عنوان: **استفاده از shim و stub برای mock کردن در آزمون واحد**
نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۱/۲۸ ۲۲:۵۰
آدرس: www.dotnettips.info
گروه‌ها: Unit testing, test, Moq, Mock, Testing, Shim, Stub

مقدمه: از آنجایی که در این سایت در مورد shim و stub صحبتی نشده دوست داشتم مطلبی در این باره بزارم. در آزمون واحد ما نیاز داریم که یک سری اشیا را moq کنیم تا بتوانیم آزمون واحد را به درستی انجام دهیم. ما در آزمون واحد نباید وابستگی به لایه‌های پایین یا بالا داشته باشیم پس باید مقلدی از object هایی که در سطوح مختلف قرار دارند بسازیم.
شاید برای کسانی که با آزمون واحد کار کردند، به ویژه با فریم ورک تست Microsoft، یک سری مشکلاتی با mock کردن اشیا با استفاده از Mock داشته اند که حالا می‌خواهیم با معرفی فریم ورک‌های جدید، این مشکل را حل کنیم.
برای اینکه شما آزمون واحد درستی داشته باشید باید کارهای زیر را انجام دهید:
-1 هر objectی که نیاز به mock کردن دارد باید حتما یا non-static باشد، یا اینترفیس داشته باشد.
-2 شما احتیاج به یک فریم ورک تزریق وابستگی‌ها دارید که به عنوان بخشی از معماری نرم افزار یا الگوهای مناسب شیءگرایی مطرح است، تا عمل تزریق وابستگی‌ها را انجام دهید.
-3 ساختارها باید برای تزریق وابستگی در اینترفیس‌های objectهای وابسته تغییر یابند.

Shims و Stubs:
نوع stub همانند فریم ورک mock می‌باشد که برای مقلد ساختن اینترفیس‌ها و کلاس‌های non-sealed virtual یا ویژگی ها، رویدادها و متدهای abstract استفاده می‌شود. نوع shim می‌تواند کارهایی که stub نمی‌تواند بکند انجام دهد یعنی برای مقلد ساختن کلاس‌های static یا متدهای non-overridable استفاده می‌شود. با مثال‌های زیر می‌توانید با کارایی بیشتر shim و stub آشنا شوید.
یک پروژه mvc ایجاد کنید و نام آن را FakingExample بگذارید. در این پروژه کلاسی با نام CartToShim به صورت زیر ایجاد کنید:

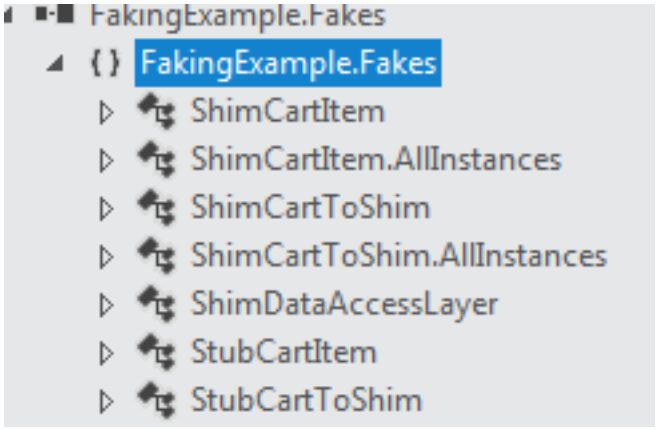
```
namespace FakingExample
{
    public class CartToShim
    {
        public int CartId { get; private set; }
        public int UserId { get; private set; }
        private List<CartItem> _cartItems = new List<CartItem>();
        public ReadOnlyCollection<CartItem> CartItems { get; private set; }
        public DateTime CreateDateTime { get; private set; }

        public CartToShim(int cartId, int userId)
        {
            CartId = cartId;
            UserId = userId;
            CreateDateTime = DateTime.Now;
            CartItems = new ReadOnlyCollection<CartItem>(_cartItems);
        }

        public void AddCartItem(int productId)
        {
            var cartItemId = DataAccessLayer.SaveCartItem(CartId, productId);
            _cartItems.Add(new CartItem(cartItemId, productId));
        }
    }
}
```

و همچنین کلاسی با نام CartItem به صورت زیر ایجاد کنید:

```
public class CartItem
    {
        public int CartItemId { get; private set; }
        public int ProductId { get; private set; }

        public CartItem(int cartItemId, int productId)
        {
            CartItemId = cartItemId;
```

```
            ProductId = productId;
        }
    }
```

حالا یک پروژه unit test را با نام FakingExample.Tests اضافه کرده و نام کلاس آن را CartToShimTest بگذارید. یک reference از پروژه FakingExample تان به پروژه‌ی تستی که ساخته اید اضافه کنید. برای اینکه بتوانید کلاس‌های پروژه FakingExample را shim و یا stub کنید باید بر روی Reference پروژه تان راست کلیک کنید و گزینه Add Fakes Assembly را انتخاب کنید. وقتی این گزینه را می‌زنید، پوشه ای با نام Fakes در پروژه تست ایجاد شده و FakingExample.fakes در داخل آن قرار دارد همچنین در referenceهای پروژه تست، FakingExample.Fakes نیز ایجاد می‌شود.
اگر بر روی فایل fakes که در reference ایجاد شده دوبار کلیک کنید می‌توانید کلاس‌های CartItem و CartToShim را مشاهده کنید که هم نوع stub شان است و هم نوع shim آنها که در تصویر زیر می‌توانید مشاهده کنید.



ShimDataAccessLayer را که مشاهده می‌کنید یک متد SaveCartItem دارد که به دیتابیس متصل شده و آیتم‌های کارت را ذخیره می‌کند.

حالا می‌توانیم تست خود را بنویسیم. در زیر یک نمونه از تست را مشاهده می‌کنید:

```
[TestMethod]
        public void AddCartItem_GivenCartAndProduct_ThenProductShouldBeAddedToCart()
        {
            //Create a context to scope and cleanup shims
            using (ShimsContext.Create())
            {
                int cartItemId = 42, cartId = 1, userId = 33, productId = 777;

                //Shim SaveCartItem rerouting it to a delegate which
                //always returns cartItemId
                Fakes.ShimDataAccessLayer.SaveCartItemInt32Int32 = (c, p) => cartItemId;

                var cart = new CartToShim(cartId, userId);
                cart.AddCartItem(productId);

                Assert.AreEqual(cartId, cart.CartItems.Count);
                var cartItem = cart.CartItems[0];
                Assert.AreEqual(cartItemId, cartItem.CartItemId);
                Assert.AreEqual(productId, cartItem.ProductId);
            }
        }
```

همانطور که در بالا مشاهده می‌کنید کدهای تست ما در اسکوپی قرار گرفته اند که محدوده shim را تعیین می‌کند و پس از پایان یافتن تست، تغییرات shim به حالت قبل بر می‌گردد. متد SaveCartItemInt32Int32 را که مشاهده می‌کنید یک متد static است و نمی‌توانیم با mock ویا stub آن را مقلد کنیم. تغییر اسم متد SaveCartItem به SaveCartItemInt32Int32 به این معنی است که

متد ما دو ورودی از نوع Int32 دارد و به همین خاطر fake این متد به این صورت ایجاد شده است. مثلا اگر شما متد Save ای داشتید که یک ورودی Int و یک ورودی String داشت fake آن به صورت SaveInt32String ایجاد می‌شد.
به این نکته توجه داشته باشید که حتما برای assert کردن باید assertها را در داخل اسکوپ ShimsContext قرار گرفته باشد در غیر این صورت assert شما درست کار نمی‌کند.

این یک مثال از shim بود؛ حالا می‌خواهم مثالی از یک stub را برای شما بزنم. یک اینترفیس با نام ICartSaver به صورت زیر ایجاد کنید:

```
public interface ICartSaver
    {
        int SaveCartItem(int cartId, int productId);
    }
```

برای shim کردن ما نیازی به اینترفیس نداشتیم اما برای استفاده از stub و یا Mock ما حتما به یک اینترفیس نیاز داریم تا بتوانیم object موردنظر را مقلد کنیم. حال باید یک کلاسی با نام CartSaver برای پیاده سازی اینترفیس خود بسازیم:

```
public class CartSaver : ICartSaver
    {
        public int SaveCartItem(int cartId, int productId)
        {
            using (var conn = new SqlConnection("RandomSqlConnectionString"))
            {
                var cmd = new SqlCommand("InsCartItem", conn);
                cmd.CommandType = CommandType.StoredProcedure;
                cmd.Parameters.AddWithValue("@CartId", cartId);
                cmd.Parameters.AddWithValue("@ProductId", productId);

                conn.Open();
                return (int)cmd.ExecuteScalar();
            }
        }
    }
```

حال تستی که با shim انجام دادیم را با استفاده از Stub انجام می‌دهیم:

```
[TestMethod]
        public void AddCartItem_GivenCartAndProduct_ThenProductShouldBeAddedToCart()
        {
            int cartItemId = 42, cartId = 1, userId = 33, productId = 777;

            //Stub ICartSaver and customize the behavior via a
            //delegate, ro return cartItemId
            var cartSaver = new Fakes.StubICartSaver();
            cartSaver.SaveCartItemInt32Int32 = (c, p) => cartItemId;

            var cart = new CartToStub(cartId, userId, cartSaver);
            cart.AddCartItem(productId);

            Assert.AreEqual(cartId, cart.CartItems.Count);
            var cartItem = cart.CartItems[0];
            Assert.AreEqual(cartItemId, cartItem.CartItemId);
            Assert.AreEqual(productId, cartItem.ProductId);
        }
```

امیدوارم که این مطلب برای شما مفید بوده باشد.

نظرات خوانندگان

نویسنده: سام ناصری
تاریخ: ۱۳۹۲/۰۱/۳۰ ۷:۲۳

من نویسنده خوبی نیستم و شاید بهتر باشه که در اینباره نظر ندهم. به هر روی چند نکته به نظرم آمد باشد که مورد توجه شما واقع شود:

مقدمه را هنوز کامل نکردی. مقدمه خواننده را در جای پرتی از ماجرا رها میکند. اگر چهار خط آخر مقدمه را دوباره بخوانید متوجه میشوید که اگر تمام کاری که برای داشتن آزمون واحد باید انجام شود همین سه مورد باشد دیگر هرگز کسی به Fakes نیاز پیدا نمیکند، پس باید در ادامه میگفتید که این حالت مطلوب است ولی همیشه عملی نیست.

شروع و پایان مثالها مشخص نبود. مثالها بدون عنوان بودند. در شروع مثال باید مقدمه ای از مثال را مطرح میکردی و بعد مراحل مثال را توضیح میدادی.

در مثال اول باید بر بیشتر بر روی DataAccessLayer تاکید میکردی و صریح مشخص میکردی که عدم توانایی برنامه نویس در تغییر این کلاس و یا معماری سیستم گزینه IoC را کنار میگذارد و به این ترتیب مثال شما سودمندی Shim را بهتر نشان میداد.

در مثال دوم، کد CardToStub را ارائه نکردی، اگر،طبق آنچه انتظار میرود، وابستگی که در CardToStub وجود دارد به اینترفیس ICartSaver است در این صورت اساساً مثال شما هیچ دلیل و انگیزشی برای Stub فراهم نمیکند. باید باز هم ذهنیت خواننده را شکل میدادی و او را متوجه این موضوع میکردی که در پیاده سازی دیگری که برنامه نویس قدرت اعمال تغییر در آن ندارد وابستگی سخت وجود دارد و به این دلیل Stub میتواند مفید واقع شود.

البته این رو به حساب اینکه من یک خواننده بسیار مبتدی هستم گفتم شاید مقاله برای دیگران بیشتر از من قابل فهم است. ولی در کل مقاله خوبی بود و برای من کابردی بود.

