codemanship

Code Craft Driving Test

Sept 16th 2017

Contents

Introduction

Welcome to the first Codemanship Code Craft Driving Test.

The aim of this exercise is to challenge your ability to create a working solution to a non-trivial programming problem that will be easy to maintain. We do not envisage inexperienced code crafters will be able to pass this test.

You have until **09:00 BST tomorrow** (Sept 17th) to complete the exercise and submit your solution.

The Problem: ProNet

ProNet is a social network for professional programmers that helps hiring managers recruit strong teams.

ProNet

Teams Made Easy

ProNet - Programmers

Programmers join ProNet with a unique identifier (their name)

Grace

Alan

Donald

codemanship

ProNet - Languages

Programmers list 1-3 programming skills they have, in descending order of ability

1. C#
2. Java
3. Ruby

Grace

1. Java
2. C++

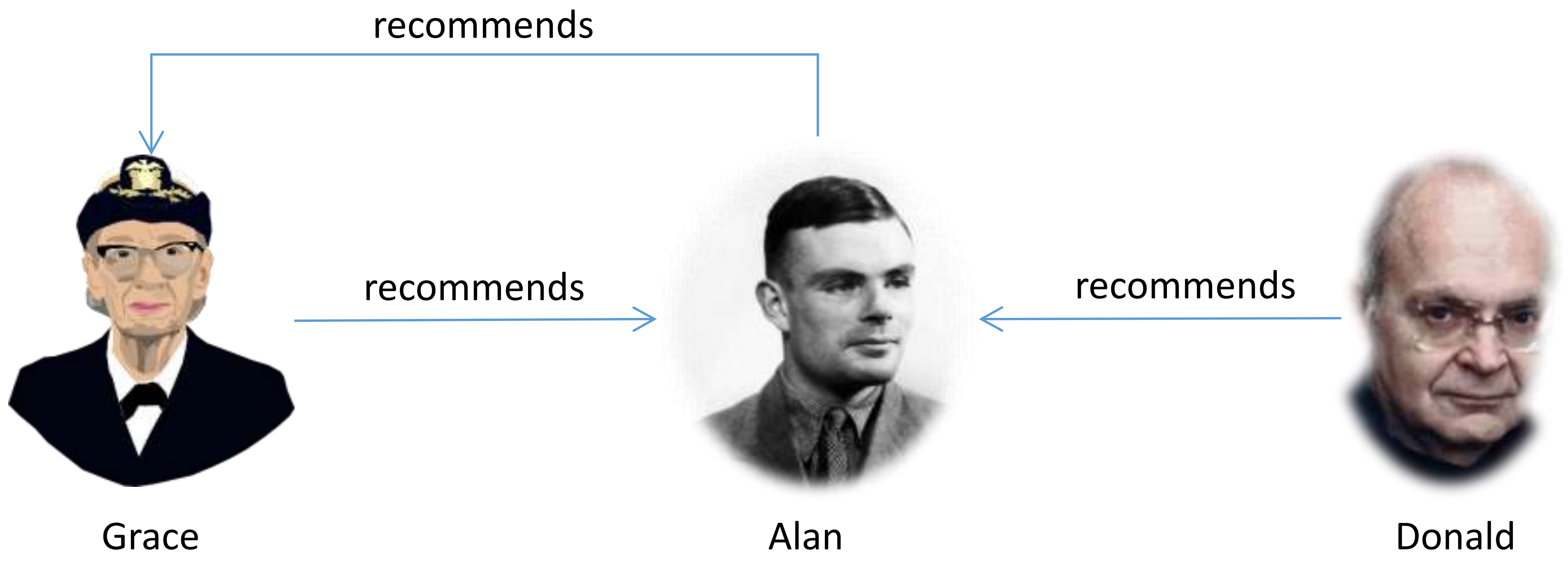
Alan

1. C
2. FORTRAN
3. Java

Donald

