**Was your Feature Plan adequate?**

Not very adequate, we change our feature plan slighter when development, to increase the efficiency, We have found that it is more efficient to have the same person doing related work, But the distribution of tasks did not change significantly.

**• What were the challenges in this assignment?**e majority challenges in assignment for our group is communication in the team and How to smoothly connect and render each step of the mesh, because everyone's ideas and what needs to be used are always changing, so the group needs to constantly discuss new requirements and communicate.  
**• How does your code respect ? Illustrate your answer with a class  
diagram representing a global overview of your project.**

Chart, scatter chart

描述已自动生成

(the high quality graph in umlClass.puml in git)

We use **Single Responsibility Principle** to make sure that one class Renderable only Responsible for one step of generation, for example our ElevationRenderer is the only part responsible for Elevation, same as other class that implement Renderable. For our class implement Generatable, they do the basic generator for the each feather. And Renderable class control the Generable class to make the feature. So If the software needs maintenance, just change the corresponding part. We use **Interface Segregation Principle** to make sure that Generable class can generate the feature, and Renderable class can Renderable the feature to mesh. Clients will not be forced to implement interfaces they do not use.

**• Which GRASP patterns have you used when attributing responsibilities?**

We use **Polymorphism**, using Generable and Renderable interface to make sure the Generator and Renderer. We have **Protected variations** that Infoset object contain the all the information we need for Tile(e.g.. temperature, Elevation, Color), and use get and set function to access.

Also we have **Controller class** implement Renderable interface to control all the class that implement generator interface. In one mesh, using many Renderable class to Render the feature, in each Feature using few Generator class to Generator the feature. **• Which GoF design patterns have you used?**

we use **Singleton** Ideas in the to calculate the Elevation. When traversing all polygons, If Elevation not exist then create and give to the value. If the value already exist return. We use **proxy** Renderer to give a surrogate for control and access the generator class. Also we use certain degree of **composition**, for example our mesh is made by background sea first, and then we give the island, the mountain and the other attribute one by one.

**• How did you design your test suite?**

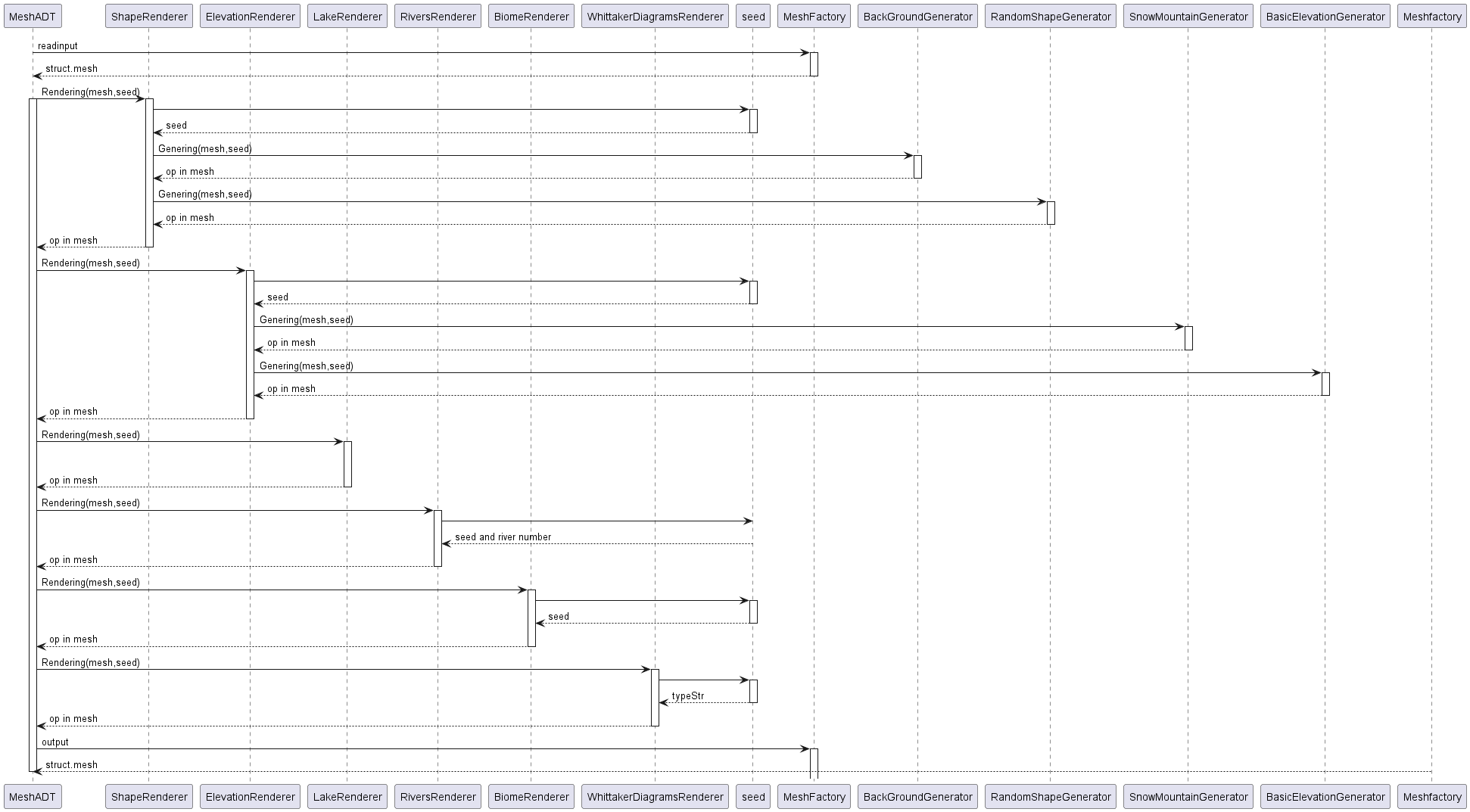
for each feature, we use individual test suite, and using the seed 148000, this seed has large enough island and flat enough that we can have good test view.

For the mesh Authenticity, Since all our shape is creator by Simulink function by seed, We use randomly generated different seeds to test our parameter generation system and drawing system, and change the weights of our internal parameters based on these images to achieve islands that match the real situation。

**屏幕上有字

低可信度描述已自动生成**

**• Using a sequence diagram, illustrate the island-building mechanism**

****

**Text

描述已自动生成Text

描述已自动生成A screenshot of a computer

低可信度描述已自动生成**