آزمون واحد

UNIT TEST

ارائه دهندگان: کیان صحافی و محمد غلامی

اصول آزمون واحد

01 

تعریف آزمون واحد قدم به قدم

اهمیت نوشتن تست واحد خوب
همه ما به نحوی تست واحد نوشته ایم

02 

مشخصه های یک آزمون واحد خوب

مشخصه ها و سوال هایی که باید در هنگام نوشتن کد از خود پرسید

03 

آزمون یکپارچه سازی

معایب آزمون های یکپارچه سازی غیرخودکار در مقایسه با آزمون های واحد خودکار

04 

علت برتری آزمون واحد

معنای نهایی آزمون واحد
لزوم آزمون واحد در قسمت های کد

05 

مثال های ساده ای از آزمون واحد

مثال ها در زبان سی شارپ هستند
C#

06 

توسعه آزمایش محور

دیدگاه های مختلف در توسعه تست محور
تکنیک های توسعه آزمایش محور

07 

سه مهارت اصلی توسعه آزمون محور

دیدگاه های مختلف در توسعه تست محور
تکنیک های توسعه آزمایش محور

08 

جمع بندی

دیدگاه های مختلف در توسعه تست محور
تکنیک های توسعه آزمایش محور

01 تعریف آزمون واحد قدم به قدم

و خوردید شکست ای پروژه در که باری اولین نوشتید، ای برنامه که باری اولین :دارد وجود اول قدم یک همیشه کنید نمی فراموش را خود بار اولین هرگز شما .شدید موفق برسانید انجام به خواستید می که کاری در که باری اولین بد، را آنها است ممکن حتی و باشید، نوشته تست چند قبلاً است ممکن .نکنید فراموش را خود های آزمون اولین امیدوارم و نگهداری نوشته باشید قابل غیر یا کند پا، و دست بی

.کنیم می مقایسه سازی یکپارچه تست مفهوم با را آن و کرده تحلیل را واحد آزمون یک "کلاسیک" تعریف ابتدا ارایه این در خواهیم نگاه سازی یکپارچه آزمون مقابل در واحد آزمون معایب و مزایا به سپس .است کننده گیج بسیاری برای تمایز این رساند، خواهیم پایان به محور آزمایش توسعه به نگاهی با ما .کرد خواهیم ارائه "خوب" واحد تست از بهتری تعریف و کرد .است مرتبط واحد آزمایش با اغلب زیرا

تعریف آزمون واحد قدم به قدم

تست واحد مفهوم جدیدی در توسعه نرم افزار نیست. این زبان از روزهای اولیه زبان برنامه نویسی اسمال تاک در دهه ۱۹۷۰ وجود داشته است و بارها و بارها خود را به عنوان یکی از بهترین راه هایی که یک توسعه دهنده می توانند کیفیت کد را بهبود بخشد و در عین حال درک عمیق تری از الزامات عملکردی یک کلاس یا تابع به دست آورد، ثابت می کند.

کنت بک مفهوم تست واحد را در اسمال تاک معرفی کرد و این مفهوم در بسیاری از زبان های برنامه نویسی دیگر ادامه یافته است و تست واحد را به یک عمل بسیار مفید در برنامه نویسی نرم افزار تبدیل کرده است. قبل از اینکه بیشتر ادامه دهم، باید تست واحد را بهتر تعریف کنم. در اینجا تعریف کلاسیک از ویکی پدیا آمده است. این به آرامی در سراسر این ارایه تکامل میدهیم، با تعریف نهایی که در بخش چهارم ظاهر می شود.

اول تعریف

تست واحد قطعه ای از یک کد (معمولاً یک روش) است که یک کد دیگر را فراخوانی می کند و صحت برخی از مفروضات را پس از آن بررسی می کند. اگر فرضیات اشتباه باشد، آزمون واحد شکست خورده است. واحد یک روش یا تابع است.

Wikipedia

تعریف واحد کار

واحد کار مجموع اقداماتی است که بین فراخوانی یک روش عمومی در سیستم و یک نتیجه نهایی قابل توجه توسط آزمایش آن سیستم انجام می شود.
یک نتیجه نهایی قابل توجه را می توان بدون نگاه کردن به وضعیت داخلی سیستم و تنها از طریق ای پی ای ها و رفتار عمومی آن مشاهده کرد.
نتیجه نهایی هر یک از موارد زیر است:
متد عمومی فراخوانی شده یک مقدار (عملکردی که خالی نیست) برمی گرداند.
تغییر محسوسی در وضعیت یا رفتار سیستم قبل و بعد از فراخوانی وجود دارد که می‌تواند بدون بازجویی از حالت خصوصی تعیین شود.
(مثال: سیستم می تواند به یک کاربر که قبلا وجود نداشت وارد سیستم شود، یا اگر سیستم یک ماشین حالت باشد، ویژگی های سیستم تغییر می کند.)
یک فراخوانی به یک سیستم شخص ثالث وجود دارد که آزمایش روی آن کنترلی ندارد،
و آن سیستم شخص ثالث هیچ مقداری را بر نمی‌گرداند، یا هر مقدار بازگشتی از آن سیستم نادیده گرفته می‌شود.
(مثال: تماس با یک سیستم گزارش شخص ثالث که توسط شما نوشته نشده است و منبع آن را ندارید.)
این ایده از یک واحد کار، برای من، به این معنی است که یک واحد می تواند به اندازه یک روش واحد و تا چندین کلاس و توابع برای رسیدن به هدف خود را شامل شود.