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۱/۳۰ ۱۱:۴۰

ممنونم از اینکه راهنماییم کردید تا مطالبم را درست‌تر بنویسم اما اون 3 موردی را که گفتم کارهایی است که برای آزمون واحد انجام می‌شود یعنی باید اینترفیس داشته باشیم برای مقلد ساختن و کلاس‌ها برای اینکه mock شوند باید non-static باشند و از این قبیل و در ادامه گفتم که اگر کلاسی ویژگی آن 3 مورد را نداشته باشد مثلا نه اینترفیس داشته باشد و هم اینکه static باشد چیکار باید کرد.
در مورد stub گفتم که این نوع همانند فریم ورک mock می‌باشد و هیچ فرقی با آن ندارد یعنی شما مجبور نیستید از stub استفاده کنید می‌توانید به جای آن از mock استفاده کنید.
در مورد کد CartToStub همان کد آخری است فقط خطی که نام کلاس را نوشته بود نگذاشتم. در مورد اینکه برای مثال مقدمه ای باید می‌گذاشتم راستش من دقیقا نمی‌دونم شاید هم حرف شما درست باشد ولی من فقط می‌خواستم طریقه نوشتن shim رو توضیح بدم یعنی در واقع حتی نیاز به ساخت پروژه و این حرفا هم نداشت.
بازم متشکرم که ایرادات منو فرمودین سعی می‌کنم از این به بعد مطالبم رو بهتر بنویسم

نویسنده: محسن خان
تاریخ: ۱۳۹۲/۰۱/۳۰ ۱۴:۷

mocking بهتره به معنای ایجاد اشیاء تقلیدی عنوان بشه تا مقلد سازی.

نویسنده: مرتضی
تاریخ: ۱۳۹۲/۰۹/۲۷ ۱:۲۱

سلام

~~(نوع stub همانند فریم ورک mock می‌باشد)~~

تعریفی که از stup تو راهنماش اومده با مطلبی که شما ذکر کردید متفاوته

Martin Fowler’s article **Mocks aren’t Stubs** compares and contrasts the underlying principles of Stubs and Mocks. As outlined in Martin Fowler’s article, a **stub provides static canned state which results in state verification** of the system under test, whereas a **mock provides a behavior verification** of the results for the system under test and their indirect outputs as related to any other component dependencies while under test

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۹/۲۷ ۸:۵۳

با سلام ممنون که این مطلب رو گذاشتین اما منظور من این نیست که هیچ فرقی با هم ندارند منظورم از اینه که همانطور هم بالا توضیح دادم برای مقلد سازی اینترفیس‌ها و abstractها و ... به کار میره همانطور که mock برای اینطور کلاس‌ها و متدها استفاده می‌شود

عنوان: **Microsoft Test Manager - قسمت اول**
نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۱/۳۱ ۲۳:۵۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Integration tests, test, Testing, Test Manager 2012, sprint, Test Case, Test Plan

مقدمه:

مدیریت آزمون مایکروسافت یا Microsoft Test Manager یک ابزار تست نویسی است که به تسترها این اجازه را می‌دهد تا بتوانند برای UI برنامه‌های خود یا sprintهای پروژه خود تست بنویسند. این ابزار برای نوشتن آزمون‌های پیشرفته و مجتمع سازی مدیریت طرح‌های تست یا test plans همراه با موردهای تست یا test case در طول توسعه برنامه است. یکی از مزایایی که این ابزار دارد این است که در طول انجام تست می‌توانید اشکالات تست را ثبت کنید و هم چنین می‌توانید شرحی در مورد انجام تست یا اشکالی که در آن تست وجود دارد، ثبت کنید. همچنین می‌توانید گزارشی از تست هایی که انجام داده اید و پاس شدن یا پاس نشدن تست‌ها و تاریخ انجام آن‌ها را نیز مشاهده کنید. قبل از کار با نرم افزار MTM باید یک سری مطالب مهم را در مورد انجام تست و مفهوم Agile بدانیم.

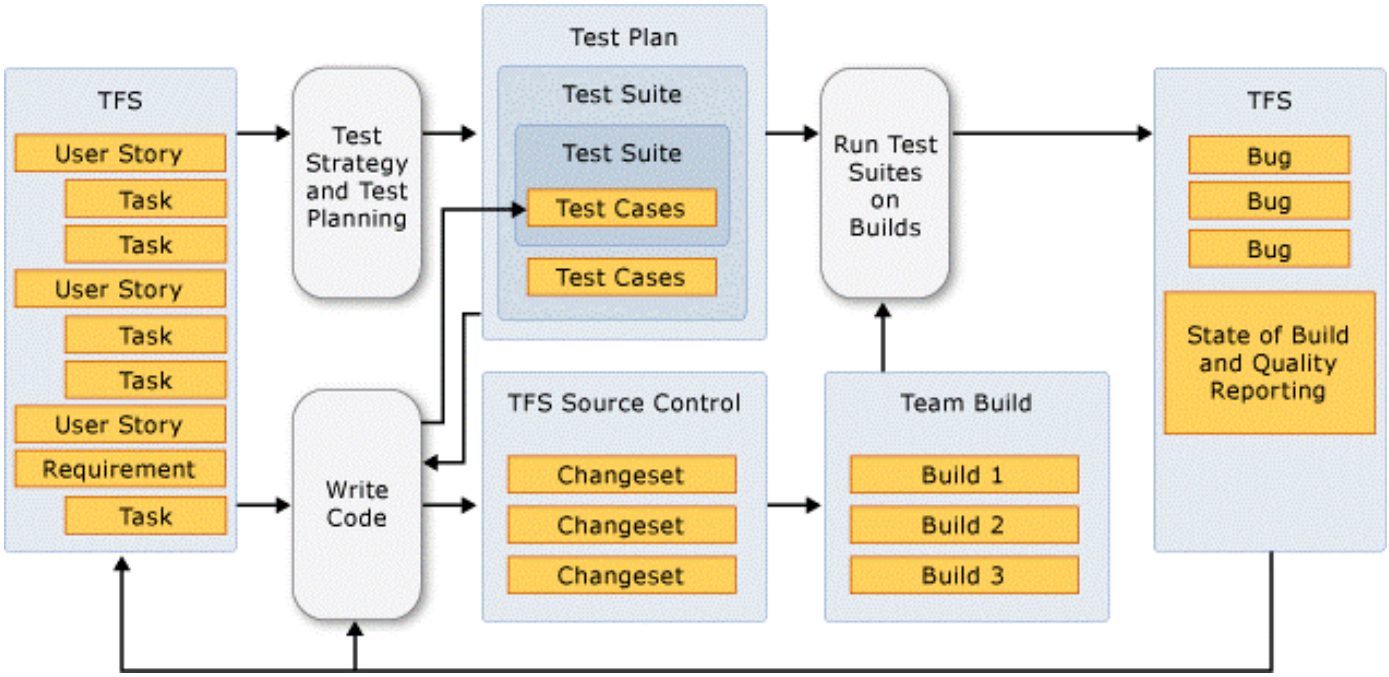
استراتژی تست:

زمانی که شما تست Agile را معرفی می‌کنید تیم برنامه نویسی شما می‌تواند بر روی تست‌های شما هم در سطح sprint و هم در سطح پروژه تمرکز کنند. تست در سطح sprint شامل تست هایی می‌شود که همه user story ها در بر بگیرد یعنی در واقع همان تست‌های واحد شما می‌شود. در سطح پروژه هم شامل تست هایی می‌شود که چندین sprint را در بر می‌گیرد که در واقع می‌توان تست‌های integrated گفت. بهتر است زمانی که تیم برنامه نویسی کدنویسی می‌کنند شما طرح تست‌های خود را بسازید و برای انجام تست کاملا آماده باشید. این تست‌ها شامل تست واحد، تست performance ، تست امنیتی و تست usability و غیره می‌باشد.

برای آماده کردن تست Agile در ابتدا شما باید یک تاریخچه یا history از برنامه یا سیستم خود داشته باشید. شما می‌توانید با استفاده از Microsoft Test Manager طرح تست خود را برای هر یک از sprint های پروژتان بسازید و موردهای تست را مشخص کنید.

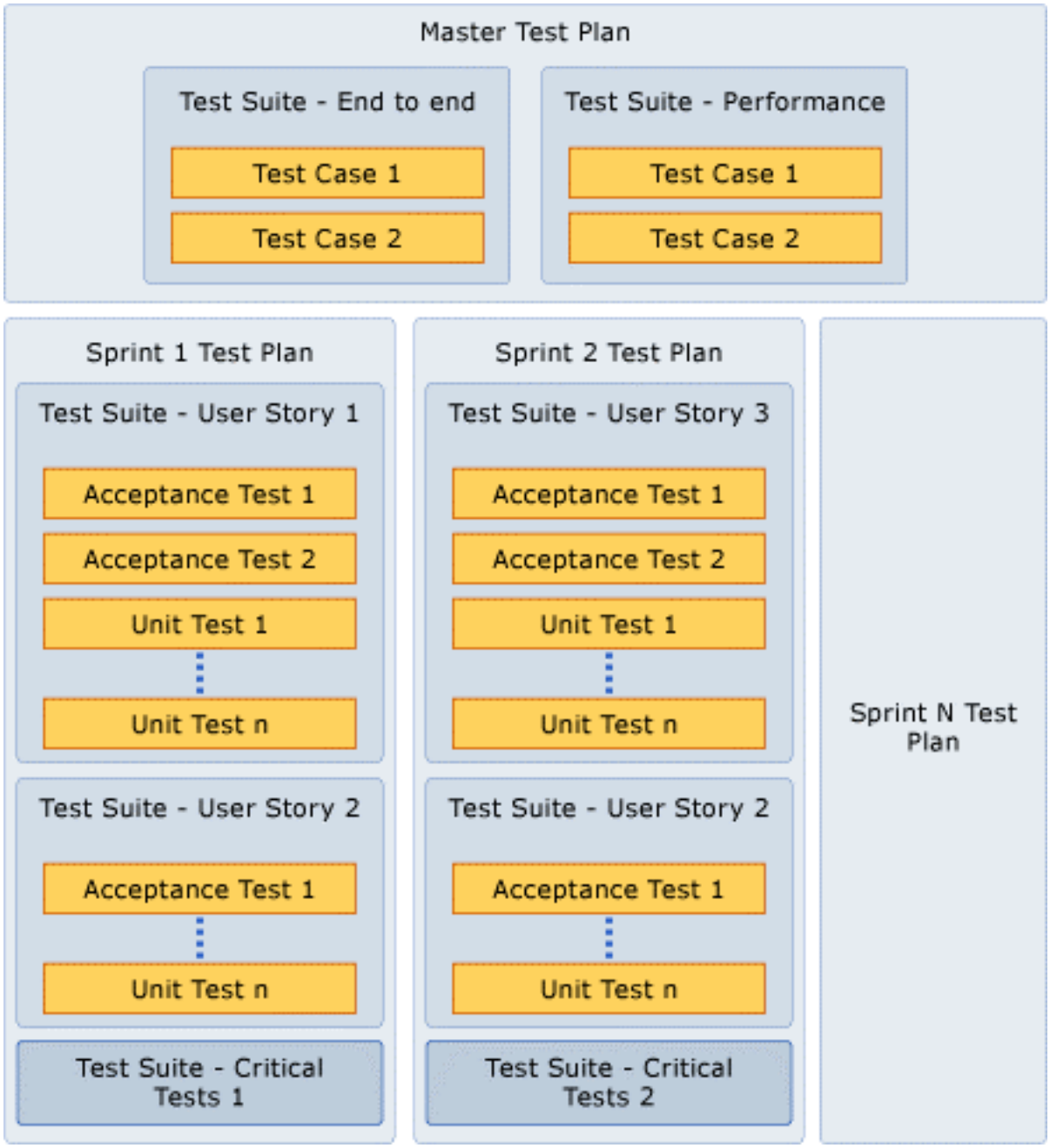
سپس باید کدهایی که برنامه نویسان می‌نویسند قابلیت تست را داشته باشند و شما به عنوان یک تستر باید آشنایی کاملی از ساختار و الگوهای برنامه تان داشته باشید.

