The provider provided us generous amount of interfaces for us to work with. They were flexible enough for us to make adapter classes that adapts our own class interfaces to their interfaces and make class that implement their interfaces. It is notable that they have a good number of interfaces, even for some sub-classes, which makes the code implementation flexible for our side. However, they are missing a controller interface, which is a very important component in a MVC pattern. This forces us to make our own Controller implementation using our own Controller interface, to make our code work with them. Furthermore, they have different overall style in the creation of animation. The provider uses shape class to store their shape names, type, and other properties of animation. It is definitely one way to make an animation, but as the number of shapes grow, there will be certain redundancies and data duplication, where they will have numbers of shapes properties, keyframe, and motions all holding the same data for the shapes. The advantage of using a shape class, is for readability and traceability. A more effective way for scalability would be to make a map of shape and type, where the map stores the name and type, and the shapes get their properties of animation using the list of keyframes and motions.

The major downside of the provider design and implementation is that the provider has significant amount of redundancies in their interfaces. They have a lot of methods in their interfaces, which are never used at view implementation. They could have polished their code better to avoid redundancies and code duplication. This confuses us when adapting our own class to their interface, because there are some methods, which are not used, but still written there for what we believe to be no reason and redundant. Furthermore, some major things is that the editor view is not fully implemented, and only the visual view is working, with some duplication code here and there. The provider attempted to do the extra credit tasks, without

initially making a working editor, which is not a good practice. It would be a more effective approach to polish the editor before moving to the extra credit and not the other way around. However, this might have been the cause of poor collaboration and communication in a team.

As for the documentation, the classes, interfaces, and methods do not provide us with a comprehensive understanding of their use. However, the provider does give some of the documentation in the readme file. It would be a convenient and elegant to have them on top of each class, interfaces, and methods to avoid going back and forth from file to code. Furthermore, the provider uses some complex methods (in VisualViewImpl and EditorViewImpl) with little to none explanation of what each method is doing. This led us, as the customer, to spend significant amount of time attempting to figure out what each method is doing. A good documentation and explanation of what each method (both private and public), would save some time for the customer.

Without regard to the completeness of their editor view, some code limitations to theirs style of animation is that as the number of animation in textual grows bigger, the number of storage required for that is going up rapidly. This is due to the amount of code duplication that they have in their overall design of code, where they have three classes, which all contain similar data. This could potentially cause inconsistency and redundancy. A suggestion would be to reduce the amount of code, classes, etc that contain the same data.

In conclusion, the provider interfaces are flexible, which makes adapting interfaces very friendly. However, there are unused methods, redundancies, and code duplication.

Furthermore, it would be great to have a good amount of code that explains what each method is doing and the logic behind it.