Personal Archival Companion (P.A.C)

Project Team: Cody Cothern, Quincy Hinson, Jacob Oleson, Andrew Menezes

GitHub Link: <https://github.com/Mask487/PAC>

[**1 Project Definition**](#_574hd61y08ce) **2**

[**2 Project Requirements**](#_3md73l8mjrsv) **2**

[2.1 Functional](#_x8x7bhwd0qqg) 2

[2.2 Usability](#_o4vzc82enram) 3

[2.3 System](#_eyspzlitc0kf) 3

[2.4 Security](#_uid7opn74ot0) 3

[**3 Project Specification**](#_b85l5juf6nuc) **3**

[3.1 Focus](#_39gd42tpefdj) 3

[3.2 Libraries](#_1jvha8df59s6) 3

[3.3 Platform](#_f0ty5xprn18e) 4

[3.4 Genre](#_85wccypd6xak) 4

[**4 System - Design Perspective**](#_54yxttra8n1q) **4**

[4.1 Database](#_umj94nyrds0a) 4

[4.2 GUI](#_yn6226j1ycc0) 4

[4.3 Inter-Device Communication](#_5xd8klrevv90) 5

[4.4 Data and Information Input](#_l4wjpcxtcahq) 6

[4.5 Sub-System Communication (Diagram and Description)](#_61zfp4xmv67b) 6

[**5 System - Analysis Perspective**](#_d8kwdh3e4ez4) **7**

[5.1 Database](#_qr9tivdjhgbe) 7

[5.2 GUI](#_u09q6vtknvk6) 7

[5.3 Inter-Device Communication](#_s7w88x5dqgdx) 8

[5.4 Data and Information Input](#_p7cpwmxttclj) 8

[5.5 System](#_ksttcr2z7744) 8

[**6 Project Scrum Report**](#_q0uzvvewuqzf) **9**

[6.1 Burndown Chart](#_vhueudqfubt4) 9

[6.2 Product Backlog](#_8bh188r35rgx) 10

[6.3 Sprint Backlog](#_gdtebdy09qeu) 11

[**7 Subsystems**](#_9zc1t5m7tt1) **12**

[7.1 Database/Data Access Layer - Jacob Oleson](#_on5oiaq8xsy6) 12

[7.2 Inter-Device Communication - Quincy Hinson](#_bela17hqw7b4) 16

[7.3 Data and Information Gathering and Input - Cody Cothern](#_8tyn0bo1df8z) 17

[7.4 GUI and Media Player - Andrew Menezes](#_28emw0l7800i) 19

# 1 Project Definition

People are always on the go; they go to and come from work, travel for business and pleasure. To entertain themselves during these typically boring periods of travel, people tend to read, either physical books or ebooks or, if their hands are busy, listen to audiobooks or podcasts. A problem arises when one has too many books and files to easily keep track of.

Now it is true that many applications exist to help make handling and organizing all of this entertainment easier, but most tend to focus on only one type. Most of these tend to be clunky, slow, not user-friendly, or lack ease of life features. No one wants to have to worry about having dozens of applications to handle a few tasks, especially if those applications are not easy and intuitive to use. Our goal with this project is to take all of these problems, bind them together, and solve them with one easy-to-use application. The method of solving this will be two-fold.

First, we will create a desktop application that acts as the repository and hub for all of the user’s ebook, audiobooks, and podcast files, as well as keeping track of the user’s physical books, both owned and desired. Secondly, we will create a mobile application that will sync with the desktop version and allow the user to access their aforementioned files wherever they happen to be. The mobile app will also include a way to scan the barcode of physical books to automatically look up the books for addition to the user’s personal archive.

# 2 Project Requirements

## 2.1 Functional

1. Catalog of user’s ebooks, audiobooks, and podcasts.
   1. User can manually enter a book’s information to catalog it.
   2. User can organize books by various categories (ex. genre, author, length, etc).
   3. User can scan a physical book’s barcode with their mobile phone to add that book to their collection.
   4. User can search the internet for a book to catalog it.
2. User can add/remove books from their wishlist.
3. User can use the app to manage podcasts and organize them based on genre.
4. User will be given different recommendations based on the books/podcasts/etc already in their library.
5. File transfer between desktop version and mobile version.
   1. User will be able to specify which ebook/podcasts will be added to their mobile phone and how many will be maintained at all times.
6. The application will use the newest information from either the desktop app or mobile app to ensure that the information is up to date across devices.

