FLIP Software for Esophageal Motility

UX Report

[redacted]

By: [redacted]

[redacted]

Client: [redacted]

Date of Submission: April 11, 2021

Table of Contents:

List of Tables:

List of Figures:

Introduction:

Esophageal motility disorders cause a patient to either have difficulty swallowing or to lose the ability to swallow at all. A lot of gastroenterologists use FLIP software to analyze and detect these Esophageal motility disorders in patients. One of the main components of this software is the FLIP device. The FLIP device is essentially a balloon that is inserted into the esophagus and inflated in order to obtain measurements that aid in the diagnosis of esophageal motility disorders. A software that is intended to process the information currently exists, but the program interface was created through crude and confusing MATLAB and Python code. As such, only specialists in esophageal motility with prior knowledge of how to use complex computer programs find the current implementation accessible. The program also does not offer a cohesive user experience. For a user to reach a diagnosis or even the information that is relevant to making a diagnosis, the user must click through numerous distinct pop-up screens requesting non-relevant user inputs. The main issue with the current version of this software is that it is so complicated and non-accessible that a gastroenterologist is not able to appropriately use the data collected by the FLIP device in order to make a same-day diagnosis.

This FLIP software can be very useful for general gastroenterologists if it can predict the diagnosis of esophageal diseases in patients based on data collected from FLIP device. Our application is an extension of the current FLIP software which will resolve all the issues that the current version of software is not able to solve. We are working to make a version of FLIP software which will be more accessible to people and more user friendly. We will also be working to extend the reach of FLIP software to allow the analysis and diagnosis of esophageal motility disorders in real time instead of just showing the bounds of the mean distensibility index and distance.

This UX report articulates our initial prototype, and how we evaluated it, using heuristic evaluations done by us and another team from the class. It also reflects how these evaluations lead us to important findings on our prototype and influenced changes that we made in our prototype and will continue to make on the product in future iterations.

Prototype:

Since our application is mostly functional, the user interface required is minimal. Consequently, our paper prototype has only 3 main screens (not including the loading screens). The first screen is the only one that accepts user input, specifically the file to be analyzed and information about the patient. The second and third screen show various modes of information the user can access on the results page. The order in which the prototypes are discussed below is synonymous with the order the users will encounter them in the application. Extraneous screens are displayed in Appendix 3: Loading Screen Prototype.

Entry Page (Screen 1)

The entry page is the first screen users encounter in the application. Since the initial version of the application is planned to be stateless, users will have to input patient information every time to run a new analysis. Our design language is consistent throughout the page, displaying input boxes with a light pink color and greyed out text while the clickable buttons have a slight shadow and raise to them.

We also provide feedback to aid the users in completing the form. If a field is empty, the button will be disabled, preventing users from proceeding. If a field is invalid, a popup will

appear in red on the top alerting the user to the problem. When all inputs are finally valid and the user presses the “RUN FLIP” button, they are moved onto the loading screen.

Figure 1: Screen 1 of FLIP Software or Entry Page

Results Page (Screen 2)

The default results page provides easy access to view output from the analysis software. From user research, we concluded distensibility index is the most important information for gastroenterologists and displaying the video of the analysis helps them make their own assessments. Furthermore, since the application is entirely stateless, the “DOWNLOAD ZIP” file allows them to save analysis data for the future.

Similar to the design language on the first page, all the buttons have the slight raise and shadow to identify them as clickable. The “RUN ANOTHER” button is appropriately placed on the left as is the natural place for a back button. And all readable modals on the page are

encased in blue boxes to highlight them on the page. Overall, the design feels clean and direct for the users of the application.

