What's the difference between a mock & stub?

Asked 9 years, 4 months ago Active 1 month ago Viewed 307k times

895

442

I've read various articles about mocking vs stubbing in testing, including Martin Fowler's Mocks Aren't Stubs, but still don't understand the difference.

testing mocking stub

edited Mar 24 '16 at 17:25
MattSayar
1,968 6 22 28

asked Aug 11 '10 at 14:19
never_had_a_name
72k 92 248 368

4 possible duplicate of An overview of unit testing terminology (stub vs mock , integration vs. interaction)?, whats-the-difference-between-faking-mocking-and-stubbing – nawfal Jul 23 '14 at 17:50

61 @OP Because there is no difference. This article, as much as loved by the community, is - with all due respect - making everything unnecessary confusing by adding additional meaning to words that are easy to understand otherwise and by making things unnecessary complicated. Mock is just a mock, something that runs fake business logic instead of real one. Checking for behavior in the end is your choice, but it is still a mock. Or whatever you want to call it, but make it ONE. Do not split a hairs. Keep it simple, so people can understand your concept easily - which above article does fail with. – wst Jun 29 '16 at 2:30

8 "Classification between mocks, fakes, and stubs is highly inconsistent across the literature." With many citations. Still one of my favorite Wikipedia quotes - if such a thing exists :) en.wikipedia.org/wiki/Mock_object – JD. Sep 22 '16 at 15:47

7 that Martin Fowler's article is really hard to understand for beginners. – lmiguelvargasf Nov 24 '16 at 15:03

1 The way i understand it is that a stub would just be a throw away object for your test, like a collection of dummy data. A Mock would be a cleverly overridden version of something more complex, like a service layer with various methods, which you might have changed the behavior of, for your tests. The two things are used together, like you could pass some stubbed objects into your mocked layer. – JsonStatham Mar 5 at 13:30

36 Answers

1 2 next

Stub

697

I believe the biggest distinction is that a stub you have already written with predetermined behavior. So you would have a class that implements the dependency (abstract class or interface most likely) you are faking for testing purposes and the methods would just be stubbed out with set responses. They would not do anything fancy and you would have already written the stubbed code for it outside of your test.

Mock

A mock is something that as part of your test you have to setup with your expectations. A mock is not setup in a predetermined way so you have code that does it in your test. Mocks in a way are determined at runtime since the code that sets the expectations has to run before they do anything.

Difference between Mocks and Stubs

Tests written with mocks usually follow an `initialize -> set expectations -> exercise -> verify` pattern to testing. While the pre-written stub would follow an `initialize -> exercise -> verify .`

Similarity between Mocks and Stubs

The purpose of both is to eliminate testing all the dependencies of a class or function so your tests are more focused and simpler in what they are trying to prove.

edited Apr 3 at 20:08

Harris
5,859 1 42 44

answered Aug 11 '10 at 14:38

Sean Copenhaver
8,161 1 14 15

816

+500

Foreword

There are several definitions of objects, that are not real. The general term is **test double**. This term encompasses: **dummy**, **fake**, **stub**, **mock**.

Reference

According to Martin Fowler's article:

> - **Dummy** objects are passed around but never actually used. Usually they are just used to fill parameter lists.
> - **Fake** objects actually have working implementations, but usually take some shortcut which makes them not suitable for production (an in memory database is a good example).

> - **Stubs** provide canned answers to calls made during the test, usually not responding at all to anything outside what's programmed in for the test. Stubs may also record information about calls, such as an email gateway stub that remembers the messages it 'sent', or maybe only how many messages it 'sent'.
> - **Mocks** are what we are talking about here: objects pre-programmed with expectations which form a specification of the calls they are expected to receive.

Style

Mocks vs Stubs = Behavioral testing vs State testing

Principle

According to the principle of *Test only one thing per test*, there may be several stubs in one test, but generally there is only one mock.

Lifecycle

Test lifecycle with stubs:

1. Setup - Prepare object that is being tested and its stubs collaborators.
2. Exercise - Test the functionality.
3. Verify state - Use asserts to check object's state.
4. Teardown - Clean up resources.

Test lifecycle with mocks:

1. Setup data - Prepare object that is being tested.
2. **Setup expectations** - Prepare expectations in mock that is being used by primary object.
3. Exercise - Test the functionality.
4. **Verify expectations** - Verify that correct methods has been invoked in mock.
5. Verify state - Use asserts to check object's state.
6. Teardown - Clean up resources.

