



# ASP .NET Core 3.0 {پیشرفته}

 Visual Studio 2019

مدرس: علیرضا ارومند  
**nikamooz;**  
آموزش برنامه نویسی و اجرای پروژه

[www.NikAmooz.com](http://www.NikAmooz.com)

جلسه دوم

# معرفی علیرضا ارومند

۱. مدرس و مشاور ASP.NET Core و معماری‌های نرم‌افزاری (نیک آموز)
۲. مدیر فنی خبرگزاری نسیم
۳. کارشناس ارشد توسعه نرم افزار داتین (فناپ)
۴. کارشناس ارشد توسعه نرم افزار ارتباط فردا (بانک آینده)
۵. متخصص انجام پروژه‌های وب و .NET
۶. و ...



# xUnit

مدرس: علیرضا ارومند

**nikamooz;**  
آموزش برنامه نویسی  
آموزش برنامه نویسی و اجرای پروژه

# چه خواهیم آموخت؟

۱. چرا تست اتوماتیک؟
۲. چه تست‌هایی باید داشته باشیم؟
۳. آشنایی با هرم تست نرم افزار
۴. معرفی xUnit
۵. Mock چیست؟



# اهمیت کیفیت نرم افزار

۱. کاربران راضی

۲. توسعه دهندگان راضی تر

۳. روئسای خوشحال تر



# تست اتوماتیک یا دستی

تست دستی	تست اتوماتیک
خطای بیشتر	خطای کمتر
هزینه نیروی انسانی	هزینه توسعه و نگهداری
اجرای چندین باره کم هزینه	هزینه به ازای تمام اجراها یکسان
یافتن خطای سریع	یافتن خطا با تاخیر
نگهداری با سورس	وابسته به فرد

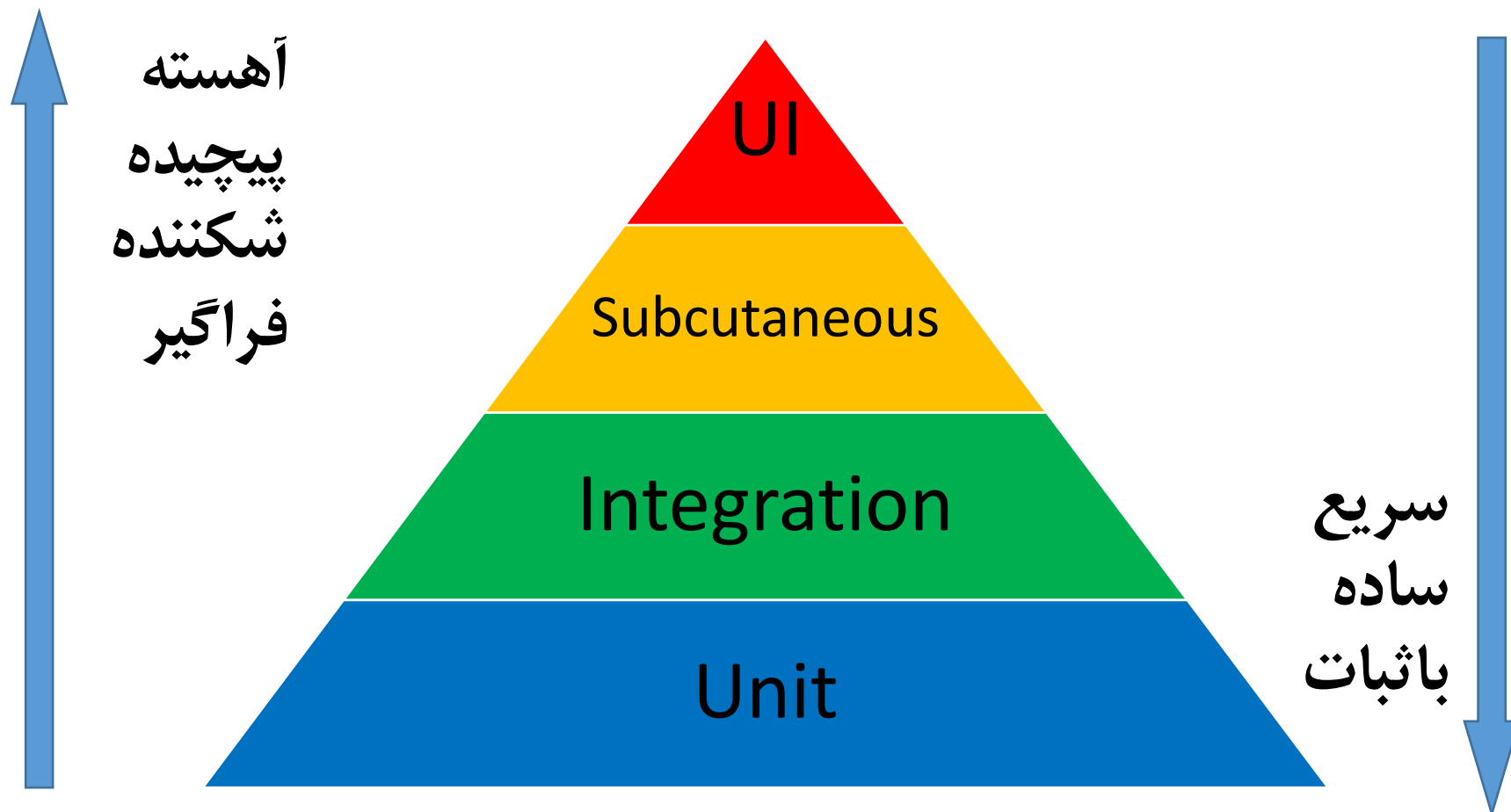
# انواع تست

UI

Subcutaneous

Integration

Unit Test





# کاربرد انواع تست

Controller

UUUUU

View Rendering

UU

Infrastructures

UU

Model

UUUUUUUU

Web API

SSSS

UI

UiUi

## نکته

۱. هر م تست تنها راهنمایی است
۲. کاربرد در اغلب موارد
۳. هر کاری باید ارزشمند باشد