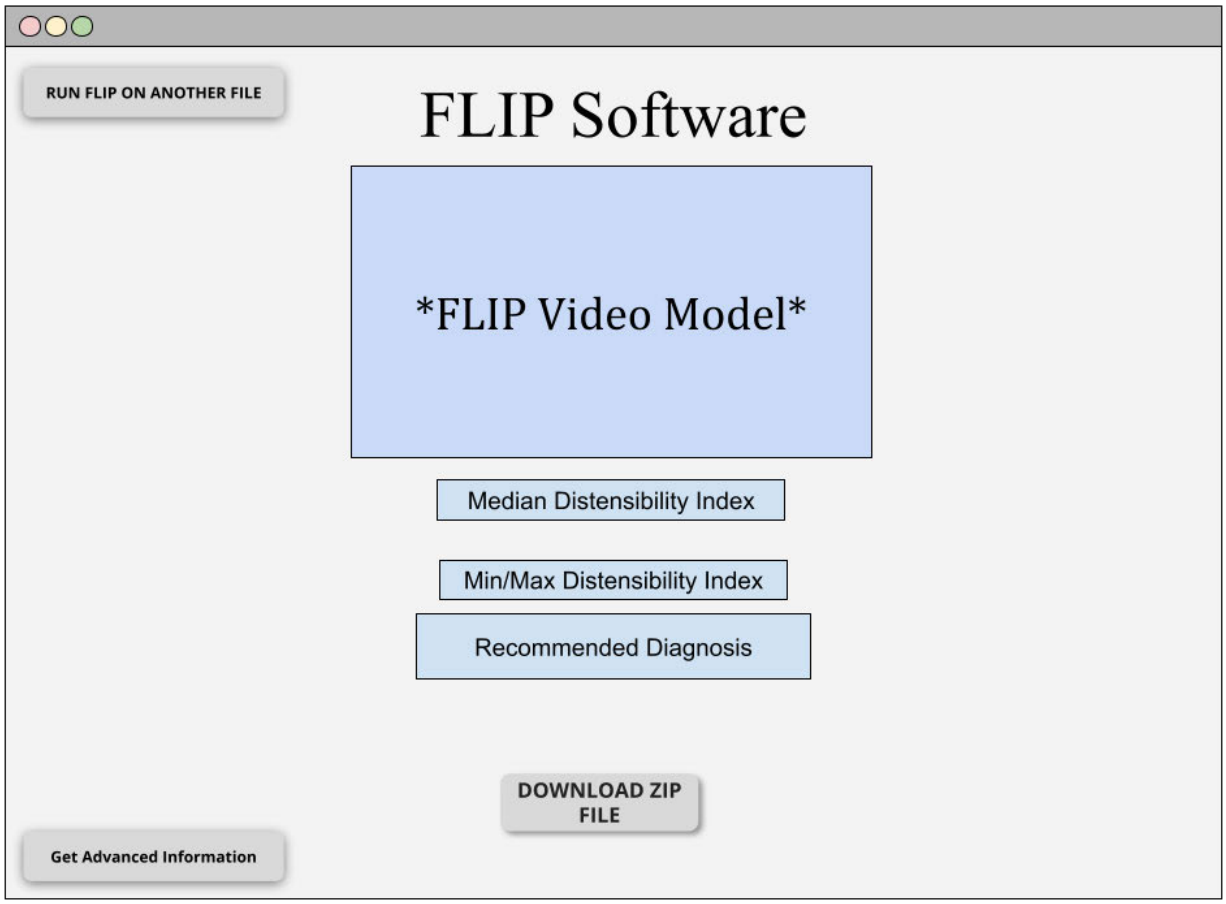

Figure 2: Screen 2 of FLIP Software or Result Page

Third Page (Screen 3)

The third page is the last screen users will encounter in the application. This is where users will get to see the recommended diagnosis based on the file, they uploaded in the first screen. They will also get more details including the probability of developing the disease(s) and further information about the diseases mentioned for the user's reference.

The screen is intuitive for the user as it uses the same blue color to indicate the information for the user to read. There are only two buttons, a back button and one to download the zip file with results. The actions of these buttons are clearly communicated to the user through

text on the buttons. When the user clicks the back button, they will be taken back to screen 2. When the user clicks the download zip button, the file will be downloaded to the user's computer.

