Our project 3 and 4 was designing a collection of four games where the user can decide which to play. The way our team integrated code was to have each person code and test a game and then when everyone finished coding and testing their game we combined them and tested them all together. I think the integration strategy our team used was all-at-once integration. I think this is the integration strategy we used because in all-at-once integration each code artifact is coded and tested separately (unit testing is done first), then after each code artifact is coded and tested they are linked together and the product is tested as a whole (integration testing is done after unit testing), and integration doesn't begin until all modules are unit tested and then all modules are integrated and tested simultaneously. This is the way we integrated our code because we each did unit testing on the game we wrote, integrated the code after everyone finished unit testing, and then performed integration testing. Some drawbacks of using all-at-once integration testing are if the product as a whole fails the fault is difficult to isolate, major design faults show up late, and operational artifacts are not adequately tested. Because of the nature of our code (4 separate games where the user chooses which to play) we didn't encounter difficulties with isolating faults but we did encounter difficulty with organizing our code. Although there aren't any strengths associated with all-at-once integration it was a convenient integration method for project 3 and 4 considering how we divided up the code.