تست یک فرآیند تکراری می‌باشد که همزمان با اجرای پروژه تان صورت می‌گیرد در زیر می‌توانید فرآیند کار تست و انجام کدنویسی را مشاهده نمایید:



: Test Planning

Test Planning فرآیندی است که به تیم شما کمک می‌کند تا درک درستی از پروژه داشته باشند و همچنین تیم را برای انجام هر گونه تستی آماده کند. تست Agile در سطح Sprint انجام می‌شود که در هر Sprint تیم شما تست هایی را ایجاد می‌کنند تا user story هایی که در هر Sprint وجود دارد، مورد بررسی قرار گیرند. در شکل زیر قالبی از test plan های شما در یک پروژه را نمایش می‌دهد:



البته این قالب‌ها بر اساس سلیقه شخصی است اما در کل می‌توانیم قالب تست را به صورت بالا در نظر بگیریم.

همیشه باید این را در نظر داشته باشیم که در طول هر sprint حتما باید تست‌ها را اجرا کرده و در صورت وجود خطا، آن خطا را رفع کنیم تا در مراحل بالاتر با مشکلی مواجه نشویم. در قسمت بعد با Microsoft Test Manager و روش‌های نوشتن sprint و تست‌ها آشنا خواهیم شد.

نظرات خوانندگان

نویسنده: سیروان عفیفی
تاریخ: ۱۳۹۲/۰۲/۰۱ ۰:۵

با تشکر از شما، مطلب خوبی بود.

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۲/۰۱ ۸:۶

خواهش می‌کنم امیدوارم مطالبم خوب نوشته شده باشه چون در نوشتن کمی ضعیف هستم

نویسنده: مهدی
تاریخ: ۱۳۹۲/۰۲/۰۳ ۱۹:۱۱

با سلام خدمت دوست عزیز و تشکر از این مقاله مفید.
لطفا اگر می‌شود در مورد اصطلاحاتی که بیان می‌کنید در اول مقاله یه تعریفی از ان‌ها بیان کنید.
با تشکر.

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۲/۰۳ ۲۱:۱۲

خواهش می‌کنم ولی منظور شما کدام اصطلاحات است؟ چون در قسمت دوم خیلی‌های این اصطلاحات رو گفتم اگر اصطلاحی رو متوجه نشدین بگین تا واستون توضیح بدم

عنوان: **Microsoft Test Manager - قسمت دوم**
نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۲/۰۱ ۲۳:۱۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Integration tests, test, Testing, Test Manager 2012, Test Case, Test Plan, Test Suite

تا اینجا متوجه شدیم که test plan چیست و چگونه ساخته می‌شود و برای نوشتن تست‌ها چه مراحلی را باید طی کنیم. در این مطلب قصد بر این است که آموزش نوشتن تست‌ها با استفاده از MTM را آموزش دهیم. در این آموزش فرض بر این است که شما آشنایی کمی با محیط این ابزار، نیازمندی‌ها و Story ها، اشکالات یا Bug ها و Task ها دارید.

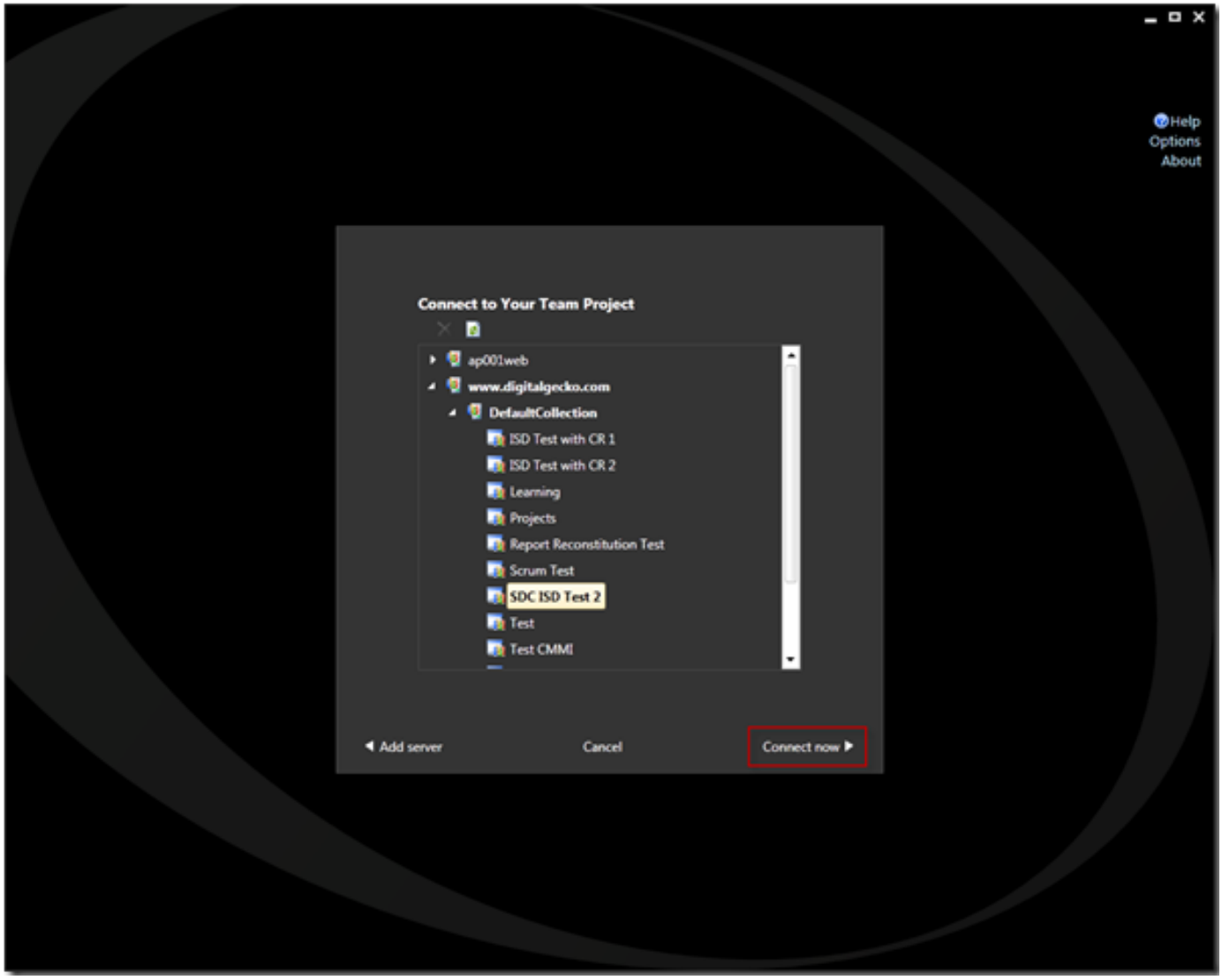
در MTM سه لایه وجود دارد:

1- **Test Plan :** شما در آغاز کار با MTM ابتدا باید Test Plan خود را ایجاد کنید.

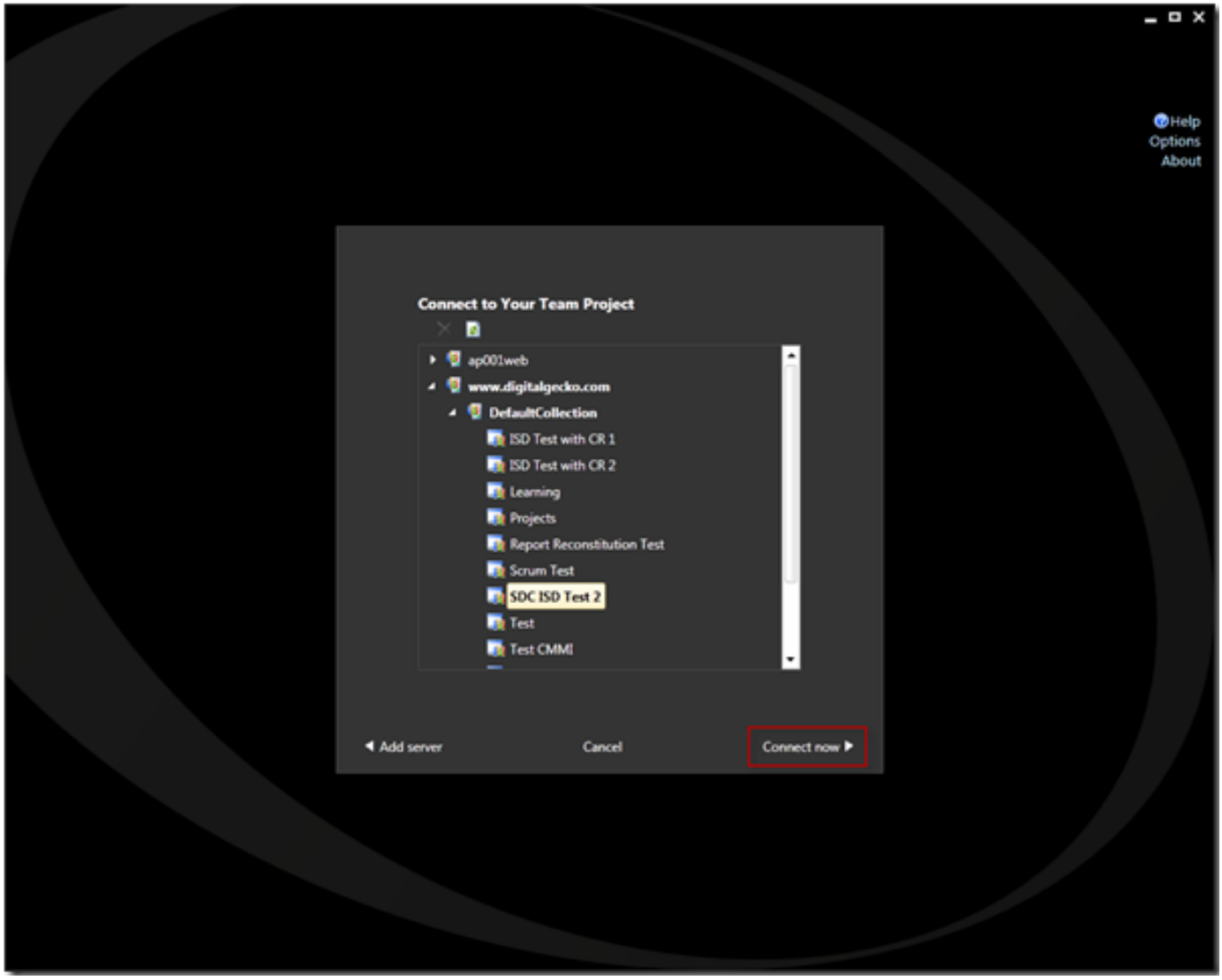
2- **Test Suite :** در هر Test Plan شما می‌توانید چندین Test Suite ایجاد کنید.

3- **Test Case :** هر Test Suite از چندین Test Case ترکیب شده است.