ممکن است احساس کنید که دوست دارید اندازه یک واحد کار در حال آزمایش را به حداقل برسانید. قبلاً این احساس را داشتم. اما من دیگر ندارم.
من معتقدم اگر بتوانید یک واحد کار بزرگتر ایجاد کنید و در جایی که نتیجه نهایی آن برای کاربر نهایی ای پی ای قابل توجه تر باشد،
آزمایش هایی ایجاد می کنید که قابل نگهداری تر هستند. اگر بخواهید اندازه یک واحد کار را به حداقل برسانید، در نهایت چیزهایی را جعل می‌کنید که واقعاً نتایج نهایی برای کاربر یک ای پی ای عمومی نیستند، بلکه فقط ایستگاه‌های قطار در مسیر ایستگاه اصلی هستند.

تعریف اصلاح شده دوم

تست واحد قطعه کدی است که یک واحد کار را فراخوانی می کند و یک نتیجه نهایی خاص از آن واحد کار را بررسی می کند.
اگر فرضیات مربوط به نتیجه نهایی اشتباه باشد، آزمون واحد شکست خورده است. دامنه آزمون واحد می تواند به اندازه یک تابع یا چند کلاس باشد.

اهمیت نوشتن آزمون واحد خوب

مهم نیست که از چه زبان برنامه نویسی استفاده می کنید، یکی از سخت ترین جنبه های تعریف تست واحد، تعریف معنای "خوب" است.
و درک اینکه یک واحد کار چیست کافی نیست.
اکثر افرادی که سعی می‌کنند کد واحد خود را آزمایش کنند یا در نقطه‌ای از کار دست می‌کشند یا در واقع تست واحد را انجام نمی‌دهند. درعوض، یا به تست‌های سیستم و یکپارچه‌سازی تکیه می‌کنند که بسیار دیرتر در چرخه عمر محصول انجام می‌شوند یا به آزمایش دستی کد از طریق برنامه‌های آزمایشی سفارشی یا با استفاده از محصول نهایی که در حال توسعه هستند برای فراخوانی کد خود متوسل می‌شوند.

نوشتن یک آزمون واحد بد فایده ای ندارد، مگر اینکه یاد بگیرید چگونه یک آزمون خوب بنویسید و این اولین قدم های شما در این زمینه است. اگر می‌خواهید یک تست واحد را بد بنویسید بدون اینکه متوجه شوید، بهتر است اصلا آن را ننویسید و از مشکلاتی که در مسیر برای نگهداری و برنامه‌های زمانی ایجاد می‌کند، نجات پیدا کنید. با تعریف اینکه یک آزمون واحد خوب چیست، می توانید مطمئن شوید که با تصور اشتباهی هدف خود را شروع نمی کنید.
برای اینکه بفهمید یک تست واحد خوب چیست، باید ببینید توسعه‌دهندگان هنگام آزمایش چیزی چه می‌کنند

اما سوالی که بوجود می آید این است که : چگونه مطمئن می شوید که کد امروز کار می کند؟

همه ما به نحوی آزمون واحد نوشته ایم

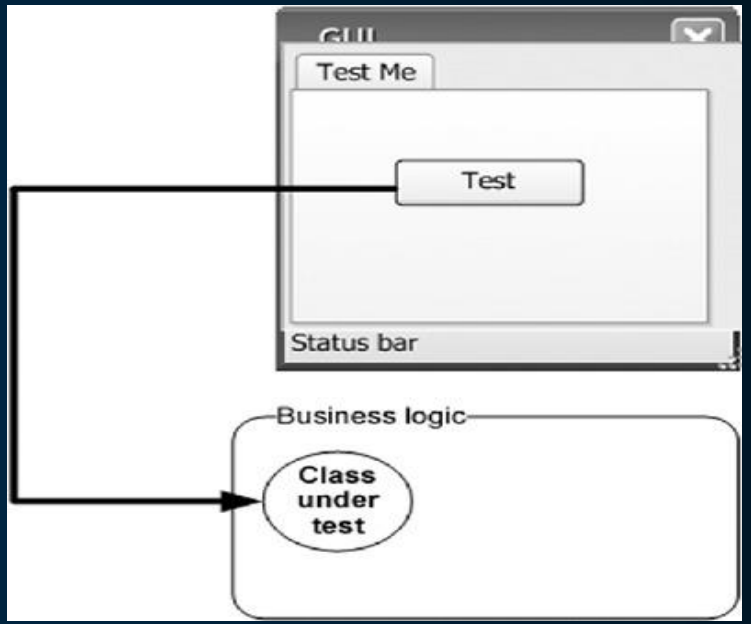
شاید برایتان جالب باشد، اما شما قبلاً برخی از انواع تست واحد را به تنهایی پیاده سازی کرده اید.
یا آیا تا به حال برنامه نویسی دیده اید که کد خود را قبل از تحویل آن تست نکرده باشد؟ من هم ندیده ام

