Codurance Code Review

I have looked into the code.  
  
Nice of him providing the already created executable. Nitpicking here, there are better ways to distribute than adding the zip to git, which is not the best use of git. But it did let me test immediately the execution of the project. Everything correct on that regard. Again nitpicking here as it is not part of the text given to James, but there is no way to exit, it seems, other than Ctrl+C. [Readme.md](http://readme.md/) was basic, it would have been better to also show the actual commands that could be used. The text is not specific about what instructions are expected, but is something that benefits by thinking from the point of view of a new user.  
  
On opening the solution I had an issue compiling: The TestUtilities project had the assembly of Moq linked, but not as part of a Nuget download. Once I added it, the application compiled.  
  
I run all the tests and all of them passed. A quick glance and which tests were created and their naming convention did seem to be correct. So I moved into looking more detailed the tests. Started at Integration level. A couple of notes here: ConfigurationShould seems not testing too much (only that there is not a null configuration object), maybe a fuller test checking that the object was correctly filled would have been better; WallHandlerShould would have benefited in terms of readability if the Dictionaries initialization covered less space, sepcially around lines 70 to 80.  
  
TestUtilities and TestUtilityUnitTests seems a bit funky. For the former it is due to design decisions taken elsewhere, for the latter is that I'm not sure that adding unit tests here would be completely warranted, as the normal unit tests where they are used can indirectly serve as tests for TestUtilities. Also, there is a small dissonance between the names. MappingTestData has the same readability issue that WallHandlerShould. Not sure why the test class MappingOptionsTestDataShould exists. It seems a bit anemic. Could have checked not only the count but also confirm what is inside. Furthermore, the comments for Arrange Act Assert tend to be superfluous in general (either the distribution of your code makes them self-evident or the test code needs some cleaning), but over here, when there is nothing for Arrange nor Act, is just wasted keypresses and space. As there are tests for MockUserRepositoryBuilder, they tend to not completely test the achieved state. As an example Build\_GenerateMockUserRepository should not only test that is not null, but it initializes with no information.  
  
Jumping to the UnitTests projects the first thing I notice is that there is a CommandHandlerFactoryShould that exists the same on the IntegrationTests project. All the command handlers tests seem nice, to the point, but it is problematic the constant use of It.IsAny on the verify calls. At no point we hare really testing the storage of the right information on the unit tests. Looking now at the usage of MockUserRepositoryBuilder, a simpler version could have been used, but does the job nicely. See my previous comments about Arrange/Assert/Act. TimeLineShould seems a bit weak on what is testing. Especially the first one, as doesn't really check that what is being passed around is the right data.  
  
Onto the main code. There is one design decision that I want to highlight here, which was the setting up of the handlers on Json. I could use that kind of initialization if the implementations could be added dynamically, after the code was compiled. Otherwise, there is no much point on having it on json, and the setup could have been done directly on code. Unlike Java, interfaces are important in C# to provide virtual methods. But even then, we have to check if something is actually needed: ICommandHandlerMappingFactory is only used on the definition of CommandHandlerMappingFactory and therefore there is no much need of it. The same applies to IRunnable. RegularExpressions seems to much scafolding for something that it is already present on C#. Keeps using as well a pattern of abstract class with private class being returned as public field. With the usages on this code base seems a lot of over-engineering. I can't really see the reason to do it like that.  
  
Commands are relatively well defined, and the pattern is well used. The only exception is the direct use of [DateTimeOffset.Now](http://datetimeoffset.now/" \t "_blank). Based on the way the code is written, property injection for the DateTimeOffset would be ideal. That will help with the previous mentioned issues of It.IsAny  
  
Looking at the git history, most of the issues described would have gone away following a strict TDD approach.  
  
From what I see over here, he has experience, command of the language, but he will benefit from our apprenticeship program. I do recommend to move him for the pair programming phase.