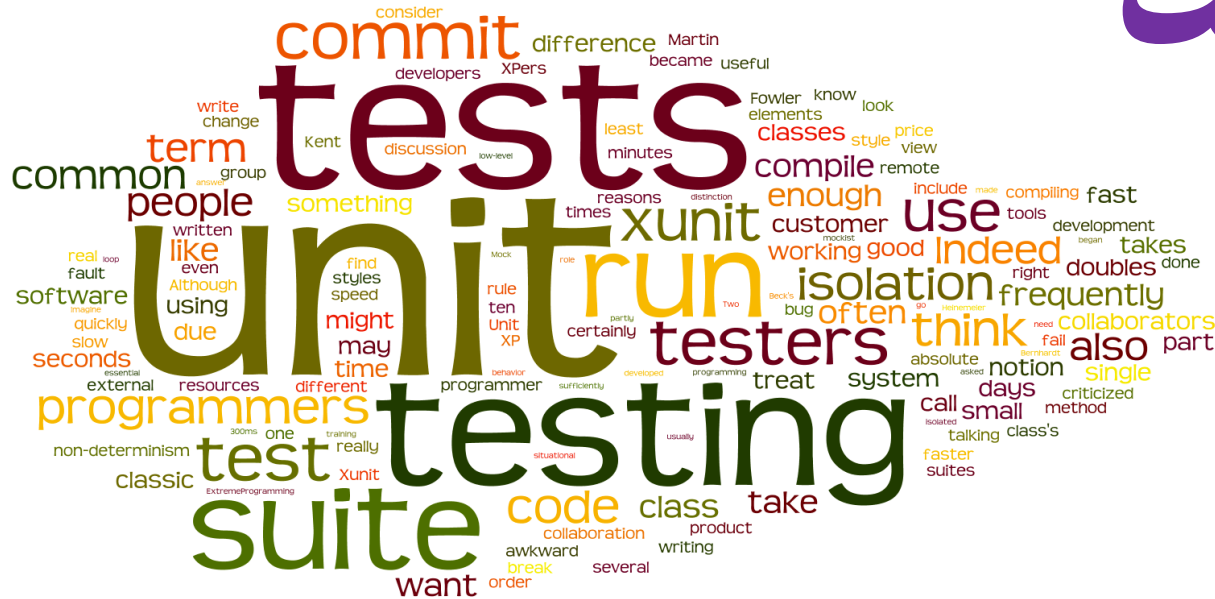
# AAA در تست نرم افزار

Arrange

Act

Assert

# چیت xUnit



# ۱. کتابخانه‌ی رایگان

# Open Source .۲

### ۳. توسعه توسط اعضای NUnit

# پشتیبانی از

.NET Core

.NET Full

UWP

Xamarin

.NET Standard

# مراحل ایجاد

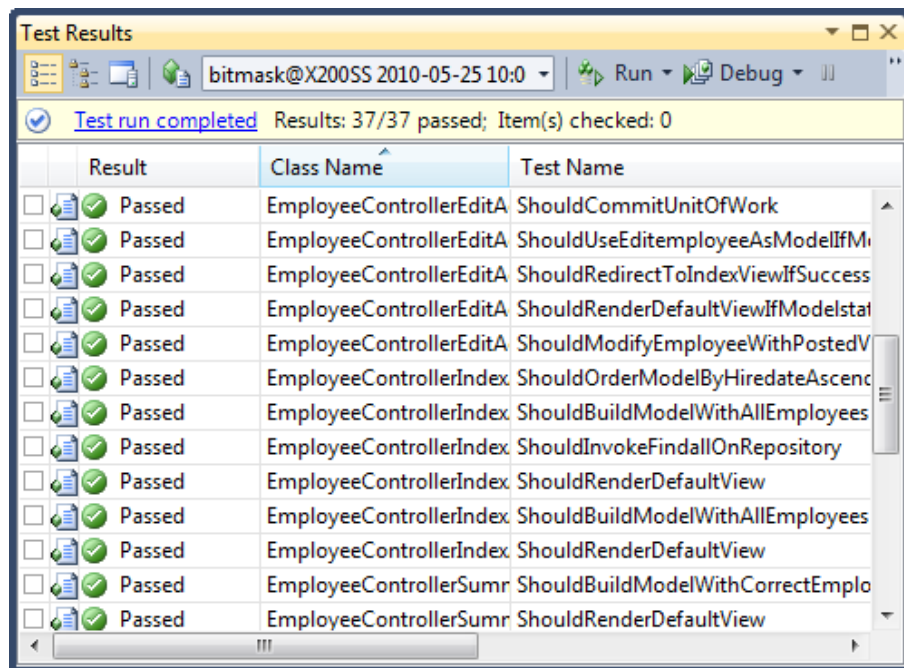
۱. ایجاد پروژه ای برای تست
۲. افزودن xUnit
۳. ایجاد رابطه با پروژه هدف



# مراحل اجرا

۱. نوشتن تابع تست

۲. اجرای تابع تست



The screenshot shows a 'Test Results' window from a testing framework. The status bar at the top indicates 'Test run completed' with 'Results: 37/37 passed; Item(s) checked: 0'. The main table lists the following test results:

Result	Class Name	Test Name
Passed	EmployeeControllerEditA	ShouldCommitUnitOfWork
Passed	EmployeeControllerEditA	ShouldUseEditEmployeeAsModelIfM...