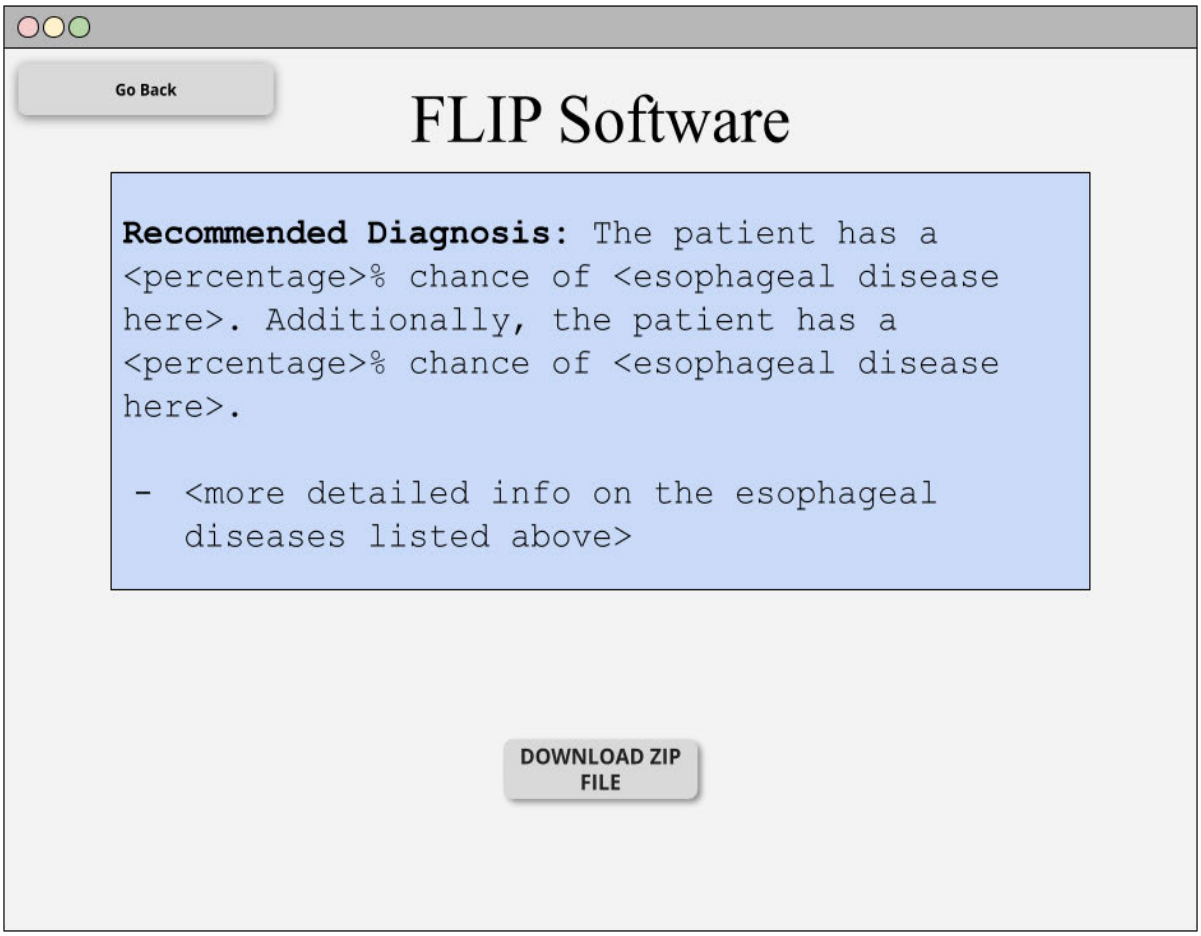

Figure 3: Screen 3 of FLIP Software or Diagnosis Page

Methods:

To conduct our heuristic evaluation, we used user scenarios that covered all the basic functionalities that we expect out of our application. Since our platform is built to run existing code, rather than us generating new code and logic, these scenarios are centered on making the existing software accessible for the average gastroenterologist. We looked at different situations in which the user could run the software, what would happen if a corrupt file was uploaded or some other error occurred along the way, and how the user could download the output, among other scenarios. Although straightforward, our scenarios were comprehensive in addressing different ways that the user may interact with our product.

Moreover, our heuristic evaluation was unique with respect to the prototype we created because it addresses how the same screens may be used or interacted with differently in different use cases. This is especially important-- since we have a simple UI with only a few screens, we need to make sure that each component of the prototype is effectively serving its intended function.

The heuristic evaluation of the prototype (including all the screens and user scenarios) was conducted on basis of the Neilsen's ten principles of interaction design or better called as the Neilsen's ten usability heuristics for user interface design. This evaluation was done both by us as well as another team from our class.

