# Questions

## Was your Feature Plan adequate?

As a team, we believe that our feature plan was adequate. We planned together as a group the ways that we should tackle this. Our first step was to go through the assignment, document every feature as well as requirement in the rubric, and create a rough sheet for what we needed to work on. Here was where we discovered all the different implementations we needed and really got to discover what was asked of us. After we compiled every task into a document with a proper naming convention, we examined what we needed to tackle first and the order of it all. We found that we organized our schedule into this order, island shape, elevation, assigning lakes, aquifers, rivers, completing moisture and soil absorption, and finally implementing biomes and Whitaker diagrams. This gave us a clear cut into what our process will be and how we should tackle it. Once we looked to see how each task met our schedule, we clearly created a timeline for us to complete our work. This left us with a finished product that we did well on.

## What were the challenges in this assignment?

The challenges that we faced were using the new code that was provided and completed. Our old code was good and created meshes. However, it was a but behind with some process that were going to be used. We could have used it, but we wanted to use a complete and error free code that was provided. This meant we needed to implement some of the code from the repo into our code. Overall, it was difficult to understand this new code and get it working in the ways that we wanted. Furthermore, creating and designing how the rivers flows was complicated for us. We first created elevation but added it to our polygon as a characteristic. We realized once we created this elation style, it ended up not work with flow too well. We decided that we had to scrap this idea and change elevation to be on the segment edges instead. Failing early and facing challenges like this were good as it ended up making our design much better.

## How does your code respect the SOLID principles? Illustrate your answer with a class diagram representing a global overview of your project.

To begin, we took the SOILD principles and applied them with high priority into our code.

First, we took on the single responsibility principle. Here we took time as to go through each feature that needs to be added. What we did was create a separate class for each feature. In our code we have a MeshAttributes folder which contains each separate attribute that our mesh can contain. Each class in this folder tackles one feature and if there is anything that is needed that is outside the given class, we make sure we appropriately relate the two classes using the skills we learned with UML design.

Second, we look at the Open Closed Principle. With this principle we implemented it by doing in it in our work. We made it a rule that we would check over our classes or interfaces and made sure that once we completed them, they would be closed for any modification. Likewise, when adding new features we had it so that our classes should only be extended instead of modified. This added to our OOP portion of the rubric and helped keep our code really well written.

Third, we have Liskov Substitution Principle. Here we can see this used in our interface called BoundedShapes. Each shape that extends this interface can easily be switched out for our parent. For bounded shape, we have our bounded method and scale method, where bounded is the coordinates of or shape and scale is the scaled factor. We then have square and circle that extend our interface. Each one can be switched and replaced to follow the principle as our shape is abstract and can contain any shape.

Fourth, we have interface segregation principle. We made sure that any functions not needed by our user were not easily accessible. For example, we have either an irregular, square or circular shape. This meant that if you were to access the interface that contains this class, you would not be able to access any of the functions that do not concern the user.

Finally, we have the Dependency Principle. Here we made our abstractions as our key priority. We realized that our code is depends too much on the visualizer class and a lot of the things in our MVP was hard coded. We decided to make sure that everything can be changed to so that it is abstract instead of being hard coded with if statements. This is seen through our lakes and rivers in our generated mesh. We made it so if the points in our bounded shape were inside a certain area, they would be lakes. We quickly saw this didn’t rely on abstraction and quickly changed this in our code.

DIAGRAM

## Which GRASP patterns have you used when attributing responsibilities?

The different GRASP patterns we used were High Cohesion, Low Coupling, and Polymorphism. For high cohesion, we designed each class in our program to have its own purpose. We made it so that every single has one responsibility and has a clear focus and purpose. This makes it so changing any code for testing or adding is easy and reliable. We made sure that each class is easily identifiable and only relates to a single responsibility. Second, we used the low coupling pattern to our advantage. In our previous assignments, we scored low on OOP design and only used four or five classes that made it so everything relied on these couple of classes. Now, we made it so that everything that we created is in a single class, and we made sure that we minimize every single dependency on all the other classes. Finally, we made sure to utilize Polymorphism. Where we found that some classes should extend others and that some classes were perfect cases when we should use interfaces. Using interfaces and extending our classes made it so our code was reusable and easy to add and test.

## Which GoF design patterns have you used?

For the Gang of Four patterns, we used the design pattern called Factory Pattern. We mainly tried to implement the pattern when creating our shapes. It was hard to grasp the task, but we were able to change it around to fit what we needed. We first created our interface which tackled the abstract shape. Next, we were able to extend from it different shapes such as an irregular, square and circular shape. After that we have a factory class that will create the given shape that we want for our island. This class extends into the interface and takes out the shape that is needed from the command line argument. This follows the Factory Pattern very closely and allows us to practice the different implementing different patterns into our code.

## How did you design your test suite?

For our design test suite, we wanted to test all our methods as well as the features we have added to our code. For example, there are certain situations where our code give us a land tile or a water tile. This means that we must test these situations to see if our code follows the right example. We followed the McUnit we did in class and tried to create something that really resembled it. We create methods that would compare if tiles were the correct ones they should be as well as tested the methods that assign each tile.

## Using a sequence diagram, illustrate the island-building mechanism.

* 1. DIAGRAM