Passed	EmployeeControllerEditA	ShouldRedirectToIndexViewIfSuccess
Passed	EmployeeControllerEditA	ShouldRenderDefaultViewIfModelstat
Passed	EmployeeControllerEditA	ShouldModifyEmployeeWithPostedV
Passed	EmployeeControllerIndex	ShouldOrderModelByHiredDateAscend
Passed	EmployeeControllerIndex	ShouldBuildModelWithAllEmployees
Passed	EmployeeControllerIndex	ShouldInvokeFindallOnRepository
Passed	EmployeeControllerIndex	ShouldRenderDefaultView
Passed	EmployeeControllerIndex	ShouldBuildModelWithAllEmployees
Passed	EmployeeControllerIndex	ShouldRenderDefaultView
Passed	EmployeeControllerSumr	ShouldBuildModelWithCorrectEmplo
Passed	EmployeeControllerSumr	ShouldRenderDefaultView

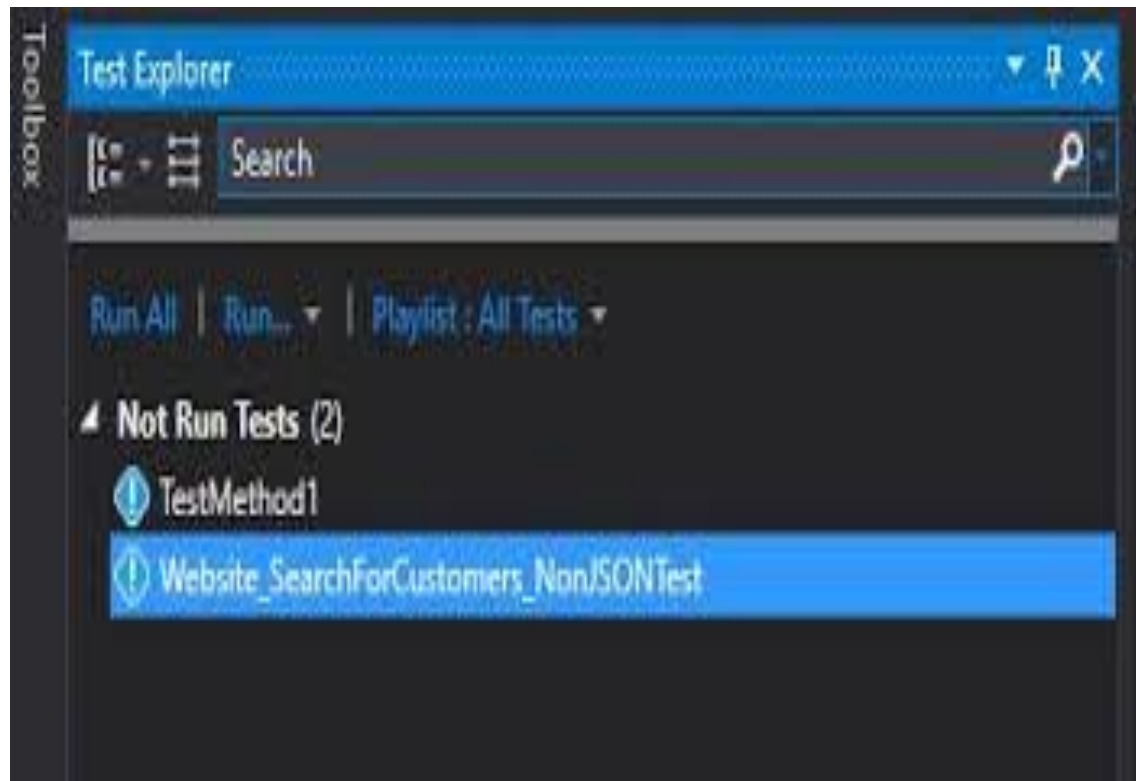
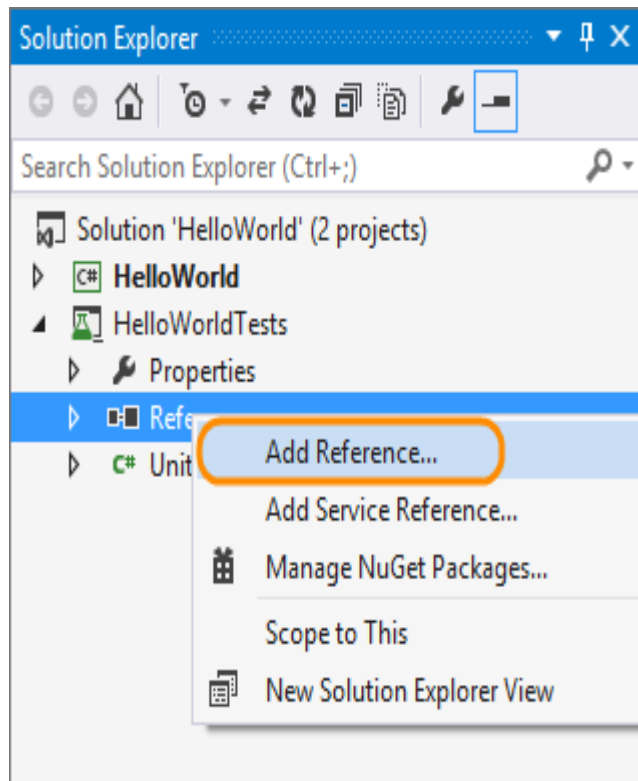
# معرفی Test Runner