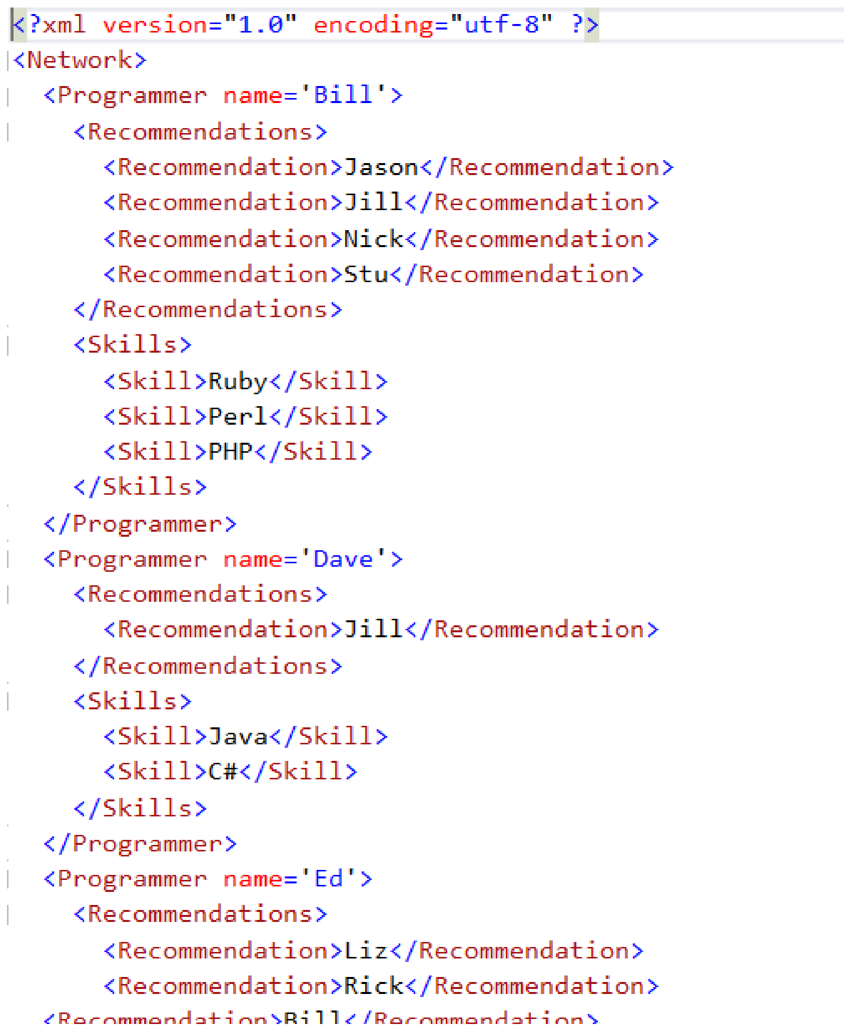
ProNet - Recommendations

Programmers can recommend each other, creating connections



ProNet - Data

ProNet data is stored in an XML file



```xml
<?xml version="1.0" encoding="utf-8" ?>
<Network>
  <Programmer name='Bill'>
    <Recommendations>
      <Recommendation>Jason</Recommendation>
      <Recommendation>Jill</Recommendation>
      <Recommendation>Nick</Recommendation>
      <Recommendation>Stu</Recommendation>
    </Recommendations>
    <Skills>
      <Skill>Ruby</Skill>
      <Skill>Perl</Skill>
      <Skill>PHP</Skill>
    </Skills>
  </Programmer>
  <Programmer name='Dave'>
    <Recommendations>
      <Recommendation>Jill</Recommendation>
    </Recommendations>
    <Skills>
      <Skill>Java</Skill>
      <Skill>C#</Skill>
    </Skills>
  </Programmer>
  <Programmer name='Ed'>
    <Recommendations>
      <Recommendation>Liz</Recommendation>
      <Recommendation>Rick</Recommendation>
  <Recommendation>Bill</Recommendation>
```

