Test Doubles

Doubling Down On TDD With Test Doubles

"When the movie industry wants to film something that is potentially risky or dangerous, they hire a *stunt double* … For testing purposes, we can replace the real depended-on component with our equivalent of the 'stunt double': the *Test Double*"

Gerard Meszaros

What Are Test Doubles?

- Test Doubles are any kind of pretend object used in place of a real object for testing purposes
- Enabled by the use of interfaces for dependent objects
- They do not implement the full business logic of the objects they are doubling
- They are used in place of real business object dependencies of the SUT to provide input data and verify behaviour

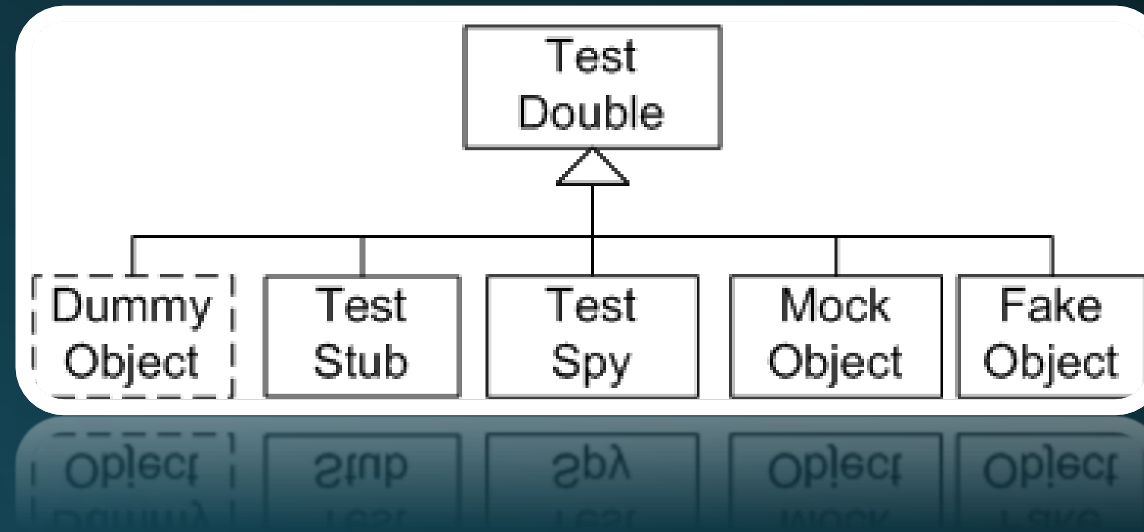
"Test doubles are indispensable tools for testing across architectural boundaries. Mocking is a good practice in general."

Robert Martin

Why Use Test Doubles?

- Dependencies of the SUT may not be suitable for use within a test environment
- They may be slow to execute
- May be expensive to use
- Might not always be available
- Can be difficult to test that the SUT has interacted correctly with the dependency
- There may be other undesirable side effects to using the real dependency

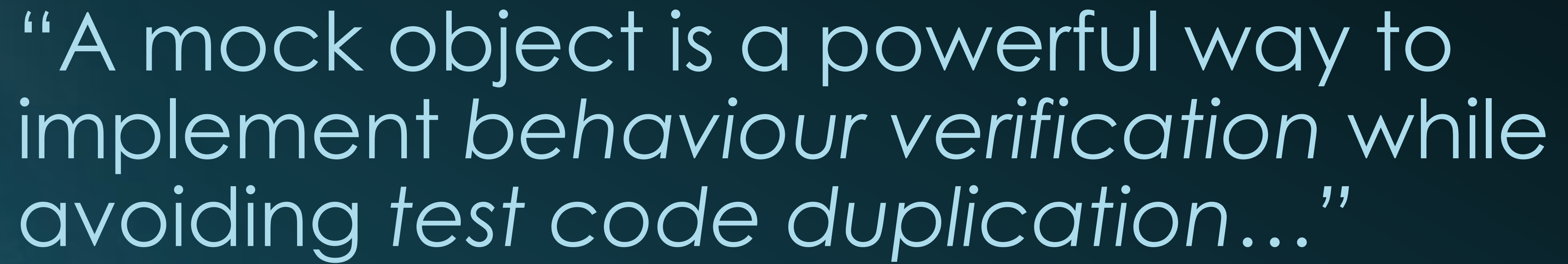
Types of Test Doubles



- Test Double is a generic term to describe several specific test object types: Dummy Object, Test Stub, Fake Object, Mock Object, Test Spy
- Terminology differs but the concepts are widely accepted