Summary

Both mocks and stubs testing give an answer for the question: ***What is the result?***

Testing with mocks are also interested in: ***How the result has been achieved?***

edited Sep 17 '13 at 7:35

answered Jul 23 '13 at 12:18

Ryszard Dżegan
19.4k 6 27 49

328

A stub is a simple fake object. It just makes sure test runs smoothly.
A mock is a smarter stub. You verify your test passes through it.

edited May 27 at 12:08

Daniel Holmes
1,264 1 7 21

answered Aug 11 '10 at 14:33

Arnis Lapsa
35.2k 26 106 186

27 I think this is the most succinct and spot on answer. Takeaway: a mock IS-A stub. stackoverflow.com/a/17810004/2288628 is the longer version of this answer. – PoweredByRice Oct 6 '14 at 0:12

7 I don't think a mock is a stub. Mocks are used to assert and should never return data, stubs are used to return data and should never assert. – dave1010 Aug 3 '15 at 14:00

1 @dave1010 Mocks most definitely can return data or even throw an exception. They should do so in response to the params passed into them. – Trenton Aug 17 '15 at 18:33

2 @trenton if an object returns or throws based on data passed in then it's a *fake*, not a mock. Stubs test how your SUT handles *receiving* messages, mocks test how your SUT *sends* messages. Mixing up the 2 is likely to lead to bad OO design. – dave1010 Aug 24 '15 at 7:41

6 I think this is great - a stub returns answers to questions. A mock also returns answers to questions (is-a stub) but it also verifies that the question was asked !! – Leif Jan 15 '17 at 1:53

225

Here's a description of each one followed by with real world sample.

- **Dummy** - just bogus values to satisfy the `API` .

  > *Example*: If you're testing a method of a class which requires many mandatory parameters in a constructor which *have no effect* on your test, then you may create dummy objects for the purpose of creating new instances of a class.

- **Fake** - create a test implementation of a class which may have a dependency on some external infrastructure. (It's good practice that your unit test does **NOT** actually interact with external infrastructure.)

> *Example*: Create fake implementation for accessing a database, replace it with `in-memory` collection.

- **Stub** - override methods to return hard-coded values, also referred to as `state-based` .

  > *Example*: Your test class depends on a method `Calculate()` taking 5 minutes to complete. Rather than wait for 5 minutes you can replace its real implementation with stub that returns hard-coded values; taking only a small fraction of the time.

- **Mock** - very similar to `Stub` but `interaction-based` rather than state-based. This means you don't expect from `Mock` to return some value, but to assume that specific order of method calls are made.

  > Example: You're testing a user registration class. After calling `Save` , it should call `SendConfirmationEmail` .

`Stubs` and `Mocks` are actually sub types of `Mock` , both swap real implementation with test implementation, but for different, specific reasons.

edited Apr 29 '15 at 22:36

Rafael Espinoza
94 2 10

answered Nov 26 '14 at 14:12

Lev
2,326 5 33 49

168

In the codeschool.com course, Rails Testing for Zombies, they give this definition of the terms:

Stub

> For replacing a method with code that returns a specified result.

Mock

> A stub with an assertion that the method gets called.

So as Sean Copenhaver described in his answer, the difference is that mocks set expectations (i.e. make assertions, about whether or how they get called).

answered Jul 4 '12 at 22:32

Dillon Kearns
3,175 2 15 8

133

Stubs don't fail your tests, mock can.

answered Apr 18 '12 at 20:05

mk_
2,347 3 22 29

2 And I think this is good, you know if tests have the same behavior after refactoring. – RodriKing May 23 '18 at 11:25

1 @RodriKing I have same feeling. As with Mock, with any changes in production code - you have corresponding changes to test code. Which is pain! With Stubs, It feels like you keep testing the behavior so no micro changes need to made with test code. – tucq88 Jul 21 at 3:13

33

I think the simplest and clearer answer about this question is given from **Roy Osherove** in his book **The art of Unit Testing** (page 85)

> The easiest way to tell we're dealing with a stub is to notice that the stub can never fail the test. The asserts the test uses are always against the class under test.
>
> On the other hand, the test will use a mock object to verify whether the test failed or not. [...]
>
> Again, the mock object is the object we use to see if the test failed or not.