codemanship

ProNet - Metrics

ProNet uses 3 metrics to help find strong teams

- Programmer Rank
- Degrees of Separation
- Team Strength

ProNet – Programmer Rank

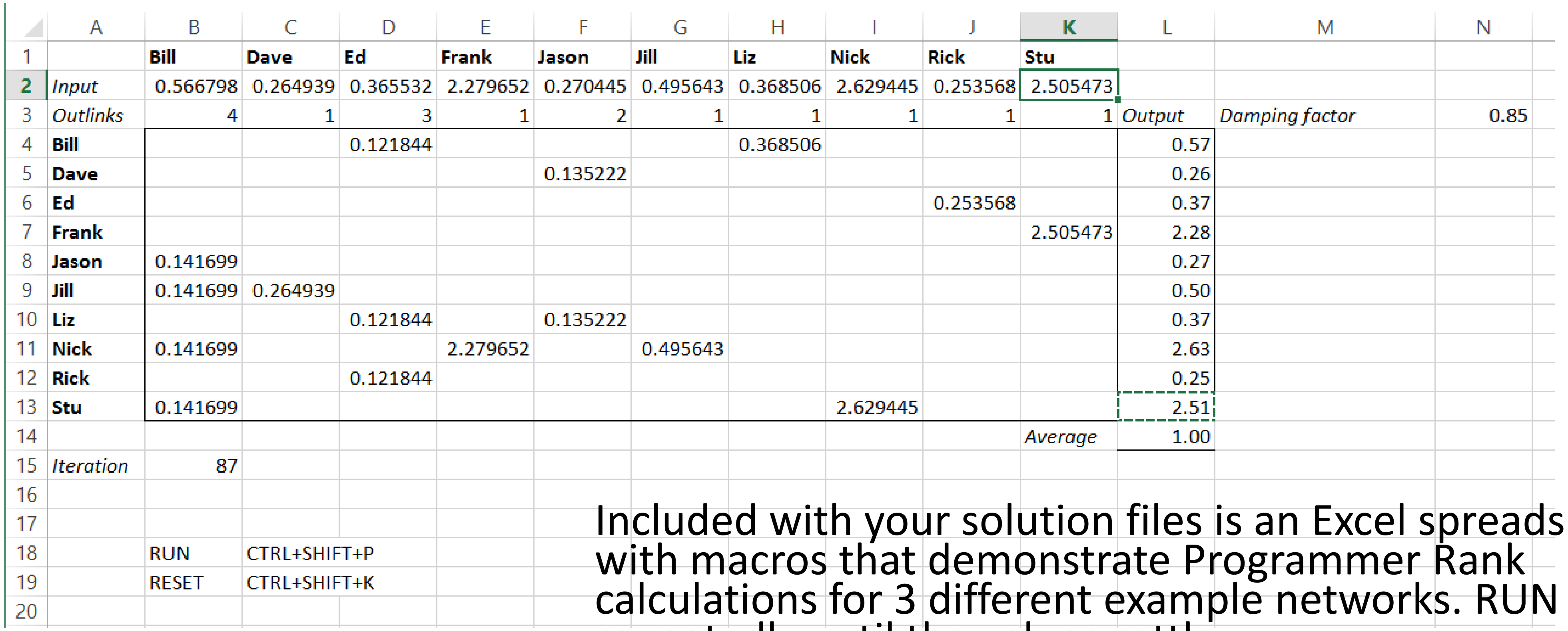
ProNet applies the Google Page Rank algorithm to iteratively calculate the rank of programmers based on recommendations

PR(A) = (1 – d) + d (PR(B)/C(B) + PR(C)/C(C) + ... PR(N)/C(N))

Where B, C…N are programmers who recommend A, d is a damping factor of 0.85 to allow PR values to "settle", and C(N) is the number of recommendations from programmer N

