AutoAdvisor Use of Design Patterns

Observer Pattern:
We are currently using the Observer Pattern, in which a subject object maintains a list of observer objects (in this case of size 1) and notifies them of certain events or state changes. In the calculating and recalculating of a users GPA, a subject-observer dependency has been implemented between the Credit.rb class (being the subject) and the User.rb class (being the observer). In use the Credit class waits to be updated (a course added, updated, changed, etc...). Upon receiving an update the Credit class calls the user.recalculate_credits function which notifies the class User of the current user to re-calculate the GPA.
Eg:

```
class Credit < ActiveRecord::Base
        …
        after_update { user.recalculate_credits}
        ...
        def trigger_gpa
           user.calculate_gpa
        end
        ...
end

class User < ActiveRecord::Base
        ...
        def calculate_gpa
           self.gpa = self.credits.average(:grade)
           save
        end
        ...
end
```

This implementation in User.rb and Credit.rb keeps the two classes loosely coupled and reduces dependency, which allows for future change in the application.

Model View Controller:
We are using the Ruby on Rails framework which uses a version of the Model View Controller(MVC) design pattern. MVC is a compound pattern that separates the user from the data it interacts with and keeps code loosely coupled. The user interacts with the controller, the controller manipulates and model, and the model updates the view for the user. In the Ruby on

Rails version of MVC, the controller can both manipulate the model and update the view. For example, in the AutoAdvisor app, for a user to upload a transcript, the user interacts with uploads_controller.rb. The uploads_controller then manipulates the User's model with the current_user.apply_transcript(transcript) call, it then redirects to the transcript_path which updates the view.
Eg.

```
class UploadsController < ApplicationController
    #upload transcript
    def upload_credits
      if (!params[:file].blank?)
        transcript = params[:file].read
        current_user.apply_transcript(transcript) if !current_user.nil?
        redirect_to transcript_path
      else
        #file field was blank
        redirect_to transcript_path
    end
end
```

The controller keeps the user separate from the models and views and keeps the code loosely coupled.