The heuristic evaluation was first done based on first heuristic of Neilsen which is the visibility of system status. This heuristic means that the user should always be informed about what is going on in the application. The second heuristic is the match between system and the real world. This heuristic means that the language and phrase used in the screens should be real life conventions instead of system-oriented terms. The third heuristic is the user control and freedom. This heuristic means that users should be able to navigate from one screen to other without any constraints. The fourth heuristic is the consistency and standards. This means that the system should follow a uniform convention and users should not get confused with the meaning of words and actions. The fifth heuristic is the error prevention. This means that errors are prevented from occurring in the first place by checking for and eliminating error-prone conditions. The sixth heuristic is the recognition

rather than recall. This heuristic means that the user's memory load should be minimized by making objects, actions, and options clearly visible, rather than needing to be inferred or memorized. The seventh heuristic is the flexibility and efficiency of use. This heuristic means that the product is flexible enough to for different users to efficiently use it in different scenarios. The eight heuristic is an aesthetic and minimalist design. This means that only necessary information and design is displayed to the user to avoid distraction and confusion. The ninth heuristic is to help users recognize, diagnose, and recover from errors. This heuristic means that users should be able to navigate from one screen to other without any constraints. The tenth heuristic is the help and documentation. This heuristic means that users should be able to navigate from one screen to other without any constraints.

The detailed heuristic evaluation of prototype done by us can be seen in the Appendix 1 of Appendices section. The detailed heuristic evaluation of our prototype done by other team can be seen in the Appendix 2 of Appendices section. These evaluations are done on the six user scenarios and the three screens of our prototype. The detailed information of the user scenarios and the tasks associated with it can be seen in the table below, which is Table 1.

Table 1: User Scenarios and Tasks

Scenarios	**Tasks in Order**
Upload and Run the text file	1. Begin with a new user 2. Put first name and last name in the text fields 3. Click on upload file and upload the txt file from the local computer 4. Click on the run flip button
View the Video Model	1. User clicks on the video model button and the video model starts. 2. User can pause the video model at any point of time. They can also see the distensibility index at every position. 3. After the video model ends, user will be able to see the median distensibility index and the max and min of mean distensibility index.
View the Recommended Diagnosis	1. User clicks on "Recommended Diagnosis" 2. User can view the programs calculated diagnosis based on the flip data. They are on screen 3 now, which also provides the other diseases along with their corresponding statistical chances that the patient may be suffering from.

	3. When the user finishes reviewing the diagnosis, it clicks the go back button.
Download the Output File	1. User clicks the `DOWNLOAD ZIP FILE` button. 2. The application compiles the data blobs on the front end and presents the information to download. 3. The user can choose where to save the zip file (if enabled in their browser) and the file is downloaded.
Run the software on a New File	1. The user clicks the `RUN FLIP ON ANOTHER FILE` button. 2. The application returns the user to screen 1. 3. User again must write name and upload a new txt file.
Close the Application	1. The user simply exits the application since the entire application is stateless.

Findings and Recommendations:

Key Findings:

From the heuristic evaluations, we have found that our prototype is generally very successful as the user functionality of the program is very simple, and thus, the interactions that the user must make are intuitive and cohesive. Overall, the program provides a very streamlined system for uploading data, processing the data, and viewing/downloading the data. Our design for the program is very minimal and color choices as well as minimal UI elements makes it aesthetically pleasing. Most user errors that may occur are handled internally such as the automatic naming of files where patient names are not inputted as well as the prevention of moving forward within the program if the user does not use appropriate file types or data files.

Through the heuristic evaluations, we encountered several successful features in our prototype. Firstly, in our prototype, at every moment of the application lifecycle, the interface will either be waiting for user input or will alert users about something happening in the background. For instance, in the **Figure 1**, we can see that if the user uploads the wrong type of input file, then the user gets the message that "uploaded file is invalid". This satisfies the visibility of system status heuristic in the heuristic evaluation. Secondly, the heuristic match between system and the real world is satisfied is also satisfied in our prototype. In the prototype, we have only used simple terms, for example "upload file" and "run FLIP" in **the Figure 1**. All these phrases can be understood by most of the people. Thirdly, the heuristic user control and freedom is also satisfied in our prototype. For instance, users can return to a previous screen or can undo or redo. This heuristic is satisfied in the screen 3 (**Figure 3)**, where we have a button "go back" that returns the user to the screen 2 (**Figure 2**).