## 2.2 Usability

1. User Interface
   1. Simple interface that is easy to understand.
   2. Files containing users’ contents will be appropriately labeled and accessible.
   3. Basic settings will be customizable and accessible to the user.
2. Performance
   1. The application will be responsive and utilize a SQL Database to load and manage their personal libraries.
   2. In the event of errors or shutdowns the user will be notified with an adequate explanation of why the error occurred.

## 2.3 System

1. Hardware
   1. A PC running the Windows OS.
   2. A mobile android device.
2. Software
   1. The Java runtime environment for Windows.

## 2.4 Security

1. Prevent against SQL Injections to protect users’ accounts and their library of media.

# 3 Project Specification

## 3.1 Focus

Users with a large collection of physical books, ebooks, audiobooks, or podcasts.

## 3.2 Libraries

1. JSON
2. Java ROME
3. jMTPe
4. Apache Tika

## 3.3 Platform

-Windows PC

-Android mobile

## 3.4 Genre

-Archival

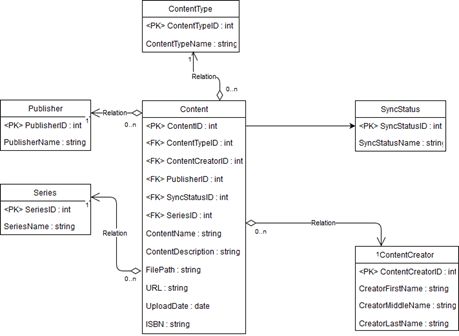
-Organization

-Entertainment (Reading and Audio)

# 4 System - Design Perspective

## 4.1 Database

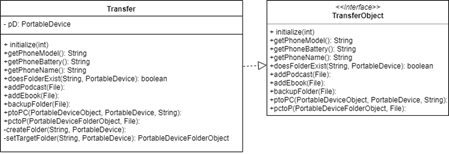
Table Diagram



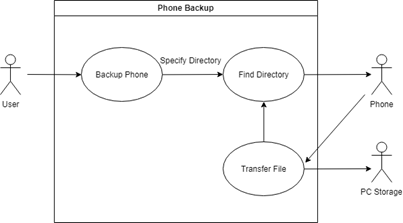
## 4.2 GUI

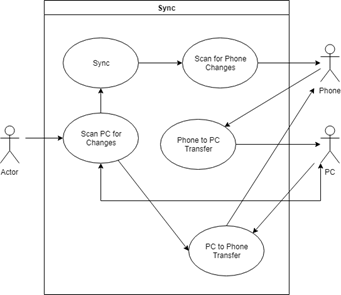
## 4.3 Inter-Device Communication

UML Diagram



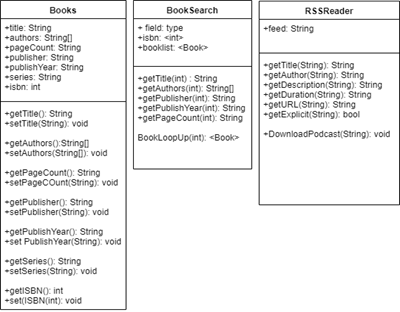
Use-Case Diagrams



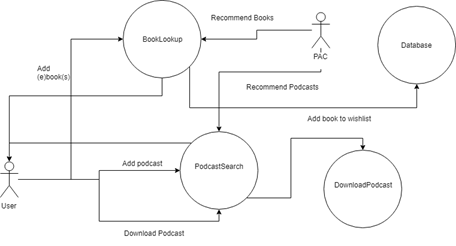


## 4.4 Data and Information Input

UML Diagram

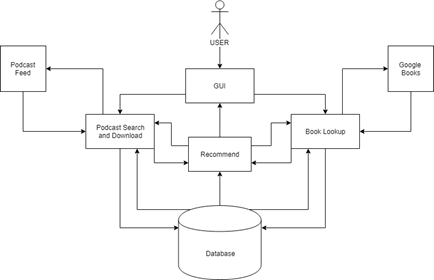


Use-Case Diagram



## 4.5 Sub-System Communication (Diagram and Description)

1. I/O
   1. Data will be input by the user which will be fed through to either the podcast or book search algorithms. After the desired information is gathered it will be given to the database for storage. From there it can be displayed as audio/visual output to the user in the form of an audio-player for podcasts and audiobooks or a text display for ebooks. The total collection can also be retrieved from the database and be displayed to the user for accessing or organizing.
2. Controls
   1. Application queries the database for user requested content (Specific books/audiobooks/podcasts to be added or viewed).
   2. Database searches for requested information and passes it to the application to be displayed in the GUI.
3. Dataflow



