The **Strategy Pattern** for software design is a key component of our system architecture.

We realized early in the development of our grouping algorithm that it had several different aspects, each with its own complications. With regards to the features we aimed to implement, we identified the tasks of making groups as:

* Handling group size effectively (if class size not exactly divisible)
* Enforcing mandatory partners
* Preventing forbidden partners
* Balancing skills among groups
* “Filling up” (completing partial) groups made necessary by other parameters

This lead us to create classes responsible for each task, coordinated by an overall “GroupMaker”, rather than making such a class that would handle all of these components. In this way, **encapsulation** was an effective technique.

But we then further realized that there were several approaches for each aspect, and the client might in the future continue to specify new ways in which the grouping should be done. So it was crucial that it be easy to add and switch between different ways of completing each task. It was here that the Strategy Pattern was vital.   
  
We conceived of 5 **interfaces**:

* GroupSizingStrategy
* MandatoryPartnersStrategy
* ForbiddenPartnersStrategy
* PartialGroupCompletionStrategy
* SkillBalancingStrategy

to which we could add more as the grouping became more sophisticated with additional features.

Thus, as new ways of completing each task were thought of or requested by the client, it would be a matter of creating new **concrete classes** which implement these interfaces. Each concrete class handles its task in the specific way as desired.

The GroupMaker, then, instantiates variables corresponding to each interface according to the client’s wishes as to how the grouping should be done. It could then simply call upon each instantiation to perform its task in its particular way, as specified in the parameters.

By **coding to interfaces** in this way, the design became very **flexible and extendable**. New ways to size the groups, enforce partners, balance skills, etc. can be added simply by adding a new concrete class that implements the relevant interface and can be called by the GroupMaker. Any changes in the implementation are also completely local.

This design was remarkably effective in building our system and has made us confident of the ease of adding further features in the future.