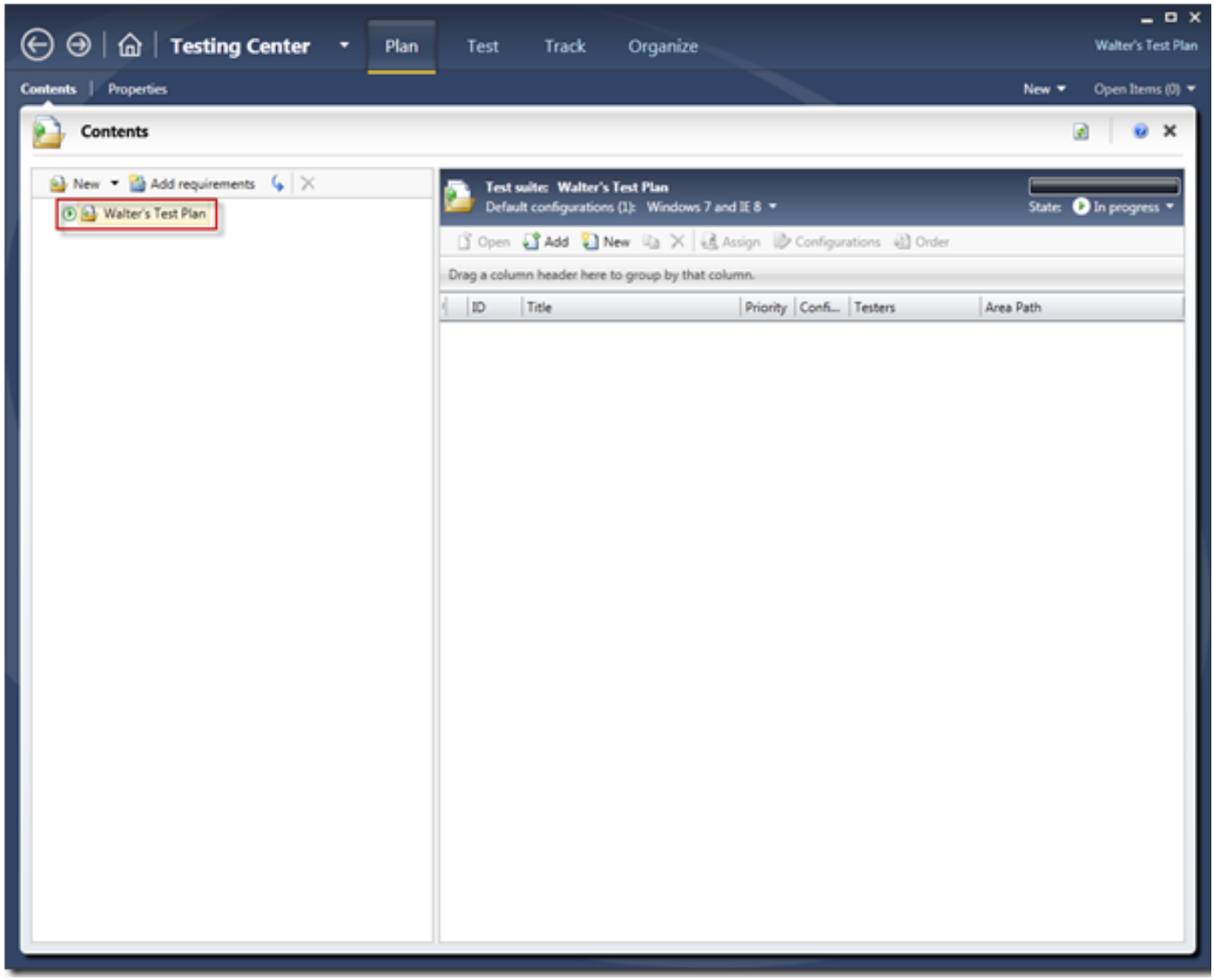
برای اولین بار که شما MTM را اجرا می‌کنید باید team project ی را که قرار است برای آن تست بنویسید را انتخاب کنید.
می‌توانید در زیر نمایی از MTM و اتصال به team project را مشاهده کنید:



بعد از اینکه پروژه خود را انتخاب کردید، می‌توانید لیستی از طرح‌های تست تان که برای این پروژه ایجاد کرده اید را مشاهده کنید که می‌توانید از این لیست یک طرحی را انتخاب نمایید و یا یک طرح جدید را ایجاد کنید همانطور که در شکل زیر مشاهده می‌کنید.



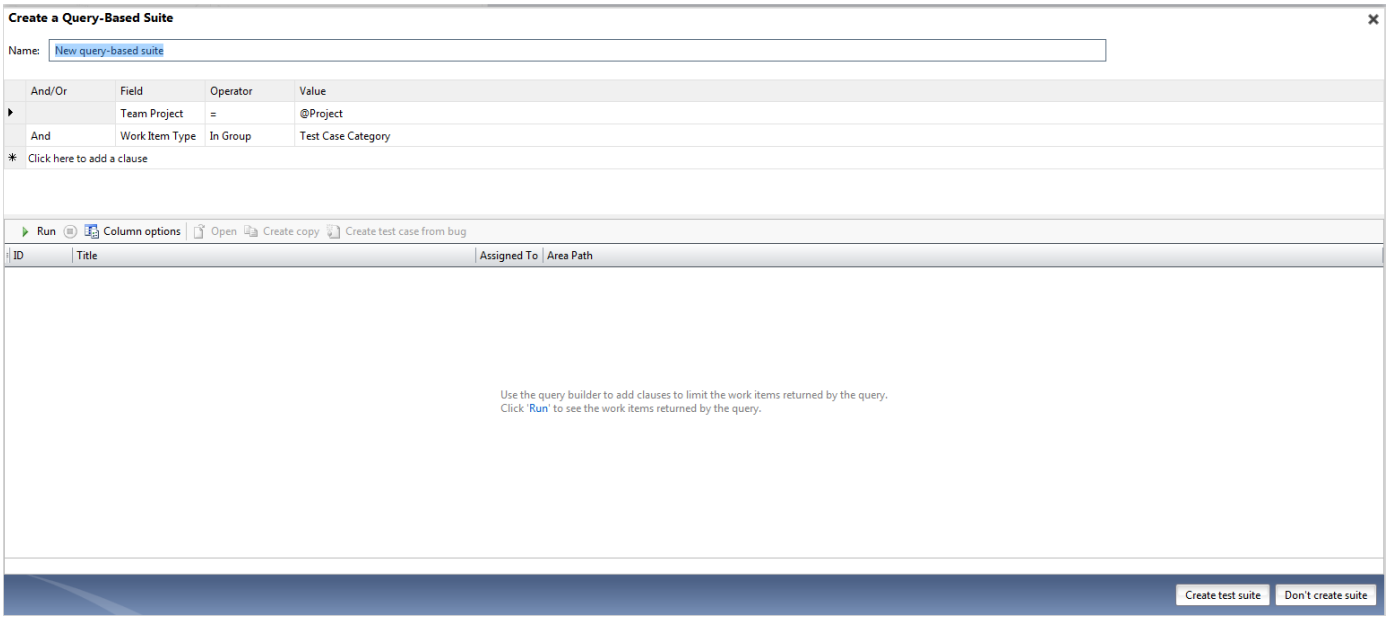
وقتی plan یا طرحی را انتخاب می‌کنید به صفحه testing center وارد می‌شوید که به صورت پیش فرض در کاربرگ plan و بخش contents قرار دارید.



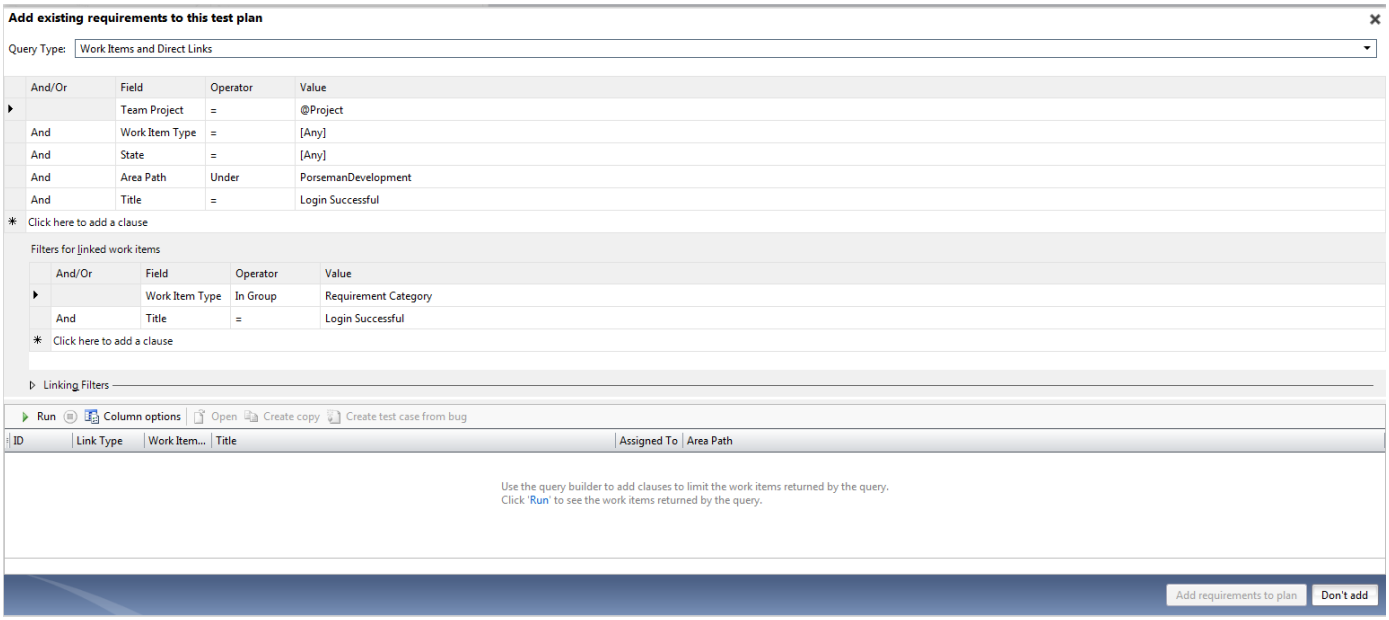
همانطور که در تصویر بالا مشاهده می‌کنید و در سمت چپ پنجره، plan شما در ریشه قرار دارد و test suite هایی را که ایجاد می‌کنید به عنوان فرزندان plan تان قرار می‌گیرند. در سمت راست test caseهای شما قرار می‌گیرند که با توجه به test suite ی که شما در سمت چپ انتخاب کرده اید test caseهای مربوط به آن در سمت راست قابل مشاهده است. برای ایجاد test suite به plan تان، باید روی plan راست کلیک کرده و گزینه new suite را انتخاب کنید و برای آن عنوانی را وارد می‌کنید. وقتی روی plan راست کلیک می‌کنید پند گزینه وجود دارد که می‌توانید با توجه به کارتان این گزینه‌ها را انتخاب کنید:

-1 وقتی new suite را انتخاب می‌کنید یک suite خالی برای شما ایجاد می‌کند.

-2 وقتی گزینه new query-based suite را انتخاب می‌کنید این اجازه را به شما می‌دهد که از test caseهای موجود در پروژه خود یک یا چندین مورد تست را انتخاب نمایید که پنجره ای مانند زیر باز می‌شود که می‌توانید با اعمال فیلتر، test caseهای موجود در پروژه را پیدا و یک یا چندین مورد را به suite خود اضافه نمایید.



3- گزینه add requirement to plan این اجازه را به شما می‌دهد تا بتوانید از planهای موجود در TFS تان استفاده نمایید. بعد از انتخاب این گزینه پنجره ای مشابه تصویر بالا باز می‌شود که می‌توانید با اعمال فیلتر موردهای تست را پیدا کرده و به آن بیافزاید.



4- با انتخاب گزینه copy suite from another plan همانطور که از اسمش پیداست می‌توانید از suiteهای مربوط به planهای دیگر کپی برداری کنید.

