WUD Film Mobile

Arup Arcalgud, Erin Goldberg, Stephanie Goldberg, Charles Johnson, Davud Porter, Francis Thompson, Susan Yang

Design and Planning Document

<2016-11-02>, version 1.1

Minimal updates were documented in blue and we crossed out information that was no longer relevant.

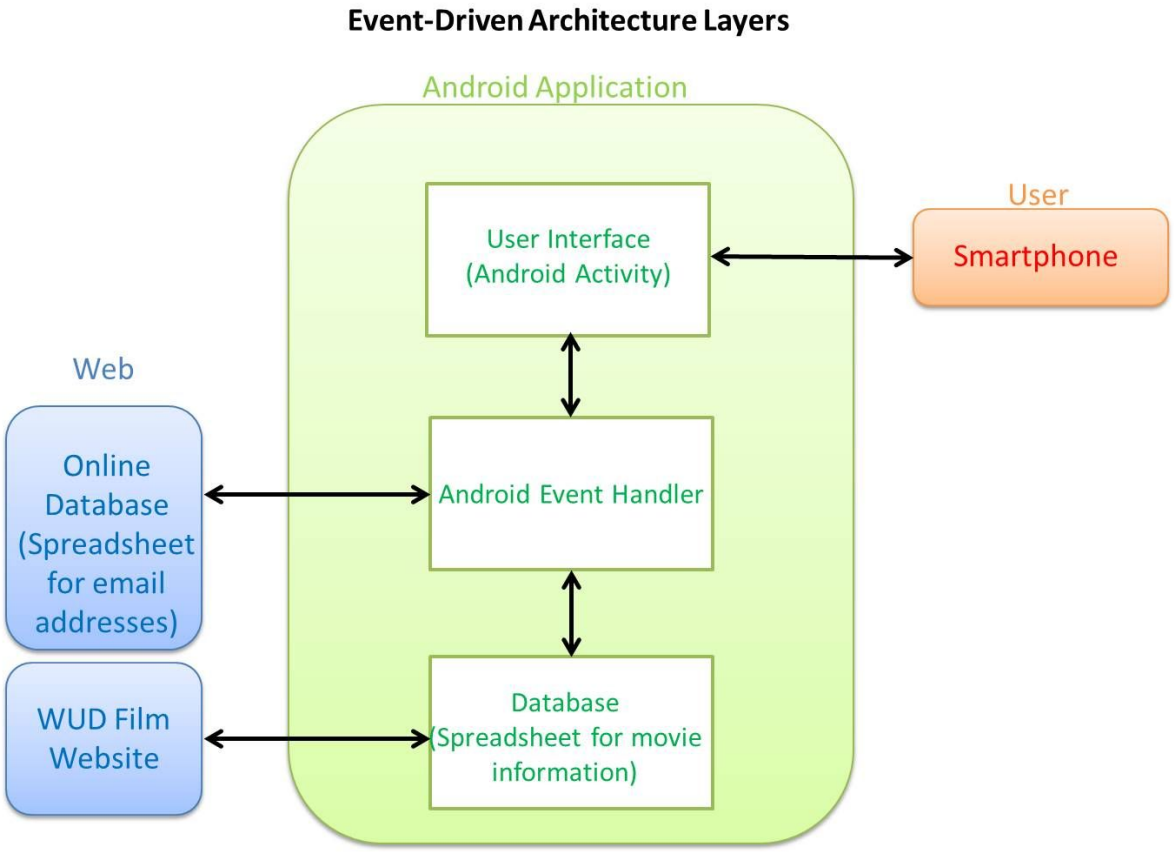
Document Revision History

Rev. 1.0 <2016-10-09>: initial version
Rev 1.1 <2016-11-1>: post iteration 1 alterations