۱. مسئول اجرای توابع تست
۲. وابسته به Test Framework
۳. Vs Test Explorer
۴. .NET Core CLI
۵. Third-Party





# بررسی ساختار پروژه و تست



# نام گذاری تست

۱. **MethodName\_StateUnderTest\_ExpectedBehavior**
۲. **MethodName\_ExpectedBehavior\_StateUnderTest**
۳. **test[Feature being tested]**
۴. **Should\_ExpectedBehavior\_When\_StateUnderTest**
۵. **When\_StateUnderTest\_Expect\_ExpectedBehavior**

# Assert و کاربرد آن

۱. نتیجه Act را بررسی می‌کند
۲. در صورت صحیح بودن همه تست پاس
۳. در صورت ناموفق بودن یکی کلا ناموفق





# انواع Assert

۱. بررسی مقادیر Boolean
۲. توابع مختلف کار با رشته‌ها
۳. انواع بررسی برای اعداد
۴. امکان کار با مجموعه‌ها
۵. Event ها

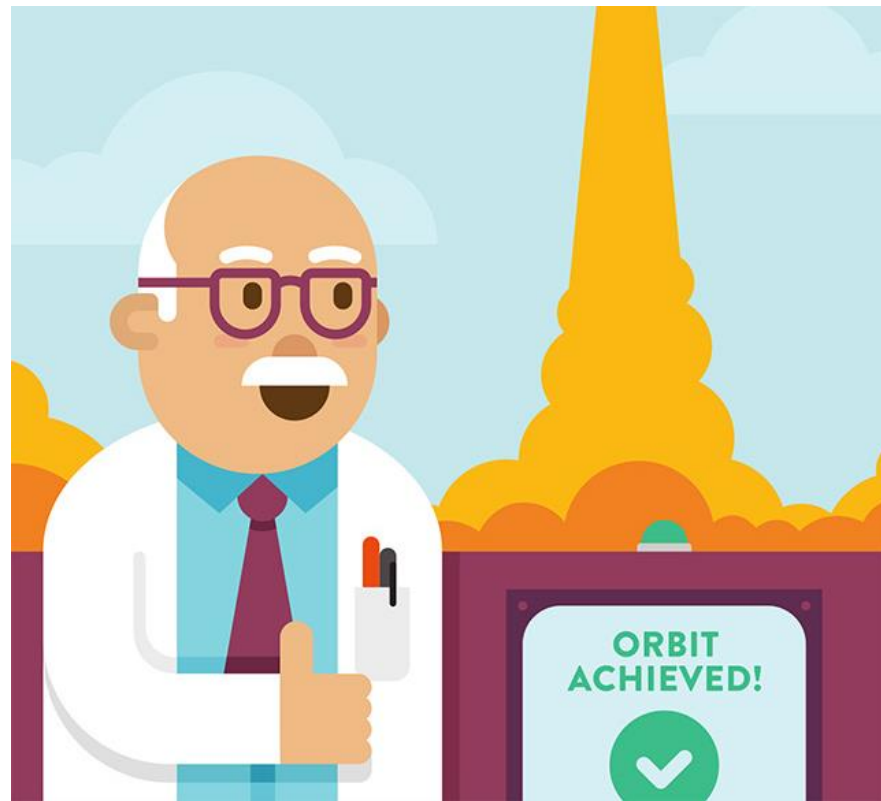
# چند Assert نیاز داریم؟



۱. حداقل یکی برای هر تست
۲. محدودیت نداریم
۳. پوشش تمام حالات ضروری

# Exception ها

۱. تمامی حالات خطا بررسی می شود.
۲. Throws بررسی خطای ارسالی



تعداد تست ها زیاد می شود

مدیریت کدها سخت تر می شود

اجرای تست ها زمانگیر

تست های ناکارآمد

# دسته بندی تست‌ها

۱. استفاده از Trait

۲. امکان اجرای دسته‌ای از تست‌ها

۳. می‌توان فیلتر انجام داد





# غیر فعال کردن تست



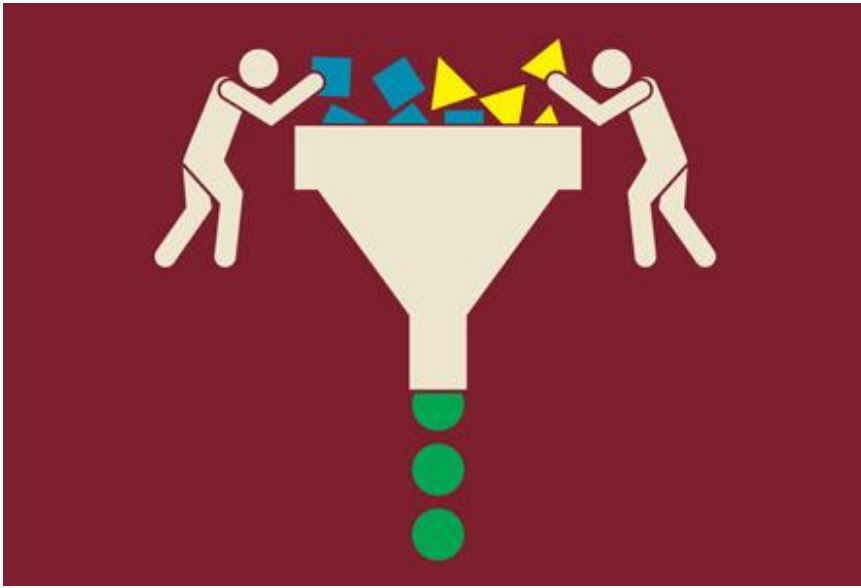
۱. مقداردهی به Skip

۲. غیر فعال سازی موقف

۳. عدم نیاز به Comment کردن

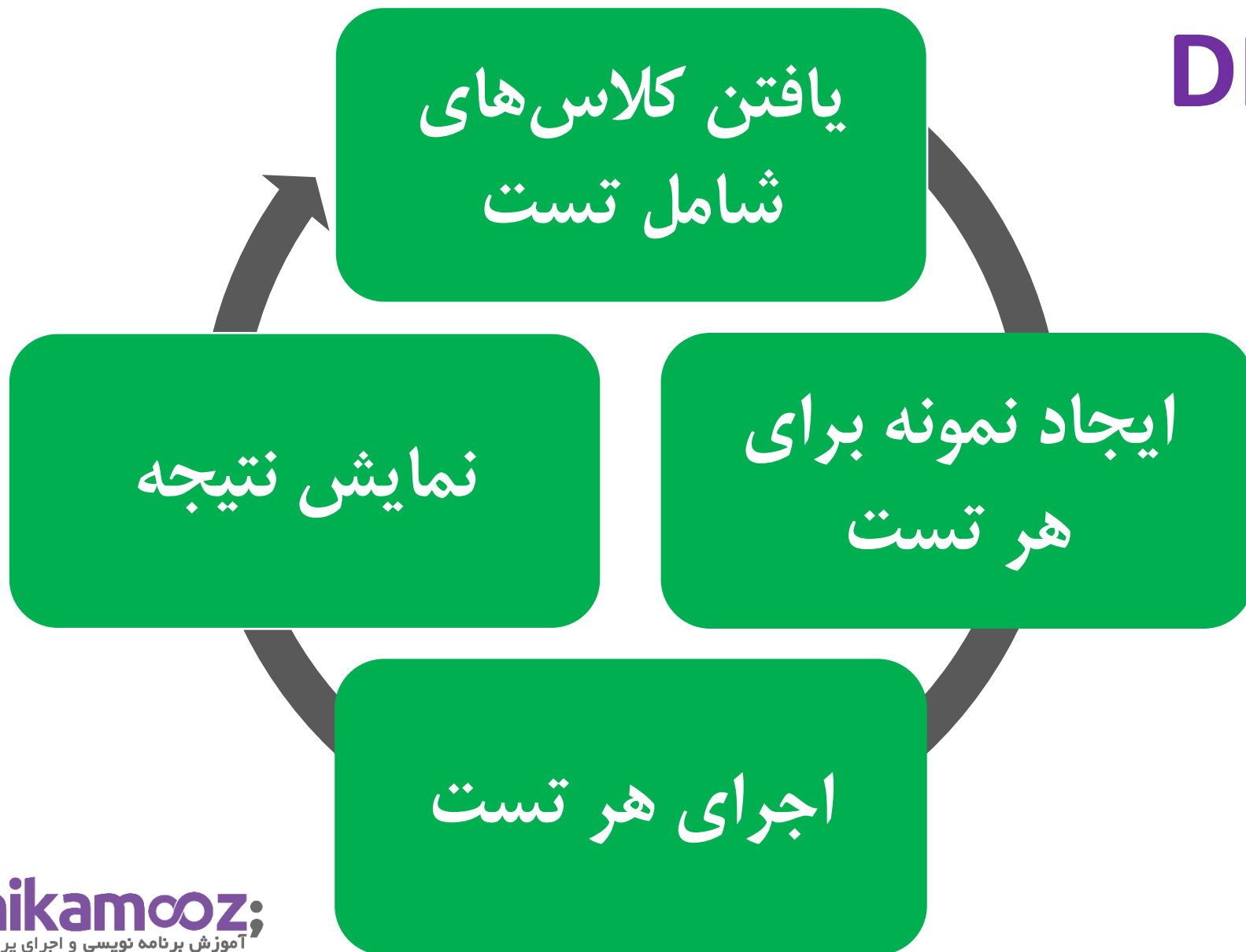
۴. مناسب برای شرایط موقف

# ایجاد خروجی



۱. خروجی‌های معمول کار نمی‌کند
۲. استفاده از `ITestOutputHelper`
۳. خروجی بعد از اجرا قابل مشاهده