http://www.slideshare.net/OmkarDash/google-page-rank-algorithm

ProNet – Programmer Rank Spreadsheet



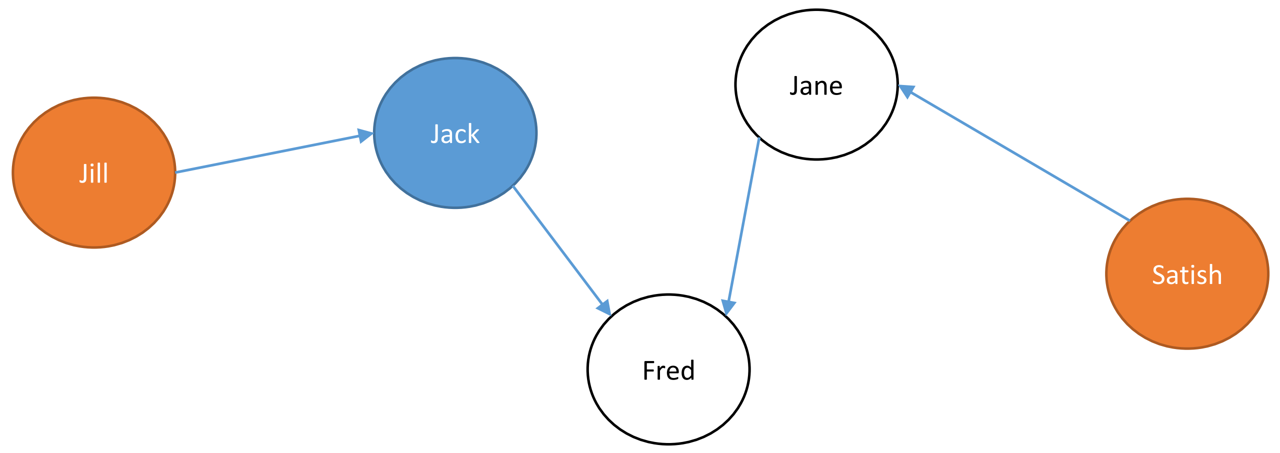
	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1		**Bill**	**Dave**	**Ed**	**Frank**	**Jason**	**Jill**	**Liz**	**Nick**	**Rick**	**Stu**			
2	*Input*	0.566798	0.264939	0.365532	2.279652	0.270445	0.495643	0.368506	2.629445	0.253568	2.505473			
3	*Outlinks*	4	1	3	1	2	1	1	1	1	1	*Output*	*Damping factor*	0.85
4	**Bill**			0.121844				0.368506				0.57		
5	**Dave**					0.135222						0.26		
6	**Ed**									0.253568		0.37		
7	**Frank**										2.505473	2.28		
8	**Jason**	0.141699										0.27		
9	**Jill**	0.141699	0.264939									0.50		
10	**Liz**			0.121844		0.135222						0.37		
11	**Nick**	0.141699			2.279652		0.495643					2.63		
12	**Rick**			0.121844								0.25		
13	**Stu**	0.141699							2.629445			2.51		
14											*Average*	1.00		
15	*Iteration*	87												
16														
17														
18		RUN	CTRL+SHIFT+P											
19		RESET	CTRL+SHIFT+K											
20														

Included with your solution files is an Excel spreadsheet with macros that demonstrate Programmer Rank calculations for 3 different example networks. RUN repeatedly until the values settle.

codemanship

ProNet – Degrees of Separation

Recommendations create links between programmers. These links can be navigated in both directions ("recommends" and "recommended by").



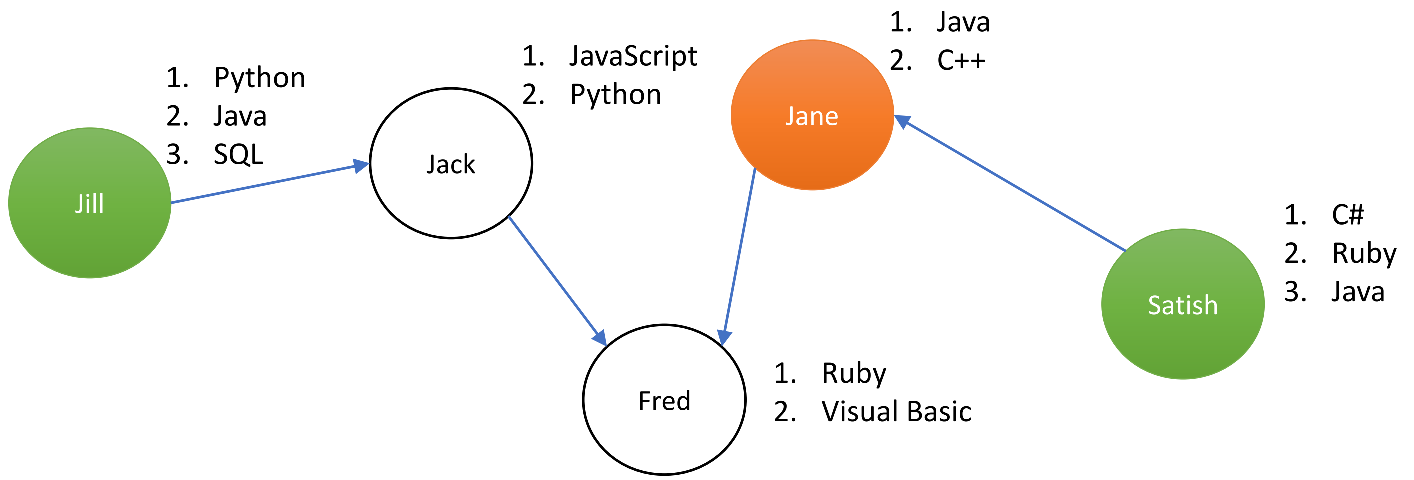
Satish is 4 degrees of separation from Jill

Jane is 1 degree of separation from Fred

Jack is 0 degrees of separation from Jack

ProNet – Teams

Teams are created by selecting a skill, choosing a team leader, then selecting other programmers from the network as members.