Table of Contents

1. System Architecture

The WUD Film Mobile application will adhere to an event-driven architectural style. Upon being opened, the application will load all of its states, and once the user decides to click on one of the components in the app, events will be triggered. The overall architecture will consist of three layers: the web layer, the Android application layer, and the user layer. In the web layer, we will gather information from the existing WUD Film website. In the Android application layer, users can trigger events by interacting with the main application user interface. The following diagram shows the layout of our event-driven architecture:



Web Layer

The WUD Film website will act as the first component of the web layer. This website will be “scraped” for information such as movie title, poster, runtime, description, genre, MPAA rating, and trailer. This information will be stored locally and will be accessed within the Android application for display on the main application page. The second component of the web layer will be an online spreadsheet that will hold user names and email addresses when they submit their name and email for subscription to WUD Film. This spreadsheet will be open for the WUD Film Committee to access and gather the email addresses of subscribers.

Android Application Layer

The Android application layer will consist of three components. The first component will be the “database”, more specifically a spreadsheet containing movie data which we will use as a local database for the other components of the application to reference. The second component will be the Android event handler, a built-in feature of Android Studio that allows events (state transitions) to be created and queried to interact with different activity screens within the application. The third and final component of the Android application layer will be the user interface layer, which will consist of Android activity screens, which is the only component that the user will be able to see and interact with.

User Layer

The user layer will consist of the human user, but also takes into consideration the hardware which will be interacting with the Android application, this will be a smartphone capable of running Android OS. Users will also need an internet connection in order to run the app.

Possible Risks

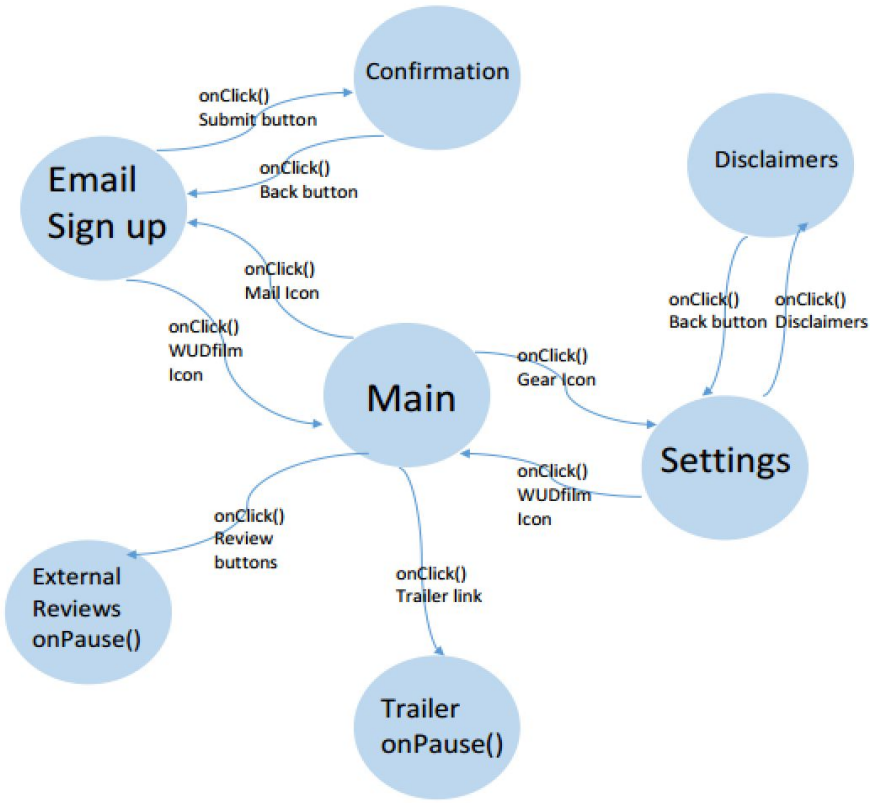
One possible risk of this design would be that it doesn’t incorporate a database server to store movie data and email data, but instead uses a local and online spreadsheet. This makes accessing data inefficient and may slow down the querying of data to our application. Another risk is that site scraping may slow down the response time of the application, since it will take time to extract data from the WUD Film site, store it, then

access the stored data. We are limited in what we can do and how efficiently we can do it by our relationship to WUD Film and our lack of access and permission to their resources. We do not have access to any of their servers or to their website from the backend, because of this we have had to find workarounds to achieve functionality that can be implemented with the resources we have available. A new potential pitfall of our application and it's dependency on WUD Film's website page is that the website may be undergoing changes and reformatting. Some of UW's websites have already been undergoing these changes and if the changes too heavily alter existing HTML tags, then our web scrape will become effectively useless, due to dependencies on the existing HTML of the website.