نظرات خوانندگان

نویسنده: علیرضا پونه
تاریخ: ۱۳۹۲/۰۲/۰۲ ۸:۴۹

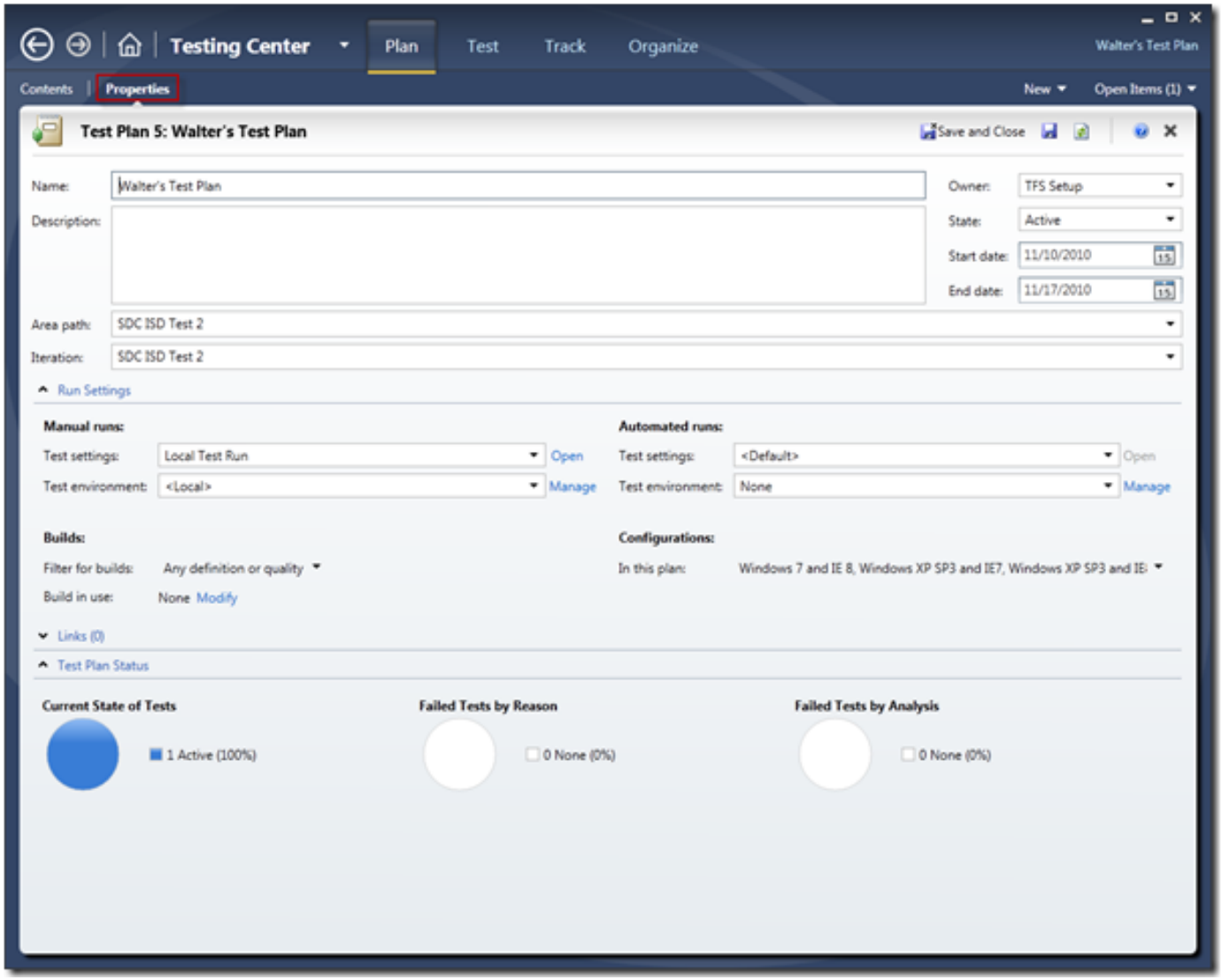
ممنونم. فقط اینکه تو هر پست مطلب رو کاملتر و قسمت بیشتری رو بگین تا در تعداد پست کمتری بشه همه چیز رو گفت و هم اینکه خواننده تا پست بعدی، خیلی از مطلب دور نشه. بازم بابت مطلب بسیار مهمی که دارین آموزش میدین خیلی خیلی ممنون.

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۲/۰۲ ۱۲:۱

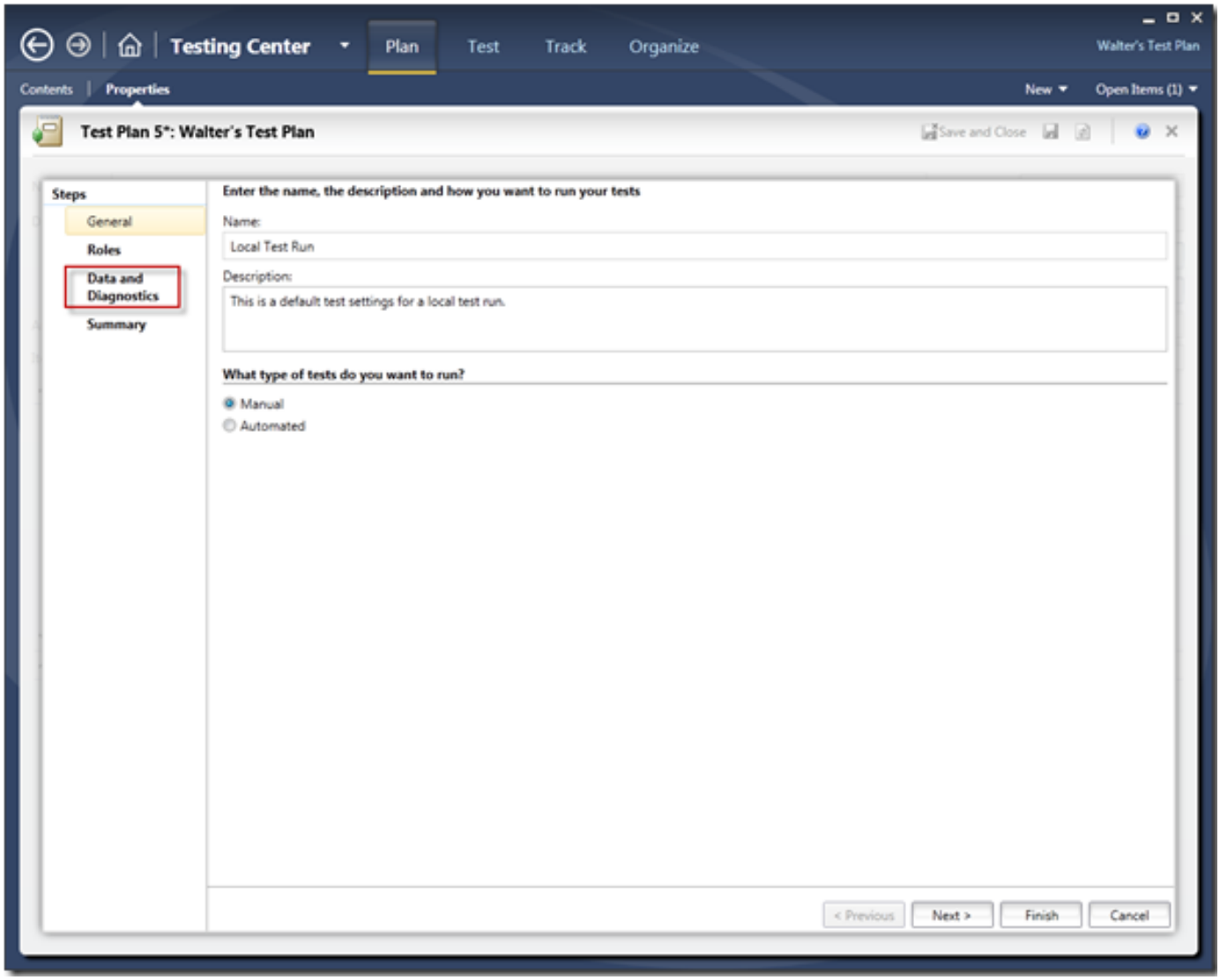
دلیل اینکه مطلب زیاد نمیزارم چون می‌گم شاید کاربران خسته شن یا حوصله‌ی خوندن مطلب زیاد رو نداشته باشن و بعد اینکه مبحث جدیدی که بخواد شروع بشه مجمبورم قسمت قبل رو قطع کنم قسمت بعدی در مورد یک سری تنظیمات در MTM است و ربطی به این بخش نداره

عنوان: **Microsoft Test Manager - قسمت سوم**
نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۲/۰۳ ۲۳:۵۰
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Integration tests, test, Testing, Test Manager 2012, Test Case, Test Plan, Test Suite

در کنار کاربرگ contents کاربرگی با نام Properties وجود دارد که می‌توانید یک سری تنظیمات را برای plan خود انجام دهید. این تنظیمات از قبیل تغییر عنوان plan ، تعیین مسیر پروژه، تاریخ شروع و پایان، کاربری که مالک این plan است، وضعیت جاری تست‌های plan و تعیین مرورگر و ویندوز نیز می‌باشد که می‌توانید در تصویر زیر آن را مشاهده کنید.

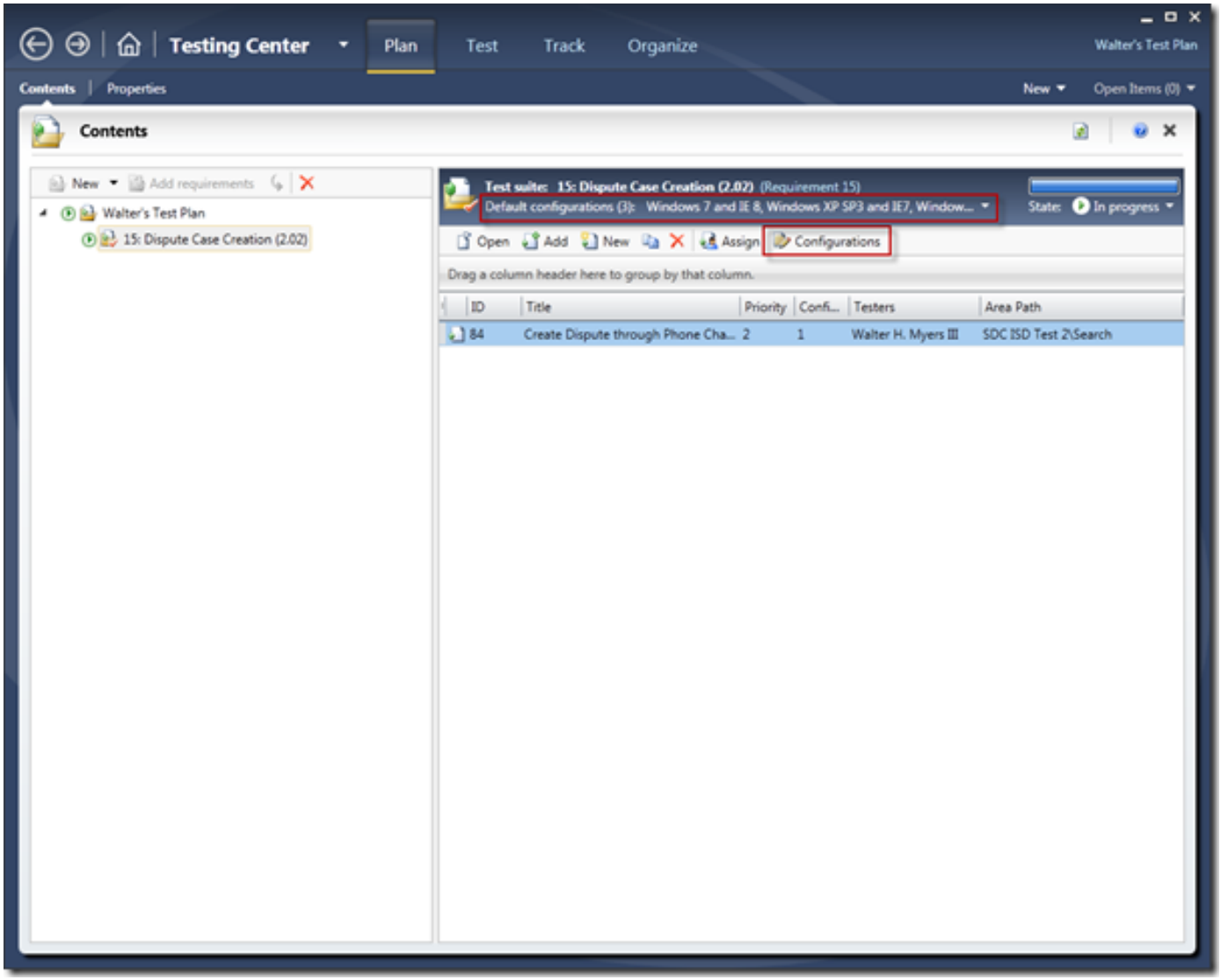


اگر در لیست کشویی مربوط به test settings مقدار <default> قرار داشت می‌توانید با انتخاب آیتم new از لیست settings جدیدی را ایجاد نمایید و یا می‌توانید لیست test settings هایی را که قبلا ایجاد کرده اید انتخاب نمایید و برای ویرایش آن با کلیک بر روی لینک open که کنار لیست قرار دارد، می‌توانید تنظیمات را ویرایش نمایید.

