عنوان: **WatIn - Web Application Testing in .Net**
نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۱/۲۴ ۲۱:۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, Integration tests, test

معرفی:

امروزه تست کردن کدها به دلیل وجود ابزارهای مختلف زیادی، کار آسانی شده است. اما بعضی‌ها در web application ها، یکی از تست‌هایی را که خیلی هم مهم است را فراموش می‌کنند که آن هم تست UI است. شما را در این مقاله با یکی از روش‌های خوب تست UI آشنا خواهم کرد. ابزارهای زیادی برای تست UI وجود دارد که کار کردن با آنها نه تنها زمان بر بلکه بسیار خسته کننده می‌باشند و به خاطر همین خیلی‌ها از انجام تست UI صرف نظر می‌کنند.

WatIn چیست؟

WatIn مخفف Web Application Testing in .Net می‌باشد؛ که یک فریم ورک تست web applicationها است. WatIn این اجازه را به شما می‌دهد که با استفاده از IE ویا FireFox عناصر داخل صفحات را مقدار دهی کنید و یا حتی رویدادی را برای عناصر فراخوانی کنید.

شروع کار با WatIn:

در زیر یک نمونه از کار با WatIn را می‌توانید مشاهده کنید:

```
[TestMethod]
public void SearchForWatiNOnGoogle()
{
  using (var browser = new IE("http://www.google.com"))
  {
    browser.TextField(Find.ByName("q")).TypeText("WatiN");
    browser.Button(Find.ByName("btnG")).Click();
    Assert.IsTrue(browser.ContainsText("WatiN"));
  }
}
```

WatIn یک فریم ورک کاربر پسند است و در ادامه متوجه می‌شوید که استفاده از این فریم ورک چه مزایایی دارد. برای نصب، WatIn را می‌توانید از اینجا دانلود کنید ویا اگر خواستید می‌توانید با NuGet هم این فریم ورک را دانلود کرده و نصب نمایید. برای شروع کار با Watin باید reference هایی را به پروژه تان اضافه کنید که یکی از این referenceها WatiN.Core.dll می‌باشد و برای استفاده از IE ویا FireFox باید فضای نام Watin.Core را اضافه کنیم. Watin چند فضای نام دیگری را هم به همراه دارد که در زیر به توضیح مختصری از آن‌ها می‌پردازیم:

1- **Watin.Core.DialogHandlers:** این فضای نام این امکان را به شما می‌دهد تا دیالوگ هایی را که مرورگر می‌تواند به کاربر نمایش دهد، مدیریت کنید. از handlerهای این فضای نام، AlertDialogHandler، ConfirmDialogHandler، FileUploadDialogHandler، PrintDialogHandler و LoginDialogHandler می‌باشد.

2- **Watin.Core.Exceptions:** این فضای نام دارای یک سری exception می‌باشد و این امکان را به ما می‌دهد تا یک سری رفتارهای ناخواسته را کنترل کنیم. بعضی از این exceptionها، IENotFoundException، ElementNotFoundException و TimeoutException و WatinException می‌باشد.

3- **Watin.Core.Logging:** این فضای نام کلاس هایی را در اختیار ما می‌گذارد تا بتوانیم عملیاتی را که در کدمان انجام می‌دهیم log کنیم.

مثالی از watin که در بالا نشان دادیم به این صورت عمل می‌کند که مرورگر IE را باز کرده و به سایت google خواهد رفت. در این صفحه جعبه متنی یا TextBox با نام "q" را پیدا کرده و عبارت "Watin" را در آن تایپ می‌کند و همچنین Buttonی با نام "btnG" پیدا کرده و آن را کلیک می‌نماید و در آخر بررسی می‌کند که در مرورگر متنی شامل WatIn وجود دارد یا خیر.

مشاهده کردید که به همین سادگی یک تست UI نوشتیم. به نظر شما جالب نبود؟ فرض کنید که اگر می‌خواستید با مثلا Microsoft Test Manager این کار را انجام دهید چه دردسرهایی را باید تحمل می‌کردید. حالا تست UI برای همه برنامه نویس‌ها جذاب خواهد شد.

به جای مثال بالا می‌توانیم به صورت زیر هم عمل کنیم:

```
[TestMethod]
public void SearchForWatiNOnGoogle()
{
  using (var browser = new IE("http://www.google.com"))
  {
    browser.TextField(Find.ByName("q")).Value="WatiN";
    browser.Button(Find.ByName("btnG")).ClickNoWait();
    Thread.Sleep(3000);
    Assert.IsTrue(browser.ContainsText("WatiN"));
  }
}
```

تفاوت کد دوم با کد اول این است چون در کد اول از متد TypeText استفاده کردیم یک مقدار سرعت تست را پایین می‌آورد ولی اگر از Value ویا از SetAttribute استفاده کنیم دیگر عمل تایپ را انجام نداده و مقدار را مستقیما در مقدار TextField قرار می‌دهد. شاید بپرسید چرا بعد از متد ClickNoWait چند ثانیه صبر می‌کنم؟ چون صفحه برای اینکه بارگذاری شود و نتیجه جستجو را نشان دهد کمی طول کشیده و Assert.IsTrue شما Failed می‌شود. البته به جای Thread.Sleep می‌توانیم از متدهای مربوط به Watin هم استفاده کنیم مانند WaitUntilComplete ویا از WaitUntilContainsText.

نظرات خوانندگان

نویسنده: آرش خوشبخت
تاریخ: ۱۳۹۲/۰۱/۲۴ ۲۱:۳۰

اگر سوالی یا مشکلی در کارکردن با این فریم ورک داشتین می‌توانید اینجا بپرسین

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:

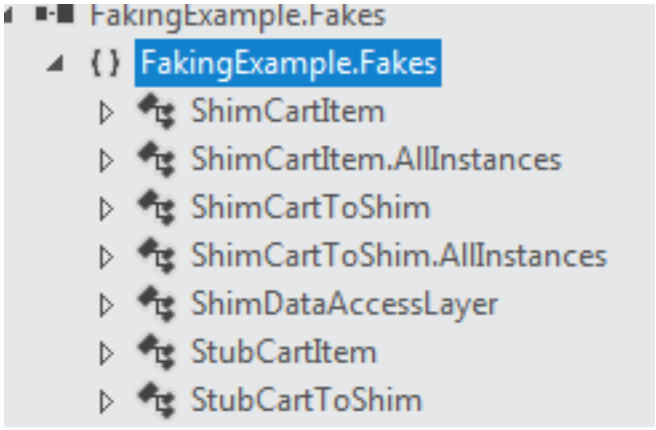
```
namespace FakingExample
{
    public class CartToShim
    {
        public int CartId { get; private set; }
        public int UserId { get; private set; }
        private List<CartItem> _cartItems = new List<CartItem>();
        public ReadOnlyCollection<CartItem> CartItems { get; private set; }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

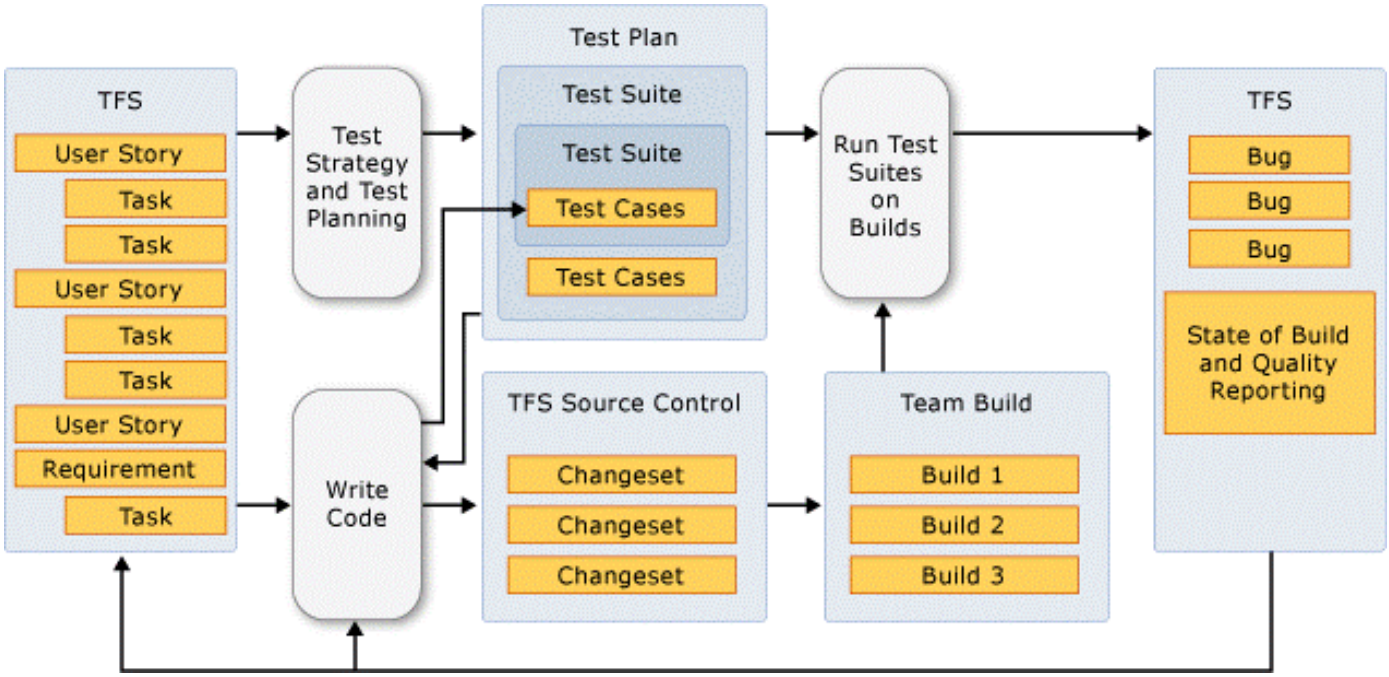
استراتژی تست:

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.

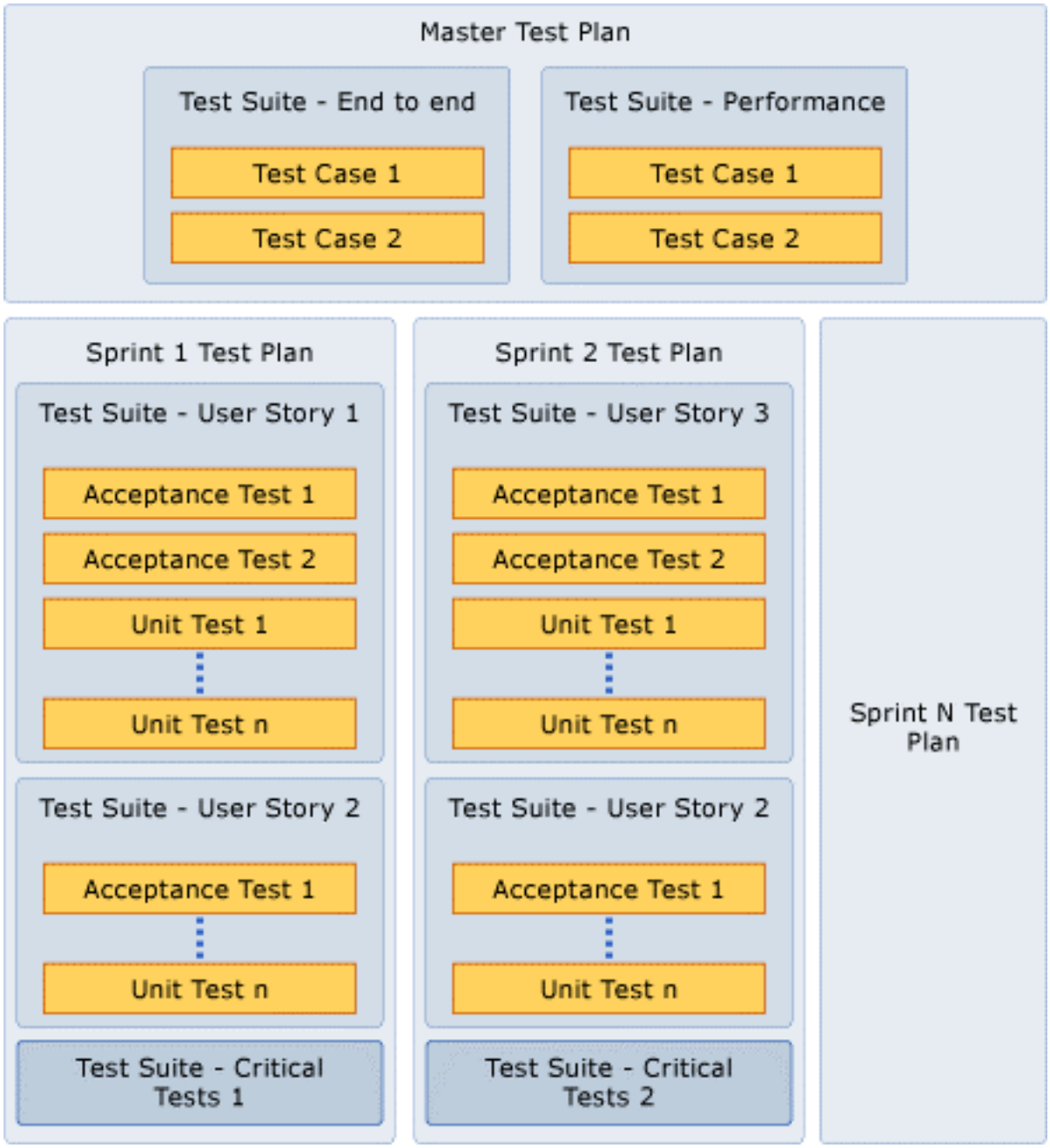
سپس باید کدهایی که برنامه نویسان می‌نویسند قابلیت تست را داشته باشند و شما به عنوان یک تستر باید آشنایی کاملی از ساختار و الگوهای برنامه تان داشته باشید.

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.

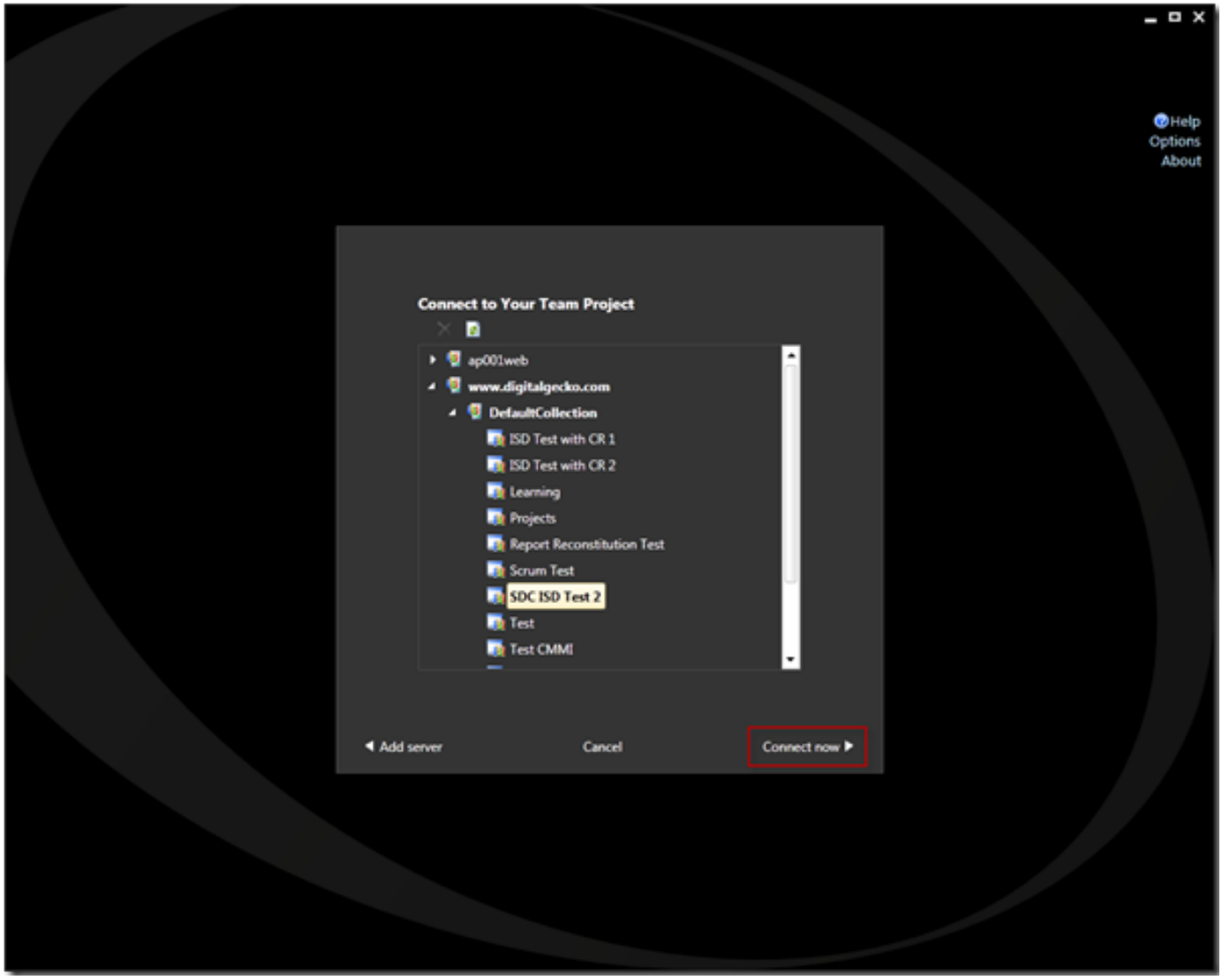
در MTM سه لایه وجود دارد:

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

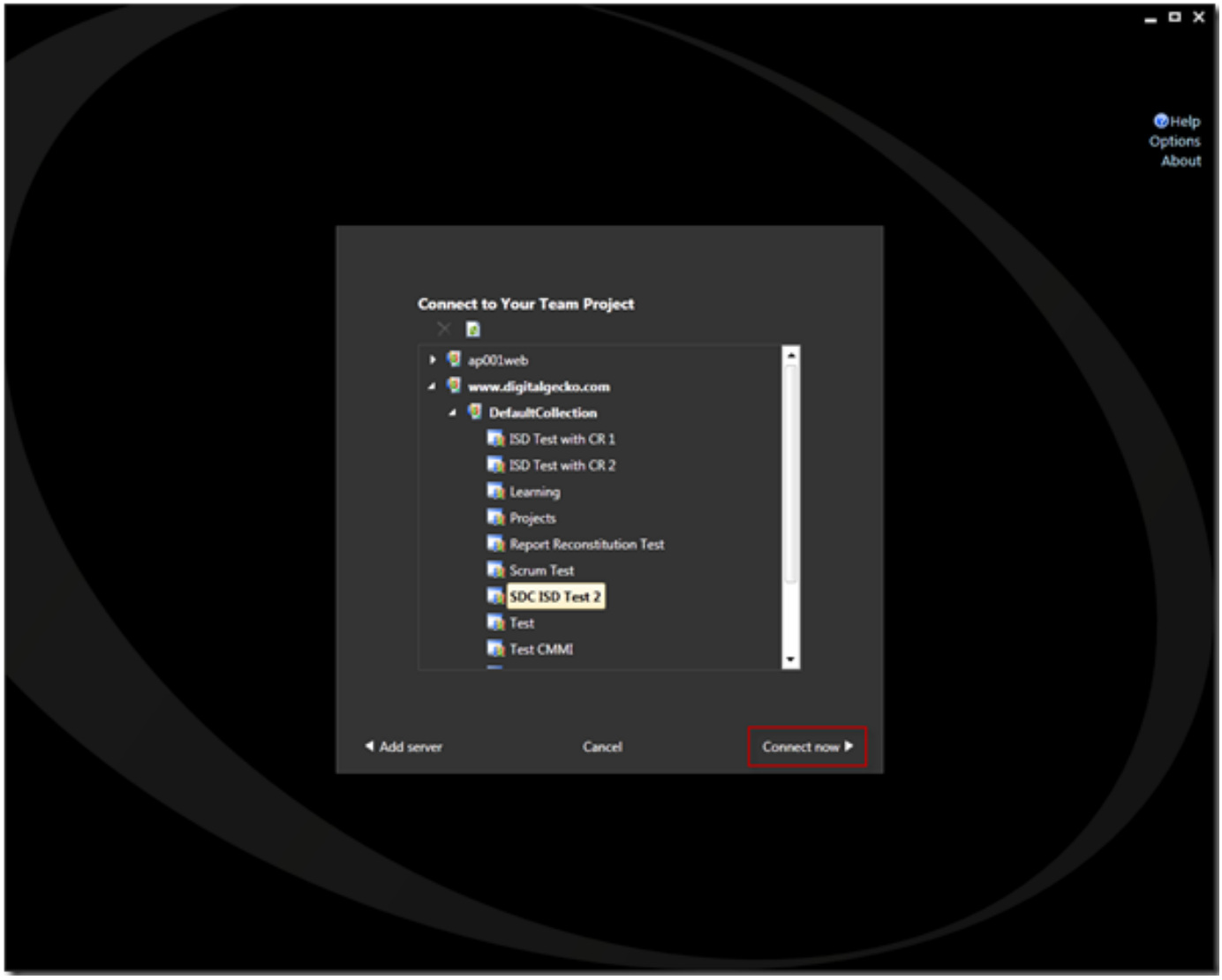
2- **Test Suite :** در هر Test Plan شما می‌توانید چندین Test Suite ایجاد کنید.

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

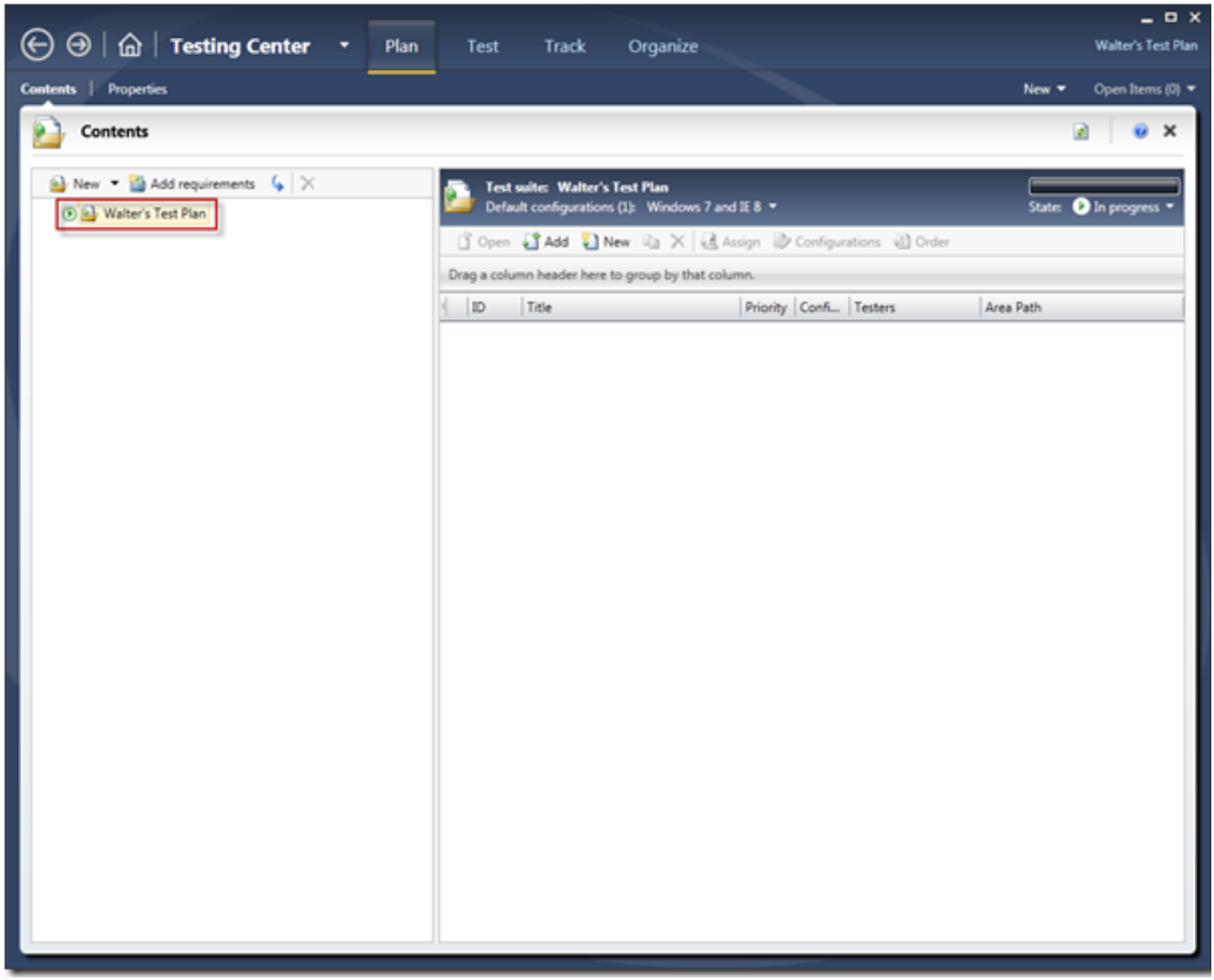
برای اولین بار که شما MTM را اجرا می‌کنید باید team project ی را که قرار است برای آن تست بنویسید را انتخاب کنید.
می‌توانید در زیر نمایی از MTM و اتصال به team project را مشاهده کنید:



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.



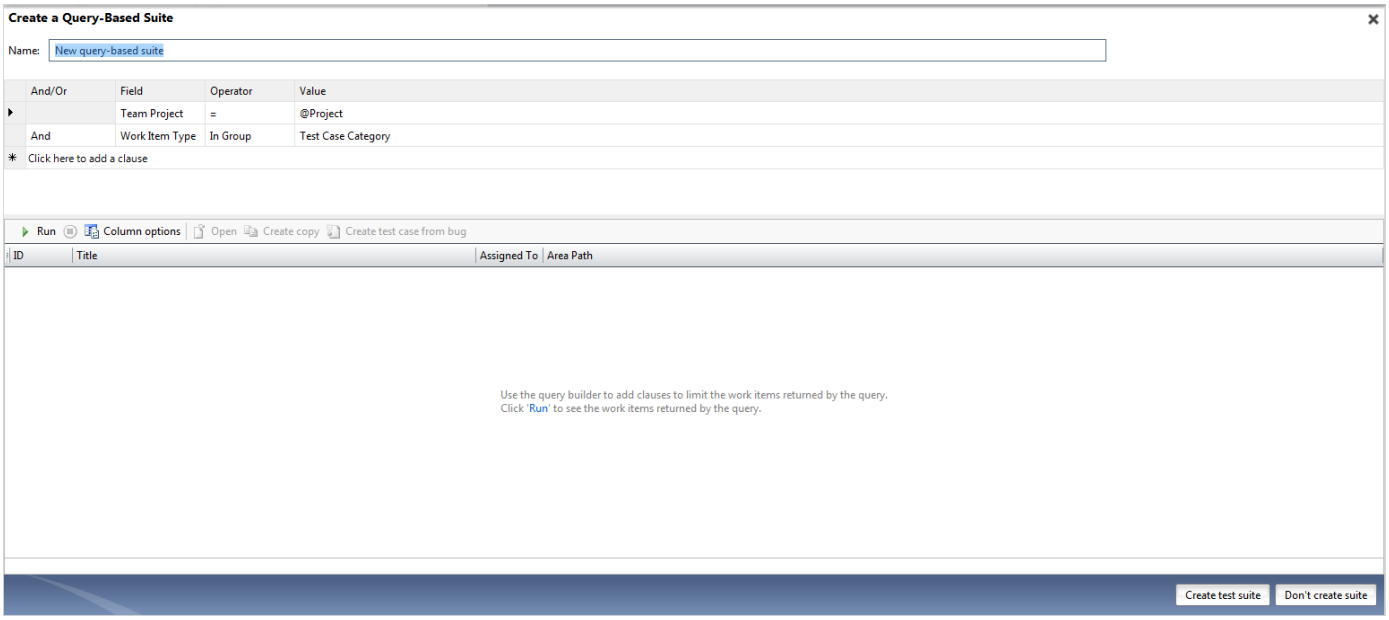
وقتی plan یا طرحی را انتخاب می‌کنید به صفحه testing center وارد می‌شوید که به صورت پیش فرض در کاربرگ plan و بخش contents قرار دارید.



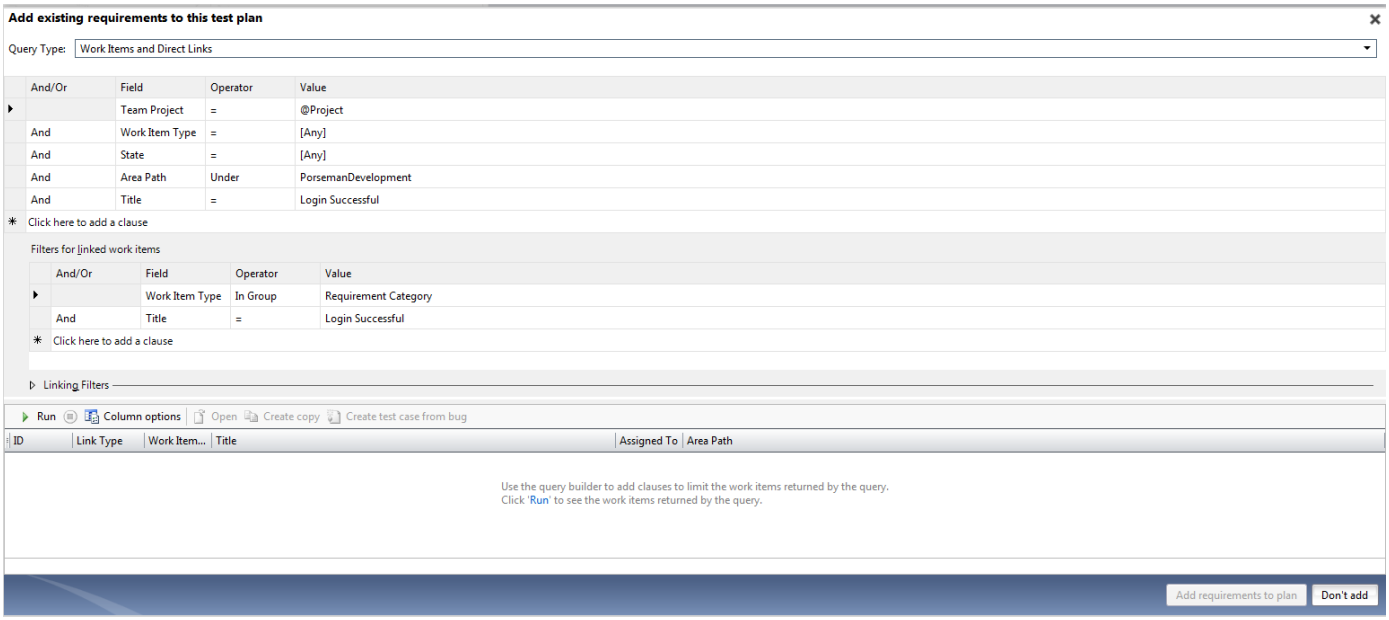
همانطور که در تصویر بالا مشاهده می‌کنید و در سمت چپ پنجره، plan شما در ریشه قرار دارد و test suite هایی را که ایجاد می‌کنید به عنوان فرزندان plan تان قرار می‌گیرند. در سمت راست test caseهای شما قرار می‌گیرند که با توجه به test suite ی که شما در سمت چپ انتخاب کرده اید test caseهای مربوط به آن در سمت راست 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

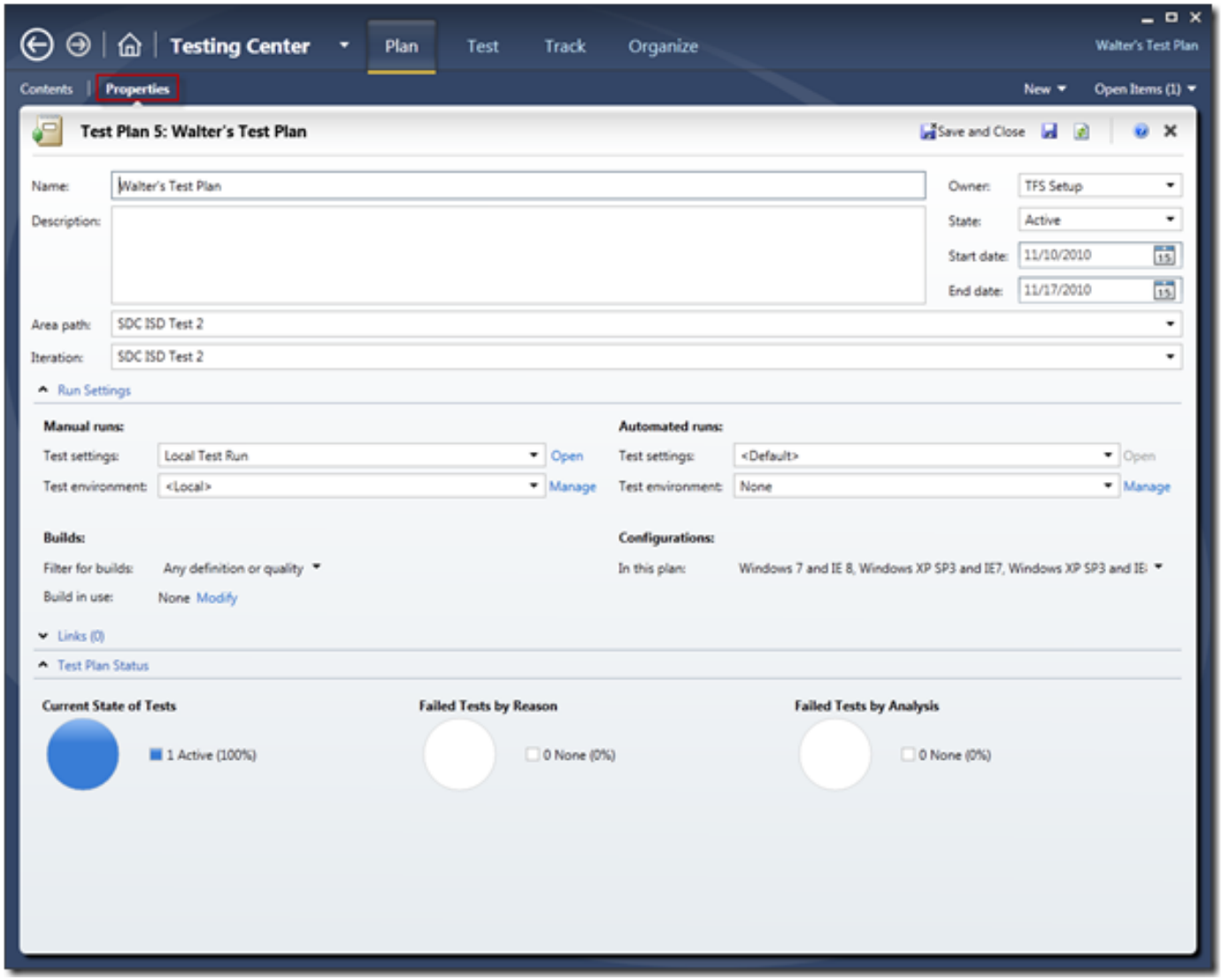
ممنونم. فقط اینکه تو هر پست مطلب رو کاملتر و قسمت بیشتری رو بگین تا در تعداد پست کمتری بشه همه چیز رو گفت و هم اینکه خواننده تا پست بعدی، خیلی از مطلب دور نشه. بازم بابت مطلب بسیار مهمی که دارین آموزش میدین خیلی خیلی ممنون.

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

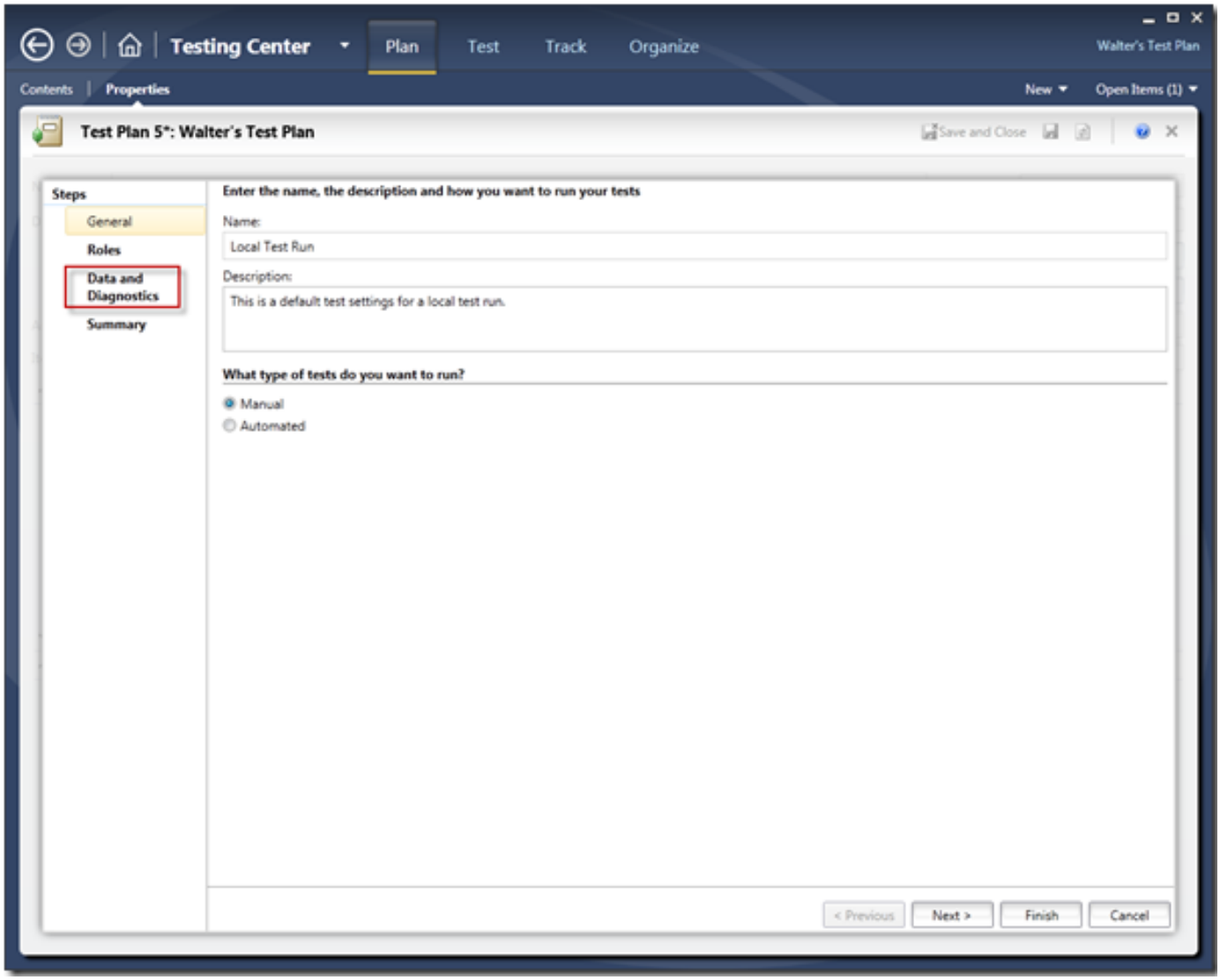
دلیل اینکه مطلب زیاد نمیزارم چون می‌گم شاید کاربران خسته شن یا حوصله‌ی خوندن مطلب زیاد رو نداشته باشن و بعد اینکه مبحث جدیدی که بخواد شروع بشه مجمبورم قسمت قبل رو قطع کنم قسمت بعدی در مورد یک سری تنظیمات در MTM است و ربطی به این بخش نداره