Alternative Designs

When originally constructing our design we considered implementing a user system which would allow for extra features in the app such as individual ratings, a message board, etc. Adding users to our app would require a server for a database containing this additional information (users, passwords, message board, ratings, etc.) rather than keeping all of the content locally. A server would require some form of payment on our end that we did not want to invest in as we will not end up receiving any profit from this app. Though we did ask the WUD department if we could use some form of server in cooperation with them, they denied this access. However, once we hand over our app to them they may choose to implement such an idea in the future. This alternate version of our app would then have likely followed the Model View Controller architecture as it is the most up-to-date/streamlined architecture that involves a heavy backend. Despite the many advantages offered by implementing our application in this way, security concerns and legality issues have deterred and prevented us from pursuing this approach. After brainstorming, we decided on an event-based architecture that would suit our app best.

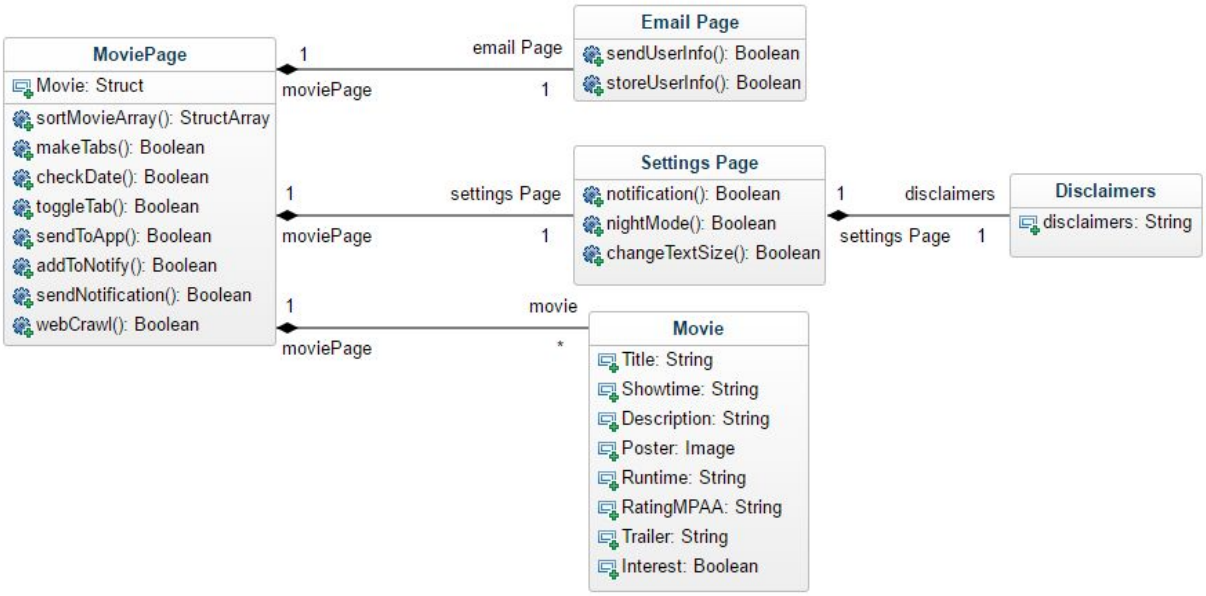
State Diagram of the Android Application Layer



Using event driven architecture means that any time a user clicks on any of the icons, links, or buttons, the current state of the system would move to the next state as shown in the above diagram. When a user is in the external links or trailer state, the main activity will be on pause as the user is redirected to an external webpage. The activity will be resumed when the user reopens the application or returns via the back button on their Android device.