**DRY**



# اشتراک منابع



۱. تولید و نگهداری منابع پرهزینه

۲. نیاز به اشتراک منابع بین تست‌ها

۳. استفاده از `IClassFixture`

۴. عدم تخریب تست با اشتراک

# اشتراک منابع بین کلاس‌ها

۱. ایجاد کلاس **CollectionFixture**

۲. تعیین نام مجموعه **CollectionDefenition**

۳. معرفی به کلاس‌ها با **Collection**



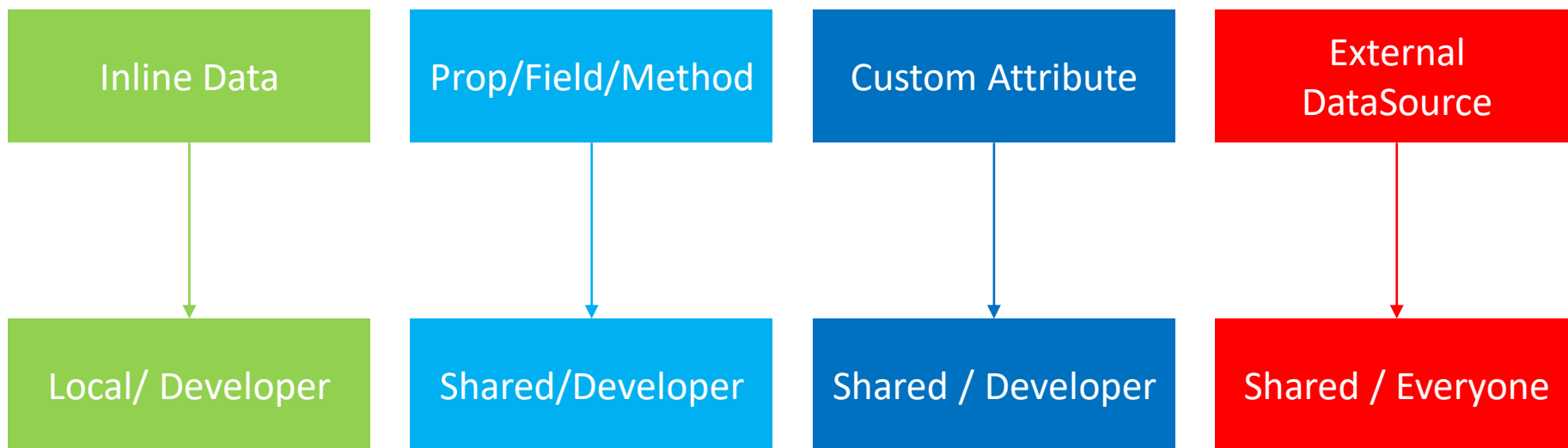
# داده‌های ورودی و تست

۱. اجرای سناریو تست برای داده‌های مختلف

۲. هزینه بالای نگهداری چندین تست



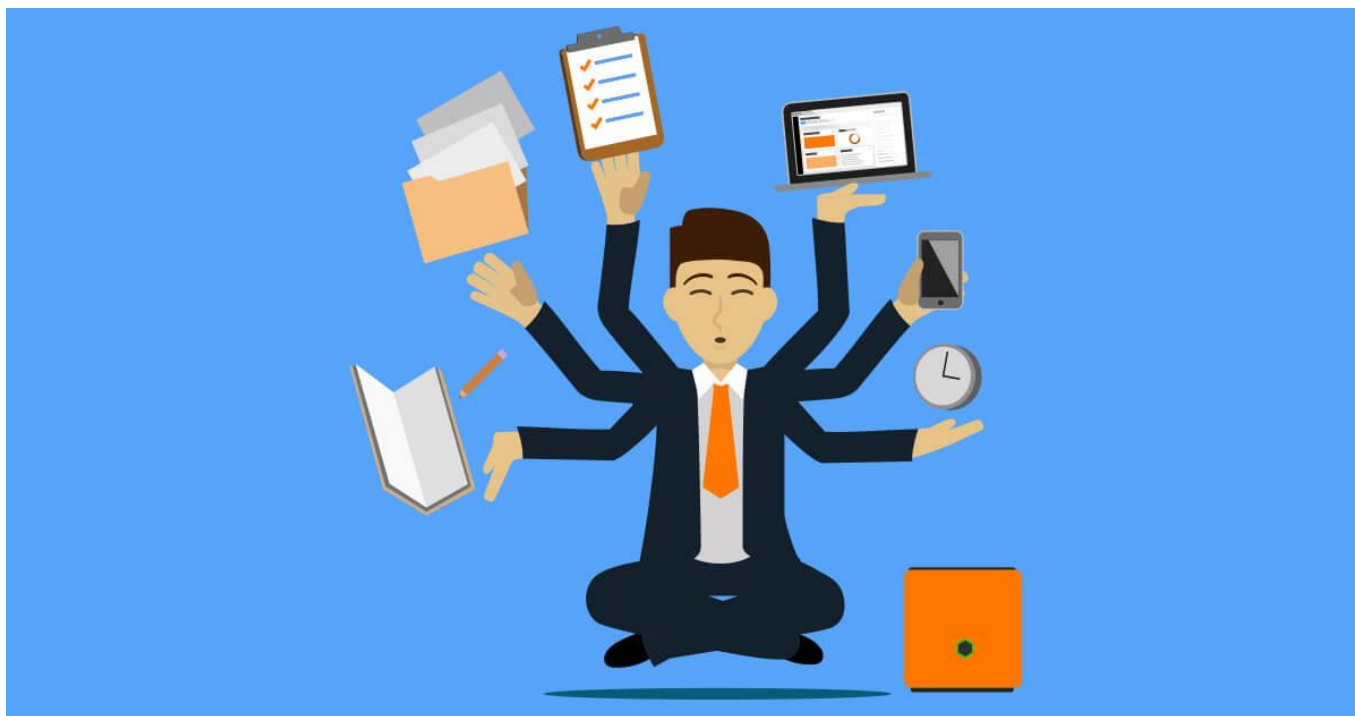
# روش‌های تامین داده



# Inline Data

۱. ساخت تست با Theory

۲. ارسال اطلاعات با InlineData





# اشتراک اطلاعات



۱. عدم امکان اشتراک در روش Inline
۲. امکان تامین داده ها از اعضای کلاس
۳. معرفی با MemberData

# توسعه Attribute اختصاصی

۱. ارث بری از DataAttribute

۲. باز نویسی GetData



# Mog

مدرس: علیرضا ارومند

**nikamooz;**  
آموزش برنامه نویسی  
آموزش برنامه نویسی و اجرای پروژه

# چه خواهیم آموخت؟

۱. Moq چیست؟

۲. بازگشت مقادیر دلخواه

۳. تایید عملکرد صحیح کلاس‌ها

۴. ...



# نگاهی کلی به Mocking

۱. برنامه بخش های مختلفی دارد
۲. نیاز به تست یک قسمت بدون سایر بخش ها
۳. تامین وابستگی ها به کمک Mocking



# تعریف

جایگزینی نسخه‌های عملیاتی وابستگی‌های  
برنامه با نسخه‌هایی صرفاً جهت تست برنامه را

Mocking گویند



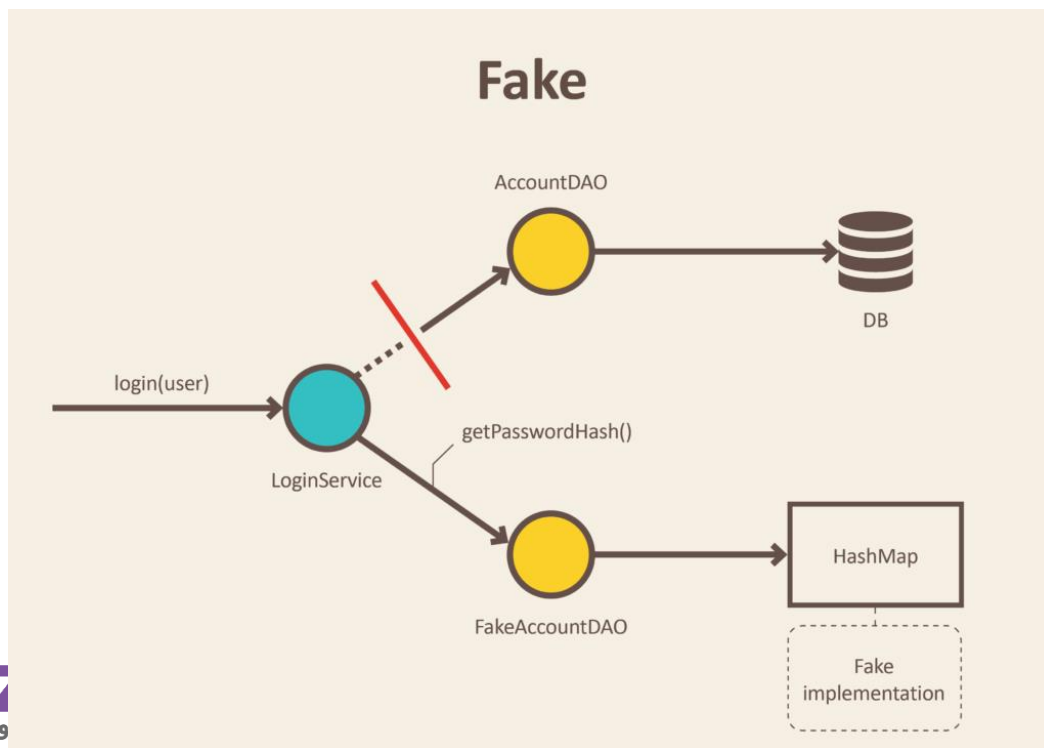


# چرا Mocking؟

۱. افزایش سرعت تست
۲. امکان توسعه موازی
۳. افزایش قابلیت اطمینان به تست‌ها
۴. کاهش هزینه‌های تست برنامه