That means if you are making assertions against the fake it means you are using the fake as a mock, if you are using the fake only to run the test without assertion over it you are using the fake as a stub.

edited Oct 16 '16 at 5:34

olibre
34.2k 21 130 170

answered Aug 24 '15 at 14:06

Ghini Antonio
1,881 17 30

2 I wish your answer would find its way to the top. Here's R. Osherove explaining this youtu.be/fAb_OnooCsQ?t=1006. – Michael Ekoka Jan 5 '18 at 20:54

Reading all the explanations above, let me try to condense:

28

- **Stub**: a dummy piece of code that lets the test run, but you don't care what happens to it.
- **Mock**: a dummy piece of code, that you VERIFY is called correctly as part of the test.
- **Spy**: a dummy piece of code, that intercepts some calls to a real piece of code, allowing you to verify calls without replacing the entire original object.

edited May 7 '18 at 1:57

answered Nov 25 '14 at 20:59

O'Rooney

1,279 16 31

3 Good answer. Mock sounds quite similar to Spy though, based on your definition. Would be nice if you updated your answer to include a few more test doubles. – Rowan Gontier May 4 '18 at 9:02

22

A Mock is just testing behaviour, making sure certain methods are called. A Stub is a testable version (per se) of a particular object.

What do you mean an Apple way?

edited May 10 '18 at 21:08

roufamatic

16.2k 6 51 80

answered Aug 11 '10 at 14:30

NebulaFox

5,052 5 37 55

18 "What do you mean an Apple way?" Use Helvetica – kubi Aug 11 '10 at 14:35

6 In an Apple way as opposed to in a Microsoft way :) – never_had_a_name Aug 11 '10 at 14:40

2 Does this help the situation any? – NebulaFox Aug 11 '10 at 22:07

20

If you compare it to debugging:

> **Stub** is like making sure a method returns the correct value
>
> **Mock** is like actually **stepping into the method** and making sure everything inside is correct before returning the correct value.

answered Nov 8 '13 at 13:29

happygilmore


2,410 3 17 32

19

I think the most important difference between them is their intentions.

Let me try to explain it in **WHY stub** vs. **WHY mock**

Suppose I'm writing test code for my mac twitter client's public timeline controller

Here is test sample code

```
twitter_api.stub(:public_timeline).and_return(public_timeline_array)
client_ui.should_receive(:insert_timeline_above).with(public_timeline_array)
controller.refresh_public_timeline
```

- STUB: The network connection to twitter API is very slow, which make my test slow. I know it will return timelines, so I made a stub simulating HTTP twitter API, so that my test will run it very fast, and I can running the test even I'm offline.
- MOCK: I haven't written any of my UI methods yet, and I'm not sure what methods I need to write for my ui object. I hope to know how my controller will collaborate with my ui object by writing the test code.

By writing mock, you discover the objects collaboration relationship by verifying the expectation are met, while stub only simulate the object's behavior.

I suggest to read this article if you're trying to know more about mocks: http://jmock.org/oopsla2004.pdf

edited Nov 14 '13 at 17:10

Adam Parkin
11.7k 11 49 76

answered Jul 8 '12 at 8:37

Joe Yang
1,243 12 10

1 I think you have the right idea, but Dillon Kearns explained it a lot more clearly. – O'Rooney Nov 25 '14 at 20:56

17

To be very clear and practical:

Stub: A class or object that implements the methods of the class/object to be faked and returns always what you want.

Example in JavaScript:

```
var Stub = {
   method_a: function(param_a, param_b){
      return 'This is an static result';
   }
}
```

Mock: The same of stub, but it adds some logic that "verifies" when a method is called so you can be sure some implementation is calling that method.

As @mLevan says imagine as an example that you're testing a user registration class. After calling Save, it should call SendConfirmationEmail.

A very stupid code Example:

```
var Mock = {
   calls: {
      method_a: 0
   }

   method_a: function(param_a, param_b){
     this.method_a++;
     console.log('Mock.method_a its been called!');
   }
}
```

edited Aug 20 '15 at 12:08

answered Jun 25 '15 at 16:56

R01010010

3,564 6 35 60

17

Using a **mental model** really helped me understand this, rather than all of the explanations and articles, that didn't quite "sink in".