2. Design Details

Our app consists of three main features: the main Movie Showtimes page, the Settings page, and the Email List Sign-Up page. A navigation bar will be implemented to allow users to navigate between these three pages. Each feature is broken down into the necessary methods in the Class Descriptions section of this document. Below is a class diagram for the WUD Film Mobile application.



Class Descriptions

1. Movie: Movie object to hold all the fields to be displayed.
    a. Fields: title, showtime, synopsis, poster, runtime, linkM, linkR, linkI, linkYT, MPAArating, interested (boolean variable that allows users to express interest in order to receive notifications)
    b. Methods
        i. Getters: getTitle(), getRuntime(), getSynopsis(), getShowtime(), getMPAArating(), getLinkM(), getLinkR(), getLinkI(), getLinkYT(), getInterested()
        ii. Setters: setTitle(String title), setRuntime(int runtime), setSynopsis(String synopsis), setShowtime(int showtime), setMPAArating(String MPAArating), setLinkM(String linkM), setLinkR(String linkR), setLinkI(String linkI), setLinkYT(String linkYT), SetInterested(boolean interested).
2. MovieListFragment~~Page~~ : Builds tabs from each Movie ~~Struct~~ Object which at first only displays basic information (Title, Showtime, and Date) but, when expanded, reveals further details (rest of movie ~~struct~~ object fields).
    ~~a. Struct Object Movie~~
        ~~i. Fields: Title, Showtime, Description, Poster, Runtime, RatingMPAA, Trailer, interest (boolean variable that allows users to express interest in order to receive notifications)~~
    b. Methods

i. **webcrawl()**: Utilizes a built in Android feature to create a url object (this object contains the entire field of HTML), we then parse this object by requesting certain fields (Title, Date, etc.) and placing them in an array for each field of the movie ~~Struct~~ Object (with indices corresponding to each film). This method fills in each Movie ~~Struct~~ Object with the data from each array and builds an array of Movie ~~Structs~~ Objects. It then calls sortMovieArray().

ii. **sortMovieArray()**: Reorders the array of Movie ~~Structs~~ Objects by earliest showtime field first. The end result will be the array of movies ordered with showtimes closest to the current time and date first.

iii. **makeTabs()**: For each movie in the sorted movie array make a tab and fill in the necessary data that the movie tab should display. It calls checkDate() first to verify if the movie should be given a tab.
    1. Iteration 1: Title, Showtime, ~~MPAA Rating, Runtime, Description.~~
    2. Iteration 2: MPAA Rating, Runtime, Description, Links to RottenTomatoes/IMDb/Metacritic.
    3. Iteration 3: Checkbox that allows users to express interest in a movie to receive notifications about it on the day of its showing (if the notification setting is turned on). Also, links to Youtube trailer for a movie.

iv. **checkDate()**: Check a movie's showtime within 2 hours of the current date and time. If 2 hours have passed since the movie has aired, remove the movie from the list so a tab is not made for it. This method takes care of clean up, only showing movies within a reasonable time frame and eliminating the possibility of displaying movies whose times have passed.

v. **toggleTab()**: Open and close the tab that contains the detailed information for each movie.

vi. **addToNotify()**: When the checkbox is clicked, the movie is added to an array of movies that the user has expressed interest in.
    1. Implementation in iteration 3

vii. **sendNotification()**: If a user has indicated that he or she is interested in a movie, the user will receive a reminder about that movie's showtime on the day of the showing.
    1. Implementation in iteration 3

viii. **sendToApp()**: This method will encompass all links which push the user out of our app ie. Youtube, RottenTomatoes, etc. The method

will first check if the user has the necessary app for the individual site. If they do not, then the method will send the user to their preferred internet app and the website corresponding to the given entry.