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.

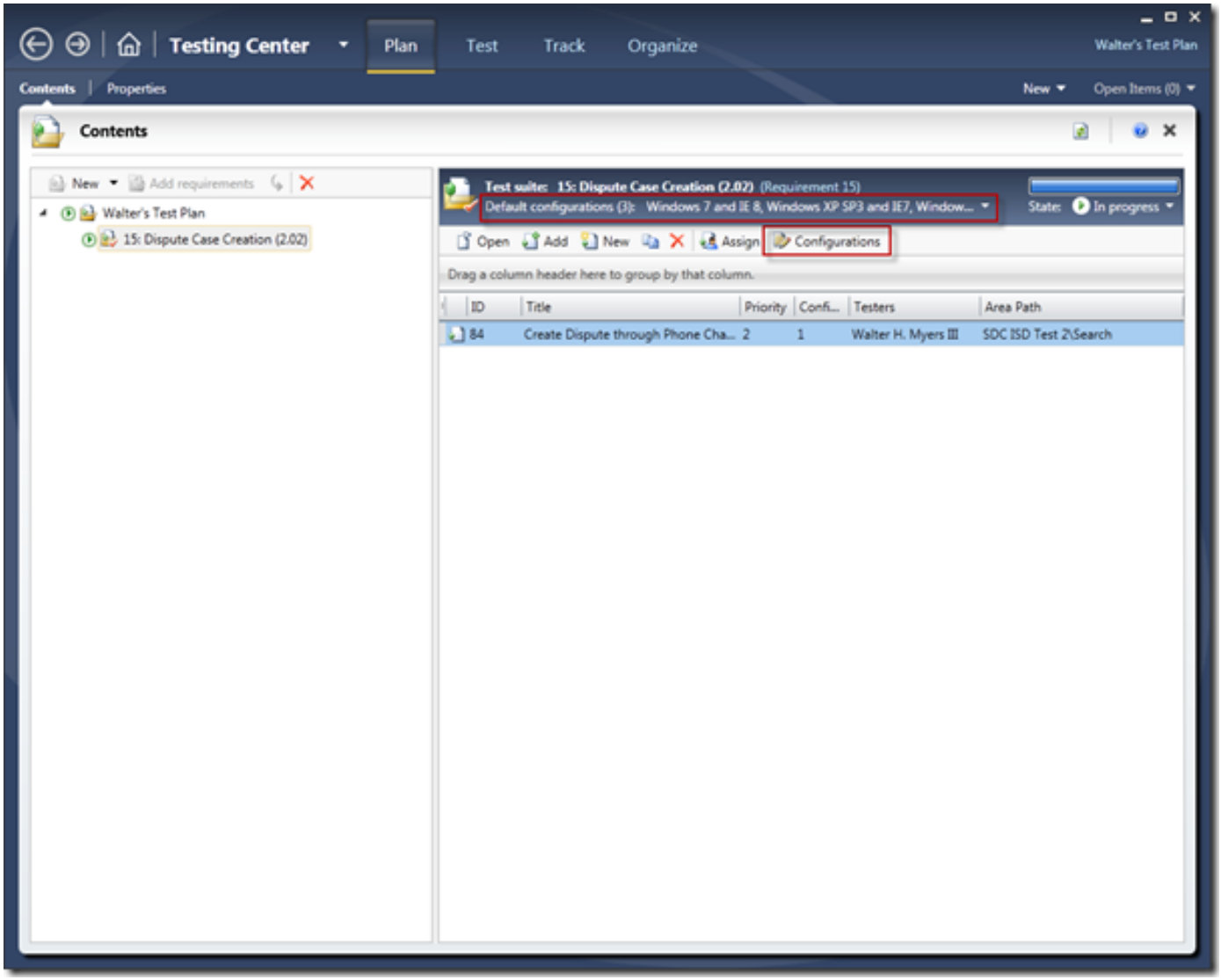


اگر در لیست کشویی مربوط به test settings مقدار <default> قرار داشت می‌توانید با انتخاب آیتم new از لیست settings جدیدی را ایجاد نمایید و یا می‌توانید لیست test settings هایی را که قبلا ایجاد کرده اید انتخاب نمایید و برای ویرایش آن با کلیک بر روی لینک open که کنار لیست قرار دارد، می‌توانید تنظیمات را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

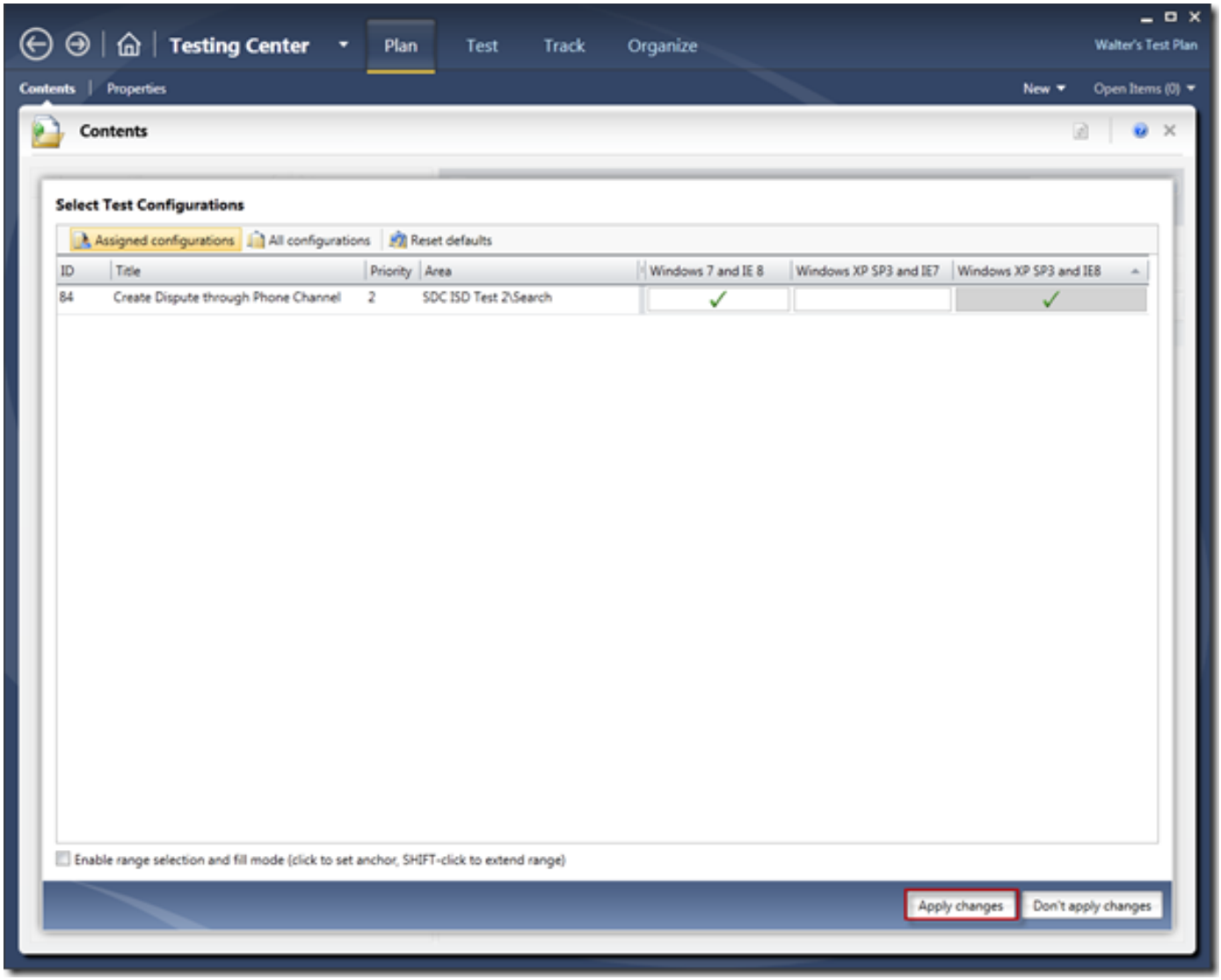


همانطور که در تصویر بالا مشاهده می‌کنید، در سمت چپ، بخش هایی برای انجام تنظیمات مربوط به تست وجود دارد. در قسمت general تنظیماتی از قبیل عنوان test settings، شرح و نوع اجرای دستی یا اتومات بودن تستتان وجود دارد. در بخش roles می‌توانید نقش هایی را برای این تست انتخاب نمایید و در قسمت data and diagnostics 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.

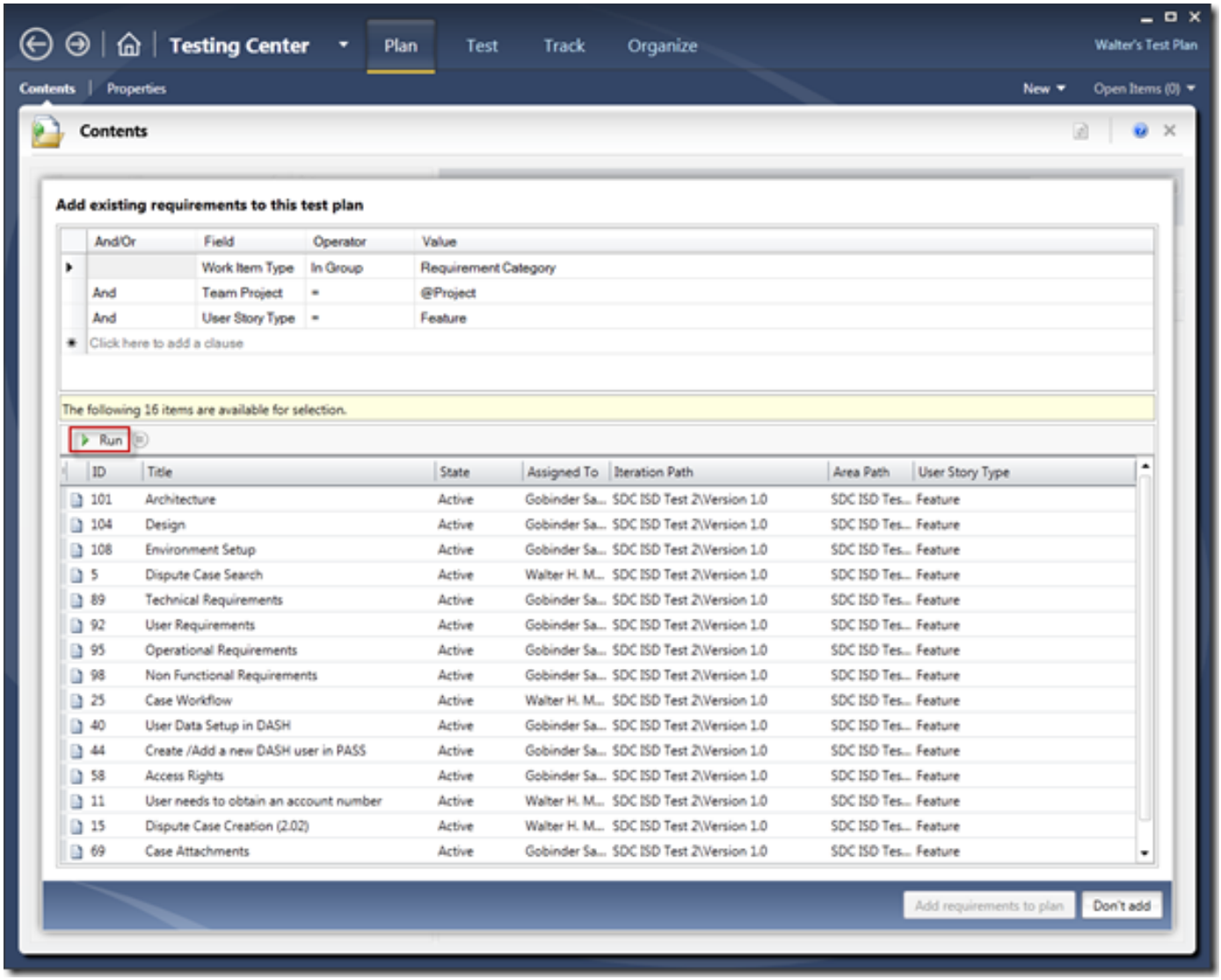


در configuration شما می‌توانید یک سری تنظیمات مربوط به test شما است انجام دهید مثلا نوع مرورگری که می‌خواهید تست خود را اجرا کنید و یا اولویت تست را مشخص نمایید یا حتی نوع سیستم عامل را مشخص کنید. هم چنین می‌توانید چندین configuration تعریف کنید و از هر کدام برای یک test suite استفاده کنید. به صورت پیش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