The heuristic error prevention is also satisfied in our prototype. The FLIP software checks for file errors before running the software to prevent errors upfront. Likewise, the type of file that can be uploaded is restricted so the user could not possibly upload a file of the incorrect format in the first place. The heuristic recognition rather than recall is also satisfied in our prototype. Our product addresses this by clearly labeling every part of the screen with words as well as colors. The words provide clear explanations for the user to recognize the

different functions, while the colors support the meaning and use of the different parts. The heuristic aesthetic and minimalist design is also satisfied in our prototype. Our minimalist design only includes necessary dialogue, buttons, and boxes, and the flow is logical so there are no sub diagrams or formatting that could be difficult for the user to follow.

However, the peer evaluation did raise some concerns for some issues that were not initially addressed or found when we performed the heuristic evaluation on our own prototype. We will classify the concerns that were raised as follows:

- Critical: If we do not fix this, users will not be able to complete the scenario.
- Serious: Many users will be frustrated if we do not fix this; they may give up.
- Minor: Users are annoyed, but this does not keep them from completing the scenario.

Although some of the features of the prototype are successful, there are indeed few serious problems and several minor problems that would be beneficial to address immediately or in future iterations. There are no critical problems in our prototype. The serious problem is that there are not many visibility statements in the prototype and does not appropriately handles the error detection statements. Although it handles the errors internally, it lacks the statements of error detection in screen 1(**figure 1).** For instance, if the user runs the FLIP without entering values in the text field, then the screen should show an error message. However, the current prototype does not show an error message. Although the software will not run further, user will face difficulties in troubleshooting the errors at the point. If the user never figures where the error is occurring, it will not be able to go to the next screen or we can say that software will not run further. It can be very frustrating for the users to understand why and where the error has occurred and if not figured then they will eventually give up.

The first minor problem is the lack of easily accessible documentation. Because the program is so simple, the documentation needed should also be simple and would likely be in the form of a pop-up accessible from any screen. The button to access the documentation should be clear in purpose (likely represented by a question mark) and will be in the same location on every screen. Another minor issue is that the return button from screen to screen are in different corners of the screen. This may cause users attempting to move back multiple

screens to mistakenly press the wrong button. This does not affect the function of the application but may be mildly frustrating to users. The third minor issue is that the text in the screens and button in the screens are very words and text heavy. This will not cause any problem in running the software. However, if we add more visuals and images in the screens instead of text, it will be easier for the users to navigate around the screens. The fourth minor issue is that the video model part of the screen 2 (**figure 2**). Our prototype lacked the basic features of running a video model: button to pause the video, button to resume the video, button to rewind the video and other features. This can be frustrating for the users. It will not stop the software or will not lead to any errors, but the users will only be able to see the live stream.

Key Recommendations:

From our heuristic evaluation, there are multiple areas where we can improve our prototype. The main area where we can improve is adding more statements in the screen so that the user will not get confused and will be aware of what is happening in the program. One of the recommendations is to add the error detection statements in the screen 1(**figure 1**). We already have the statement for uploading a wrong text file. The recommendation is to add a statement if the first and last name are not entered in the text fields or we can make those fields with asterisk, real life convention to show that some value need to be entered in that field otherwise program will not run forward.

The other area that could be improvement is providing help and documentation for the user. This was overlooked in our prototype due to focus on the simplicity of our design, but in our next iteration, we will be sure to provide tooltips and other means of information so there is no confusion for the user when running the software.

Another recommendation is to follow the basic convention of placing buttons in the screen, changing the place of button “advanced information” with button “run FLIP on another file” in screen 2 **(figure 2)**. This is just to make the application more user friendly.

Another recommendation is to change the texts in the screen with symbols and visuals. For example, download the zip file button in screen 2 (**figure 2)** can be replaced with a downward arrow, signifies the download button in general. Upload File button in screen 1 **(figure 1)** can be replaced with a paper clip button or better called as an attachment sign button. These symbols are often used in real life and are understood by most people. These symbols make the screens look more appealing and user friendly.

