3 Mocking with Python and C#

Table of Contents

3.0 Introduction

This lab will introduce mocking of objects (a fake environment around your source code AKA test harness) when running your unit tests in Python and C#.

According to the rules a unit test is not a unit test if:

- It talks to the database
- It communicates across the network
- It touches the file system
- It cannot run at the same time as any of your other unit tests
- You have to do special things to your environment (such as editing config files) to run it

Source: https://www.artima.com/weblogs/viewpost.jsp?thread=126923

We can in the above cases mock the functionality. But a more common reason to mock may be that the functionality around your module is not implemented yet, got very slow access, produces non-deterministic values or is not easily available. For example a colleague is developing the dependent module, but nevertheless you need to be able to test your own code against the colleagues module.

Prerequisites
See lab 1 and 2.

Problems, usage, documentation and hints
Some mocking frameworks:

- Python 3.x unittest.mock: https://docs.python.org/3/library/unittest.mock.html
- Moq framework for C# Mocking: https://github.com/moq
- NSubstitute for C# Mocking: https://github.com/nsubstitute/nsubstitute
- Rhino Mocks for C#: https://hibernatingrhinos.com/oss/rhino-mocks

Remember that since you are mocking the apps functionality the actual code do **NOT** need to exist. But it can be very helpful to see how the missing code works in order to mock it correctly.

To clarify you can mock read/write to file from disk or mock read/write to a database. You can also mock network access in some form, both reading and writing via REST and JSON

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

for example or making other library/module calls. If your app is threaded or running in parallel you can mock that as well.

Try to follow the SOLID principles and the Dependency Injection (DI) pattern which is related to DIP (Dependency Inversion Principle) and IoC (Inversion of Control).
See the lecture example code, slides and presentation links for more info. Good resources for this is: the DevIQ reference for high-level software development topics at: https://deviq.com/ or https://www.tutorialsteacher.com/ioc

3.1 Mocking in Python with unittest.mock

Report:
Hand in the full source code of your solution. Report your code coverage and Cyclomatic Complexity Metrics (CCM, MT_05_CCM_theory.pdf from the lecture) which is attached. Cover the task requirements.

Task requirements:

- Implement a solution which mocks the functionality of at least two of the four first rules in the introduction sections list.
- Write an appropriate amount of unit tests for the functionality of the app when parts of it are mocked.
- Try to follow the SOLID principles and the Dependency Injection pattern.

Further info
You can create an application yourself from scratch, extend a lab or project from this or earlier courses or use some of the attached examples as an idea to move further on. You should be familiar with using files, databases, network etc. from earlier courses in the program.
If you want you can look into the Dependency Injection framework for Python at: https://pypi.org/project/dependency-injector/ but it may be to complicated and extensive to learn and use in this lab.

3.2 Mocking in C# with suitable framework

Report:
Hand in the full source code of your solution. Report your code coverage and Cyclomatic Complexity Metrics (CCM, MT_05_CCM_theory.pdf from the lecture) which is attached. Cover the task requirements.

Task requirements:

- Use .NET 5.x or newer framework.
- Implement a solution which mocks the functionality of at least two of the four first rules in the introduction sections list.
- Write an appropriate amount of unit tests for the functionality of the app when parts of it are mocked.
- Some kind of logging to file must be implemented, since it is a must to find and locate errors in an app when it has shipped to the customer or users.

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/

- Using app configuration via App.config makes running advanced configurations and updating them during run-time easier: https://docs.microsoft.com/en-us/dotnet/api/system.configuration.configurationmanager.appsettings
- Try to follow the SOLID principles and the Dependency Injection pattern.

Further info
You can create an application yourself from scratch, extend a lab or project from this or earlier courses, project from github or use some of the attached examples as an idea to move further on. You should be familiar with using files, databases, network etc. from earlier courses in the program.

Examples how to use asynctask, logging and much more etc. is found here: http://users.du.se/~hjo/cs/gik2f7/ > in the week number folders.

How to put logging into your app and read from App.config is demonstrated in the NoobChain1 and in some of the sample programs from above link.

Check out this article regarding logging: NLog vs log4net vs Serilog: Compare .NET Logging Frameworks: https://stackify.com/nlog-vs-log4net-vs-serilog/

Remember that when using .NET many MS API:s must be imported and used via NuGet packages. This is the case using App.config with System.Configuration.ConfigurationManager as well: https://www.nuget.org/packages/System.Configuration.ConfigurationManager/

Visual Studio EE also have tools for calculating code metrics via the menu: Analyze > Calculate Code Metrics. More info here:

- https://docs.microsoft.com/en-us/visualstudio/code-quality/how-to-generate-code-metrics-data
- https://docs.microsoft.com/en-us/visualstudio/code-quality/code-metrics-values

Try to use the Dependency Injection pattern from the C# Mocking lecture with some DI IoC container framework as

- Castle Windsor: http://www.castleproject.org/
- Autofac: https://autofac.org/
- Other DI IoC container frameworks: https://www.claudiobernasconi.ch/2019/01/24/the-ultimate-list-of-net-dependency-injection-frameworks/

3.3 Lab feedback

a) Were the labs relevant and appropriate and what about length etc.?
b) What corrections and/or improvements do you suggest for these labs?

Högskolan Dalarna
Röda vägen 3
781 88 BORLÄNGE
Telefon: 023-77 80 00
Telefax: 023-77 80 50
URL: http://www.du.se/