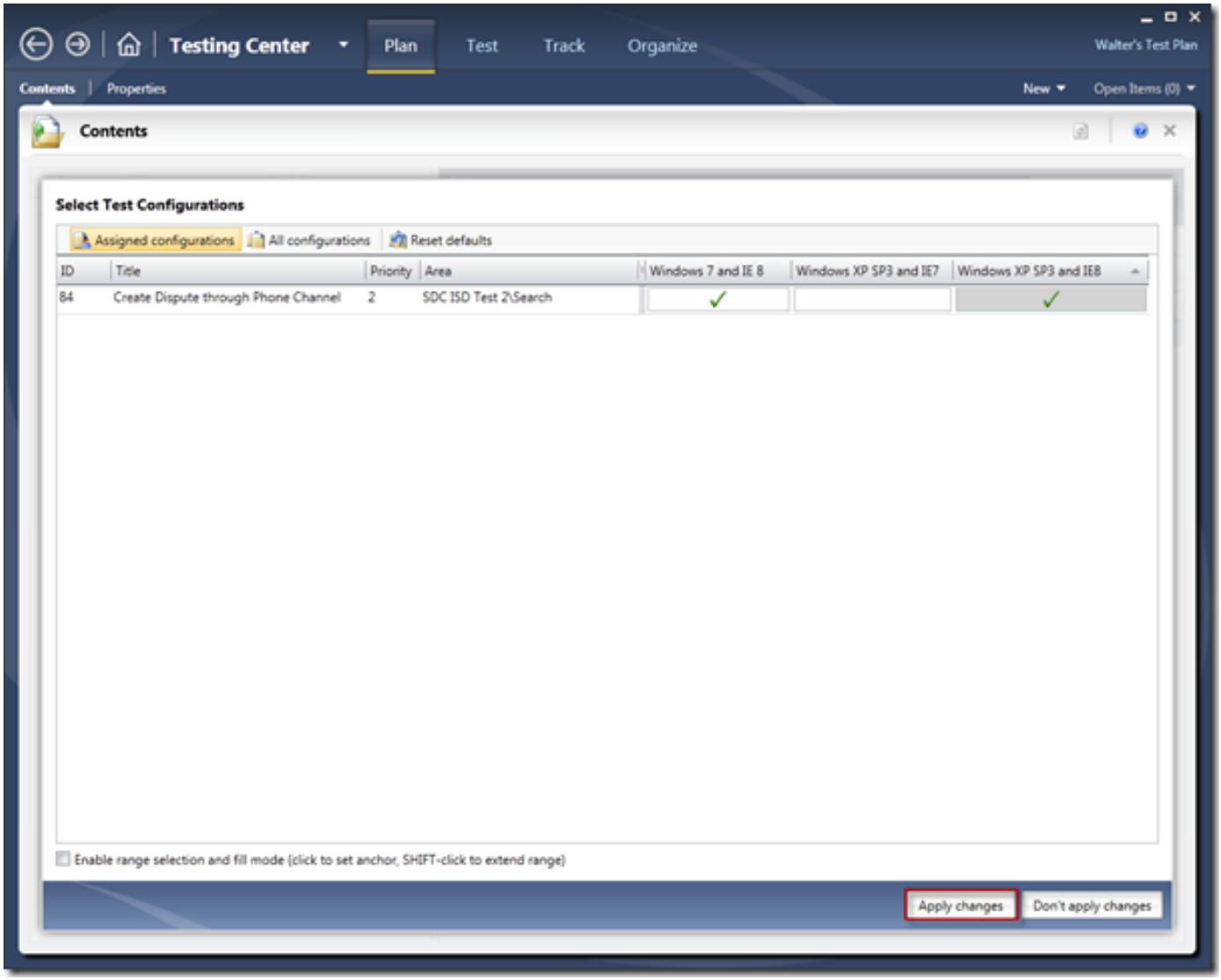


همانطور که در تصویر بالا مشاهده می‌کنید، در سمت چپ، بخش هایی برای انجام تنظیمات مربوط به تست وجود دارد. در قسمت general تنظیماتی از قبیل عنوان test settings، شرح و نوع اجرای دستی یا اتومات بودن تستتان وجود دارد. در بخش roles می‌توانید نقش هایی را برای این تست انتخاب نمایید و در قسمت data and diagnostics می‌توانید یک سری اطلاعاتی را که می‌خواهید در زمان تست دریافت کنید، انتخاب کنید. برای اطلاعات بیشتر در مورد این بخش می‌توانید در سایت مایکروسافت مطالعه کنید.

حالا بر می‌گردیم به بخش contents و موارد تست خود را می‌سازیم. همانطور که در تصویر پایین مشاهده می‌کنید در بخش contents و در سمت راست پنجره یک گزینه ای به نام configuration وجود دارد.

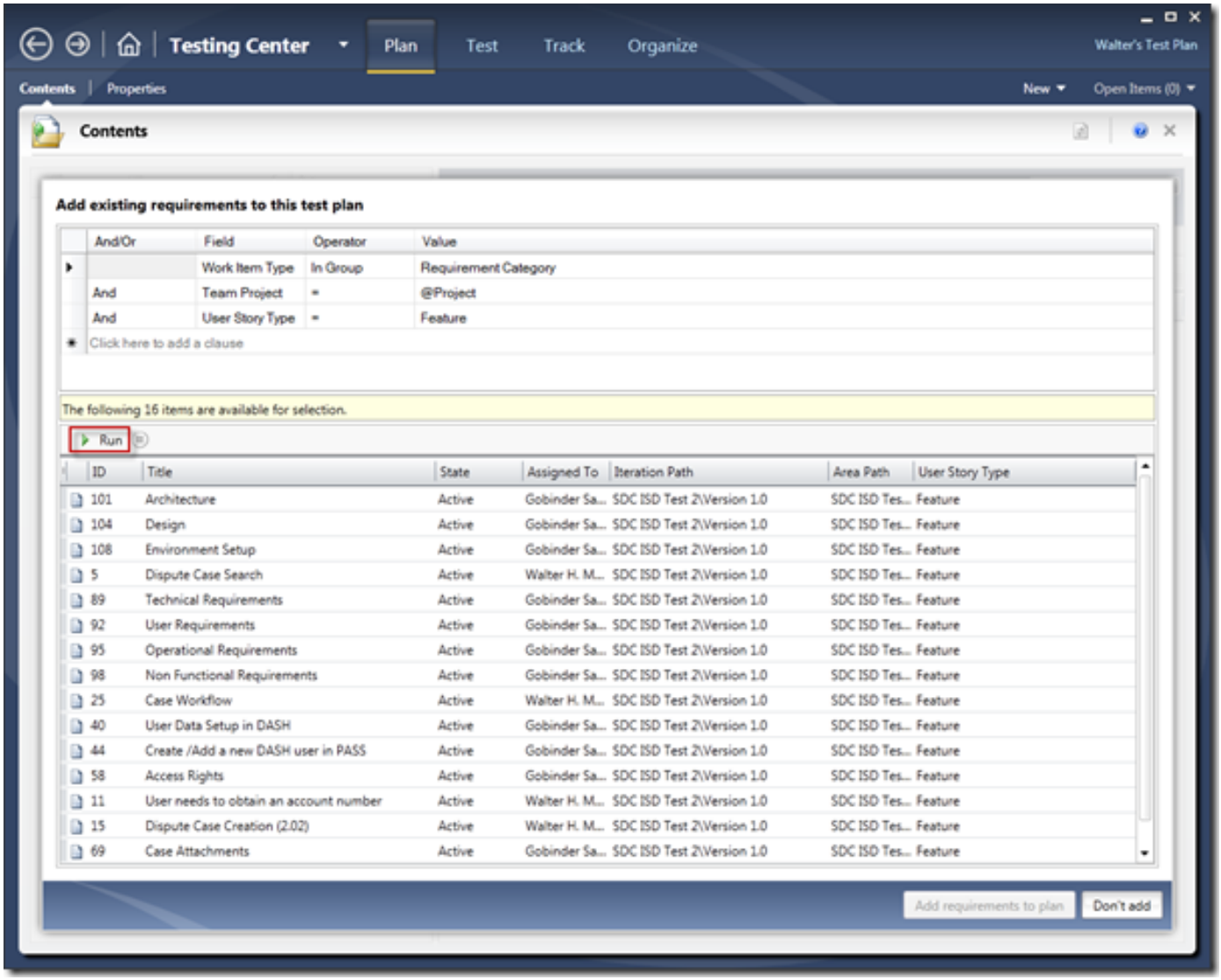


در configuration شما می‌توانید یک سری تنظیمات مربوط به test شما است انجام دهید مثلا نوع مرورگری که می‌خواهید تست خود را اجرا کنید و یا اولویت تست را مشخص نمایید یا حتی نوع سیستم عامل را مشخص کنید. هم چنین می‌توانید چندین configuration تعریف کنید و از هر کدام برای یک test suite استفاده کنید. به صورت پیش فرض test suite از تنظیمات config والد خودش یعنی test plan استفاده می‌کند.