Team (skill: Java)

Leader: Jane
Members: Satish, Jill

codemanship

ProNet – Team Strength

The strength of a team for a specific skill is calculated using the formula:

$$\frac{1}{\text{Team Size}} * \left(\frac{\text{Rank (leader)}}{\text{Skill Index (leader)}} + \sum^{\text{members}} \frac{\text{Rank (member)}}{\text{Skill Index (member)} * \text{Degrees of Separation from leader}} \right)$$

Smaller teams with more closely connected members who have greater ability in the desired skill will be stronger

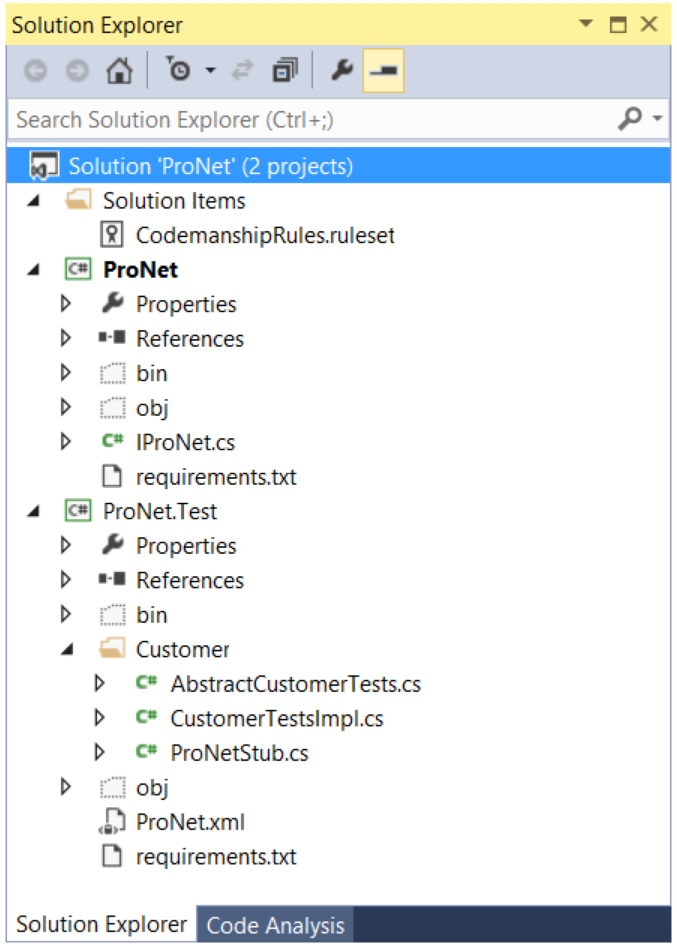
ProNet – Robustness

Your solution must be correct for all these behaviours, and also it must meaningfully handle every input it allows:

1. When a programmer is not found in the network, it must throw an *ArgumentException*
2. When a team size < 1 is specified, it must throw an *ArgumentException*
3. When the data file specified is not found, or is not a valid ProNet data file, it must throw an *ArgumentException*
4. Cases 1-3 are the only times your implementation should throw an exception. In all other cases, it must return a valid response
    1. When an empty team is specified, it's strength = 0
    2. When a team leader or member does not have the specified skill, they contribute 0 to team strength
    3. When the same programmer is included in a team twice, their contribution is only counted once in team strength
    4. No method in the ProNet API should ever return null

codemanship

ProNet – Visual Studio Solution



```
Solution 'ProNet' (2 projects)
  Solution Items
    CodemanshipRules.ruleset
  ProNet
    Properties
    References
    bin
    obj
    IProNet.cs
    requirements.txt
  ProNet.Test
    Properties
    References
    bin
    Customer
      AbstractCustomerTests.cs
      CustomerTestsImpl.cs
      ProNetStub.cs
    obj
    ProNet.xml
    requirements.txt
```

In the VS 2013 solution, you should find 2 projects:

