# Code Integration for Image Text Translator

1. How we integrated code from team members:

   The purpose of our program was to implement an image translator, in which the input is an image file with non-English text and the output is another image file that looks the same except the text is translated into English.

   We integrated code that we wrote with several libraries and software developed outside of our team. Imported code included the `tesseract` OCR (Optical Character Recognition) engine, the `pytesseract` Python library (a wrapper for tesseract, so that we can use tesseract in our Python code) and the `translate` Python library (so that we can translate the text file generated by `tesseract`). The first person who started our code file set up the input process for our code. This required selecting and importing the necessary libraries, and writing Python code so that we could customize how everything worked. Two other people then wrote the code for the output of the image file with translated text, and also refined and polished both the output and the input process (for example, giving the user the ability to choose the input language for the OCR and translation, and optimizing the output to closely match the input image, so that background color and size are same). At this stage, our prototype was completed. A fourth person added a test suite to the prototype. Three people collaborated to fully develop the prototype, by adding checks for bad user input, and by adding more user options (more input languages and more output styles).

2. The integration method we used was most like a sandwich integration, and to a smaller extent, all-at-once integration. The operational artifacts had been developed and tested by the developers of the Python libraries and tesseract OCR engine. The top-down logic was the development and testing of a properly functioning basic setup (able to input and output an image file). And then the middle: we developed, refined, and tested the interface between logic artifacts and operational artifacts, which was actually the bulk of the code that our team developed.