ممکن است از یک برنامه کنسولی استفاده کرده باشید که متدهای مختلف یک کلاس یا مؤلفه را فراخوانی می‌کند، یا شاید از برخی وب فرم ها که عملکرد آن کلاس یا مؤلفه را بررسی می‌کند استفاده کرده باشید، یا حتی آزمایش‌های دستی را که با انجام اقدامات مختلف در داخل برنامه اجرا می‌شوند، استفاده کرده باشید.نتیجه نهایی این است که شما تا حدی مطمئن شده اید که کد به اندازه کافی خوب کار می کند تا آن را به شخص دیگری منتقل کنید.

شکل ۱/۱ نشان می دهد که بیشتر توسعه دهندگان چگونه کد خود را آزمایش
می کنند. رابط کاربری ممکن است تغییر کند، اما الگوی آن معمولاً یکسان است:
استفاده از یک ابزار خارجی دستی برای بررسی مکرر چیزی یا اجرای کامل برنامه
و بررسی رفتار آن به صورت دستی

در تست کلاسیک، توسعه دهندگان از یک رابط کاربری گرافیکی برای راهاندازی یک عمل
در کلاسی که میخواهند آزمایش کنند، استفاده میکنند.
سپس نتایج را بررسی می کنند

این تست‌ها ممکن است مفید بوده باشند، و ممکن است به تعریف کلاسیک
آزمون واحد نزدیک شوند، اما با نحوه تعریف یک آزمون واحد خوب در بسیار فاصله دارند.
این ما را به اولین و مهمترین سوالی که یک توسعه دهنده باید در هنگام تعریف کیفیت
یک آزمون واحد خوب با آن روبرو شود میرساند:
آزمون واحد چیست و چه چیزی نیست؟



شکل ۱/۱

02 مشخصه های یک آزمون واحد خوب

یک تست واحد باید دارای ویژگی های زیر باشد:

۱.باید خودکار و قابل تکرار باشد.
۲.اجرای آن باید آسان باشد.
۳.باید مربوط به فردا باشد.
۴.هر کسی باید بتواند آن را با فشار یک دکمه اجرا کند.
۵.باید به سرعت اجرا شود.
۶.باید در نتایج خود ثابت باشد (اگر چیزی را بین اجراها تغییر ندهید، همیشه همان نتیجه را برمی گرداند).
۷.باید کنترل کامل واحد تحت آزمایش را داشته باشد.
۸.باید کاملاً ایزوله باشد (مستقل از آزمایشات دیگر اجرا می شود).
۹.هنگامی که با شکست مواجه می شود، تشخیص آنچه مورد انتظار بود و چگونگی تعیین دقیق مشکل باید آسان باشد.

بسیاری از مردم عمل تست نرم افزار خود را با مفهوم تست واحد اشتباه می گیرند.
برای شروع، سوالات زیر را در مورد آزمون هایی که تاکنون نوشته اید از خود بپرسید:

۱.آیا می توانم از آزمون واحدی که دو هفته یا ماه یا سال پیش نوشتم اجرا کنم و نتیجه بگیرم؟
۲.آیا هر یک از اعضای تیم من می تواند اجرا شود و از تست های واحدی که دو ماه پیش نوشتم نتیجه بگیرد؟
۳.آیا می توانم تمام تست های واحدی که نوشته ام را در کمتر از چند دقیقه اجرا کنم؟
۴.آیا می توانم تمام تست های واحدی که نوشته ام را با فشار دادن یک دکمه اجرا کنم؟
۵.آیا می توانم یک آزمون پایه را در کمتر از چند دقیقه بنویسم؟
اگر به هر یک از این سؤالات پاسخ منفی داده اید، به احتمال زیاد چیزی که اجرا می کنید یک آزمون واحد نیست.
این قطعاً نوعی آزمون است و به اندازه یک آزمون واحد مهم است، اما در مقایسه با آزمون‌هایی که به شما امکان می‌دهد به همه آن سؤالات بله پاسخ دهید، دارای اشکالاتی است.
"تا الان چیکار میکردم؟" ممکن است بپرسید شما در حال انجام تست یکپارچه سازی بوده اید.