# Test doubles

۱. هر چیزی که جایگزین شی اصلی شود
۲. با هدف کاهش وابستگی





# Fakes



۱. یک نسخه عملیاتی از وابستگی
۲. برای محیط نهایی مناسب نیست

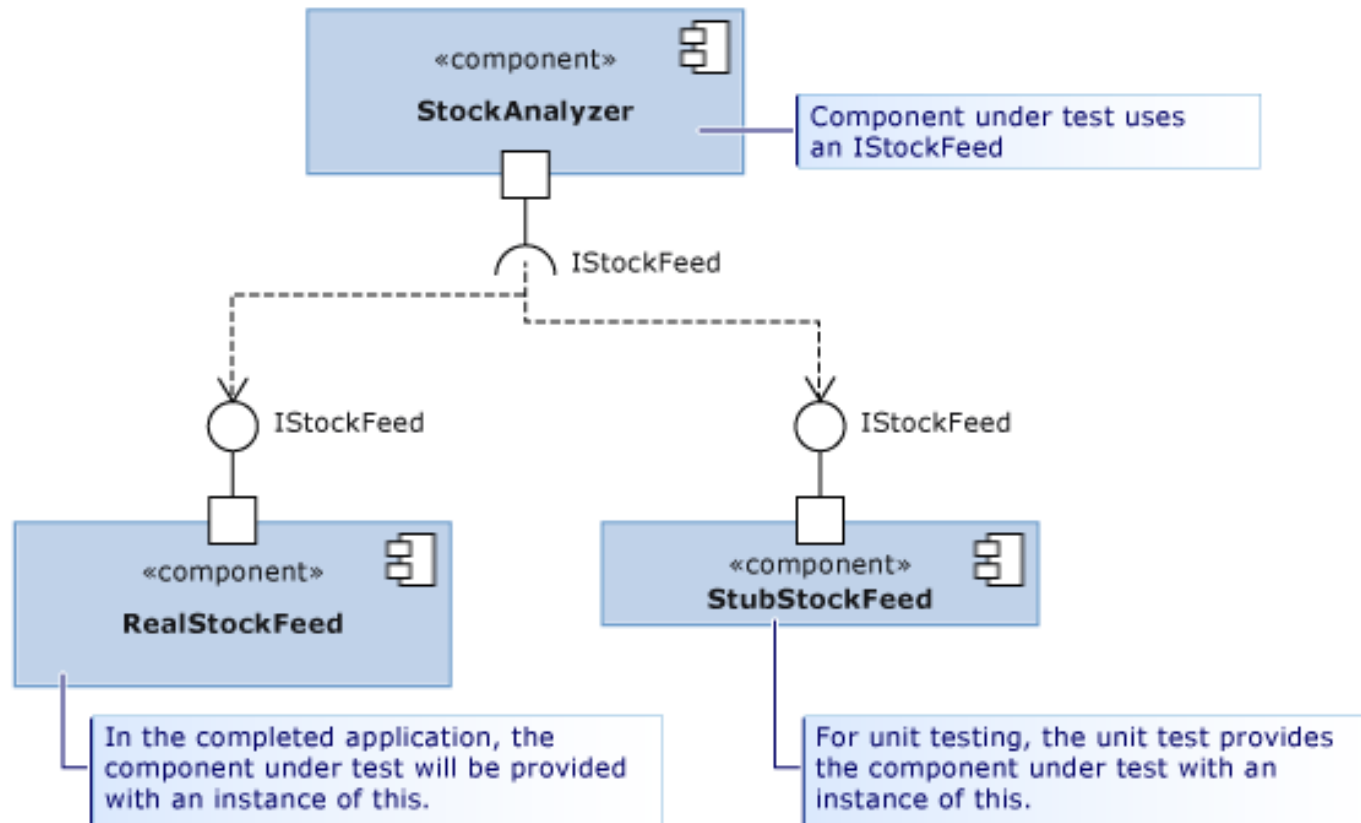
۳. برای مثال In memory Provider

# Dummies

۱. یک نسخه غیر عملیاتی از وابستگی
۲. هیچ کاربردی ندارد
۳. صرفاً جهت تامین پارامترها است



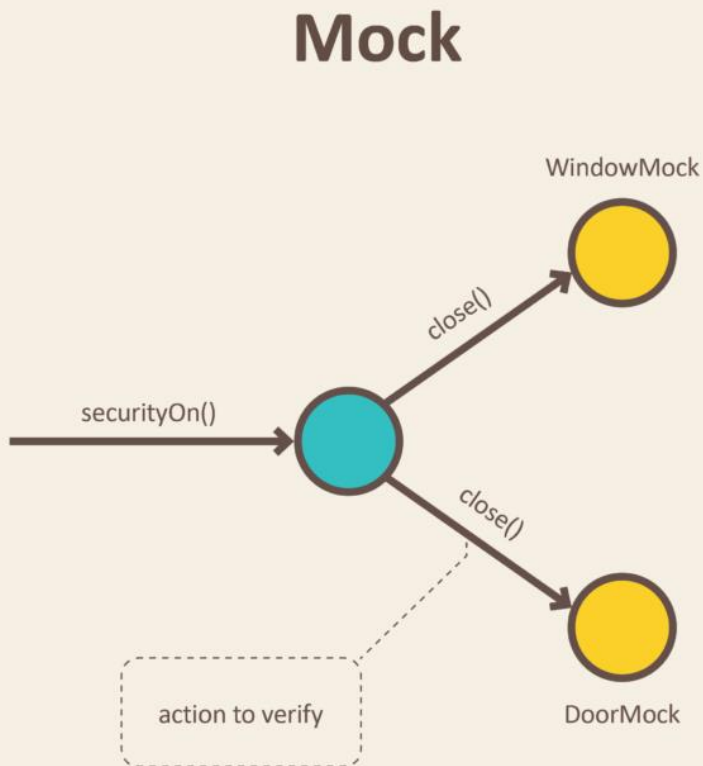
# Stubs



۱. یک نسخه عملیاتی از وابستگی

۲. تامین داده‌های بازگشتی از متدها و خواص

# Mocks



۱. یک شی جهت شبیه سازی وابستگی

۲. بخش های مورد نیاز را شبیه سازی می کنیم

# Moq و پشتیبانی از Test Doubles



**۱. Dummies**

**۲. Stubs**

**۳. Mocks**

# آشنایی با Moq

۱. یک پروژه Open Source

۲. بیش از ۳۲ میلیون دانلود از Nuget



# اهداف طراحی

۱. سادگی در طراحی
۲. ارائه راهکارهای عملیاتی به جای آکادمیک
۳. استفاده ساده و راحت