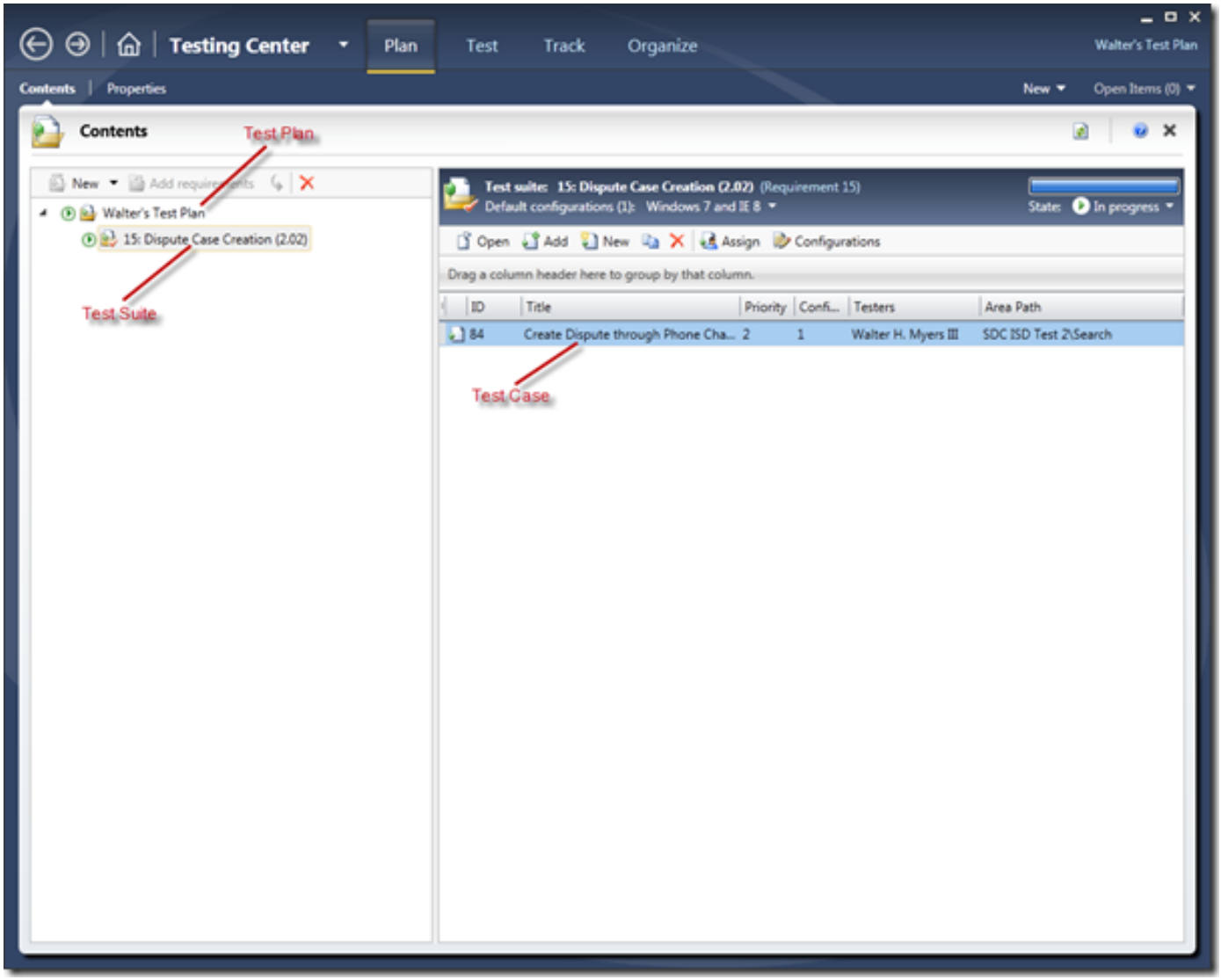
دوباره برمی گردیم به بخش contents و می‌خواهیم یک test suite با استفاده از add requirements بسازیم. همانطور که در بخش‌های قبل توضیح دادم می‌توانیم به چند روش test suite بسازیم که یکی از آن‌ها همین add requirements بود که می‌توانستید از test suite هایی که قبلا ساخته اید به این پروژه تستتان اضافه کنید.

با انتخاب گزینه add requirements پنجره ای باز می‌شود که می‌توانید همه test suiteها را مشاهده کنید و حتی می‌توانید براساس عنوان و یا وضعیت تست و ... فیلتر کنید.



بعد از اینکه در قسمت بالا کوئری خود را تنظیم کردید با انتخاب گزینه run می‌توانید کوئری خود را اجرا کرده و لیست testsuiteها را براساس آن کوئری فیلتر کنید. می‌توانید یک یا چند سطر را انتخاب کرده و با زدن دکمه add requirements to plan آن‌ها را به plan خود اضافه نمایید. حالا ما یک test suite با استفاده از test suite هایی که قبلا ساخته ایم ایجاد کردیم. حالا باید مورد تست‌های مان را به این test suite اضافه کنیم. در سمت راست با کلیک بر روی گزینه add پنجره ای مشابه پنجره بالا باز می‌شود که شما می‌توانید test caseها را فیلتر کنید و یک یا چند مورد را انتخاب کرده و با زدن دکمه add test cases آن‌ها را به test suite تان اضافه کنید. برای اضافه کردن مورد تست جدید هم می‌توانید با کلیک بر روی new که در کنار گزینه Add قرار دارد مورد تست جدیدی را بسازید.

در تصویر زیر می‌توانید بخش‌های مختلف تست را که در بخش‌های قبل هم توضیح دادم ببینید.

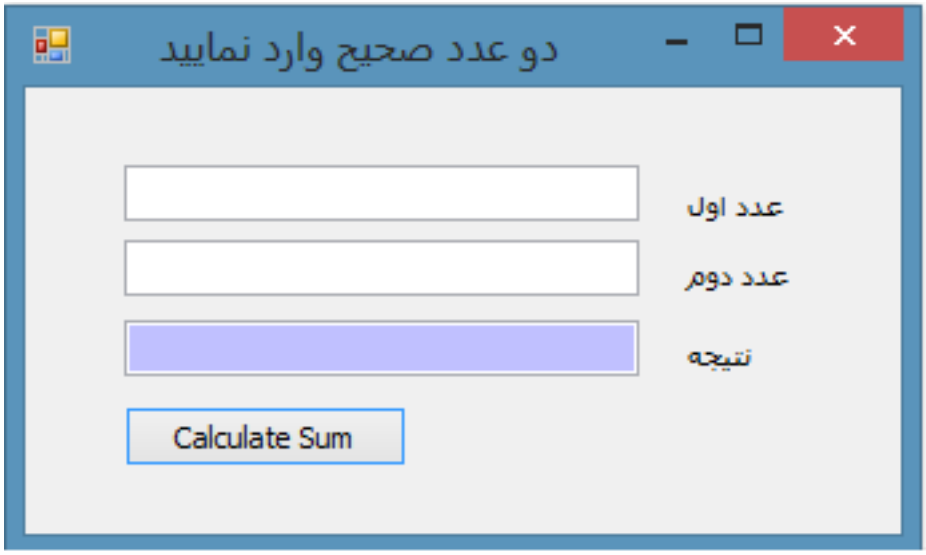

Testing Center
Plan
Test
Track
Organize
Walter's Test Plan
Contents
Properties
New
Open Items (0)
Contents
Test Plan
New
Walter's Test Plan
15: Dispute Case Creation (2.02)
Test Suite
Test suite: 15: Dispute Case Creation (2.02) (Requirement 15)
Default configurations (1): Windows 7 and IE 8
State: In progress
Open
Add
New
Assign
Configurations
Drag a column header here to group by that column.
ID
Title
Priority
Confi...
Testers
Area Path
84
Create Dispute through Phone Cha...
2
1
Walter H. Myers III
SDC ISD Test 2\Search
Test Case

عنوان: **آموزش Coded UI Test #1**
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۴/۰۷ ۸:۳۰
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Testing, Coded UI Test

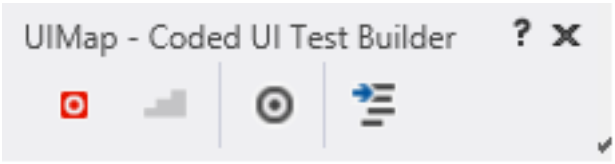
اکثر برنامه نویسان با مباحث Unit Testing آشنایی دارند و بعضی برنامه نویسان هم، از این مباحث در پروژه‌های خود استفاده می‌کنند. ساختار الگوهای MVC و MVVM به گونه ای است که به راحتی می‌توان برای این گونه پروژه‌ها Unit Test بنویسیم. در پروژه‌های MVC به دلیل عدم وابستگی بین View و Controller به طور مستقیم، امکان نوشتن Unit Test برای Controller امکان پذیر است و از طرفی در الگوی MVVM به دلیل منطق وجود ViewModel می‌توان برای اینگونه پروژه‌ها نیز Unit Test نوشت. اما ساختار سایر پروژه‌ها به گونه ای است که نوشتن Unit Test برای آن‌ها مشکل و در بعضی مواقع غیر ممکن می‌شود. برای مثال در پروژهای Desktop نظیر Windows Application و حتی وب به صورت Asp.Net Web Forms به دلیل وابستگی مستقیم کنترل‌های UI به منطق اجرای برنامه، طراحی و نوشتن Unit Test بسیار مشکل و در برخی موارد بیهوده است. در VS.Net ابزاری وجود دارد به نام Coded UI Test که برای تست این گونه پروژه‌ها طراحی شده است و همان طور که از نامش پیداست صرفا برای تست کنترل‌های UI و رویدادهای کنترل‌ها و تست درستی برنامه با توجه به داده‌های ورودی به کار می‌رود. یکی از مزیت‌های اصلی آن تسریع عملیات تست در حجم بالا است و زمان ایجاد unit test را به حداقل می‌رساند. مزیت دوم آن امکان ایجاد unit test برای پروژه‌های که در مراحل پایانی تولید هستند ولی هنوز اطمینانی به عملکرد صحیح برنامه در حالات مختلف نیست.
در این پست قصد دارم روش استفاده از این گونه پروژه‌های تست را با ذکر یک مثال بررسی کنیم و در پست‌های بعدی به بررسی امکانات دیگر خواهیم پرداخت.

نکته : فقط در Vs.Net با نسخه‌های Ultimate و Premium می‌تونید از Code UI Test استفاده کنید که البته به دلیل اینکه در ایران پیدا کردن نسخه‌های دیگر Vs.Net به غیر از Ultimate سخت‌تر است به طور قطع این محدودیت برای برنامه نویسان ما وجود نخواهد داشت. برای اینکه از نسخه Vs.Net خود اطمینان حاصل کنید از منوی Help گزینه About Microsoft Visual Studio رو انتخاب کنید. پنجره ای به شکل زیر مشاهده خواهید کرد که در آن مشخصات کامل Vs.Net ذکر شده است.

در این مرحله قصد داریم برای فرم زیر Unit Test طراحی کنیم. پروژه به صورت زیر است:



کاملا واضح است که در این فرم دو عدد به عنوان ورودی دریافت می‌شود و بعد از کلیک بر روی CalculateSum نتیجه در textbox سوم نمایش داده می‌شود. برای تست عملکرد صحیح فرم بالا ایتدا به Solution مورد نظر از منوی test Project یک Coded UI Test Project اضافه می‌کنیم. به دلیل اینکه این قبلا در این Solution پروژه تست از نوع Coded UI نبود بلافاصله یک پنجره نمایش داده می‌شود. مطمئن شوید گزینه اول انتخاب شده و بعد بر روی Ok کلیک کنید.(گزینه اول به معنی است که قصد داریم عملیات مورد نظر بر روی UI را رکورد کنیم و گزینه دوم به معنی است که قصد داریم از عملیات رکورد شده قبلی استفاده کنیم). یک کلاس به نام CodeUITest1 به همراه یک متد تست به نام CodedUITestMethod1 ساخته می‌شود. اولین چیزی که جلب توجه می‌کند این است که این کلاس به جای TestClassAttribute دارای نشان CodeUITestAttrbiute است. در گوشه سمت راست Vs.Net خود یک پنجره کوچک به نام UI Map Test Builder مانند شکل زیر خواهید دید.



دکمه قرمز رنگ به نام Record Button است و عملیات تست را رکورد خواهد کرد. دکمه دایره ای به رنگ مشکی برای تعیین Assertion به کار می‌رود. و در نهایت گزینه آخر کدهای مورد نظر مراحل قبل را به صورت خودکار تولید خواهد کرد.

#روش کار

روش کار به این صورت است که ابتدا شما مراحل تست خود را شبیه سازی خواهید کرد و بعد از آن Test Builder مراحل تست شما را به صورت کامل به صورت کدهای قابل فهم تولید خواهد کرد. (دقیقا شبیه به ایجاد UnitTest به روش Arrange/Act/Assert است با این تفاوت که این مراحل توسط UI Map رکورد شده و نیازی به کد نویسی ندارد). در پایان باید یک Data Driven Coded UI Test طراحی کنید تا بتوانید از این مراحل رکورد استفاده نمایید.