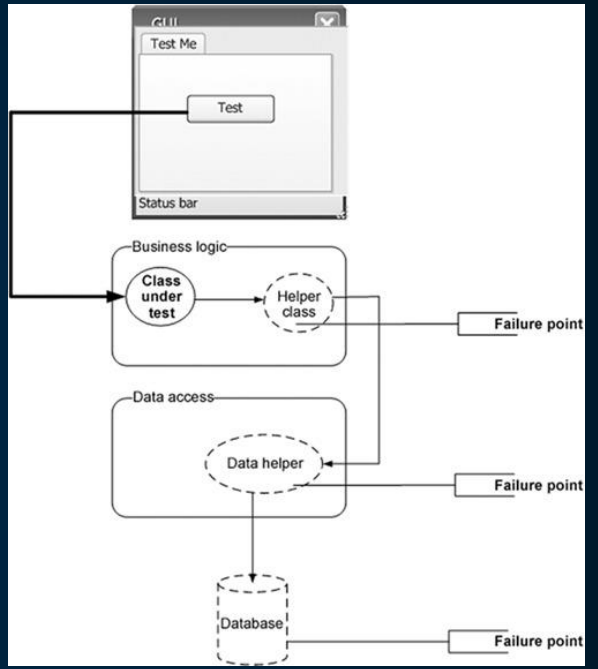
03 آزمون یکپارچه سازی

من تست‌های یکپارچه‌سازی را به‌عنوان تست‌هایی در نظر می‌گیرم که سریع و سازگار نیستند و از یک یا چند وابستگی واقعی از واحدهای مورد آزمایش استفاده می‌کنند. به عنوان مثال، اگر تست از زمان واقعی سیستم، سیستم فایل واقعی یا یک پایگاه داده واقعی استفاده کند، به حوزه تست یکپارچه سازی قدم گذاشته است.
برای مثال، اگر آزمایشی کنترل زمان سیستم را نداشته باشد و از DateTime فعلی استفاده کند. اکنون در کد تست، هر بار که آزمایش اجرا می‌شود، اساساً آزمایش متفاوتی است زیرا از زمان متفاوتی استفاده می‌کند. دیگر سازگار نیست این به خودی خود چیز بدی نیست. من فکر می‌کنم تست‌های ادغام همتای مهمی برای آزمون‌های واحد هستند، اما باید آنها را از آنها جدا کرد تا احساس "منطقه سبز امن" حاصل شود، که بعداً در این کتاب مورد بحث قرار می‌گیرد.
اگر آزمایشی از پایگاه داده واقعی استفاده کند، دیگر فقط در حافظه اجرا نمی شود، زیرا پاک کردن عملکردهای آن سخت تر از زمانی است که فقط از داده های جعلی درون حافظه استفاده می کند. آزمایش همچنین طولانی تر خواهد بود، واقعیتی که هیچ کنترلی بر آن ندارد. تست های واحد باید سریع باشد. تست های یکپارچه سازی معمولا بسیار کندتر هستند.
وقتی شروع به انجام صدها تست می کنید، هر نیم ثانیه مهم است.
تست‌های یکپارچه‌سازی خطر یک مشکل دیگر را افزایش می‌دهند: آزمایش چند چیز در یک زمان.
وقتی ماشین شما خراب می شود چه اتفاقی می افتد؟ چگونه متوجه می شوید که مشکل چیست، چه رسد به اینکه آن را حل کنید؟
یک موتور متشکل از زیرسیستم‌های زیادی است که با هم کار می‌کنند، که هر کدام به دیگری برای کمک به تولید نتیجه نهایی متکی هستند: یک ماشین متحرک. اگر خودرو از حرکت باز ایستد، خطا ممکن است مربوط به هر یک از این زیرسیستم ها باشد - یا بیش از یک. این ادغام آن زیرسیستم ها (یا لایه ها) است که باعث حرکت خودرو می شود.
شما می توانید حرکت ماشین را به عنوان تست نهایی یکپارچه سازی این قطعات در حین حرکت ماشین در جاده در نظر بگیرید. اگر تست ناموفق باشد، تمام قطعات با هم از کار می افتند. اگر موفق شد، همه قسمت ها موفق می شوند.

در نرم افزار هم همین اتفاق می افتد. روشی که اکثر توسعه دهندگان عملکرد خود را آزمایش می کنند، از طریق عملکرد نهایی رابط کاربری است. با کلیک بر روی برخی از دکمه‌ها، مجموعه‌ای از رویدادها ایجاد می‌شود - کلاس‌ها و مؤلفه‌ها با هم کار می‌کنند تا نتیجه نهایی را ایجاد کنند. اگر آزمایش ناموفق باشد، همه این مؤلفه‌های نرم‌افزار به‌عنوان یک تیم شکست می‌خورند، و تشخیص اینکه چه چیزی باعث شکست عملیات کلی شده است دشوار است (شکل ۱٫۲ را ببینید)

شما می توانید در یک تست یکپارچه سازی نقاط شکست زیادی داشته باشید.
همه واحدها باید با هم کار کنند و هر کدام ممکن است دچار نقص شوند و پیدا کردن منبع اشکال را دشوارتر کند.

همانطور که در راهنمای کامل تست نرم افزار توسط بیل هتزل تعریف شده است،
تست یکپارچه سازی "پیشرفت منظم آزمایش است که در آن عناصر نرم افزار و/یا سخت افزار ترکیب و آزمایش می شوند تا زمانی که کل سیستم یکپارچه شود.
" این تعریف از تست یکپارچه سازی کمی کمتر از آنچه بسیاری از افراد همیشه انجام می دهند، نه به عنوان بخشی از تست یکپارچه سازی سیستم،
بلکه به عنوان بخشی از تست های توسعه و واحد است.