Dependency: Our app will use the current URL for the WUD film web page as a reference for getting the data that we need to display (title, poster, trailer, ratings, etc). For this reason, if the location of the web page were to change in the future, we will not be able to locate the information.

3. SettingsFragment~~Page~~ : Where users can customize the settings of the app and view legal disclaimers.
    a. Methods
        i. **notification()**: User is able to set whether or not they would like to receive notifications from the app.
            1. Implementation in iteration 3
        ii. **nightMode()**: Changes the colors of the background and text of the app.
            1. Implementation in iteration 2
        iii. **changeTextSize()**: Switch between having large and small text.
            1. Implementation in iteration 2
    b. Disclaimers Page: Displays the legal disclaimers of the app (regarding RottenTomatoes, IMDb, Metacritic, Youtube). This is done by displaying a string containing the information.

4. EmailFragment~~Page~~: Users can provide their name and email to be passed along to WUD Film to be added to their emailing list.
    a. Methods
        i. **storeUserInfo():** Takes in the user-entered name and email address and stores it into a Google spreadsheet using the Google Sheets API. Sends message to the user that their submission has been received. Text boxes will be then be cleared. Also opens popup which displays to the user that their submission was received correctly.
        ii. **sendUserInfo()**: Sends the link to the Google Spreadsheet of the user-entered name and email addresses to WUD Film to be added to their emailing list.

Methods: Will return True on success and False on failure for every class unless otherwise specified.

Notifications

Notifications will be configured using the android builder class NotificationCompat.Builder. The UI information and actions will be specified in this object. As mentioned in the ~~Struct~~ Object Movie above, notifications will be sent to users who have indicated interest in a movie. The user will be notified when the day of a movie's showtime arrives or in the unlikely event that a showing is cancelled.

Libraries

~~We will be using Python for our web crawl script. In order to smoothly parse through the HTML or XML web pages, we will use a simple HTTP library for Python.~~ We will be using the JSoup library for the webscrape. We will also be using the Java Client library required for Google sheets API. This will enable us to implement the email signup functionality in our application.The emails will be collected in a Google spreadsheet and sent to the WUD film committee routinely.

Details

Main Page

Content	Component	Description
Email List Sign Up	Button	On selection brings users to the Email List Sign Up page
Movie Showtimes Page	Button	On selection brings users to the Movie Showtimes Page(the main page)
Settings	Button	On selection brings users to the Settings page
Movie Tab	Toggle Button	On selection expands to show information about the movie

Expanded Movie Tab

Content	Component	Description
Poster	image	Theatrical poster for the selected movie
MPAA Rating	Text	MPAA rating given to the selected movie
IMDb/RottenTomatoes/ Metacritic scores/links	Button/Link	On selection brings users to rating site or app chosen
Plot Synopsis	Text	Plot synopsis displayed for the chosen movie
Runtime	Text	Runtime of the movie displayed
YouTube Trailer Link	Button/Link	On selection brings users to the movie's YouTube trailer

YouTube Trailer/RottenTomatoes/IMDb/Metacritic Link (Iteration 2, YouTube trailers in Iteration 3)

Users taken to the corresponding app if they have it downloaded, otherwise users are sent through their preferred browser to the corresponding page. On return from browser the main page (Movie Showtimes) is displayed with film details still expanded.

Email List Sign Up Page

Content	Component	Description
Name	Text Field	The area for users to input their first and last name
Email	Text Field	The area for users to enter their email address
Submit	Button	On selection passes user's name and email to WUD Film through a

		spreadsheet for email subscription
Received	Popup	Shows the user that their submission has been received
Close	Button	Closes received popup

Settings Page

Content	Component	Description
Back	Button	On selection takes users back to the Main Page without saving settings
Save	Button	On selection saves the setting changes and takes users back to the main page
App Notifications	Toggle Button	On selection toggles between on and off
Text Size	Toggle Button	On selection toggles between large text and small text
Night Mode (Color Scheme Customization)	Toggle Button	On selection toggles between on and off
Show Disclaimers	Button	On selection opens Disclaimers page