Another recommendation is to add all the basic features of running a video in our video model subpart of the screen 2 **(figure 2)**. This can be done by adding the rewind button, pause button, and continue button in the video model part of screen 2 **(figure 2)**.

Revision of Prototype:

Since the EndoFLIP analysis application requires minimal UX (most of the work happens on the backend), our prototypes needed little adjustment for clarification. Besides adding some loading screens for more feedback on what is happening on the backend, we changed up the positions of some buttons for more a more natural feeling, which is visible in the Button Position Revision section.

At present, there are no critical problems in our prototype. Thus, we don't need to make any immediate changes in our prototype. Users may get frustrated while using this software, but it will run without errors. We plan to incorporate other recommendations from the key recommendation section in the future iterations of the project.

Button Position Revision:

We switched the position of the "Get Advanced Information" and "RUN FLIP ON ANOTHER FILE" buttons to match with the natural feeling of the back button being located in the top left of the screen. Figure 4 represents the prototype after the change.

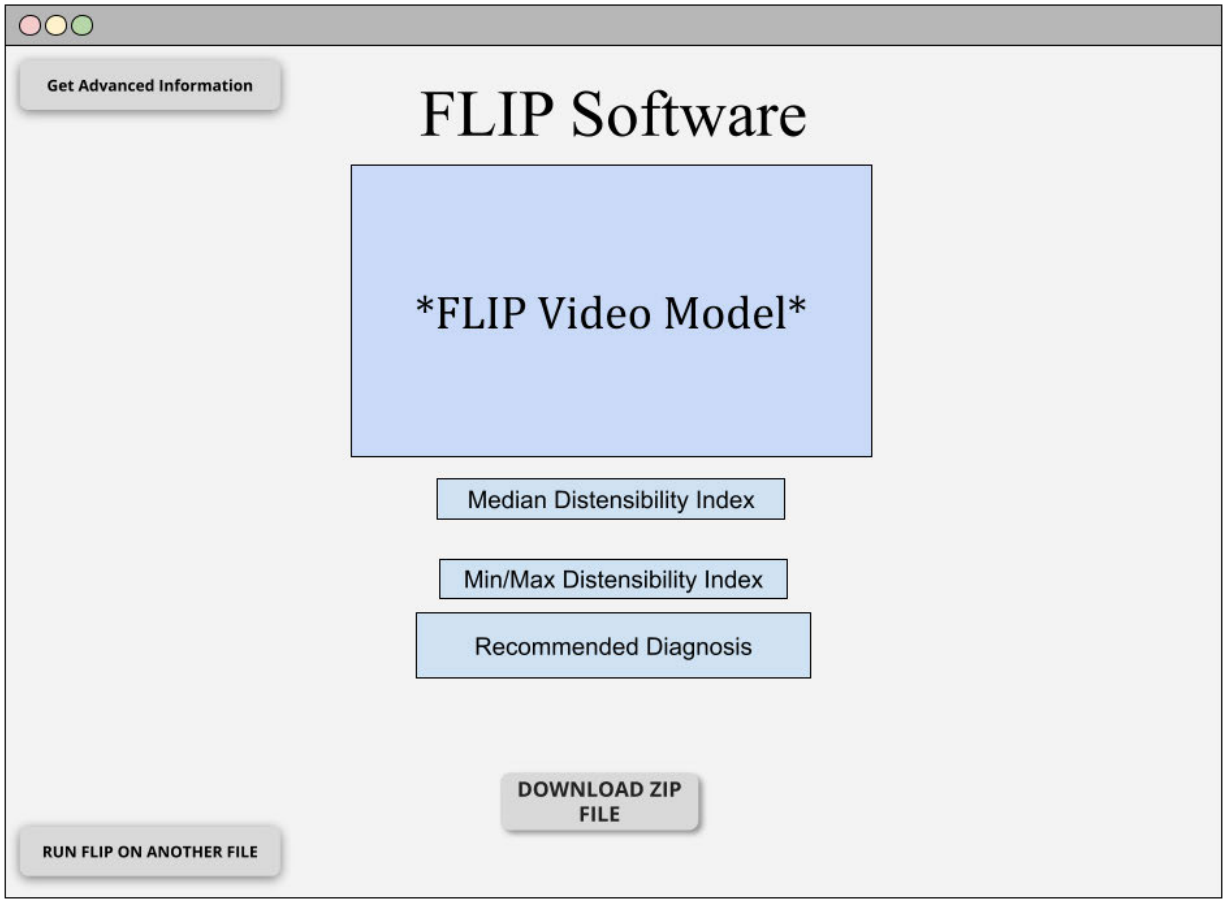

Figure 4: Before button position switch

Appendices:

Appendix 1: Heuristic Evaluation done by Our Team

Table 2: Heuristic Evaluation of Prototype Interface

Heuristic	***Evaluation*** In the space below, enter your observation and evaluation of the degree to which the heuristic has been satisfied. Use as much space as you see fit.
1. Visibility of system status Always keep users informed about what is going on. Provide appropriate feedback within reasonable time.	*In every moment of the applications lifecycle, the interface will either be waiting for user input or will alert users about something happening in the background. If there is an error on the user's side, the user will be immediately informed, always keeping the user in the know.*
2. Match between system and the real world Speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.	*The jargon used throughout the application is either simple web language (e.g. "upload file") that the majority of people can understand or gastroenterologist language relevant to the target users (e.g. "distensibility index").*
3. User control and freedom Users often choose system functions by mistake. Provide a clearly marked "out" to leave an unwanted state without having to go through an extended dialogue. Support undo and redo.	*Users will start on the first page, which is where they input all the information to run the analysis. Since our application is stateless, there is no way for them to exit the application and return to a previous diagnosis. This is by design because medical data is sensitive. However, on the second page, we have a prevalent button to return to the beginning screen. Using a different color for that button might be beneficial.*
