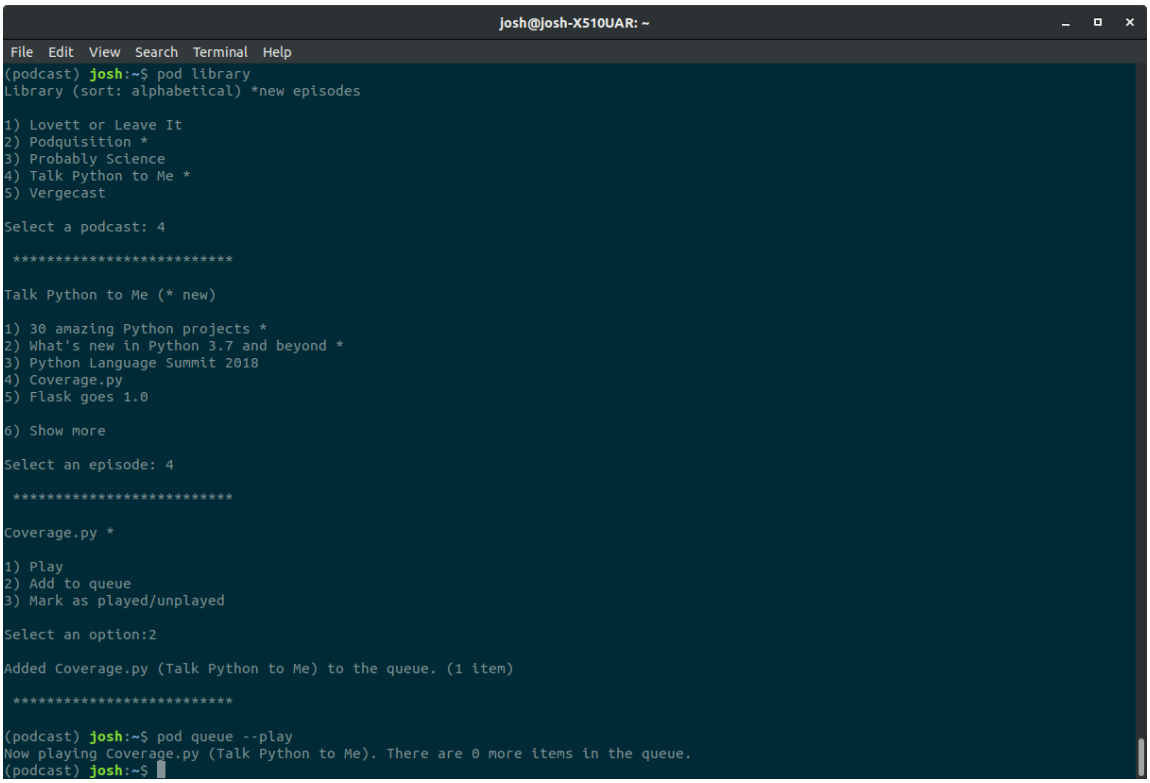

Figure 4: An example of the Podcast CLI in use. The user browses their podcast library, selects a podcast series and then an episode, adds the episode to the queue, and plays it.

available podcasts. Given no arguments, it will display an unfiltered list of podcasts sorted by popularity. The user can select a podcast to add it to the library, view episodes, or read the podcast description. The `--search` option accepts a string argument and will filter result to include only those that contain the search term in the title or description of the podcast. The `--sort` option is used to specify a sorting scheme (*e.g.*, newest, by popularity, recommended). The `--catg` option accepts a string argument and will filter results to include only podcasts of a specific category (*e.g.*, comedy, science, politics). Figure 4 demonstrates an example of the interface in use.

4 Perspective Analysis

In the following sections, I will analyze these designs in the context of cognitive psychology, specifically focusing on the domain of learning and memory. A critical dimension of interface usability is the time and effort required to learn how to use it, as well as the ability to retain this information over time. I will discuss a few cognitive phenomena related to learning and memory and their relation to the usability of the proposed interface designs.