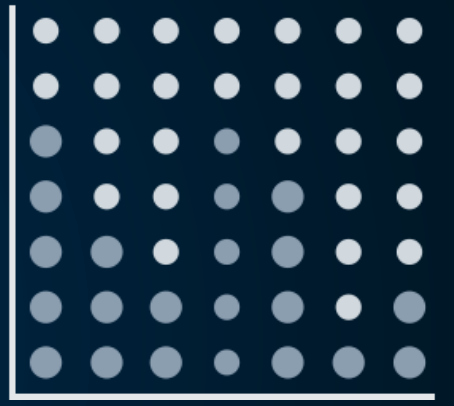
شکل ۱٫۲

04 آزمون یکپارچه سازی

به طور خلاصه: یک آزمون یکپارچه سازی از وابستگی های واقعی استفاده می کند. تست‌های واحد واحد کار را از وابستگی‌های آن جدا می‌کنند تا به راحتی در نتایج خود ثابت باشند و بتوانند به راحتی هر جنبه از رفتار واحد را کنترل و شبیه‌سازی کنند.

سوالات بخش ۳ می تواند به شما کمک کند تا برخی از اشکالات تست یکپارچه سازی را تشخیص دهید.
بیایید سعی کنیم کیفیت هایی را که در یک آزمون واحد خوب به دنبال آن هستیم، تعریف کنیم.

۱.آیا می توانم از آزمون واحدی که دو هفته یا ماه یا سال پیش نوشتم اجرا کنم و نتیجه بگیرم؟
۲.آیا هر یک از اعضای تیم من می تواند اجرا شود و از تست های واحدی که دو ماه پیش نوشتم نتیجه بگیرد؟
۳.آیا می توانم تمام تست های واحدی که نوشته ام را در کمتر از چند دقیقه اجرا کنم؟
۴.آیا می توانم تمام تست های واحدی که نوشته ام را با فشار دادن یک دکمه اجرا کنم؟
۵.آیا می توانم یک آزمون پایه را در کمتر از چند دقیقه بنویسم؟
اگر به هر یک از این سؤالات پاسخ منفی داده اید، به احتمال زیاد چیزی که اجرا می کنید یک آزمون واحد نیست.
این قطعاً نوعی آزمون است و به اندازه یک آزمون واحد مهم است، اما در مقایسه با آزمون‌هایی که به شما امکان می‌دهد به همه آن سؤالات بله پاسخ دهید، دارای اشکالاتی است.
"تا الان چیکار میکردم؟" ممکن است بپرسید شما در حال انجام تست یکپارچه سازی بوده اید.



اکنون که ویژگی‌های مهمی را که یک تست واحد باید داشته باشد را پوشش دادم، یک بار برای همیشه تست‌های واحد را تعریف می‌کنم:

تعریف نهایی آزمون واحد

تست واحد یک قطعه کد خودکار است که واحد کار مورد آزمایش را فراخوانی می‌کند و سپس برخی فرضیات را در مورد یک نتیجه نهایی واحد آن واحد بررسی می‌کند. یک آزمون واحد تقریباً همیشه با استفاده از چارچوب تست واحد نوشته می شود. می توان آن را به راحتی نوشت و به سرعت اجرا می شود. قابل اعتماد، خواندنی و قابل نگهداری است. تا زمانی که کد تولید تغییر نکرده باشد، در نتایج خود ثابت است.

یک مثال ساده از آزمون واحد

```
public class SimpleParser
{
    public int ParseAndSum(string numbers)
    {
        if(numbers.Length==0)
        {
            return 0;
        }
        if(!numbers.Contains(","))
        {
            return int.Parse(numbers);
        }
        else
        {
            throw new InvalidOperationException(
            "I can only handle 0 or 1 numbers for now!");
        }
    }
}
```

```
class SimpleParserTests
    {
        public static void TestReturnsZeroWhenEmptyString()
        {
            try
            {
                SimpleParser p = new SimpleParser();
                int result = p.ParseAndSum(string.Empty);
                if(result!=0)
                {
                 Console.WriteLine(
                 @"***SimpleParserTests.TestReturnsZeroWhenEmptyString:
                 -------
                 Parse and sum should have returned 0 on an empty string");
                }
            }
            catch (Exception e)

            {
                Console.WriteLine(e);
            }
        }
    }
```

```
public static void Main(string[] args)
    {
        try
        {
            SimpleParserTests.TestReturnsZeroWhenEmptyString();
        }
        catch (Exception e)
        {
            Console.WriteLine(e);
        }
    }
```

```
public class TestUtil
{
    public static void ShowProblem(string test,string message)
    {
        string msg = string.Format(@"
        ---{0}---
              {1}
        --------------------
        ", test, message);
        Console.WriteLine(msg);

    }
}
public static void TestReturnsZeroWhenEmptyString()
{
    //use .NET's reflection API to get the current
    //           method's name
    // it's possible to hard code this,
    //but it's a useful technique to know
    string testName =
MethodBase.GetCurrentMethod().Name;
```

```
 try
    {
        SimpleParser p = new SimpleParser();
        int result = p.ParseAndSum(string.Empty);
        if(result!=0)
        {
    //Calling the helper method
            TestUtil.ShowProblem(testName,
                "Parse and sum should have returned 0 on
an
                empty string");
        }
    }
    catch (Exception e)
    {
        TestUtil.ShowProblem(testName, e.ToString());
    }
}
```

