# Runner Project

## Introduction

* Modelled on the Studio\_Game project from Pragmatic Studios
* Represents a software development project called “Runner”
* The project has team members
* The team members get regular ratings
* The team members acquire new skills and improve existing skills
* The project has a manager
* Be sure to “namespace” all code right from the beginning
* Use the standard lib, bin, spec subdirectories.
* Be sure to run “rspec” from the top of the project directory

## Session 1

### Classes and Methods

* Class = TeamMember
* Methods = initialize, to\_s

### Spec

* Before clause: create a TeamMember object
* Tests:
  + The team member has a capitalized name
  + The team member has an initial rating
  + Has a string representation
  + Has a default rating of 10

### Code – Step 1

* Start with the Team Member
* Create a Team Member class.
* Create the “initialize” method with 2 parameters (name, rating). In initialize:
  1. Capitalize the first letter of the name parameter.
  2. Default rating to a value of 10.
* Create attribute readers for both name and rating.
* Create a method (to\_s) that displays the name and rating
* At the bottom of the file, add the “if \_\_FILE\_\_ == $PROGRAM\_NAME” test to run some test when the file itself is run.

### Test – Step 1

* Run rspec team\_member\_spec.rb to unit test the TeamMember class
* Run ruby team\_member.rb to test the class

## Session 2 - Allow a rating of a Team Member & track how often they have been reviewed using a virtual attribute

### Classes and Methods

* Class = TeamMember
* Methods = rated\_up, rated\_down

### Spec

* Before clause: create a TeamMember object
* Tests:
  + Increases rating by 15 when rated\_up
  + Decreases rating by 15 when rated\_down
  + Cannot have a rating < 0

### Code – Session 2

* Add additional attribute for “reviews”
* Add methods for “rated\_up” and “rated\_down”
* In each method, increment the reviews attribute
* Print the number of times a team member has been reviewed.

### Test – Session 2

* Run rspec team\_member\_spec.rb to unit test the TeamMember class
* Run ruby team\_member.rb to test the class

## Session 3 – Add the Project class and add team members to a project

### Classes and Methods

* Class = Project
* Attributes = name (string); members (array)
* Methods = add\_member, print\_ratings

### Spec

* Before clause: create a project object; create a team member and add to the project.
* Tests:
  + Project has a capitalized name
  + Adds a team member to a project
  + Can have multiple team members

### Code – Session 3

* Add methods add\_member and print\_ratings
* The add\_members adds team members to an array
* Iterate through the array to display their starting ratings

### Test – Session 3

* Run rspec team\_member\_spec.rb to unit test the TeamMember class
* Run ruby team\_member.rb to test the class

## Session 4 – Add a Review cycle and introduce randomness in ratings (i.e. the Dice)

Create a method for “reviews” that will cycle through each member and have each member review the other members. A team member cannot review themselves. The rating given will be the result of the dice roll (1..6). A high number (5 or 6) will result in a rating up. A low number (1 or 2) will result in a rating down. A middle number (3) will keep the team member on the same rating.

### Classes and Methods

* Class = Die; Methods = initialize, roll
* Class = Project; method = review

### Spec

* Before clause: create a TeamMember object
* Tests:
  + When a number is rolled, the number is added to rating.

### Code – Session 3

* For the dice:
  1. Create the class for Die
  2. Add methods initialize and roll
  3. Roll returns the value rolled
* For the new “review” method:
  1. Loop through each team member so that they act as a “reviewer”
  2. Loop through each team member again as “reviewee”
  3. For each reviewee, check that they are not the reviewer and roll the dice
  4. Based on result of the roll, rate the reviewee up, down or stay the same
  5. Print out the final review ratings

### Test – Session 3

* Run all rspec’s

## Session 5 – Separate strong performers from weak performers

Separate out the strong from the weak at the end of the review cycles and print out each list at the end of every rating period.

Get input from the user to determine how many cycles should be run.

### Classes and Methods

* Uses array partitioning

### Spec

* TeamMember: context is “when a team member has a rating > 79”
* Tests:
  + They are considered strong performers.
* TeamMember: context is “when a team member has a rating < 50”
* Tests
  + They are considered weak performers.

### Code – Session 3

* Test for strong, weak and average performers
* Get the number of cycles to be run from the user
* Print out a list of each type of performers

### Test – Session 3

* Run rspec team\_member\_spec.rb to unit test the TeamMember class
* Run ruby team\_member.rb to test the class

## Session 6 – Introduce Manager as a subclass of TeamMember. Provide additional ratings boost if the person is a manager

The review of the manager resulting from the die roll should be doubled i.e. its really bad if his people rate him low and really good if they rate him high.

### Classes and Methods

* Class=Manager; methods=initialize, rated\_up, rated\_down, manager?

### Spec

* Before clause: create a TeamMember object
* Tests:
  + Can be identified as a manager
  + Up rating is doubled
  + Down rating is doubled

### Code – Session 3

* Update initialize to run super and also set a type variable to be ‘manager’
* Update rating\_up and down to include a multiplication factor parameter

### Test – Session 3

* Run rspec manager\_spec.rb to unit test the TeamMember class
* Run ruby manager.rb to test the class

## Session 7 – Move some elements into a separate module and introduce loading team members from a file

Move the rating methods into a separate file as a means of tidying up some of the code. Introduce the ability to load all your team members from a csv file.

### Classes and Methods

* Class=Manager; methods=initialize, rated\_up, rated\_down, manager?

### Spec

* Before clause: create a TeamMember object
* Tests:
  + Can be identified as a manager
  + Up rating is doubled
  + Down rating is doubled

### Code – Session 3

* Update initialize to run super and also set a type variable to be ‘manager’
* Update rating\_up and down to include a multiplication factor parameter

### Test – Session 3

* Run rspec manager\_spec.rb to unit test the TeamMember class
* Run ruby manager.rb to test the class