# 5 System - Analysis Perspective

## 5.1 Database

1. We will utilize a SQLExpress database embedded into the application to store important information about the user’s content.
2. The database is composed of the main content table which contains information relevant to each individual piece of media. It has a many to one relationship with the other tables which include information about a publisher, author, series, and sync status. The reason those tables exist is to reduce the possibility of repeating data.
3. DB structure may be subject to change and additional attributes may be added or removed from the tables as we progress through the project and optimize certain features.

## 5.2 GUI

## 5.3 Inter-Device Communication

1. This will ensure that a user’s library will be kept up to date between the desktop application and the user’s phone.
2. The user can also back up their phone through the application, so this section of the program focuses on the transfer of data between the desktop application and the phone.

## 5.4 Data and Information Input

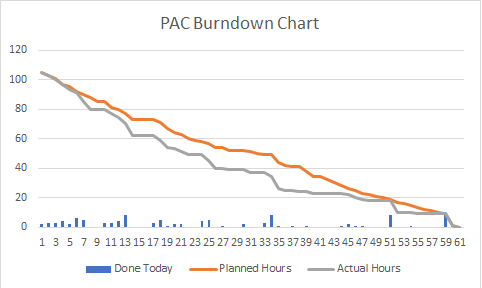
1. Books and eBooks will be searched via their ISBN numbers that can be input manually or eventually via barcode scan.
2. Podcasts will be found using their RSS feed URL and be downloaded and stored as required.
3. Recommendations will be found using the above search methods but using data gathered from the database (ex. Genre, author, etc.).

## 5.5 System

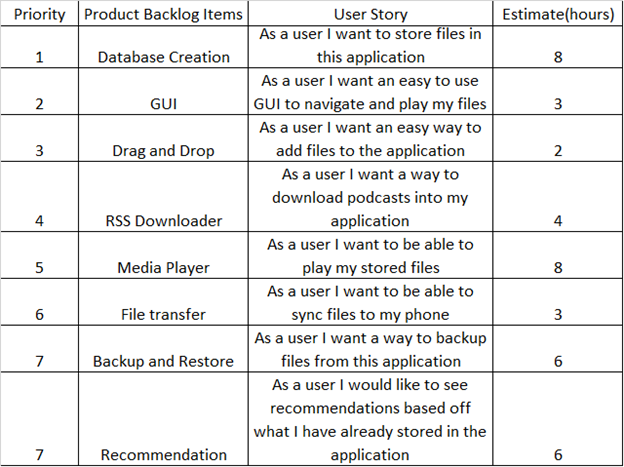
1. Data Dictionary
   1. Genre - String - What category of content the data is (fantasy, sci-fi, etc.).
   2. ContentCreator - String - Author of the book, or speaker in the podcast.
   3. ISBN - String - unique identifier on every book
   4. PageCount - Integer - How many pages a book contains
   5. Publisher - String - The company that published the work
   6. PublishYear - String - The year that the work was published
   7. Title - String - The title of the work
   8. SubTitle - String - The subtitle of the work, may not apply to all entries
   9. Description - String - A short description of the content of a podcast
   10. Duration - String(Potentially change to Time/Int - the length of a podcast or audiobook
   11. URL - String - link to access or download a podcast
   12. Explicit - Boolean - True/False value of whether or not a podcast contains explicit content
   13. Synced - Boolean - If content from the desktop is synced to the mobile device.
2. Algorithm Analysis
   1. BookLookup Algorithm
      1. Assuming adequate internet speed, this algorithm has a complexity of O(n) where n is the number of books the user is searching for.
      2. This algorithm simply reads in a list of ISBNs and uses the Google Books API to find the desired information and then creates a book object using this information
   2. PodcastSearch and Download Algorithms
      1. Again assuming adequate internet speed, this algorithm should have a complexity of O(n).
      2. Downloading depends on how many podcasts must be downloaded along with the size of each file and the internet speed available.
   3. Phone Backup algorithm
      1. To backup files on a phone, an array list of the file/folder locations that must be backed up must be passed to the backup method. If no array is being passed, then the entire phone will be backed up. In the best case scenario, only one file is specified, and it's on the root of the phone making O(n).To find files and folders with the library I'm using folders are looped through to find files. Unfortunately, most normal people will have several folders with folders in them. This means that for most cases the speed would be O(n2) since it loops to find specific files which themselves could be in a folder within a folder.
   4. SQL Algorithms
      1. Inserting new entries into the database is O(1).
      2. Selecting entries from the database takes approximately O(n) with n being the number of entries in a table to sort through.

