**Integration Strategy**

In our project, the flappy bird build using the Pygame library has many closely related components and the dependency of one object on the other is also high. For an instance, the major workflow of the product is based on the orientation and movements of different objects like a bird, pipe, and other miscellaneous features like coins and random fire-up elements. So, it created a necessity to work on both the logic artifacts and operational artifacts at the same time and to use the Sandwich Integration approach.

We used the git hub as a primary method to pull the recent code changes made by the peer and to push the modified code to the repository to integrate with the other components. As each team member was assigned a component to work on, we hardly had a merge conflict that could have made the whole process time-consuming, and also it helped us to have a recent version available at any time and start the integration process.

The logic artifacts like building the level which is responsible for setting the obstacles, difficulty levels, themes for the level and orientation of the features like coins, ghost powerup, and fire powerup are coded, unit tested and integrated in the order of top-down integration.

Whereas operational artifacts like bird animation control, accurate Collision detection, tweaking the skin of the bird object based on the element hit, checking for boundary collision and populating the screen with an updated score and also better control over the game over and game start events are built simultaneously. They are unit tested and integrated in the order of bottom-up.

Once we are done with the Implementation of both the logic and operational artifacts, we tested the integration between the two modules as a single entity and ran a test suite that covers a series of test cases to make sure all the functionalities are working as expected.