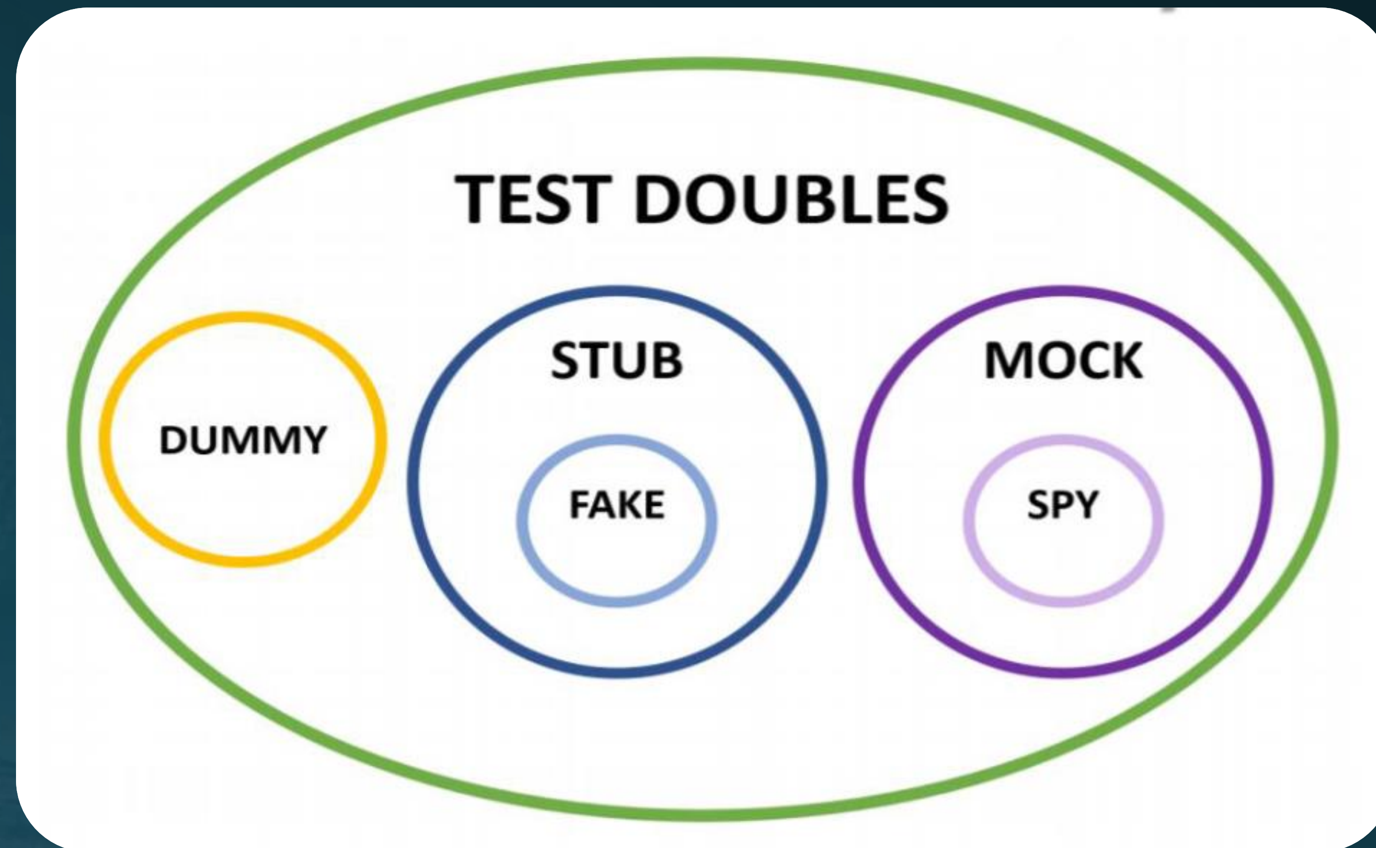
Types of Test Doubles

- A useful way to separate test double types is using the CQRS (Command Query Responsibility Segregation) principle:-
  - Queries return information about the current state but don't modify the state
  - Commands modify the state but don't return any information about the state
- Stubs are used in place of queries and return some pre-determined output
- Fakes are hand made stubs
- Mocks are used instead of commands and confirm that the commands have been triggered correctly
- Spies are hand make mocks
- Various libraries to help create stubs and mocks (NSubstitute, Moq, etc.)



Gerard Meszaros

Test Doubles Taxonomy



Example: Stub

```
public interface IGetUsers
{
    User[] GetAll();
}

//////////

[Test]
public void notify_expired_users()
{
    var userRepository = Substitute.For<IGetUsers>();
    userRepository.GetAll().Returns(new[] { new User("Test User") });
    var userService = new UserService(userRepository);

    userService.NotifyExpired();

    Assert.AreEqual(1, userService.NotifiedUsers());
}
```

Example: Fake

```
public interface IGetUsers
{
    User[] GetAll();
}


public class UserRepositoryFake : IGetUsers
{
    public User[] GetAll()
    {
        return new[] { new User("Test User") };
    }
}

[Test]
public void notify_expired_users()
{
    var userRepository = new UserRepositoryFake();
    var userService = new UserService(userRepository);

    userService.NotifyExpired();

    Assert.AreEqual(1, userService.NotifiedUsers());
}
```

Example: Mock

```
public interface IUserRepository
{
    User[] GetAll();
    void Update(User user);
}


[Test]
public void block_out_of_contract_users()
{
    var userRepository = Substitute.For<IUserRepository>();
    var user = new User(1, "Test User");
    var outOfContractUser = new User(2, "OOC User");