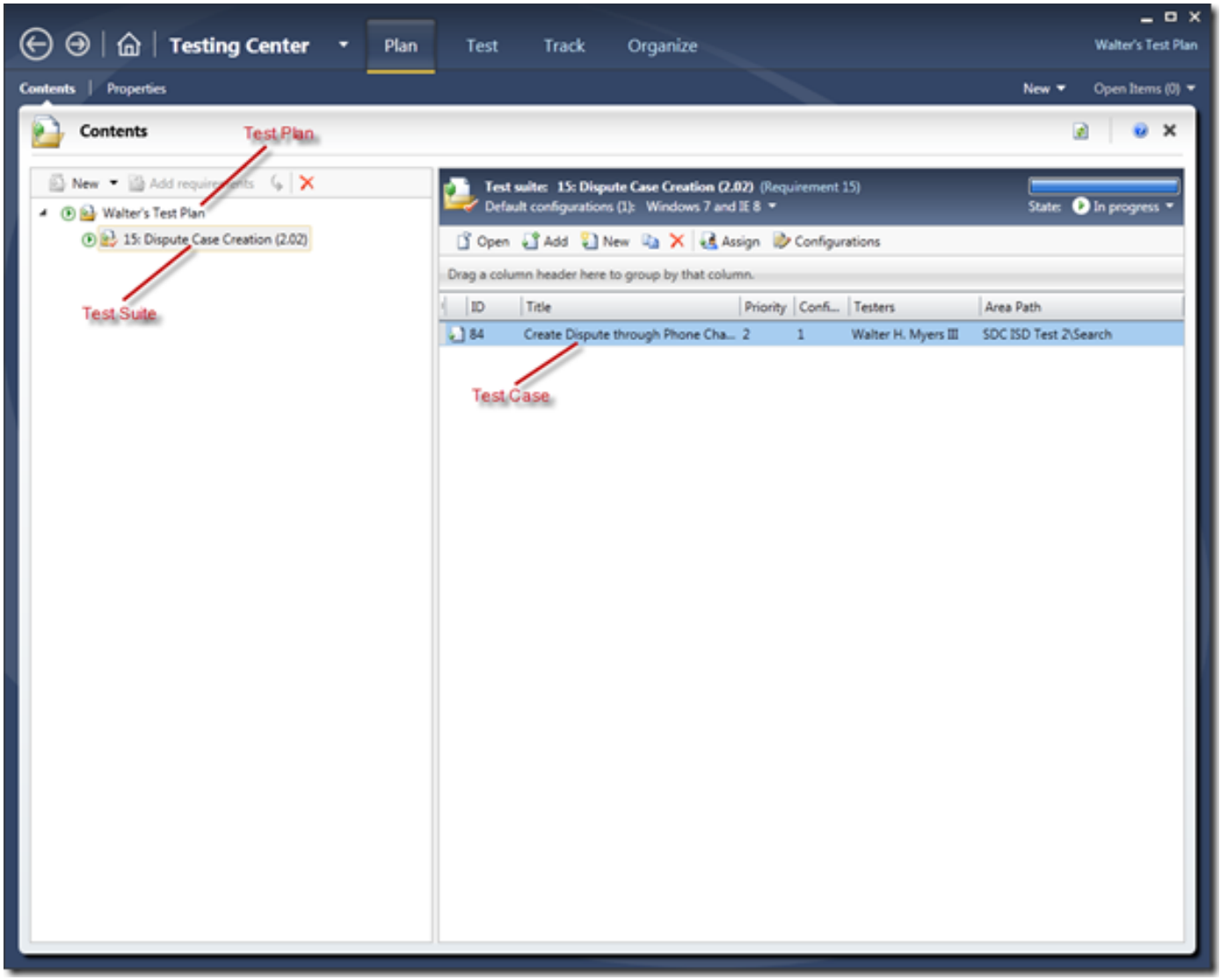
دوباره برمی گردیم به بخش contents و می‌خواهیم یک test suite با استفاده از add requirements بسازیم. همانطور که در بخش‌های قبل توضیح دادم می‌توانیم به چند روش test suite بسازیم که یکی از آن‌ها همین add requirements بود که می‌توانستید از test suite هایی که قبلا ساخته اید به این پروژه تستتان اضافه کنید.

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: آماده سازی Jasmine برای پروژه های Asp.Net MVC
نویسنده: مسعود پاکدل
تاریخ: ۱۳۹۲/۰۷/۰۸ ۹:۴۵
آدرس: www.dotnettips.info
برچسب‌ها: Unit testing, test, TDD, JavaScriptFrameWork, Jasmine

با گسترش روز افزون برنامه‌های تحت وب، نیاز به یک سری ابزار برای تست و اطمینان از نحوه عملکرد صحیح کدهای نوشته شده احساس می‌شود. Jasmine یکی از این ابزارهای قدرتمند برای تست کدهای JavaScript است.
چندی پیش در سایت جاری چند مقاله خوب توسط یکی از دوستان درباره Qunit منتشر شد. Qunit یک ابزار قدرتمند و مناسب برای تست کدهای جاوااسکریپت است و در اثبات صحت این گفته همین کافیست که بدانیم برای تست کدهای نوشته شده در پروژه‌های متن بازی هم چون Backbone.Js و JQuery از این فریم ورک استفاده شده است. اما به احتمال قوی در ذهن شما این سوال مطرح شده است که خب! در صورت آشنایی با Qunit چه نیاز به یادگیری Jasmine یا خدای نکرده Mocha و FuncUnit است؟ هدف صرفا معرفی یک ابزار غیر برای تست کد است نه مقایسه و نتیجه گیری برای تعیین میزان برتری این ابزارها. اصولا مهم‌ترین دلیل برای انتخاب، علاوه بر امکانات و انعطاف پذیری، فاکتور راحتی و آسان بودن در هنگام استفاده است که به صورت مستقیم به شما و تیم توسعه نرم افزار بستگی دارد.

اما به عنوان توسعه دهنده نرم افزار که قرار است از این ابزار استفاده کنیم بهتر است با تفاوت‌ها و شباهت‌های مهم این دو فریم ورک آشنا باشیم:

»Jasmine یک فریم ورک تست کدهای جاوا اسکریپ بر مبنای Behavior-Driven Development است در حالی که Qunit بر مبنای Test-Driven Development است و همین مسئله مهم‌ترین تفاوت بین این دو فریم ورک می‌باشد.
»اگر قصد دارید که از Qunit نیز به روش BDD استفاده نمایید باید از ترکیب Pavlov به همراه Qunit استفاده کنید.
»Jasmine از مباحث مربوط به Spies و Mocking به خوبی پشتیبانی می‌کند ولی این امکان به صورت توکار در Qunit فراهم نیست. برای اینکه بتوانیم این مفاهیم را در Qunit پیاده سازی کنیم باید از فریم ورک‌های دیگر نظیر SinonJS به همراه Qunit استفاده کنیم.
»هر دو فریم ورک بالا به سادگی و راحتی کار معروف هستند
»تمام موارد مربوط به الگوهای Matching در هر دو فریم ورک به خوبی تعبیه شده است
» هر دو فریم ورک بالا از مباحث مربوط به Asynchronous Testing برای تست کدهای Ajax ای به خوبی پشتیبانی می‌کنند.

بررسی چند مفهوم

قبل از شروع، بهتر است که با چند مفهوم کلی و در عین حال مهم این فریم ورک آشنا شویم