Disclaimers Page

Content	Component	Description
Disclaimers	Text	Disclaimers for the application
Close	Button	Closes Disclaimers page and returns user to

		Settings page

Design Risks

One major risk of the design is that it relies very heavily on the WUD Film Website format and availability (which is beyond our control). Because we are webscraping, if any of the formatting of the HTML is changed, such as the IDs or class names of elements, it will affect our webscrape and our code would not function as intended. Another major risk is the way in which we are storing the user names and emails that are entered when users sign up to be added to WUD's email list. Strings will be passed to a Google spreadsheet owned by WUD Film, which would then allow them to add the names to their subscription list (we do not have direct access or security permissions to add the names ourselves, we have to pass them onto WUD). If the URL of the document changes, or if the document gets deleted, potential problems may arise and result in the names and emails being lost. Additionally, the document would have to be able to be written to by anyone, since the phones themselves are the ones sending the data, not a middleman (i.e. server). This could lead to a security concern if someone reverse engineers our code to get the URL for said document to also get access to the list of names and emails.

3. Implementation Plan

The following tables are representations of our Use Cases broken down into programming tasks. Difficulty will be ranked in tiers: low(1, 2, 3), medium(4, 5, 6), or high(7, 8, 9). Time unit will be based on the completion of a low-difficulty-ranked task.

Task	Create Main Screen
Description	Create the Main Page for the user to be brought to upon opening the app. This is the Movie Showtimes Page, upon first opening the app it should create tabs for each movie found on WUD Film's website. This is dependent on the web crawl and the information it gets.
Difficulty	High: 9
Time Units	5
Assigned To	Charles, Erin, Stephanie, Arup, Susan, Frank
Iteration	1

Task	Create Webcrawler Functionality within Main Screen
Description	As this is likely the most complicated method, it deserves its own task: Create function to pull details from WUD film website and/or spreadsheet and hold them within an array. This function is dependent upon the sortArray() function - after the arrays are sorted, place the data within each Movie ~~Struct~~ Object correctly. The ~~structs~~ objects themselves will exist inside either a linked list or an array depending upon which structure is most efficient (if growth is needed or if it is static).
Difficulty	High: 9
Time Units	4
Assigned To	Charles, Susan
Iteration	1

Task	Create Email List Sign Up Page
Description	Create the Email List Sign Up Page for users to submit their names and emails in order to sign up for WUD's emailing list.
Difficulty	Medium: 5
Time Units	3
Assigned To	Davud, Arup
Iteration	1

Task	Create Settings Page
Description	Create the Settings Page for users to customize the settings of the app and get to the Disclaimers Page.

Difficulty	Low: 3
Time Units	2
Assigned To	Davud, Arup
Iteration	1

Task	Disclaimers Page
Description	Display the Disclaimers Page in the settings page. The page displays all of the necessary copyright information as well as content disclaimers.
Difficulty	Low: 1
Time Units	1
Assigned To	Frank, Stephanie, Erin
Iteration	1

Task	Toggle Film Details
Description	When a movie's tab is selected, it expands to reveal information about the movie This is dependent on the web crawl's information.
Difficulty	Low: 2
Time Units	2
Assigned To	Davud, Arup
Iteration	2

Task	Display Link to RottenTomatoes

Description	Display the link to RottenTomatoes from the app when a movie tab is expanded.
Difficulty	Low: 2
Time Units	2
Assigned To	Stephanie, Erin
Iteration	2

Task	Display Link to IMDb
Description	Display the link to IMDb from the app when a movie tab is expanded.
Difficulty	Low: 2
Time Units	2
Assigned To	Charles, Frank
Iteration	2

Task	Display Link to Metacritic
Description	Display the link to Metacritic from the app when a movie tab is expanded.
Difficulty	Low: 2
Time Units	2
Assigned To	Susan, Frank
Iteration	2

Task	Display Link to YouTube
Description	Display the link to YouTube from the app when a movie tab is expanded.