# 6 Project Scrum Report

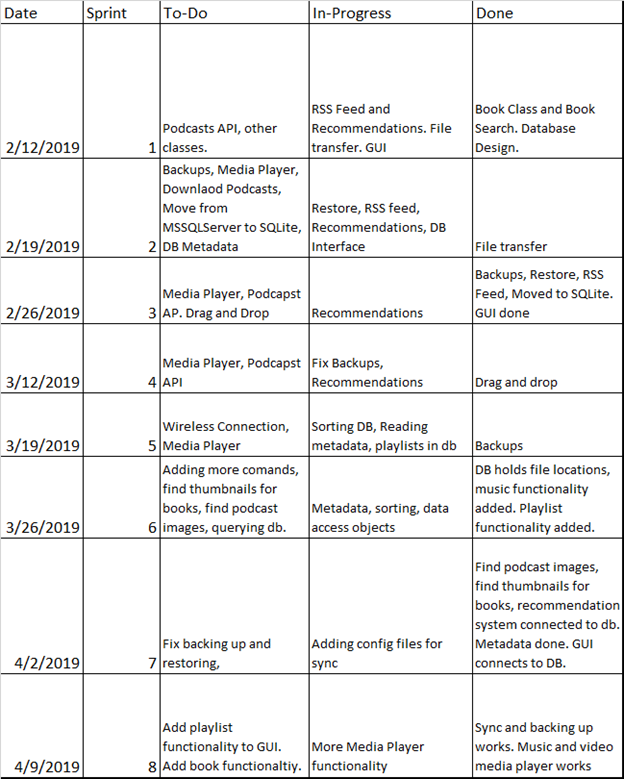
## 6.1 Burndown Chart



## 6.2 Product Backlog

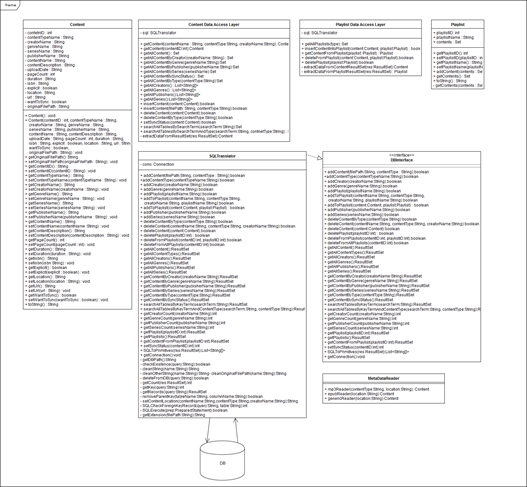
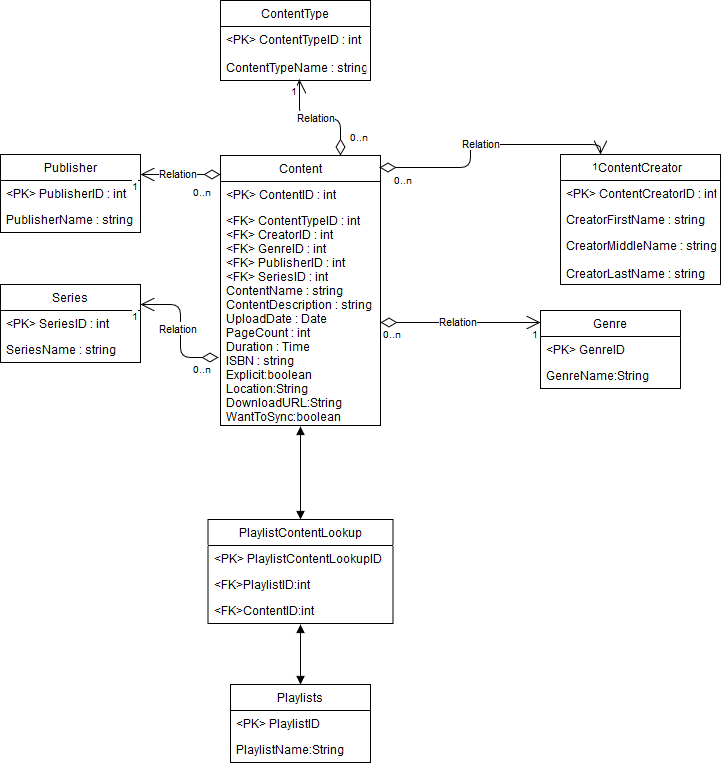