4.1 Recognition vs. Recall

An important distinction in the learning and memory literature is between **recognition** and **recall**. While the two processes are considered to be continuous [6] and to rely on the same retrieval mechanisms, a critical difference lies in the context in which each is invoked. Recognition occurs when a memory is retrieved due to a cue or prompt that triggers it. Remembering about an appointment by seeing a note on a calendar is an example of recognition memory; the note is a *cue* that triggers the retrieval of the associated information. Conversely, recall occurs when there is no cue or prompt to trigger memory. For example, when writing an exam a student must remember a particular mathematical formula, they must use some degree of cognitive effort to retrieve it from memory. Thus, a major point of distinction is that recognition is *passive* (*i.e.*, it is elicited without the need for cognitive effort), whereas recall is *active*, requiring conscious effort to retrieve the memory from long-term storage. Another way of describing this dichotomy is that recognition involves knowledge *in the world*, while recall involves knowledge *in the head* [4]. There is necessarily overlap between these two concepts, but for the purpose of this discussion it is beneficial to examine them individually.

Direct Manipulation: According to Shneiderman's [5] view, a requirement of direct manipulation is that objects of interest are always represented and do not cease to exist. Clearly then, DMIs rely more strongly on recognition memory than recall. The manipulable objects in a DMI can be viewed as cues; the user is immediately reminded of their functionality without having to search through their memory to remember what to do. This approach presumably decreases the cognitive

burden associated with the interface. This approach benefits the learnability and long-term retention of the interface. If a user does not use the interface for some period of time, there are plenty of cues with obvious hints as to their function available to assist users in remembering how to use it. The knowledge of the interface is largely stored *in the world* (or *in the interface*).

Command Line: CLIs rely almost completely on recall memory. The only information conveyed by the interface is a basic text prompt which gives no hints as to what commands are expected or, indeed, what commands are even available. All information about the use of the interface must be stored *in the head*. This is a problematic feature of CLIs and severely impacts their usability, especially for non-expert or non-technical users. If one does not use a particular CLI for some time, the design does not support an easy return to use; the user will likely have to re-learn the system and, while there may be some recognition component as the user encounters commands that they have used before, there is necessarily a greater cognitive overhead due to the need to actively retrieve information from memory. It is, of course, possible that a user could refer to documentation (CLIs usually have `--help` commands which provide a list of commands and their functions), or a cheat-sheet, or even write down instructions on paper for how to complete various tasks. Still, it can be argued that an interface that requires constant references to outside sources in order to complete basic tasks is not what most would consider to be a highly *usable* interface.

Based on this analysis of recall and recognition, there is a clear benefit of the DMI over the CLI approach. With a DMI, the information that a user needs is presented within the interface; all of the items of interest are visible, and the interactions needed to accomplish a specific action are relatively straightforward. CLIs, conversely, provide little or no information to the user unless they go out of their way to reference documentation, and the commands needed to engage with the interface are complex. The need to reference external materials is a barrier to the user's primary task which detracts from the usability of the system.

4.2 Chunking

Chunking is a memorization strategy that involves combining pieces of information into meaningful segments (*i.e.*, **chunks**), which allows them to be more easily remembered [1, 2]. This phenomenon was demonstrated in a classic experiment involving chess masters and novices. The number of possible legal chess board configurations is somewhere in the neighbourhood of 10^{45}; remembering all possible configurations should be an impossible task. Chase and Simon [1] found that, while truly random combinations of pieces were not reproducible by masters or novices after viewing them for five seconds, more experienced chess players were able to reproduce a board more accurately than novices only when the configuration of pieces was one that could feasibly occur in a real game. Upon further examination, it appeared that the chess masters were able to accomplish this by considering the board configurations not as individual pieces, but as a combination of meaningful **chunks** of pieces. Interestingly, this chunking method seems to be implicit; chess masters did not rehearse or practice learning chunks, but rather they were understood implicitly as a consequence of a great deal of experience and exposure to these patterns. Later evidence (reviewed by Gobet et al. [2]) found that chunking mechanisms are found in a wide variety of human learning tasks, suggesting that chunking is not specific to chess players but rather is a fundamental mechanism underlying learning and memory of complex meaningful structures.

The chunking phenomenon has particular relevance to CLIs. The possible command configurations that a user can supply to a terminal are so numerous as to be practically infinite. An important consideration, however, is the *meaningfulness* of commands; they are not random, but consist of meaningful text commands organized in a predictable structure. This fact makes CLIs an excellent candidate for chunking techniques. With enough expertise and experience using a system, the problems with memorability of CLIs should disappear as the user develops a greater understanding of the meaningful structure of commands and can represent them in memory as chunks, rather than granular components. Indeed, there is a general consensus that CLIs are difficult for beginners, but powerful in the hands of experienced users. This maps nicely onto the findings of chunking with chess players [1], suggesting that chunking techniques underlie the impressive memory abilities of

experts for information in their domain of expertise.