4. Consistency and standards Users should not have to wonder whether different words, situations, or actions mean the same thing. Follow platform conventions.	*The language used across the application is consistent with the basic expectations of how the application will perform. Additionally, specific data reporting areas for the program (I.e. motility index, diagnosis, etc.) are very familiar to the users of the program and hold the exact meaning as it pertains to esophageal diagnostics.*
5. Error prevention Even better than good error messages is a careful design which prevents a problem from occurring in the first place.	*Our program should be able to handle any dataset, so we will do as much debugging as possible beforehand to ensure no error results because of our program calculations. Otherwise, any error messages will be presented early on, as soon as the file is uploaded. That is where the errors should be isolated to, unless something happens with our UI.*

6. Recognition rather than recall Make objects, actions, and options visible. User should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. Buildings are all visible and clickable Experienced users can directly check on the buildings they are interested in by clicking on them Recent searches feature is useful for users that use the same buildings regularly	*We follow a common design language throughout each screen to ensure users can easily recognize various elements in our application. All of our buttons have the same shadow outline to show that they are buttons. Input boxes are slightly pinkish and have placeholders. Static text is outlined in a blue box for users to read.*
7. Flexibility and efficiency of use Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user so that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.	*Buttons like "get advanced information" and "download zip file" allow for the user to quickly get the information he/she needs in an efficient manner. Additionally, because the needs from this program are simple, the UI is also simple. There are not too many buttons on the screens, so it is flexible and efficient.*
8. Aesthetic and minimalist design Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.	*The program uses a consistent color theme throughout the screens, with certain colors highlighting specific functionalities. Additionally, the buttons are aligned in an aesthetically pleasing manner, while not making the screen look "too busy." This ensures a simple, minimalist UX.*
9. Help users recognize, diagnose, and recover from errors Expressed in plain language (no codes) Precisely indicate the problem Constructively suggest a solution.	*Our errors are very straightforward and noticeable. Errors will only happen on the first page, since after the diagnosis has run, the user simply observes the application rather than inputting. And errors on the first page are alerted to the user through a distinct red error bar, ensuring no questionability about the state of the application.*
10. Help and documentation Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation.	*Currently, there is no help or information button to help guide the user, as there are not too many buttons. However, in our error messages, we will provide steps to follow in order to fix the data/program to receive accurate results.*

Help information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.	