Imagine your kid has a glass plate on the table and he starts playing with it. Now, you're afraid it will break. So, you give him a plastic plate instead. That would be a **Mock** (same behavior, same interface, "softer" implementation).

Now, say you don't have the plastic replacement, so you explain "If you continue playing with it, it will break!". That's a **Stub**, you provided a predefined state in advance.

A **Dummy** would be the fork he didn't even use... and a **Spy** could be something like providing the same explanation you already used that worked.

edited Apr 3 at 18:39

Harris

5,859 1 42 44

answered Nov 7 '18 at 11:56

Moshisho

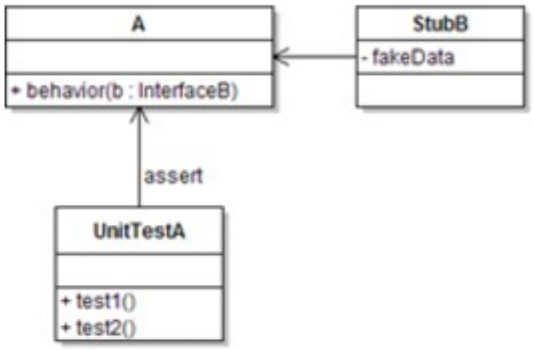
1,959 1 14 31

14

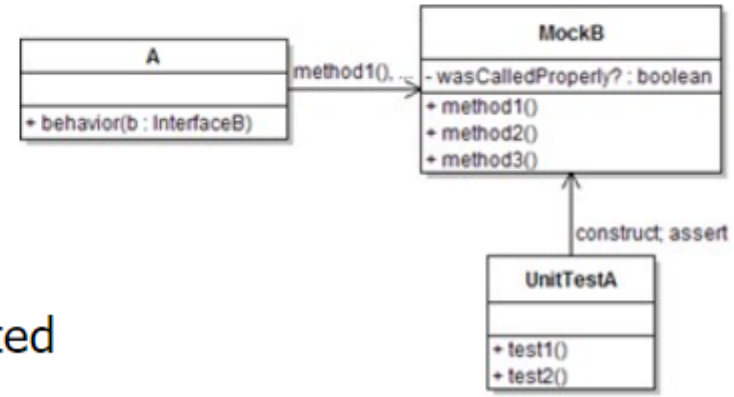
This slide explain the main differences very good.

Stubs vs. mocks

- A **stub** gives out data that goes to the object/class under test.
- The unit test directly asserts against class under test, to make sure it gives the right result when fed this data.



- A **mock** waits to be called by the class under test (A).
  - Maybe it has several methods it expects that A should call.
- It makes sure that it was contacted in exactly the right way.
  - If A interacts with B the way it should, the test passes.



21

*From CSE 403 Lecture 16 , University of Washington (slide created by "Marty Stepp")

answered Jun 5 '17 at 12:19

Aviram Fireberger

2,372 3 32 50

11

I like the explanantion put out by Roy Osherove [video link].

> Every class or object created is a Fake. It is a Mock if you verify calls against it. Otherwise its a stub.

answered Feb 20 '16 at 6:59

nitishagar
6,552 3 18 33

11

- Stubs vs. Mocks
  - Stubs
    1. provide specific answers to methods calls
       - ex: myStubbedService.getValues() just return a String needed by the code under test
    2. used by code under test to isolate it
    3. cannot fail test
       - ex: myStubbedService.getValues() just returns the stubbed value
    4. often implement abstract methods
  - Mocks
    1. "superset" of stubs; can assert that certain methods are called
       - ex: verify that myMockedService.getValues() is called only once
    2. used to test behaviour of code under test
    3. can fail test
       - ex: verify that myMockedService.getValues() was called once; verification fails, because myMockedService.getValues() was not called by my tested code
    4. often mocks interfaces

edited Jul 2 '18 at 12:35

answered Feb 19 '18 at 23:18

Relu Mesaros
3,166 2 16 30

9

A **fake** is a generic term that can be used to describe either a stub or a mock object (handwritten or otherwise), because they both look like the real object.

Whether a fake is a stub or a mock depends on how it's used in the current test. If it's used to check an interaction (asserted against), it's a mock object. Otherwise, it's a stub.

Fakes makes sure test runs smoothly. It means that reader of your future test will understand what will be the behavior of the fake object, without needing to read its source code (without needing to depend on external resource).