# نصب Moq



۱. دانلود و نصب از روی Nuget

۲. افزودن Moq using به کلاس‌های تست



# ایجاد نمونه از Mock Object

۱. ایجاد نمونه از کلاس `Moq<>`

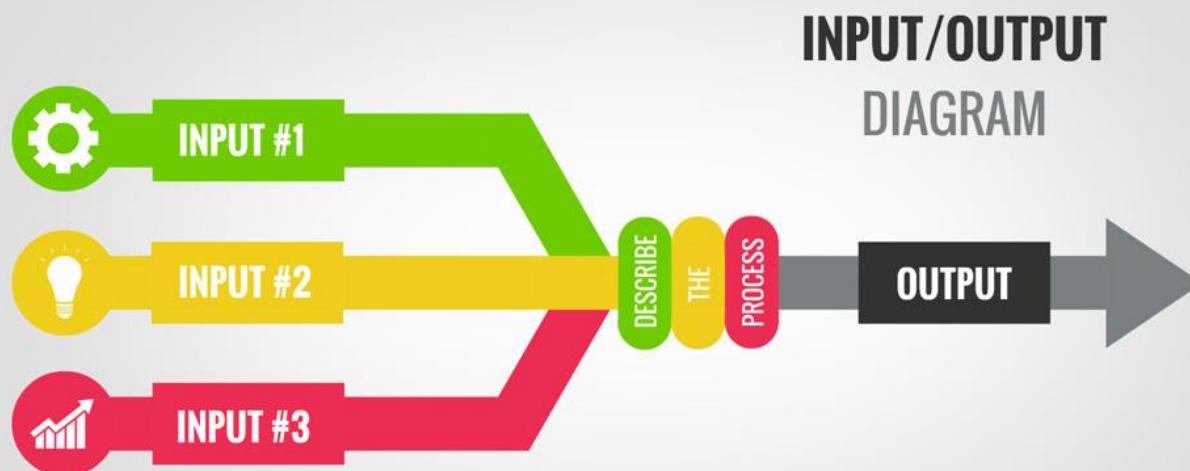
۲. نگهداری نمونه کلاس در `Object`



# تعیین مقدار خروجی از متدها

۱. متد Setup برای انتخاب متد دلخواه

۲. بازگشت مقدار دلخواه با Returns

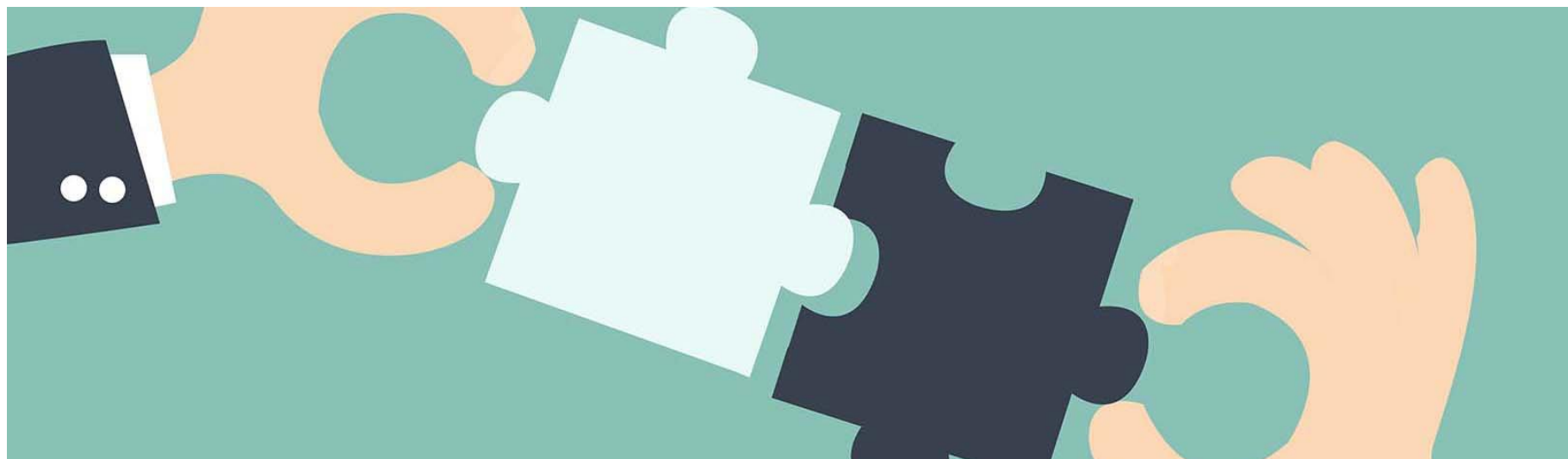


# Argument Matching

۱. استفاده از کلاس `It`

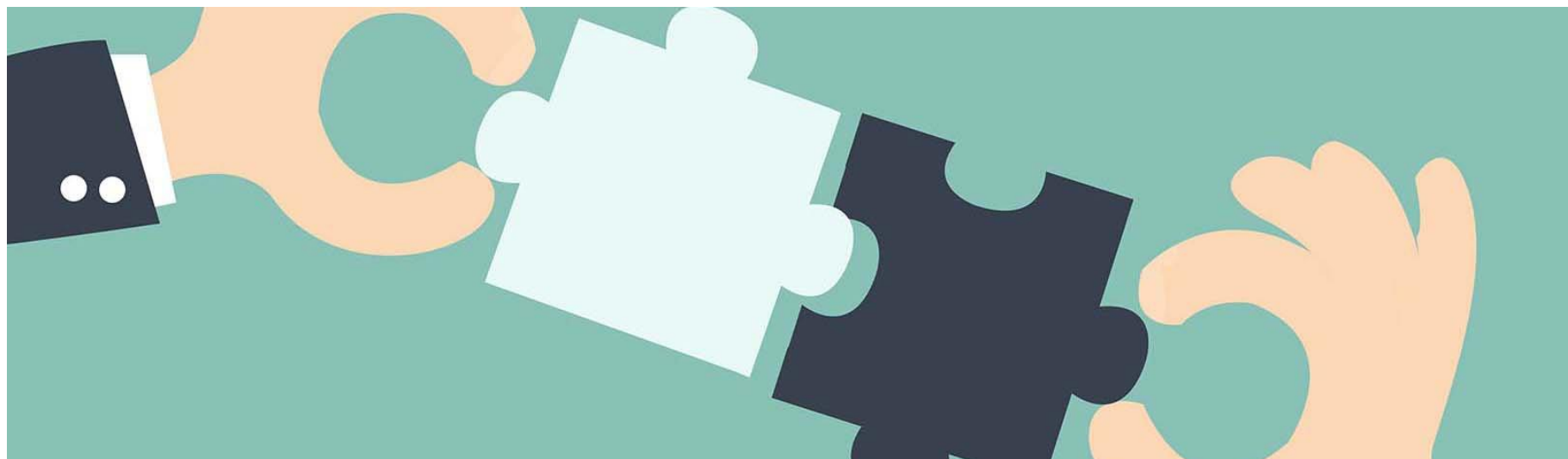
۲. تابع `IsAny` یعنی هر مقداری

۳. تابع `Is` برای ارسال شرط به جای هر مقداری



# Argument Matching

- ۴. **IsIn** بررسی وجود ورودی در لیست اشیا
- ۵. **IsInRange** برای تعیین محدوده مقبول
- ۶. **IsRegex** بررسی **Regex** روی ورودی



# Mock های Strict و Loose

۱. Loose عدم نیاز به Setup کردن

۲. Strict اجبار به Setup کردن متدها



# Mock های Strict و Loose

۳. حالت پیشفرض Loose

۴. امکان تنظیم هنگام نمونه سازی Mock



Loos	strict
Setup ساده	مراحل Setup سخت تر
بازگرداندن مقادیر پیش فرض	تعیین مقدار برای هر متد
کمتر به مشکل می خورد	با کوچکترین تغییری تست Failed می شود
تست های جاری با تغییر کلاس ادامه پیدا می کند	تست های موجود متوقف می شود

## نکته

فقط زمانی که اجباری وجود دارد از حالت Strict استفاده کنید و در سایر موارد حالت Loos که پیشفرض سیستم هم می باشد بهترین انتخاب است.