Difficulty	Low: 2
Time Units	2
Assigned To	Davud, Charles
Iteration	3

Task	Create Notifications
Description	Create notifications through Android notification builder object. These notifications should remind a user of upcoming movies that the user has indicated interest in.
Difficulty	Medium: 5
Time Units	3
Assigned To	Susan
Iteration	3

Iteration 3 potential features (time permitting)

Our approach to the iterations, and dividing the tasks among the iterations, prioritizes functionality of the app first and foremost. Iteration 1 establishes the core mechanics of critical components (like webscraping and formatting the different pages of the app). Iteration 2 builds off of that by adding more details and features from what was established in iteration 1. Aesthetic additions available in the Settings page, such as night mode and text size options, will simply be added by adjusting attributes in the code. Iteration 3 will cover all of the features that are not essential to the core functionality of the app but would make the user experience more personal and engaging. The potential features were designed to integrate easily into the established logic and the code framework previously set up.

Potential features for iteration 3 include a customizable profile page for the user, the ability to mark movies as favorites, and the ability to sort and filter movies on the Movie Showtimes page by certain options, such as genre or MPAA rating. In order for users to access their profiles, there will be an added button to the already existing navigation bar. To allow alternate sorting options, different methods similar to the sortMovieArray() would be implemented with the appropriate sorting logic and be based on different fields of the Movie ~~Struct~~ Object. Adding a user favorites list will be implemented in the same

way that users are able to express interest in movie, except this list will be explicitly displayed on the user's profile.

4. Testing Plan

Unit Testing

The majority of the modules within our design will be independent, thus most of our testing throughout the development process will be unit testing. We'll do unit testing after developing each unit and then automate each for larger scale testing later on as necessary. Questions for the client, such as what should happen when a user enters an email with invalid special characters, should be asked during this phase.

Integration Testing

Most of the units will have little to no overlap, and thus integration testing will be fairly minimal. The most important feature to integration test will be the settings module, which will be very simple to test empirically. We will conduct minor integration testing as we finalize modules, to ensure that the things are coming together smoothly, but accidentally crossing modules that should be independent is an unlikely scenario.

System Testing

To test our system, we will test all expected and unexpected behaviors. Our goal will be to try and break the system. We will draft a list of test cases for the complete system. Then, we'll run it through the entire development team and the client to get it verified and approved. After all behaviors are clarified, execution of all the test cases can be performed by the team. Should any tests fail, we will reiterate as efficiently as possible and determine the source of the fault in our software, or if the fault is with the tests themselves. Then we'll either correct the software, or the tests as needed.
We will test our product on different devices to see if it works properly across subtly different hardware configurations. We will also conduct usability testing along the lines of whether the header icons are distinguishable and intuitive, as well as whether or not the interfaceable tabs are an accessible size for both usability and maximum information display. This will be done at the end of each iteration.

Performance Testing

The majority of our performance testing will be empirical during other testing processes. The main possible performance bottleneck will be the process of pulling movie data from the data source, which we will monitor closely. Otherwise we will try to implement

efficient code and use feedback to determine if performance is lacking. We will be unable to correct for performance deficiencies on individual devices that result from unique hardware configurations. This will be done at the end of each iteration.

Compatibility Testing

We will be testing our software on the Android devices available to us. This should give us enough of a reference frame for the greater Android ecosystem to produce an adaptable quality product. We will not be developing for iOS or other non-Android devices. This will be done at the end of each iteration.

Regression Testing

As we develop code incrementally, we can automate tests along the way as well. That way, at each iteration, we can run these test scripts with each addition of code to see that everything still works the same as it did before. Automating these test scripts will also allow us to focus on the bugs instead of spending too much time manually testing. We will be using Android Studio to build a test suite and automate it using Selendroid, which is similar to Selenium but for Android.