```
describe('JavaScript addition operator', function () {
 it('adds two numbers together', function () {
  expect(1 + 2).toEqual(3);
 });
});
```

در کد بالا یک نمونه از تست نوشته شده با استفاده از Jasmine را مشاهده می‌کنید. دستور describe برای تعریف یک تابع تست مورد استفاده قرار می‌گیرد که دارای دو پارامتر ورودی است. ابتدا یک نام را به این تست اختصاص دهید(بهتر است که این عنوان به صورت یک جمله قابل فهم باشد). سپس یک تابع به عنوان بدنه تست نوشته می‌شود. به این تابع Spec گفته می‌شود.
در تابع it کد بالا شما می‌توانید کدهای مربوط بدنه توابع تست خود را بنویسید. برای پیاده سازی Assert در توابع تست مفهوم expectationها وجود دارد. در واقع expect برای بررسی مقادیر حقیقی با مقادیر مورد انتظار مورد استفاده قرار می‌گیرد و شامل مقادیر true یا false خواهد بود.
برای Setup و Teardown توابع تست خود باید از توابع beforeEach و afterEach که بدین منظور تعبیه شده اند استفاده کنید.

```
describe("A spec (with setup and tear-down)", function() {
```

```
  var foo;

  beforeEach(function() {
    foo = 0;
    foo += 1;
  });

  afterEach(function() {
    foo = 0;
  });