# Mock و پارامترهای Out

۱. مقدار را در Setup ارسال می‌کنیم

۲. نیازی به Returns ندارد





# خواص تو در تو



۱. خاصیت می‌تواند شامل خاصیت دیگر باشد

۲. پیاده سازی خواص تو در تو سخت و زمانبر

# Auto-Mocking خواص تو در تو

۱. تنظیم داخلی ترین Property
۲. حجم کدنویسی کاهش می یابد



# مقادیر پیشفرض

۱. برای Value Type ها مقدار Default
۲. Null برای Value Type ها



# مقادیر پیشفرض

۳. مجموعه خالی برای مجموعه‌ها

۴. تغییر عملکرد با خاصیت DefaultValued

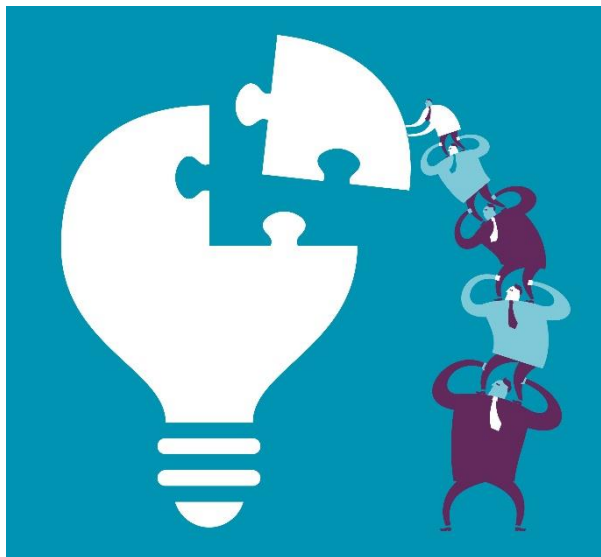


# نگهداری تغییرات

۱. به صورت پیش فرض نگهداری نمی شود

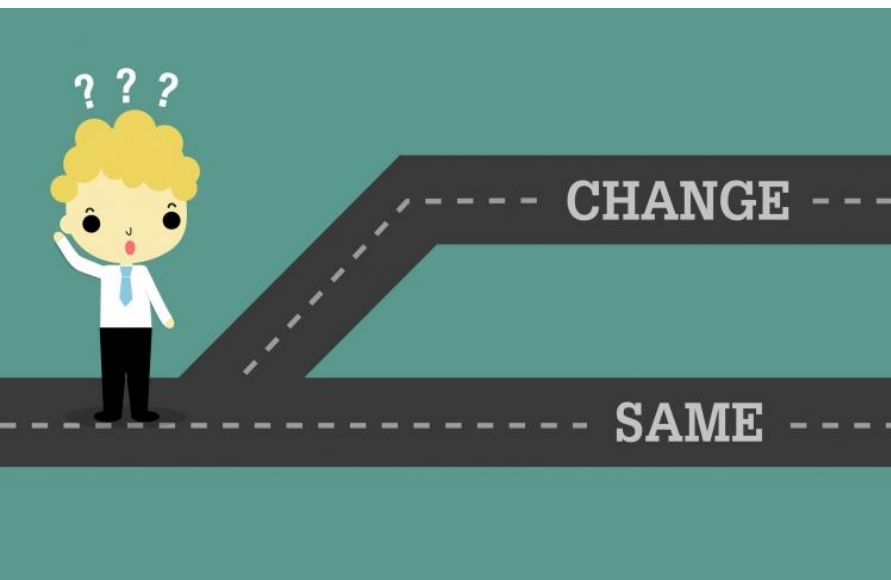
۲. استفاده از `SetupProperty`

۳. در صورت زیاد بود `SetupAllProperties`



# تست وضعیت

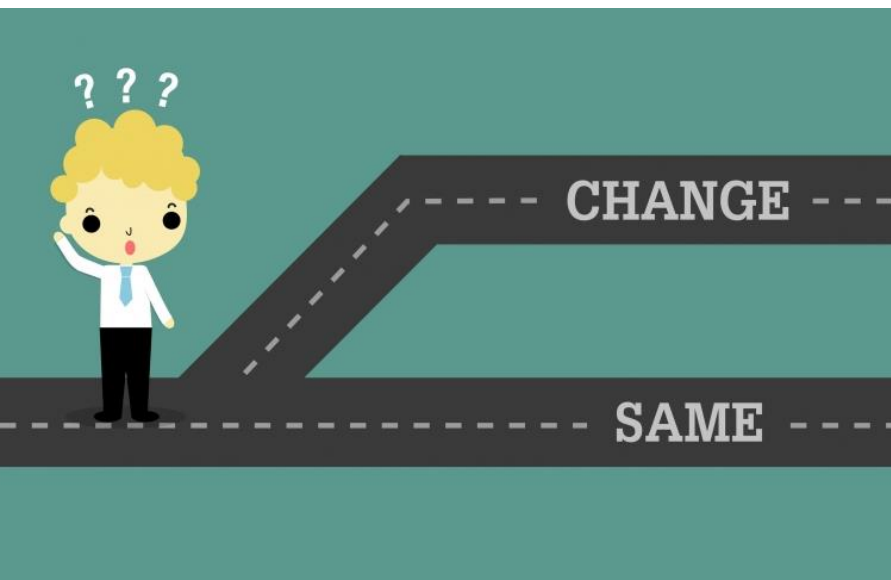
۱. سیستم اجرا می شود
۲. وضعیت سیستم چک می شود
۳. اغلب تست های موجود از این گروه است





# تست عملکرد

۱. وضعیت سیستم مهم نیست
۲. اجرا شدن بخش خاصی از کد مهم است
۳. مثلاً صدا زده شدن تابع به تعداد خاص



# بررسی صدا زدن تابع

۱. انتخاب تابع مورد نظر به کمک Verify
۲. عدم صدا زدن تابع برابر با خطا در تست
۳. امکان ارسال پیام خروجی برای Verify



# بررسی تعداد دفعات صدا زدن

۱. انتخاب تابع مورد نظر به کمک Verify

۲. ارسال تعداد به پارامتر Times

۳. بررسی عدم صدا زدن با Times.Never



# بررسی صدا زدن Getter خواص

۱. استفاده از VerifyGet

۲. امکان بررسی تعداد دفعات مانند توابع



# بررسی صدا زدن Setter خواص

۱. استفاده از VerifySet
۲. امکان تعیین مقدار مورد انتظار
۳. تعداد مراجعه مانند قبل قابل بررسی



# ایجاد Exception

۱. استفاده از Throws به جای Returns

۲. امکان تعیین نوع Exception با Generic



# بازگشت مقادیر مختلف

۱. استفاده SetupSequence

۲. امکان تنظیم چندین Returns



# Moq و کلاس‌های Concrete

۱. بهتر است Interface داشته باشیم
۲. متدها باید قابلیت بازنویسی داشته باشند
۳. در صورت اجبار باید متدها Virtual





# Moq و مقادیر Protected

۱. باید توابع و خواص قابل مشاهده باشد

۲. افزودن Moq.Protected



# Moq و مقادیر Protected

۴. نوع بازگشتی به صورت Generic

۵. انتخاب ورودی به صورت رشته



# شبکه‌های اجتماعی نیک آموز

اطلاع رسانی سریع کارگاه‌های نسبتاً رایگان،

کوپن‌های تخفیف، مقالات، فیلم و دوره‌های نیک آموز



Instagram



Telegram