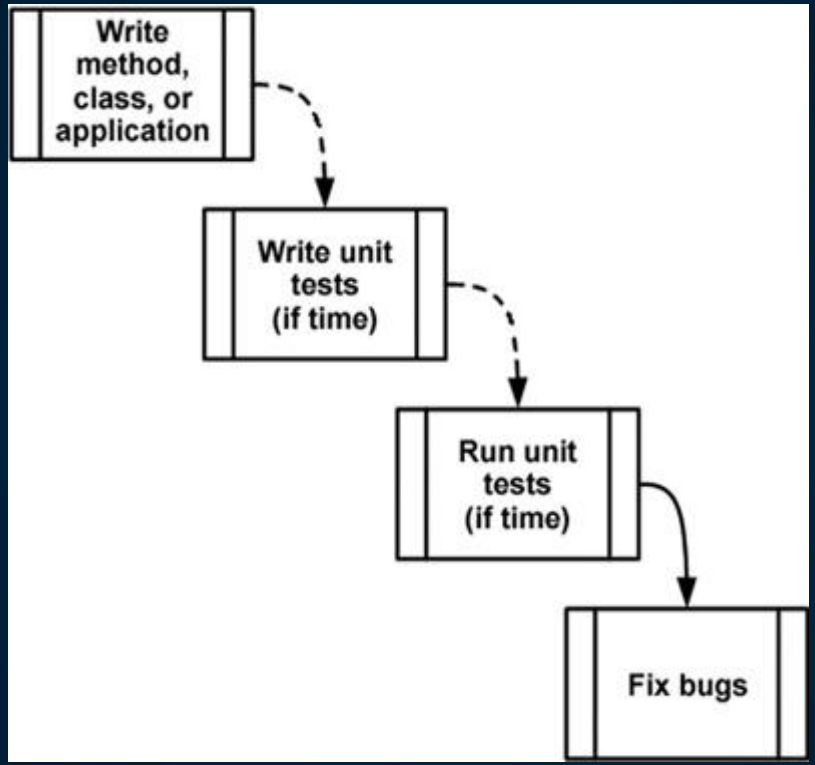
توسعه آزمون محور TDD

ساختاریافته،قابل نگهداری،solid

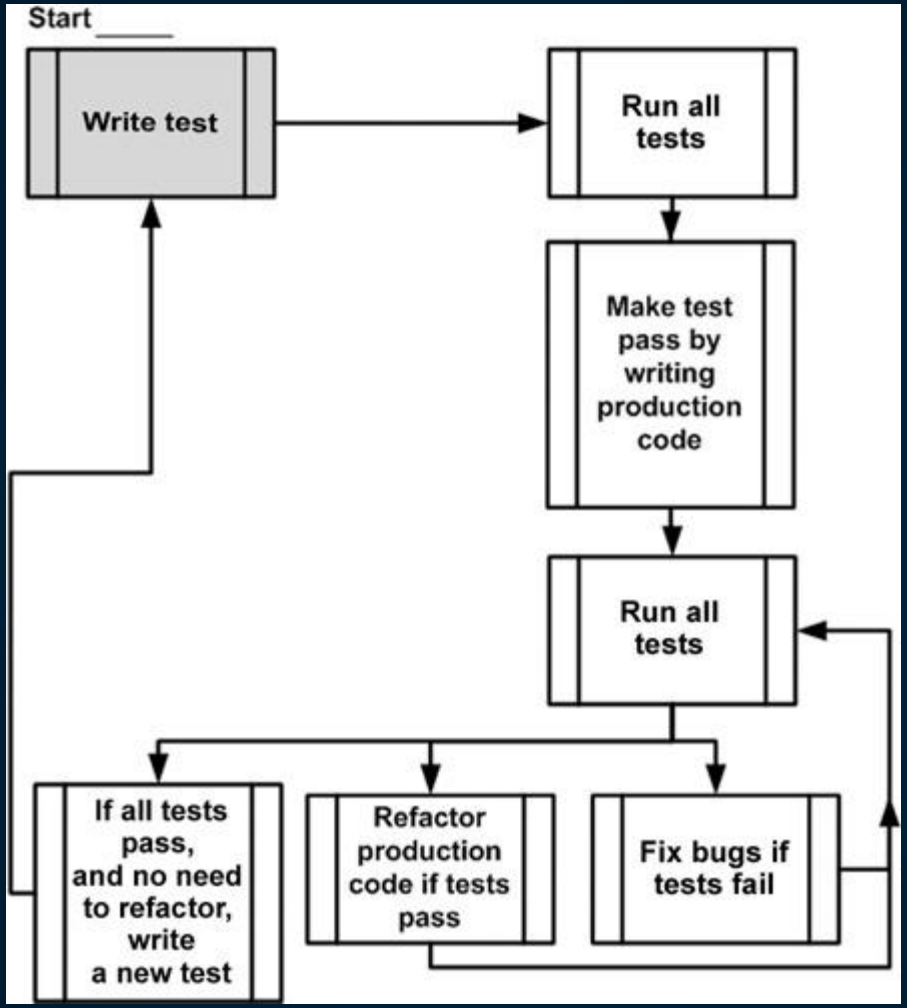
زمان نوشتن تست؟؟