  it("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });

  it("can have more than one expectation", function() {
    expect(foo).toEqual(1);
    expect(true).toEqual(true);
  });
});
```

کاملا واضح است که در تابع beforeEach مجموعه دستورالعمل‌های مربوط به setup تست وجود دارد. سپس دو تابع it برای پیاده سازی عملیات Assertion نوشته شده است. در پایان هم دستورات تابع afterEach ایجاد می‌شوند.

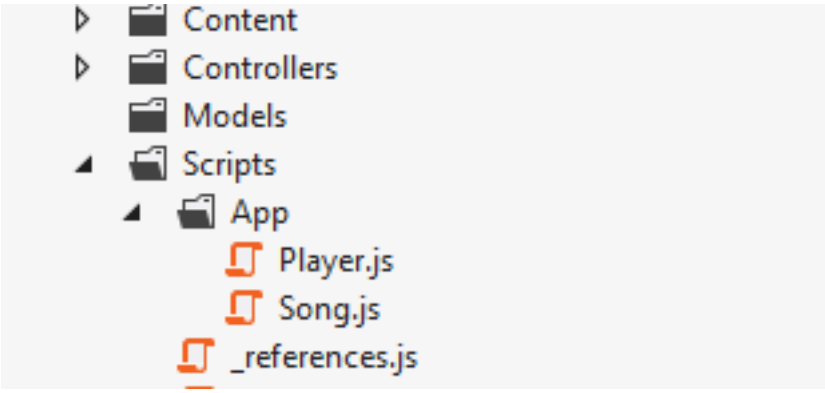
اگر در کد تست خود قصد دارید که یک تابع describe یا it را غیر فعال کنید کافیست یک x به ابتدای آن‌ها اضافه کنید و دیگر نیاز به هیچ کار اضافه دیگری برای comment کردن کد نیست.

```
xdescribe("A spec", function() {
  var foo;

  beforeEach(function() {
    foo = 0;
    foo += 1;
  });

  xit("is just a function, so it can contain any code", function() {
    expect(foo).toEqual(1);
  });
});
```

توابع describe و it بالا در هنگام تست نادیده گرفته می‌شوند و خروجی آن‌ها مشاهده نخواهد شد.

درادامه قصد پیاده سازی یک مثال را با استفاده از Jasmine و RequireJs در پروژه Asp.Net MVC دارم.
برای شروع آخرین نسخه Jasmine را از اینجا دریافت نمایید. یک پروژه Asp.Net MVC به همراه پروژه تست به صورت Empty ایجاد کنید(در هنگام ایجاد پروژه، گزینه create unit test را انتخاب نمایید). فایل دانلود شده را unzip نمایید و دو پوشه lib و spec ،به همراه فایل specRunner.html را در پروژه تست خود کپی نمایید.
فولدر lib شامل فایل‌ها کدهای Jasmine برای setup و tear down و spice و تست کدهای شما می‌باشد.
فایل specRunner.html به واقع یک فایل برای نمایش فایل‌های تست و همچنین نمایش نتیجه تست است.
فولدر spec نیز شامل کدهای Jasmine برای کمک به نوشتن تست می‌باشد.

در این مثال قصد داریم فایل‌های player.js و song.js که به عنوان نمونه به همراه این فریم ورک قرار دارد را در قالب یک پروژه MVC به همراه RequireJs، تست نماییم. در نتیجه این فایل‌ها را از فولدر src انتخاب نمایید و آن‌ها را در قسمت Scripts پروژه اصلی خود کپی کنید(ابتدا یک پوشه به نام App بسازید و فایل‌ها را در آن قرار دهید)



برای استفاده از requireJs باید دستور define را در ابتدا این فایل‌ها اضافه نماییم. در نتیجه فایل‌های Player.js و Song.js را باز کنید و تغییرات زیر را در ابتدای این فایل‌ها اعمال نمایید.

Song.js

```
define(function () {
    function Song() {
    }

    Song.prototype.persistFavoriteStatus = function (value) {
        // something complicated
        throw new Error("not yet implemented");
    };
});
```

Player.js

```
define(function () {
    function Player() {
    }
    Player.prototype.play = function (song) {
        this.currentlyPlayingSong = song;
        this.isPlaying = true;
    };

    Player.prototype.pause = function () {
        this.isPlaying = false;
    };

    Player.prototype.resume = function () {
        if (this.isPlaying) {
            throw new Error("song is already playing");
        }

        this.isPlaying = true;
    };

    Player.prototype.makeFavorite = function () {
        this.currentlyPlayingSong.persistFavoriteStatus(true);
    };
});
```

حال فایل SpecRunner.html را بازکنید و کدهای مربوط به تگ script که به مسیر اصلی فایل‌های تست اشاره می‌کند را Comment نمایید و به جای آن تگ Script مربوط به RequireJs را اضافه نمایید. برای پیکر بندی RequireJs باید از baseUrl و paths استفاده کرد.

```
<link rel="shortcut icon" type="image/png" href="lib/jasmine-1.2.0/jasmine_favicon.png">
<link rel="stylesheet" type="text/css" href="lib/jasmine-1.2.0/jasmine.css">
<script type="text/javascript" src="lib/jasmine-1.2.0/jasmine.js"></script>
<script type="text/javascript" src="lib/jasmine-1.2.0/jasmine-html.js"></script>

<script type="text/javascript" src="../RequireJsMvcStarter/Scripts/require.js"></script>

<!-- include source files here... -->
<!--<script type="text/javascript" src="spec/SpecHelper.js"></script>-->
<!--<script type="text/javascript" src="spec/PlayerSpec.js"></script>-->

<!-- include spec files here... -->
<!--<script type="text/javascript" src="src/Player.js"></script>
<script type="text/javascript" src="src/Song.js"></script>-->

<script type="text/javascript">

    require.config({
        baseUrl: '../RequireJsMvcStarter/Scripts/App',
        paths: {
            spec: '../../../RequireJsMvcStarter.Scripts.Test/spec'
        }
    });
</script>
```

baseUrl در پیکر بندی requireJs به مسیر فایل‌های پروژه که در پروژه اصلی MVC قرار دارد اشاره می‌کند. paths برای تعیین مسیر فایل‌های تست که در پوشه spec در پروژه تست قرار دارد اشاره می‌کند. اگر دقت کرده باشید به دلیل اینگه تگ‌های script مربوط به لود فایل‌های SpecHelper.js و PlayerSpec.js به صورت comment در آمده اند در نتیجه این فایل‌ها لود نخواهند شد و خروجی مورد نظر مشاهده نمی‌شود. در این جا باید از مکانیزم AMD موجود در RequireJs استفاده نماییم و فایل‌های مربوطه را لود کنیم. برای این کار نیاز به اضافه کردن دستور require در ابتدای تگ script به صورت زیر در این فایل است. در نتیجه فایل‌های PlayerSpec و SpecHelper نیز توسط RequireJs لود خواهند شد.

```
<script type="text/javascript">

    require(['spec/PlayerSpec', 'spec/SpecHelper'], function() {
        var jasmineEnv = jasmine.getEnv();
        jasmineEnv.updateInterval = 1000;

        var htmlReporter = new jasmine.HtmlReporter();

        jasmineEnv.addReporter(htmlReporter);

        jasmineEnv.specFilter = function(spec) {
            return htmlReporter.specFilter(spec);
        };

        var currentWindowOnload = window.onload;

        window.onload = function() {
            if (currentWindowOnload) {
                currentWindowOnload();
            }
            execJasmine();
        };

        function execJasmine() {
            jasmineEnv.execute();
        }
    });
</script>
```

نیاز به یک تغییر کوچک دیگر نیز وجود دارد. فایل PlayerSpec را باز نمایید و وابستگی فایل‌های آن را تعیین نمایید. از آن جا که این فایل برای تست فایل‌های Player , Song ایجاد شده است در نتیجه باید از define برای تعیین این وابستگی‌ها استفاده نماییم.

PlayerSpec.js

```
1  define(['Player', 'Song'], function(Player, Song) {
2      describe("Player", function() {
3          var player;
4          var song;
5
6          beforeEach(function() {
7              player = new Player();
8              song = new Song();
9          });
10
```

یادآوری :

»دستور describe در فایل بالا برای تعریف تابع تست است. همان طور که می‌بینید بک نام به آن داده می‌شود به همراه بدنه تابع تست.

»دستور beforeEach برای آماده سازی مواردی است که قصد داریم در تست مورد استفاده قرار گیرند. همانند متدهای Setup در UnitTest.

» دستور expect نیز معادل Assert در UnitTest است و برای بررسی صحت عملکرد تست نوشته می‌شود.

اگر فایل SpecRunner.html را دوباره در مرورگر خود باز نمایید تصویر زیر را مشاهده خواهید کرد که به عنوان موفقیت آمیز بودن پیکر بندی پروژه و تست‌های آن می‌باشد.