## 6.3 Sprint Backlog



# 7 Subsystems

## 7.1 Database/Data Access Layer - Jacob Oleson

1. Initial design and model
   1. See Section 4.1 initial diagram
   2. The purpose of this design was to generalize what a piece of content is and utilize foreign keys to build content records in the database. All content, whether music, podcasts or books can fit inside this model. By using foreign keys we can assign various content to their appropriate artist and genre while saving space by not having to repeat the same instance of a specific genre or author, just reference a piece of content to their key.
2. Data Dictionary
   1. SQLTranslator - The class that converts primitive requests into SQL statements for the database.
   2. DAO (Data Access Object) - The object that communicates with the SQLTranslator and converts the SQL result sets into an object the application can use.
3. Refinement
   1. In the next image, the tables of the database have been changed slightly. Instead of a sync status table, we now just have a boolean in a content’s record whether that piece of content is set to be synced to a device or not. Additionally, tables have been added to incorporate playlist functionality. Playlist names are stored in the Playlists table while the logic behind what content belongs to what playlist, if any, is handled in the lookup table. The lookup table has its own primary key and stores a content ID and playlist ID. That match indicates what content belongs to what playlist. 
   2. This gross mess of smeared text above is actually not that scary (and if you zoom in you can make out the text). This is the data access layer. The purpose of this design is to have two data access objects, the content, and playlist, that the application can communicate with. These data access objects call the SQLTranslator class with primitive arguments and receive SQL result sets in turn. They convert these results into content and playlist objects as needed. They then pass these objects, usually as a Set, to the application to use and display the content.
   3. The SQL statements are actually quite fast but converting them into objects takes O(n) where n is the number of records being returned.
4. Scrum Backlog
   1. Scrum 1 - Tuesday
      1. Design Database - done
      2. figure out relations - done
      3. fix database - by Thursday
      4. make it work - tentative
   2. Scrum 2 - Thursday
      1. Getting away from Microsoft by Thursday
      2. making tables - today
      3. Interface tentative
   3. Scrum 3 - Tuesday
      1. interface - next Thursday
      2. fix database - tentative
      3. metadata - next Thursday
      4. sorting - tentative
   4. Scrum 4 - Thursday
      1. interface - tentative
      2. fix database - done
      3. sorting - tentative
   5. Scrum 5 - Thursday
      1. Everything is redone
   6. Scrum 6 - Tuesday
      1. Fix database - done
      2. sorting - tentative
      3. metadata - tentative
   7. Scrum 7 - Thursday
      1. added file locations
      2. Updates
      3. added music
   8. Scrum 8 - Tuesday
      1. Metadata done
      2. adding playlists
      3. made it easier to use
   9. Scrum 9 - Thursday
      1. still working on metadata because its not actually done
   10. Scrum 10 - Tuesday
       1. Still working on metadata
   11. Scrum 11 - Thursday
       1. more metadata
5. Coding
   1. I followed an object oriented approach in building the data-access layer by creating a content and playlist object. The content object held all pertinent details about a piece of content, such as its type, author, name, and most importantly, its location on the device. The playlist object held details about a playlist’s name, as well as a set of all the content in a playlist. This approach made it easier to communicate with the other subsystems and made it easier to communicate with the database.
   2. Java is the sole language used to build the data access layers to the database. I utilized the slqite jdbc library to connect to the SQLite database. I built a translator class that would take parameters from the application and convert them into the SQL statements necessary to query the database. Java was used to build the content objects that the application would be able to use to display and play.
6. User Training
   1. Training / User manual (needed for final report)
   2. The user should know that the files they add into this application are taken from wherever they were stored on the computer and are placed inside the application directory. This is just so files that the application has to keep track of aren’t scattered all across the device the user is using.
   3. The user should be very wary of not deleting the database file as well. If they delete the file there is no backup on the application so they will have to restore the file manually. The database does not hold the actual files they the user had downloaded though, only the location, so they won’t have to worry about redownloading anything.
   4. The way the file system is set up to store the user’s files in the application is as follows :