4.3 Summary

Having discussed direct manipulation and command line interface styles in the context of learning and memory, a few conclusions can be drawn. At the most basic level of usability, DMIs have an advantage in terms of learnability and memorability based on the fact that they rely on recognition rather than recall; all of the information that a user needs to interact with the application is presented in plain view within the interface. The need for users to memorize and recall the commands needed to use a CLI means that there is an inherently steeper learning curve which makes these sorts of interfaces poorly suited to novice or casual users. The world of CLIs is, however, not totally bleak. The memory phenomenon of chunking can significantly reduce the space of possible commands and facilitate efficient use of the interface, but this technique requires a substantial amount of experience with CLIs to be applicable.

5 Discussion and Comparison

Thus far I have presented designs of a podcast management application in two distinct styles: direct manipulation and command line. The purpose of this endeavor was not to create the best possible interface *per se*, but to explore the advantages and disadvantages of these interface styles in relation to each other. It would likely be unfeasible to design an interface to maximize usability using only one design paradigm–modern user interfaces generally borrow from many styles to leverage their strengths and accommodate their weaknesses. The final section of this paper will reflect on the two interface designs in relation to the field of human-computer interaction (HCI).

The direct manipulation framework was proposed by Shneiderman [5] and aimed to bring together several components of interface design into a comprehensive framework. Many components of direct manipulation stem from **WIMP** (Windows, Icons, Menus, and Pointers) designs, which have been prevalent since the Xerox Star and its **desktop metaphor** [3]. The goal of the direct manipulation framework is to develop systems that are immediately comprehensible to novice users

and which facilitate exploration and mastery. In particular, immediate reversible operations allow the user to explore the interface with immediate feedback to know whether the system is doing what they intend, and without the fear of permanently damaging the system or data because any operation can be immediately reversed. This type of exploration interaction is critical for the user's ability to become comfortable with the interface. A new user's first experience with a software is generally not goal-oriented; they aim to understand how the interface works, what it is capable of, and to develop a mental model of the various actions that they can undertake and the corresponding results.

The podcast DMI discussed here attempts to adhere to these principles to the greatest extent possible. In particular, the aim was to ensure that anything that the user could conceivably want to do at a given moment would be readily available and apparent, and that any given interaction could be achieved by grabbing, dragging, or pressing (*i.e.*, directly manipulating) the objects of interest. For example, important interface elements such as the queue and playback controls may need to be accessed at any time, whether the user is in the library, browse, or search screens; these elements are, therefore, present on every screen of the interface with identical positions and layouts. I aimed for consistency in the types of actions that would be used to manage the interface. Specifically, the act of dragging an item (whether a podcast series or a single episode) onto the queue will add it to the queue, regardless of what screen the user is on and whether the particular podcast is subscribed to or not.

In the design of this interface, a major limitation of direct manipulation became apparent: the interface, while easy to use in theory, allows the user to do only one action at a time. This may not be a problem if the user's needs for podcast management are simple, it could quickly become tedious if there are many operations that the user wishes to complete repetitively. For example, a user may wish to mark all episodes of a podcast as unplayed, or add episodes from a particular range of dates to the queue. The DMI as such provides no mechanism to do this expediently or easily – the user must manually perform the desired action on every relevant object. This could quickly get out of control as the user's podcast library increases in size, becoming frustrating, impractical, and potentially completely unfeasible as the size of desired operations increased. The inability of direct

manipulation systems to perform batch operations is a major limitation of this interaction style.

The podcast DMI could be improved by implementing elements of other interaction styles. In particular, a comprehensive menu system could be implemented to allow users to, for example, select all unplayed episodes of a particular podcast and perform some action on them, such as marking them as played or adding to the queue. The search functionality in the interface was intentionally kept limited to maintain the direct manipulation style, however a more comprehensive search that makes use of form fill-in could be used to expand the search system to include searching episode titles, searching within the library or browse menu, and more comprehensive filtering of search results.

Command line applications are the oldest form of HCI and were the sole method of interaction in some of the earliest consumer computers. Though this method has usability issues outlined above, the continued development and popularity of some command line applications (*e.g.* git) demonstrates that there is still a place in HCI design for this type of interface. CLIs have a relatively high skill floor, meaning that a user must learn a great deal before being able to accomplish meaningful tasks. But, that said, command line applications provide a high level of functionality once the user has surmounted the learning curve. Batch operations allow a user to perform complex tasks on an essentially limitless number of objects with a single command, and the time required for the task to complete is determined only by computational factors, rather than user inputs. Contrast this with direct manipulation, wherein every task must be manually performed and the time requirement scales both with computational complexity of the task *and* the time required for the user to give the appropriate inputs every time.