What does test run smoothly mean?
Forexample in below code:

```
public void Analyze(string filename)
       {
           if(filename.Length<8)
           {
               try
               {
                   errorService.LogError("long file entered named:" + filename);
               }
               catch (Exception e)
               {
                   mailService.SendEMail("admin@hotmail.com", "ErrorOnWebService",
"someerror");
               }
           }
       }
```

You want to test *mailService.SendEMail()* method, to do that you need to simulate an Exception in you test method, so you just need to create a Fake Stub errorService class to simulate that result, then your test code will be able to test mailService.SendEMail() method. As you see you need to simulate a result which is from an another External Dependency ErrorService class.

edited Jan 13 '16 at 1:25

answered Apr 8 '14 at 18:13

Mustafa Ekici
6,162 6 46 67

let see Test Doubles:

0

- **Fake**: Fakes are objects that have working implementations, but not the same as production one. **Such as**: in-memory implementation of Data Access Object or Repository.
- **Stub**: Stub is an object that holds predefined data and uses it to answer calls during tests. **Such as**: an object that needs to grab some data from the database to respond to a method call.
- **Mocks**: Mocks are objects that register calls they receive. In test assertion, we can verify on Mocks that all expected actions were performed. **Such as**: a functionality that calls e-mail sending service. for more just check this.

edited Aug 24 '18 at 6:12

answered Aug 24 '18 at 6:05

Alireza Rahmani Khalili
903 2 14 20

7

Right from the paper Mock Roles, not Objects, by the developers of jMock :

> Stubs are dummy implementations of production code that return canned results. Mock Objects act as stubs, but also include assertions to instrument the interactions of the target object with its neighbours.

So, the main differences are:

- expectations set on stubs are usually generic, while expectations set on mocks can be more "clever" (e.g. return this on the first call, this on the second etc.).
- **stubs** are mainly used to **setup indirect inputs of the SUT**, while **mocks** can be used to test **both indirect inputs and indirect** outputs of the SUT.

To sum up, while also trying to disperse the confusion from Fowler's article title: **mocks are stubs, but they are not only stubs**.

edited Sep 7 '17 at 19:01

answered Jan 15 '16 at 22:00

Dimos
5,767 22 32

1 i think you're right, but this is why the Fowler article is confusing, the article title is "Mocks Aren't Stubs"...but they ARE?! ¯_(ツ)_/¯ – stonedauwg Dec 7 '16 at 16:14

I came across this interesting article by UncleBob The Little Mocker. It explains all the terminology in a very easy to understand manner,

5

so its useful for beginners. Martin Fowlers article is a hard read especially for beginners like me.