- ProNet – skeleton for your source code
    - Contains the *IProNet* interface you must implement
- ProNet.Test – skeleton NUnit 2.6 test project
    - Contains an abstract *AbstractCustomerTests* fixture
    - *CustomerTestsImpl* extends *AbstractCustomerTests* and currently returns a failing stub that implements *IProNet*

You should also find a copy of the **ProNet.xml** test data file to be used in the customer tests

ProNet – Instructions

```
public interface IProNet
{
    string[] Languages(string programmer);
    string[] Recommendations(string programmer);
    double Rank(string programmer);
    int DegreesOfSeparation(string programmer1, string programmer2);
    double TeamStrength(string language, string[] team);
    string[] FindStrongestTeam(string language, int teamSize);
}
```

Implement *IProNet* with a general solution, so that it passes all of the customer tests using the test data in **ProNet.xml**

Complete *CustomerTestsImpl*, overriding *LoadProNet()* to return your implementation of *IProNet*

codemanship

Timings

The driving test began as soon as you were emailed these instructions.

Your finished solution must be committed to a public GitHub repository.

Email a link to that repository to jason.gorman@codemanship.com no later than 09:00 BST on Sunday Sept 17th 2017.

Jason will be available for remote support for technical issues during the following hours:

Sept 16th

09:00 – 12:00

18:00 – 21:00

Sept 17th

08:00-09:00

If you require hands-on remote support, you will need to have Skype installed as well as TeamViewer (www.teamviewer.com)

codemanship

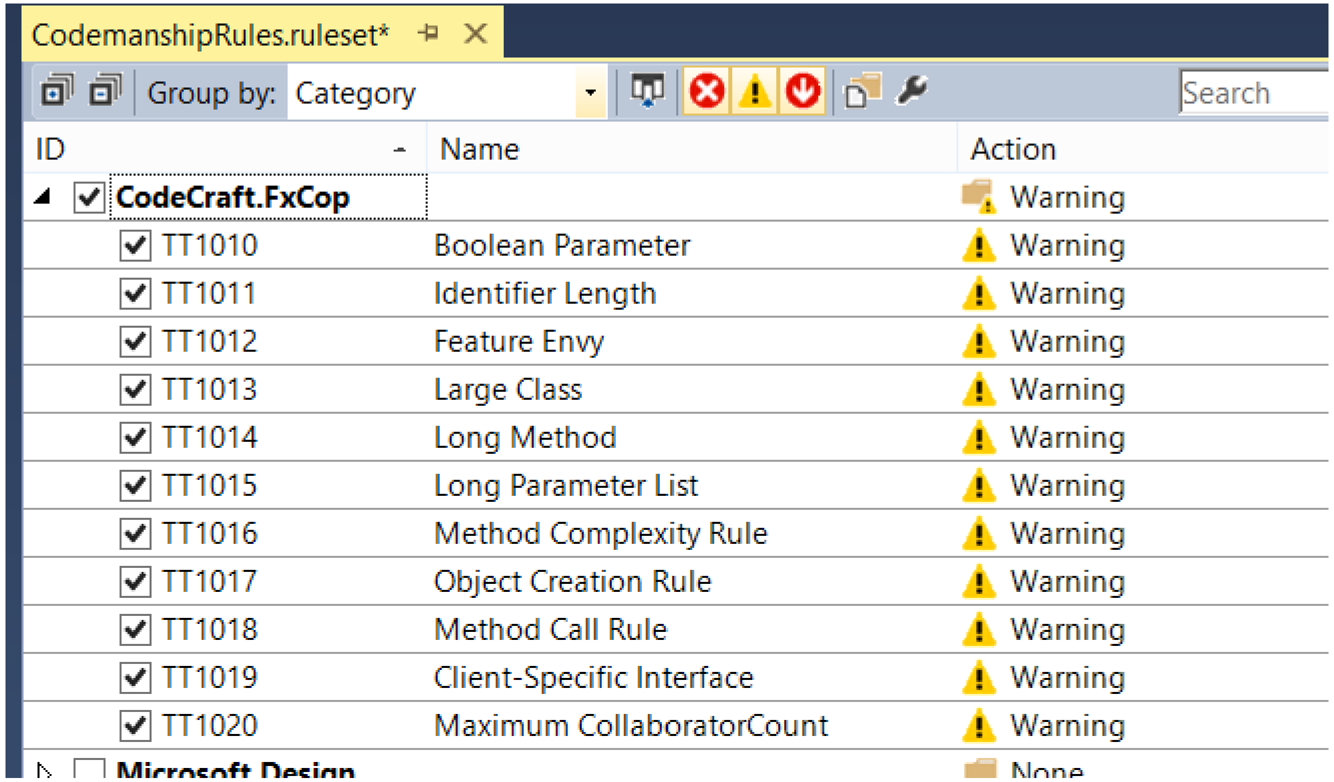
The Rules

To pass this driving test, you must satisfy all of the following:

