# 1.0 Motivation

(Eric)

- Have a purpose/motivation? -> what is the problem abstractly

Problems:

- if a class consists of many unrelated subclasses, and needs new operations added to it frequently, or modify its operations frequently, then it can become difficult to maintain

- you would have to modify each child of the parent class to add the new operation

Solution:

- have a separate class to house and implement the new operation

- this is called the visitor class

- it will accept all children of the object class and can treat them differently

- Intended use case? -> what are some examples

- will uses taxes to explain above

- many unrelated operations on an object structure are required

- new operations are added frequently

- a class has many children with shared functions, but would like to have the functions maintained and managed in a single location

(Karan)

**- Have its own vocabulary? (define)**

In a visitor design pattern, a visitor class is used, which changes the execution of an element class’s algorithm. This pattern is used when we have to perform an operation on a group of similar kinds of objects.

It consists of two parts:

1. A visit() method which is implemented by a visitor and is called for every element of the program.
2. A visitable class that provides an Accept() method which accepts a visitor.

**- Specific structure or runtime behaviour?**

 - Have known consequences (both positive and negative)?

Advantages:

* If logic of operations in visitable items changes ie, we want to handle them differently, we only need to modify code in the concrete visitor visit methods.
* Visitors can have and maintain state relating to the different visitable items they visit
* Adding new visitable items does not affect current visitable functionality

Disadvantages:

* Return type of the concrete *visit()* methods must be known when designing otherwise all the signatures in the interface and implemented methods
* As we increase the number of visitor implementations, it makes the system difficult to maintain and extend.
  + If a new visitable item is added, all the visitors need to implement its *visit()* method

 - Improve / degrade NFPs discussed during architectural discussions?

-Improves scalability in the case when new operations on classes are needed frequently, and the object structure consists of many unrelated classes. This is because it is inflexible to add new subclasses each time a new operation is required.

-It can degrade maintainability in the case when we need to add new elements to an object’s structure. Once a new element is added, all existing visitors must be updated with the new method responsible for processing this new addition.In the case when we have multiple visitors,this update process can take some time. Furthermore, using the visitor pattern can cause the business logic of an object to be spread be spread all over visitor implementations.

-It degrades testability due to the extensive use of polymorphism and the fact that its implementation is based on double dispatch.