../ContentFiles/ContentType/Genre/Series/File.mp3

As an example: ../ContentFiles/EBook/Fantasy/TheLordOfTheRings/The\_Fellowship\_of\_the\_Ring.epub

* 1. The application currently reads in metadata from a file using the tika api. This metadata reader, while effective, is not always successfully and if certain parameters cannot be found they are defaulted to the value: UNKNOWN. This value can be assigned to the publisher name, content name, series name, genre name and content description.

1. Testing
   1. My tests included making sure that the user can add and delete content from the database clearly and efficiently. I’ve tested with making sure that just by passing a file path to a piece of content that the user has downloaded, the application can read the metadata of that file, create a content object, and insert that object into the database, as well as moving that file to the main directory where the application maintains and accesses its files for the user.
   2. All my tests were conducted on the database subsystem alone until rather recently.

## 7.2 Inter-Device Communication - Quincy Hinson

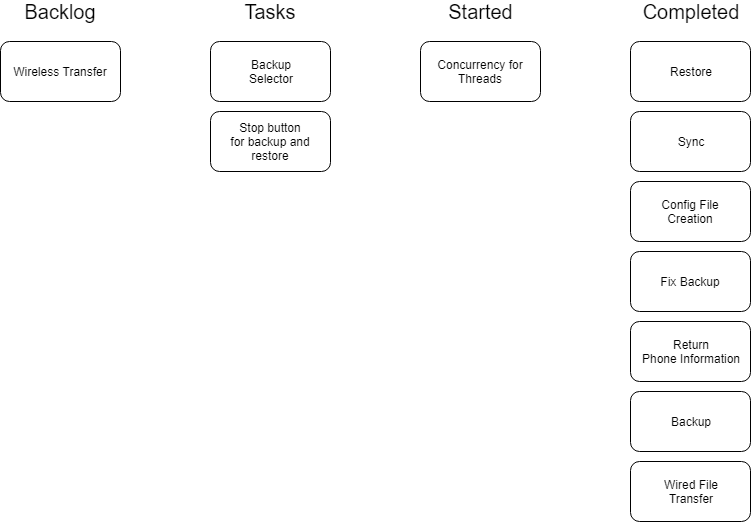
1. Initial Design
   1. Initially, we wanted to create an android app to go alongside the desktop app to make syncing to the phone much easier, but that was more difficult than we anticipated. Google has an API that could be used to help an android device communicate with a PC, but that would have required me to create my own driver which I don’t have the time to learn how to do. This made my subsystem easier, and difficult at the same time since now I had to take a different approach to transfer files.
2. Data Dictionary
   1. pD - (PortableDevice) An array of every phone connected to the PC
   2. pDM - (PortableDeviceManager) Allows you to select a device in the array and perform operations on them
   3. pDO - (PortableDeviceObject) A file, or folder located on the device
   4. pDFO - (PortableDeviceFolderObject) A folder located on the device
   5. FileA - File Accessor Class that stores the file location on the PC, the file’s name, and the File type (Music, Podcast, eBook, …)
3. Refinement

To navigate the phone I had to get an array of all the files on the root of the phones storage, and compare those files with the file I wan’t. I then enter that directory, and make an array of files and folders in that folder and repeat the process. I’ve also changed how backups of the phone is stored on the PC. They are sorted by date, so the user can tell how recent a backup is. I also created a new class called FileA that makes it much easier for my team members to use my sync method. They only have to pass an array of files, and the sync method will sort them by type to add them to the correct folders on the device.

1. Coding Approach

My approach to my subsystem would have to be mostly functional. I pull information from the database to move around files, but I rarely change the data at all.

1. Scrum Backlog



1. Training

My subsystem is extremely simple and doesn’t require much training at all. All the buttons are self explanatory and should not require any training at all.