As an example, consider the case of moving files between directories. In a pure direct manipulation interface, the user would need to grab each file individually and manually move them into the target directory. Alternatively, with a command line interface, the user can accomplish this with a single command (`mv * target_dir`), and with greater flexibility (*e.g.*, moving only `.png` files using `mv *.png target_dir`). Furthermore, experienced users can use shell scripting to create customized commands that build on the functionality of built-in commands in an iterative fashion, dramatically increasing functionality.

The podcast CLI was designed in a relatively simple way, with only a few commands used to manage the podcast library. It would be a simple task to expand this application to support more complex operations by wrapping the current commands into more complex ones. To illustrate, a simple way to expand the functionality of this interface would be to add a function to add all unplayed podcast episodes in the library to the queue. This could take the form of `pod library --queue-all-unplayed`, for example.

Another factor to consider is the platforms upon which these interfaces could be implemented. The CLI is a somewhat difficult case, as it is difficult to see how it could be implemented on, for example, a mobile device. While not impossible, the typing experience on mobile devices leaves a lot to be desired, and it would likely be better to use an interaction style other than CLI for mobile devices. The DMI is more flexible: with a few tweaks to accommodate the different form factor, the podcast DMI could be implemented on a mobile device with relative ease without compromising functionality. This is an advantage of direct manipulation: it can be implemented with ease on a variety of different technology platforms, the only requirement being a suitable display and input device such as a mouse or touchscreen, whereas command line applications are really only feasible with a physical keyboard such as on a laptop or desktop computer.

5.1 Conclusions

Based on this exploration of interface styles, I am left to conclude that the direct manipulation design is probably the better of the two approaches for a few reasons:

- The learning curve is much more forgiving for the DMI than for the CLI–no memorization is required as it relies on recognition memory rather than recall.
- There is greater visibility of the objects of interest in the DMI and the opportunity to integrate affordance into the design of those objects.
- The DMI gives users a greater ability to explore and develop a mental model of how the application works and what it can do. The CLI approach generally requires consulting with documentation or tutorials.

- Despite the power and flexibility of CLIs in general, the requirements of a podcast management app are fairly minimal and the tradeoff of poorer learnability is probably not worth it.
- The nature of the DMI makes it possible to implement across multiple platforms while the CLI is restricted (practically speaking) to those with a physical keyboard.

However, it must be noted that there are severe limitations of the DMI as discussed previously. In particular, the requirement for users to manually perform all actions may benefit their understanding of the system and ability to learn, but this benefit becomes a detriment when it comes to operations on many objects. Overall, if one were to design a podcast management application for consumer use (as opposed to a purely exploratory application such as this), the best approach would be to take the best features of the various available interaction styles and combine them in such a way that they compensate for each others' weaknesses.

References

[1] William G Chase and Herbert A Simon. "Perception in Chess". In: *Cognitive Psychology* 4 (1973), pp. 55–81. ISSN: 00219193. DOI: 10.1128/jb.173.17.5253-5255.1991. eprint: arXiv:1011.1669v3.

[2] Fernand Gobet et al. "Chunking mechanisms in human learning." In: 5.6 (2012), pp. 236–243. ISSN: 13646613. DOI: 10.1016/S1364-6613(00)01662-4. eprint: S1364-6613(00)01662-4 (10.1016).

[3] J Johnson and a.l. Et. "The Xerox Star: A Retrospective". In: *IEEE Computer* 22.9 (1989), pp. 11–29.

[4] David H Jonassen and Philip Henning. "Mental Models: Knowledge in the Head and Knowledge in the World". In: *Proceedings of the 1996 international conference on Learning sciences* (1996), pp. 433–438. ISSN: 21620989. DOI: 10.22608/APO.201780.

[5] Ben Shneiderman. “Direct manipulation for comprehensible, predictable and controllable user interfaces”. In: *Proceedings of the 2nd international conference on Intelligent user interfaces - IUI '97* (1997), pp. 33–39. ISSN: 00010782. DOI: 10.1145/238218.238281. URL: http://portal.acm.org/citation.cfm?doid=238218.238281.

[6] Endel Tulving and Michael J Watkins. “Continuity between Recall and Recognition”. In: *Americal Journal of Psychology* 86.4 (1973), pp. 739–748.