- Jason must be able to clone your completed solution from GitHub, open it and build it using the appropriate version of Visual Studio (2013, 2015 or 2017)
- Your implementation of *IProNet* must pass all of the customer tests
- Your implementation must also pass all of our more exhaustive tests for those same behaviours
- Your implementation must satisfy a range of code rules, as explained next…

Code Rules - FxCop

The source code in the *ProNet* project must not break the **CodeCraft.FxCop** rules on more than 3 occasions, where you must use the *SuppressMessage* attribute to document a compromise



CodemanshipRules.ruleset*

Group by: Category

ID	Name	Action
☑ **CodeCraft.FxCop**		Warning
☑ TT1010	Boolean Parameter	⚠ Warning
☑ TT1011	Identifier Length	⚠ Warning
☑ TT1012	Feature Envy	⚠ Warning
☑ TT1013	Large Class	⚠ Warning
☑ TT1014	Long Method	⚠ Warning
☑ TT1015	Long Parameter List	⚠ Warning
☑ TT1016	Method Complexity Rule	⚠ Warning
☑ TT1017	Object Creation Rule	⚠ Warning
☑ TT1018	Method Call Rule	⚠ Warning
☑ TT1019	Client-Specific Interface	⚠ Warning
☑ TT1020	Maximum CollaboratorCount	⚠ Warning
☐ **Microsoft Design**		None

codemanship

Code Rules - Simian

The source code in the *ProNet* project must contain no more than 10% duplicated code, and no more than 20% in the test code, as reported by Simian with the following command-line settings:

```
-threshold=2 -ignoreLiterals -reportDuplicateText -includes="*.cs"
```

http://www.harukizaemon.com/simian/

Code Rules – Conceptual Correlation

Included in both projects is *requirements.txt*, which contains a plain text version of the ProNet description in this file.

The console application *Conceptual.exe* will compare the language you used in naming things in your code with words found in requirements.txt, and report the % correlation.

To pass this driving test, your code – source and test – must have a Conceptual Correlation >= 75%

You can find instructions for Conceptual.exe at
http://codemanship.co.uk/parlezuml/blog/?postid=1470

codemanship

Test Rules – Assertions

Tests must contain no more than one assertion (including Verify on mock objects and expected exceptions)

Test Rules – Coverage

Coverage of source code by your **unit tests** (not including CustomerTests) must be <u>>= 97%</u>

Test Rules – Integration Tests

No more than 10% of your tests can have external dependencies (e.g., file access)

codemanship

Test Rules – Execution Time

It should take < 10 seconds in total to execute all of your tests, including integration and customer tests, on a PC with 4GB RAM and an Intel i5 CPU (i.e., Jason's travel laptop!)

Continuous Integration Rules

In order to assess your approach, we need a record of the build history of your solution.

Set up CI for your solution, and add the following email address to the list of recipients for build notifications:

builds@codemanship.com

To pass the driving test, you must commit frequently (>= 10 times), and you should not break the build more than once after it's up and running

codemanship

Submitting Your Solution

When you are ready to have your finished solution assessed, email a link to its GitHub repository – together with a link to your screencast – to:

jason.gorman@codemanship.com

Submitting Your Screencast

At some point while you're working on your solution, you will need to record a screencast demonstrating your approach.

In your screencast, you must go through the red-green-refactor cycle *at least 4 times*, explaining not just what you're doing, but *why*.

Upload your screencast (20-30 minutes duration) to YouTube or Vimeo and send the link with your GitHub solution link to:

jason.gorman@codemanship.com

Grading & Your Results

Provided you have submitted your solution no later than 09:00 BST on Sept 17th, you will receive your results within 7-10 days.

Please ensure your solution satisfies *all* of our rules before submitting, to save our time and your disappointment.