1. Testing
   1. The first couple of tests were mainly focused around phone connectivity. I started the program making sure that my phone was not plugged in, and then tried to backup, restore, and sync the phone. These, of course, shouldn’t work, and my program handles the exceptions nicely. I then plugged in my phone without closing the program and ran restore, sync, and backup making sure that it would work this time.
   2. My other tests centered around making sure that the same file could not be added to the device, or PC if it already exists. I would add files that I know exist already on the phone, and then I used the program to try to add it again.
   3. I have already mentioned that the backup/restore methods recursively navigate the phone’s storage, so I went in step by step to make sure that those methods don’t get caught in an infinite loop. It’s kind of hard to make absolutely sure that the method doesn’t get caught in an infinite loop, but those methods have a lot of checks to ensure that the method is able to escape.

## 7.3 Data and Information Gathering and Input - Cody Cothern

1. Initial design
   1. See section 4.4 for initial UML
2. Refinement

The original plan was to make objects for each media type, rather than just books and ebooks. However, we realised that it would be easier to retrieve and access the information with metadata extraction via the database. We had also planned on being able to search local libraries for book, but due to legal difficulties of accessing these resources, this idea was scrapped. Another change is the lack of recommendations for media types other than books. The original plan was to be able to recommend various media based on the contents of the database, however due to the fact that there is not a free, open source, database for most types of

1. Scrum Backlog
   1. I have to integrate the book look-up with the G.U.I.
   2. We have to integrate the book recommendation into the G.U.I. as well.
2. Coding
   1. Approach

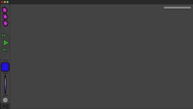
For my subsystem I used an object-oriented approach. This is most easily seen in the book recommendation and the book look up. Both of these retrieve information from the Google Books API and then compile this information as a Book object which is then stored in a list and then returned.

* 1. The language used for this subsystem is Java

1. User Manual
   1. To download podcasts, one must merely enter the URL of the RSS feed into the import box.
   2. To look up books, one must enter the ISBN number for the desired book.
   3. Recommendations should be generated automatically based on the user’s typical preferences.
2. Testing
   1. My initial testing was done with my subsystem separated from the others. For this, I had a testing class that would call the various functions that the other systems would be using and then output the returned information. For the book search, I used various ISBNs to ensure that the data could be gathered correctly. Similarly, I used various RSS feeds to test the podcast download.
   2. Secondary testing was done after I could connect to the database. This was mainly testing that the recommendation would work correctly.
   3. Tertiary testing was done after all of the subsystems were integrated. This mainly consisted of making sure that the podcast download would not interfere with the functionality of the project.

## 7.4 GUI and Media Player - Andrew Menezes

1. Initial Design
   1. My initial design for the Gui and the Media Player is shown below, I wanted to make a dark themed interface with a side panel that controlled everything. The Controls were supposed to animate in and out in a circular motion in order to switch between media control types and there were only three types at the time.



1. Data Dictionary
   1. Media (Media File Type)

Provides source file for media player to play

* 1. Media Player (Media Player Type)

provides controls for media file

* 1. Media Viewer (Media Viewer Type)

Provides visuals and organization for media player

* 1. Media List (List Type)

Provides structure for keeping media files

1. Refinement
   1. After many different file types were added to the program I had to change the gui. Since everything could not fit inside the side panel anymore I had to make a new bottom panel that could store the controls. As other functionality was added they got moved to the bottom pane as well, that keeps things mostly organized.
   2. The media playing was much more complex than I originally anticipated so a lot of internal reorganization was done in order to make it work properly.
2. Coding Approach
   1. Originally I was using Gluon scene builder to help make the GUI but after a while I wrote the whole thing over again in JavaFx.
3. Scrum Backlog
   1. Base gui ✓
   2. media controls ✓
   3. resizability ✓
   4. lists of database content ✓
   5. search pane ✓
   6. drag and drop ✓
   7. ordering select ✗
   8. rss feed box ✓
   9. content selection for sync ✓
   10. books/ recommendation box ✗
   11. text reader ✗
   12. delete song selection ✗
4. Training
   1. The gui and the media player required no training
5. Testing
   1. Many of the parts of the GUI and media player had to be tested
   2. the media controls had to be tested often to make sure players weren’t getting destroyed, controls actions were happening properly and the logic was correct
   3. The GUI was tested to make sure the right colors were showing
   4. the GUI was also tested repeatedly for proper drag and drop functionality.