    outOfContractUser.OutOfContract();
    userRepository.GetAll().Returns(new[] { user, outOfContractUser });
    var userService = new UserService(userRepository);

    userService.BlockOutOfContractUsers();

    userRepository.Received(1).Update(new User(2, "OOC User", Status.Blocked));
}
```

Example: Spy

```
public class UserRepositorySpy : IUserRepository
{
    private List<User> updatedUsers = new List<User>();
    public User[] GetAll()
    {
        var outOfContractUser = new User(2, "OOC User");
        outOfContractUser.OutOfContract();
        return new[] { new User(1, "Test User"), outOfContractUser };
    }

    public void Update(User user)
    {
        updatedUsers.Add(user);
    }

    public List<User> SpyUpdatedUsers()
    {
        return updatedUsers;
    }
}

[Test]
public void block_out_of_contract_users()
{
    var userRepository = new UserRepositorySpy();
    var userService = new UserService(userRepository);

    userService.BlockOutOfContractUsers();

    Assert.AreEqual(new User(2, "OOC User", Status.Blocked), userRepository.SpyUpdatedUsers());
}
```

Best Practice

- Only use Test Doubles for your own objects
- Use a proxy/wrapper to call external services to allow mocks/stubs to be written
- Don’t use Test Doubles for isolated objects
- Only use Test Doubles for immediate neighbours or objects at the same abstraction level
- Don’t add complex behaviour in Test Doubles

Questions?

Alan Alford
alan.alford@fdbhealth.co.uk

References:

Mark Seemann, Mocks for Commands, Stubs for Queries
https://blog.ploeh.dk/2013/10/23/mocks-for-commands-stubs-for-queries/

Gerard Meszaros, xUnit Patterns
http://xunitpatterns.com/Test%20Double.html

Martin Fowler, Mocks Aren't Stubs
https://martinfowler.com/articles/mocksArentStubs.html

Robert Martin (Uncle Bob), The Little Mocker
https://blog.cleancoder.com/uncle-bob/2014/05/14/TheLittleMocker.html