edited Jan 22 '17 at 13:50

answered Jan 22 '17 at 13:40

A.I

1,178 1 11 16

4

See below example of mocks vs stubs using C# and Moq framework. Moq doesn't have a special keyword for Stub but you can use Mock object to create stubs too.

```
namespace UnitTestProject2
{
    using Microsoft.VisualStudio.TestTools.UnitTesting;
    using Moq;
    [TestClass]
    public class UnitTest1
    {
        /// <summary>
        /// Test using Mock to Verify that GetNameWithPrefix method calls Repository
GetName method "once" when Id is greater than Zero
        /// </summary>
        [TestMethod]
        public void GetNameWithPrefix_IdIsTwelve_GetNameCalledOnce()
        {
            // Arrange
            var mockEntityRepository = new Mock<IEntityRepository>();
            mockEntityRepository.Setup(m => m.GetName(It.IsAny<int>()));

            var entity = new EntityClass(mockEntityRepository.Object);
            // Act
            var name = entity.GetNameWithPrefix(12);
            // Assert
            mockEntityRepository.Verify(m => m.GetName(It.IsAny<int>()), Times.Once);
        }
        /// <summary>
        /// Test using Mock to Verify that GetNameWithPrefix method doesn't call
Repository GetName method when Id is Zero
        /// </summary>
        [TestMethod]
        public void GetNameWithPrefix_IdIsZero_GetNameNeverCalled()
        {
            // Arrange
            var mockEntityRepository = new Mock<IEntityRepository>();
            mockEntityRepository.Setup(m => m.GetName(It.IsAny<int>()));
            var entity = new EntityClass(mockEntityRepository.Object);
```

```
            // Act
            var name = entity.GetNameWithPrefix(0);
            // Assert
            mockEntityRepository.Verify(m => m.GetName(It.IsAny<int>()), Times.Never);
        }
        /// <summary>
        /// Test using Stub to Verify that GetNameWithPrefix method returns Name with a
Prefix
        /// </summary>
        [TestMethod]
        public void GetNameWithPrefix_IdIsTwelve_ReturnsNameWithPrefix()
        {
            // Arrange
            var stubEntityRepository = new Mock<IEntityRepository>();
            stubEntityRepository.Setup(m => m.GetName(It.IsAny<int>()))
                .Returns("Stub");
            const string EXPECTED_NAME_WITH_PREFIX = "Mr. Stub";
            var entity = new EntityClass(stubEntityRepository.Object);
            // Act
            var name = entity.GetNameWithPrefix(12);
            // Assert
            Assert.AreEqual(EXPECTED_NAME_WITH_PREFIX, name);
        }
    }
    public class EntityClass
    {
        private IEntityRepository _entityRepository;
        public EntityClass(IEntityRepository entityRepository)
        {
            this._entityRepository = entityRepository;
        }
        public string Name { get; set; }
        public string GetNameWithPrefix(int id)
        {
            string name = string.Empty;
            if (id > 0)
            {
                name = this._entityRepository.GetName(id);
            }
            return "Mr. " + name;
        }
    }
    public interface IEntityRepository
    {
        string GetName(int id);
    }
    public class EntityRepository:IEntityRepository
    {
        public string GetName(int id)
```

```
        {
            // Code to connect to DB and get name based on Id
            return "NameFromDb";
        }
    }
}
```

answered Jan 1 '14 at 16:51

Adarsh Shah
6,401 2 20 36

4

Stub and Mock testing point of view:

- **Stub** is dummy implementation done by user in **static** way mean i.e in Stub writing the implementation code. So it can not handle service definition and dynamic condition, Normally this is done in JUnit framework without using mocking framework.
- **Mock** is also dummy implementation but its implementation done **dynamic** way by using Mocking frameworks like Mockito. So we can handle condition and service definition as dynamic way i.e. mocks can be created dynamically from code at runtime. *So using mock we can implement Stubs dynamically.*

edited Dec 19 '17 at 4:28

answered Jun 29 '17 at 1:45

Premraj
41.8k 21 188 133

3

Stub helps us to run test. How? It gives values which helps to run test. These values are itself not real and we created these values just to run the test. For example we create a HashMap to give us values which are similar to values in database table. So instead of directly interacting with database we interact with Hashmap.

Mock is an fake object which runs the test. where we put assert.

answered Nov 30 '13 at 9:44

user965884
3,955 14 62 94

I was reading The Art of Unit Testing, and stumbled upon the following definition:

3

A ***fake*** is a generic term that can be used to describe either a stub or a mock object (handwritten or otherwise), because they both look like the real object. Whether a fake is a stub or a mock depends on how it's used in the current test. if it's used to check an interaction (asserted against), it's a ***mock object***. Otherwise, it's a ***stub***.

edited Nov 7 at 13:57

DdW
641 1 15 25

answered Jul 31 at 14:13

Afonso Matos
1,795 12 23

2

I have used python examples in my answer to illustrate the differences.

Stub - Stubbing is a software development technique used to implement methods of classes early in the development life-cycle. They are used commonly as placeholders for implementation of a known interface, where the interface is finalized or known but the implementation is not yet known or finalized. You begin with stubs, which simply means that you only write the definition of a function down and leave the actual code for later. The advantage is that you won't forget methods and you can continue to think about your design while seeing it in code. You can also have your stub return a static response so that the response can be used by other parts of your code immediately. Stub objects provide a valid response, but it's static no matter what input you pass in, you'll always get the same response:

```
class Foo(object):
    def bar1(self):
        pass

    def bar2(self):
        #or ...
        raise NotImplementedError

    def bar3(self):
        #or return dummy data
        return "Dummy Data"
```

Mock objects are used in mock test cases they validate that certain methods are called on those objects. Mock objects are simulated objects that mimic the behaviour of real objects in controlled ways. You typically creates a mock object to test the behaviour of some other object. Mocks let us simulate resources that are either unavailable or too unwieldy for unit testing.