#چگونگی شبیه سازی :

پروژه را اجرا نمایید. زمانی که فرم مورد نظر ظاهر شد بر روی گزینه Record در TestBuilder کلیک کنید. عملیات ذخیره سازی شروع شده است. در نتیجه به فرم مربوطه رفته و در Textbox اول مقدار 10 و در textbox دوم مقدار 5 را وارد نمایید. با کلیک بر روی دکمه CalculateSum مقدار 15 نمایش داده خواهد شد. از برنامه خارج شوید و بعد بر روی گزینه Generate Code در TestBuilder کلیک کنید با از کلیدهای ترکیبی Alt + G استفاده نمایید.(اگر در این مرحله، از برنامه خارج نشده باشید با خطا مواجه خواهید شد.) در پنجره نمایش داده شده یک نام به متد اختصاص دهید. عملیات تولید کد شروع خواهد شد. بعد کدی مشابه زیر را در متد مربوطه مشاهده خواهید کرد.

```
[TestMethod]
        public void CodedUITestMethod1()
        {
            this.UIMap.CalculateSum();
            this.UIMap.txtSecondValueMustBe10();
        }
```

بخشی از سورس کد تولید شده برای متد CalculateSum به شکل زیر است:

```
public void CodedUITestMethod1
```

```
()
        {
            #region Variable Declarations
            WinEdit uITxtFirstNumberEdit =
this.UIدوعددصحیحواردنماییدWindow.UITxtFirstNumberWindow.UITxtFirstNumberEdit;
            WinEdit uITxtSecondNumberEdit =
this.UIدوعددصحیحواردنماییدWindow.UITxtSecondNumberWindow.UITxtSecondNumberEdit;
            WinButton uICalculateSumButton =
this.UIدوعددصحیحواردنماییدWindow.UICalculateSumWindow.UICalculateSumButton;
            #endregion

            // Type '10' in 'txtFirstNumber' text box
            uITxtFirstNumberEdit.Text = this.CalculateSumParams.UITxtFirstNumberEditText;

            // Type '{Tab}' in 'txtFirstNumber' text box
            Keyboard.SendKeys(uITxtFirstNumberEdit,
this.CalculateSumParams.UITxtFirstNumberEditSendKeys, ModifierKeys.None);

            // Type '10' in 'txtSecondNumber' text box
            uITxtSecondNumberEdit.Text = this.CalculateSumParams.UITxtSecondNumberEditText;

            // Click 'Calculate Sum' button
            Mouse.Click(uICalculateSumButton, new Point(83, 12));

            // Type '10' in 'txtFirstNumber' text box
            uITxtFirstNumberEdit.Text = this.CalculateSumParams.UITxtFirstNumberEditText1;

            // Type '{Tab}' in 'txtFirstNumber' text box
            Keyboard.SendKeys(uITxtFirstNumberEdit,
this.CalculateSumParams.UITxtFirstNumberEditSendKeys1, ModifierKeys.None);

            // Type '10' in 'txtSecondNumber' text box
            uITxtSecondNumberEdit.Text = this.CalculateSumParams.UITxtSecondNumberEditText1;

            // Type '{Tab}' in 'txtSecondNumber' text box
            Keyboard.SendKeys(uITxtSecondNumberEdit,
this.CalculateSumParams.UITxtSecondNumberEditSendKeys, ModifierKeys.None);

            // Click 'Calculate Sum' button
            Mouse.Click(uICalculateSumButton, new Point(49, 11));

            // Type '10' in 'txtFirstNumber' text box
            uITxtFirstNumberEdit.Text = this.CalculateSumParams.UITxtFirstNumberEditText2;

            // Type '{Tab}' in 'txtFirstNumber' text box
            Keyboard.SendKeys(uITxtFirstNumberEdit,
this.CalculateSumParams.UITxtFirstNumberEditSendKeys2, ModifierKeys.None);

            // Type '5' in 'txtSecondNumber' text box
            uITxtSecondNumberEdit.Text = this.CalculateSumParams.UITxtSecondNumberEditText2;

            // Type '{Tab}' in 'txtSecondNumber' text box
            Keyboard.SendKeys(uITxtSecondNumberEdit,
```

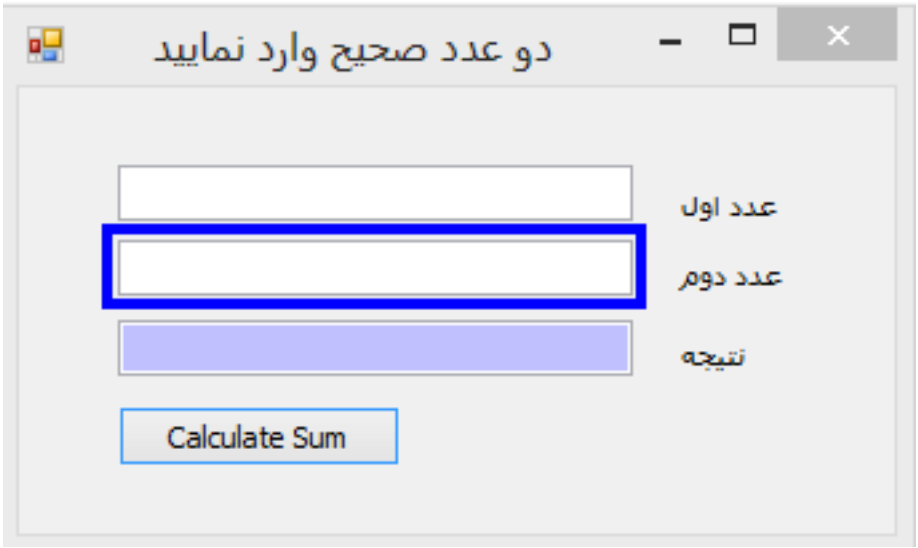
```
this.CalculateSumParams.UITxtSecondNumberEditSendKeys1, ModifierKeys.None);

            // Click 'Calculate Sum' button
            Mouse.Click(uICalculateSumButton, new Point(74, 16));
        }
```

همان طور که می‌بینید تمام مراحل تست شما رکورد شده است و به صورت کد قابل فهم بالا ایجاد شده است.

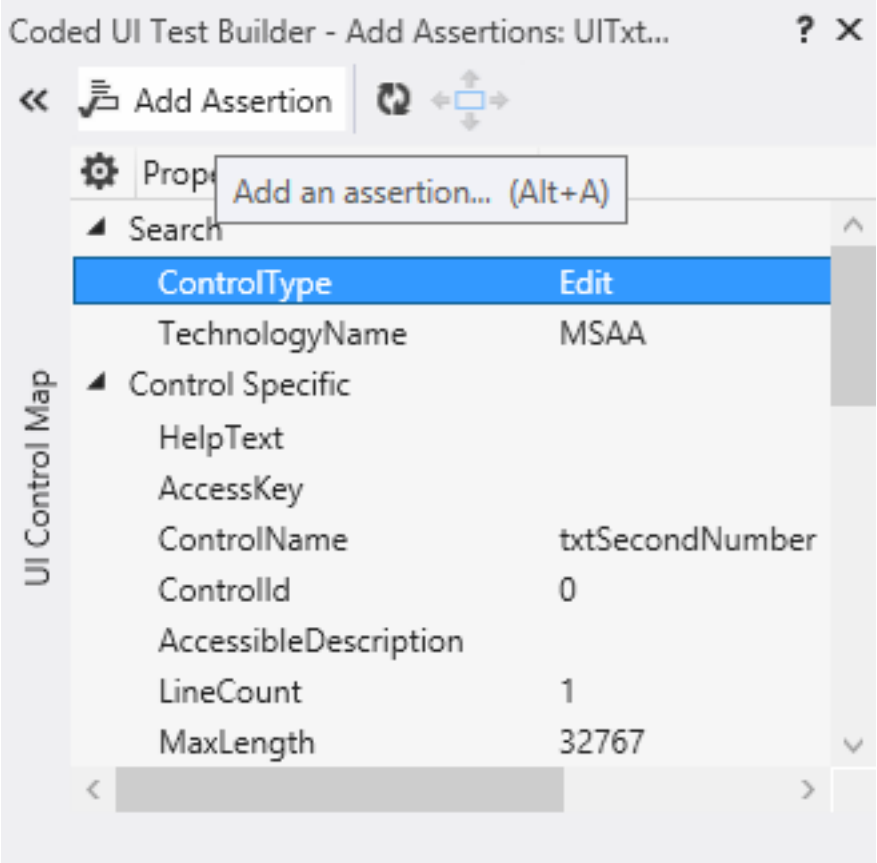
چگونگی ایجاد Assertion

اگر به کد متد تست CodedUITestMethod1 در بالا دقت کنید یک متد به صورت this.UIMap.txtSecondValueMustBe10 فراخوانی شده است. این در واقع یک Assertion است که در هنگام عملیات رکورد ایجاد کردم و به این معنی است که مقدار TextBox دوم حتما باید 10 باشد. حال روش تولید Assertionها را بررسی خواهیم کرد.

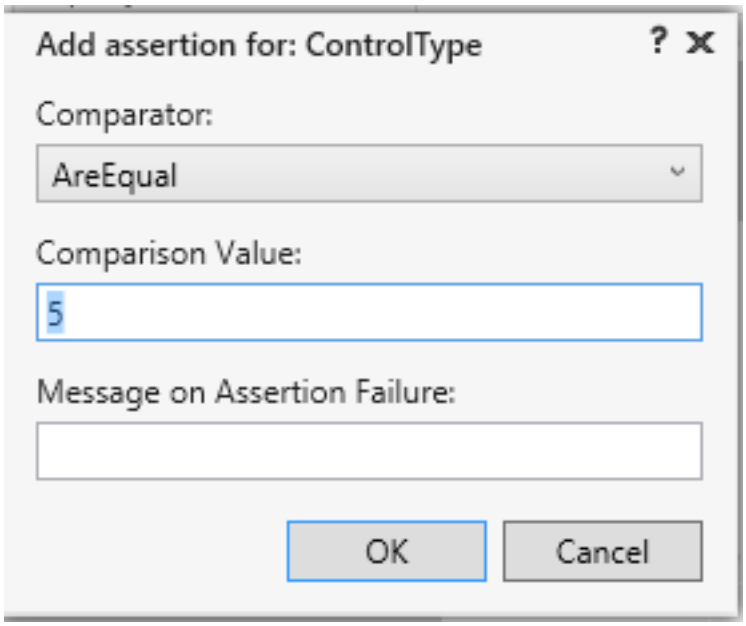
بعد از شروع شدن مرحله رکورد اگر قصد دارید برای یک کنترل خاص Assert بنویسید، دکمه assertion (به رنگ مشکی و به صورت دایره است) را بر روی کنترل مورد نظر drag&drop کنید. یک border آبی برای کنترل مورد نظر ایجاد خواهد شد:



به محض اتمام عملیات drag&drop منوی زیر ظاهر خواهد شد:



از گزینه Add Assertion استفاده کنید و برای کنترل مورد نظر یک assert بنویسید. در شکل زیر یک assert برای textbox دوم نوشتم به صورتی که مقدار آن باید با 5 برابر باشد.



از گزینه آخر برای نمایش پیغام مورد نظر خودتون در هنگامی که aseert با شکست مواجه می‌شود استفاده کنید.
کد تولید شده زیر برای عملیات assert بالا است:

```
public void txtSecondValueMustBe10()
        {
            #region Variable Declarations
            WinEdit uITxtSecondNumberEdit =
this.UIدوعددصحیحواردنماییدWindow.UITxtSecondNumberWindow.UITxtSecondNumberEdit;
            #endregion

            // Verify that the 'ControlType' property of 'txtSecondNumber' text box equals '10'
            Assert.AreEqual(this.txtSecondValueMustBe10ExpectedValues.UITxtSecondNumberEditControlType,
uITxtSecondNumberEdit.ControlType.ToString());
        }
```

مرحله اول انجام شد. برای تست این مراحل باید یک Data DrivenTest بسازید که در پست بعدی به صورت کامل شرح داده خواهد شد.