While working with the exchanged code we ran into issues, mainly in design, but the program was overall able to implement mosaicking via text-based functionality and displayed on the GUI with the load functionality - however, we were ultimately unable to implement mosaicking via the GUI due to the limitations. An early issue we noticed was in the design of the ImageControllerSwing class which implements ActionListener but not the ImageController interface. Because of this, there would be issues later on when implementing a main method with the ability to run either the GUI or text-based program. This is likely what caused the code to have two classes, each with their own main method: one for the text-based program, and one for the GUI program, but this makes it difficult for a single jar to be produced that supports both implementations. Another limitation in the design that could be fixed is the incompatibility of the methods in the model, view for GUI (ImageGUIView), and controller for the GUI (ImageControllerSwing). A big problem is the GetImage() method in ImageProcessorModel that it returns an ImageModel, but the loadImage(Image image) method in ImageGUIView loads a java.awt.Image, making the GUI controller unable to have the view load an image from the model, other than an image that is loaded from a file as the load() method in ImageControllerSwing returns a java.awt.Image which is compatible with the loadImage(Image image) method in the view.

For the implementation of code, the commands and their functionality are implemented in the model, ImageProcessorModel, rather than the function object. This causes the unorganization of model classes and interfaces and also causes the ImageProcessorModel deal with the storage of images but also how to manipulate them. This can easily be addressed by simply moving each image-manipulation function (brighten, flip, etc.) to their respective command object. Shared functionality can be added to the AbstractCommand. A limitation in documentation was outdated javadoc such as the documentation of the ImageProcessorModelImpl class, which only mentions support for PPM images. Other than the outdated javadoc, the comments and provided README and USEME files were helpful. In addition to this, the design and implementation of the text-based program's view and controller were designed well. A specific strength was the implementation of a map to store the commands in the ImageControllerImpl class which allows for an easy integration of our Mosaic command into the text-based program.

Our suggestions are based on solutions to the previously mentioned design and implementation issues. In order to allow a singular JAR file to be created to run the program in either mode and scripting capabilities, we would combine both classes containing the main method into one, which we implemented in our submission with the mosiacking capabilities - but required changing the type of some local variables such as the view to ImageGUIViewImpl rather than the view interface. To further improve this, we suggest having the GUI controller implement the ImageController interface to have proper inheritance. We would also suggest creating separate function objects to handle the logic of the commands in order to follow SOLID principles and delegate the responsibilities more appropriately. A final suggestion would be to create compatibility between the model and the GUI controller by either: creating and adapter which allows an ImageModel to be translate to a java.awt.Image; or overloading the loadImage() method to allow for an ImageModel to be loaded into the GUI view. Overall, the code for the text-based image processing application was done well, but the design and implementation to support the GUI program had issues that created limitations for the addition of the mosiacking functionality namely: in our final code, we are able to load an image in the model and apply a mosaic through the text-based program, but we are only able to display it one the GUI once that image is saved as a separate file i.e. the GUI program is unable to keep track of the images in the model and apply commands to the "current" image.