این جاست که این نوع تست نویسی به کمک ما میاد.

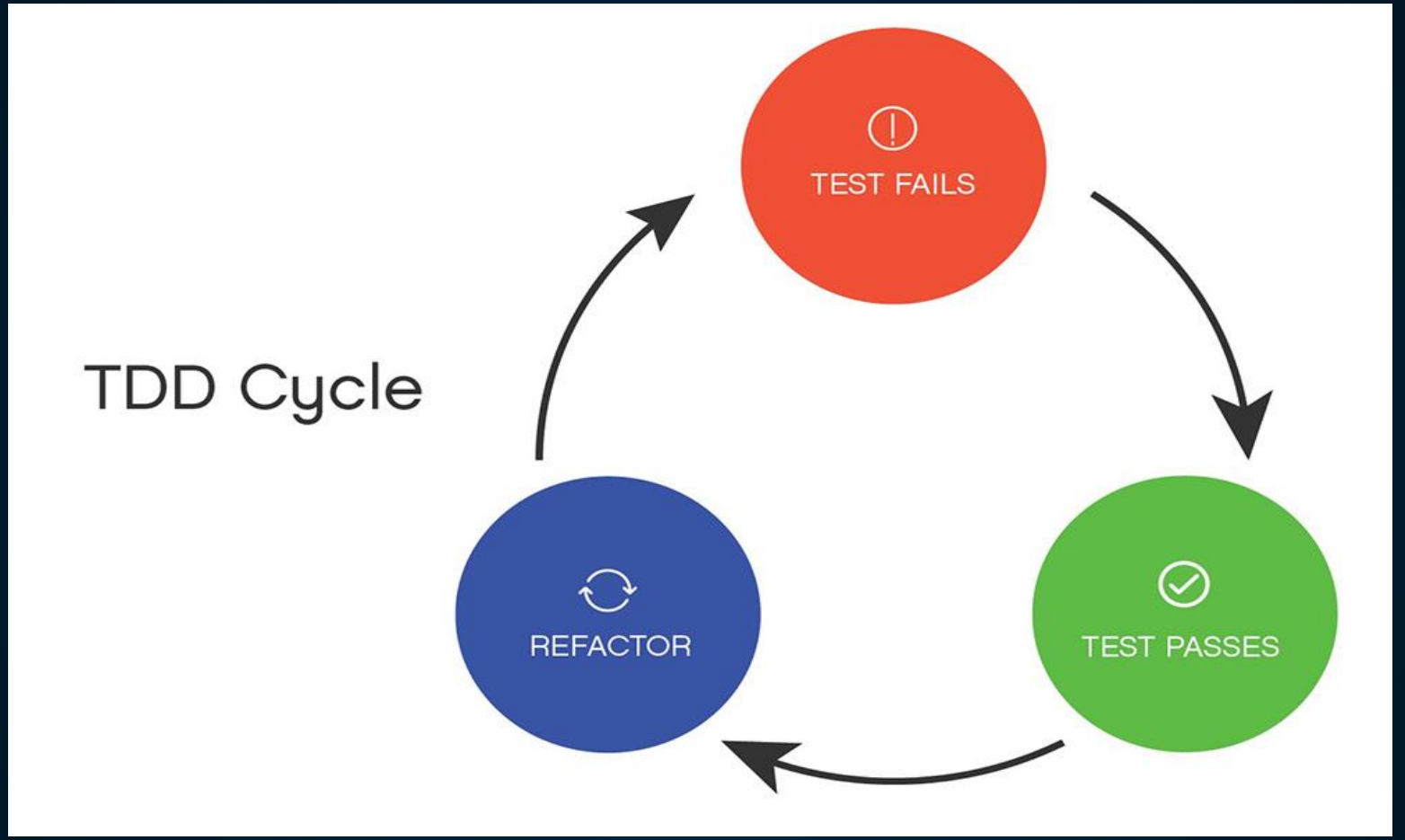
اختلاف نظرهای زیادی راجب این رویکرد وجود داره.