Appendix 2: Heuristic Evaluation done by Other Team

Table 3: Heuristic Evaluation of Prototype Interface

Heuristic	***Evaluation*** In the space below, enter your observation and evaluation of the degree to which the heuristic has been satisfied. Use as much space as you see fit.
1. Visibility of system status Always keep users informed about what is going on. Provide appropriate feedback within reasonable time.	*Overall, the system visibility status is good. However, there are some areas that could have some more detail in the form of help notes or more concise wording. It also might be good to include where the user is in the process of setting up each of these screens.*
2. Match between system and the real world Speak the users' language, with words, phrases and concepts familiar to the user, rather than system-oriented terms. Follow real-world conventions, making information appear in a natural and logical order.	*This heuristic is met on the first screen but is only semi satisfied on the second and third screens. The wording for each button and text box could be better and more readable from a real-world perspective, or it could simply include notes describing what each text field and button description is referring to.*
3. User control and freedom Users often choose system functions by mistake. Provide a clearly marked "out" to leave an unwanted state without having to go through an extended dialogue. Support undo and redo.	*This system doesn't contain any extra function since as a redo/ undo; however, it satisfies this heuristic since that is not needed. The user has free control over what they will enter and change in each text box and file location. One thing that could be added is a save form button, or some type of save function to allow the user to save their current progress, although that may be unneeded.*
4. Consistency and standards Users should not have to wonder whether different words,	*Design is consistent across individual screens. Design (especially buttons) is somewhat inconsistent across entire application.*

situations, or actions mean the same thing. Follow platform conventions.	
5. Error prevention Even better than good error messages is a careful design which prevents a problem from occurring in the first place.	*Design needs to improve with regards to error prevention.* *An error should pop up if data is not entered on screen 1. An error should also pop up if the uploaded text file is empty or unreadable on Screen 2. An error should pop up if the zip file is unable to be read on screen 2.*
6. Recognition rather than recall Make objects, actions, and options visible. User should not have to remember information from one part of the dialogue to another. Instructions for use of the system should be visible or easily retrievable whenever appropriate. Buildings are all visible and clickable Experienced users can directly check on the buildings they are interested in by clicking on them Recent searches feature is useful for users that use the same buildings regularly	*Design adheres to this heuristic to a high degree. Only suggestion is to add an area on screen 2 which shows details concerning the text file that was uploaded.*
7. Flexibility and efficiency of use Accelerators -- unseen by the novice user -- may often speed up the interaction for the expert user so that the system can cater to both inexperienced and experienced users. Allow users to tailor frequent actions.	*Platform is set up nicely to provide for streamlined use. It follows a logical flow that the user can clearly follow. The process is outlined and provides go back buttons or an easy redo, allowing for user flexibility.*

8. Aesthetic and minimalist design Dialogues should not contain information which is irrelevant or rarely needed. Every extra unit of information in a dialogue competes with the relevant units of information and diminishes their relative visibility.	*In general, the design is minimalistic and uncluttered. For aesthetic purposes, consider keeping button styles consistent and simple (ie: no shadows). Consider changing type boxes on front page to something other than red as that usually signifies an error. Button spacing and sizing could be fixed in screen 2.*
9. Help users recognize, diagnose, and recover from errors Expressed in plain language (no codes) Precisely indicate the problem Constructively suggest a solution.	*It was great that the team already accounted for the possibility that a user could upload a file incorrectly on Screen 1. However, there are also other potential error points, such as text fields that are required, to keep the user from submitting information that is incomplete. Consider adding * for required text fields or having a message pop up if the information is invalid.*
10. Help and documentation Even though it is better if the system can be used without documentation, it may be necessary to provide help and documentation. Help information should be easy to search, focused on the user's task, list concrete steps to be carried out, and not be too large.	*It may be beneficial to include documentation on how to operate and manage zip files. Most people will likely know how to do this already, but it would be a failsafe just in case. Consider having a "?" button somewhere on a screen (perhaps in a corner) for accessing documentation.*

Appendix 3: Loading Screen Prototype

The loading screen when a user has correct inputs and runs the FLIP analysis software on a TXT file. Provides the user with better feedback on what is happening.

Figure 5: Loading screen