mymodule.py:

```
import os
import os.path
```

```
def rm(filename):
    if os.path.isfile(filename):
        os.remove(filename)
```

test.py:

```
from mymodule import rm
import mock
import unittest

class RmTestCase(unittest.TestCase):
    @mock.patch('mymodule.os')
    def test_rm(self, mock_os):
        rm("any path")
        # test that rm called os.remove with the right parameters
        mock_os.remove.assert_called_with("any path")

if __name__ == '__main__':
    unittest.main()
```

This is a very basic example that just runs rm and asserts the parameter it was called with. You can use mock with objects not just functions as shown here, and you can also return a value so a mock object can be used to replace a stub for testing.

More on unittest.mock, note in python 2.x mock is not included in unittest but is a downloadable module that can be downloaded via pip (pip install mock).

I have also read "The Art of Unit Testing" by Roy Osherove and I think it would be great if a similar book was written using Python and Python examples. If anyone knows of such a book please do share. Cheers :)

answered Dec 13 '14 at 19:35

radtek
19.6k 7 101 83

2

A stub is a fake object built for test purposes. A mock is a stub that records whether expected calls effectively occurred.

answered Dec 21 '16 at 18:56

simon.denel
679 6 20

2

A stub is an empty function which is used to avoid unhandled exceptions during tests:

```
function foo(){}
```

A mock is an artificial function which is used to avoid OS, environment or hardware dependencies during tests:

```
function foo(bar){ window = this; return window.toString(bar); }
```

In terms of assertions and state:

- Mocks are asserted before an event or state change
- Stubs are not asserted, they provide state before an event to avoid executing code from unrelated units
- Spies are setup like stubs, then asserted after an event or state change
- Fakes are not asserted, they run after an event with hardcoded dependencies to avoid state

References

- Geek Glossary: Mock
- Geek Glossary: Stub
- Geek Glossary: Spy
- Test Doubles: Fakes, Mocks and Stubs

edited Sep 12 '17 at 15:19

answered Jun 23 '15 at 15:40

Paul Sweatte

21.3k 6 106 224

2 +1 for adding spies to the glossary. Also, I think you mean "Spies are setup like mocks" not "Spies are setup like stubs" – Sameh Deabes Nov 23 '15 at 21:48

2

a lot of valid answers up there but I think worth to mention this form uncle bob: https://8thlight.com/blog/uncle-bob/2014/05/14/TheLittleMocker.html

the best explanation ever with examples!

answered Dec 18 '17 at 15:01

Kasper
427 5 21

1

A Stub is an object that implements an interface of a component, but instead of returning what the component would return when called, the stub can be configured to return a value that suits the test. Using stubs a unit test can test if a unit can handle various return values from its collaborator. Using a stub instead of a real collaborator in a unit test could be expressed like this:

unit test --> stub

unit test --> unit --> stub

unit test asserts on results and state of unit

First the unit test creates the stub and configures its return values. Then the unit test creates the unit and sets the stub on it. Now the unit test calls the unit which in turn calls the stub. Finally the unit test makes assertions about the results of the method calls on the unit.

A Mock is like a stub, only it also has methods that make it possible determine what methods where called on the Mock. Using a mock it is thus possible to both test if the unit can handle various return values correctly, and also if the unit uses the collaborator correctly. For instance, you cannot see by the value returned from a dao object whether the data was read from the database using a Statement or a PreparedStatement. Nor can you see if the connection.close() method was called before returning the value. This is possible with mocks. In other words, mocks makes it possible to test a units complete interaction with a collaborator. Not just the collaborator methods that return values used by the unit. Using a mock in a unit test could be expressed like this:

unit test --> mock

unit test --> unit --> mock

unit test asserts on result and state of unit

unit test asserts on the methods called on mock

More Detail >> Here

answered Jan 23 '15 at 8:00

M_ Fa
3,277 1 10 11

0

following is my understanding...

- if you create test objects locally and feed your local service with that, you are using mock object. this will give a test for the method you implemented in your local service. it is used to verify behaviors
- when you get the test data from the real service provider, though from a test version of interface and get a test version of the object, you are working with stubs the stub can have logic to accept certain input and give corresponding output to help you perform state verification...

answered Aug 11 '10 at 14:37

Andy Lin
555 2 4

1 2 next