Write method, class, or application
Write unit tests (if time)
Run unit tests (if time)
Fix bugs

شکل ۲/۱


Start
Write test
Run all tests
Make test pass by writing production code
Run all tests
If all tests pass, and no need to refactor, write a new test
Refactor production code if tests pass
Fix bugs if tests fail

شکل ۲/۲



شکل ۲/۳

سه مهارت اصلی برای توسعه ازمون محور موفق

۱.از آنجایی که ابتدا تست های خود را می نویسید به این معنی نیست که آنها قابل نگهداری، خوانا یا قابل اعتماد هستند.

۲.صرفاً به این دلیل که تست‌های خوانا و قابل نگهداری را می‌نویسید به این معنی نیست که از همان مزایایی که هنگام نوشتن آزمون اول آن‌ها را می‌نویسید، بهره مند می‌شوید.

۳.فقط به این دلیل که ابتدا تست های خود را می نویسید و آنها خوانا و قابل نگهداری هستند، به این معنی نیست که در نهایت با یک سیستم خوب طراحی شده خواهید بود.

خلاصه

ما یک تست واحد خوب را به عنوان تستی تعریف کردم که دارای این ویژگی ها باشد:

*این یک قطعه کد خودکار است که روش دیگری را فراخوانی می کند و سپس برخی از فرضیات را در مورد رفتار منطقی آن متد یا کلاس بررسی می کند.

*با استفاده از یک چارچوب تست واحد نوشته شده است.

*می توان آن را به راحتی نوشت.

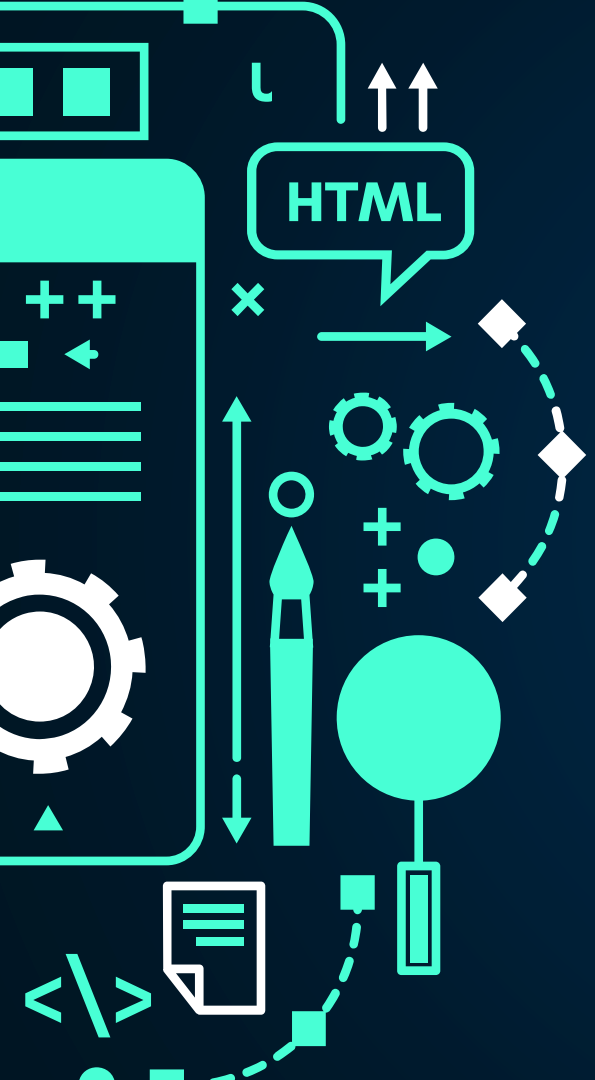
*به سرعت اجرا می شود.

*این می تواند بارها و بارها توسط هر کسی در تیم توسعه اجرا شود.

برای اینکه بفهمید یک واحد چیست، باید بفهمید که تا به حال چه نوع آزمون را انجام داده اید. شما آن نوع آزمون را به عنوان تست ادغام شناسایی کردید، زیرا مجموعه ای از واحدها را آزمایش می کند که به یکدیگر وابسته هستند.

ما همچنین به معایب انجام تست ادغام بدون چارچوب پشت آن نگاه کردیم: نوشتن و خودکار شدن این نوع آزمون سخت است، اجرای آن کند است و نیاز به پیکربندی دارد. اگرچه می‌خواهید تست‌های یکپارچه‌سازی را در یک پروژه داشته باشید، تست‌های واحد می‌توانند ارزش زیادی را در مراحل اولیه ارائه کنند، زمانی که اشکالات کوچک‌تر و راحت‌تر پیدا می‌شوند و کد کمتری برای مرور وجود دارد.

در آخر، ما به توسعه مبتنی بر آزمایش، تفاوت آن با کدنویسی سنتی و مزایای اساسی آن نگاه کردیم. TDDبه شما کمک می کند تا مطمئن شوید که پوشش کد آزمون شما (چه مقدار از کد آزمون های شما را اجرا می کند) بسیار زیاد است (نزدیک به ۱۰۰ درصد کد منطقی). TDDبه شما کمک می کند مطمئن شوید که تست های شما قابل اعتماد هستند. TDDتست های شما را آزمایش می کند به این ترتیب که به شما امکان می دهد آنها را ببینید که شکست می خورند و در زمانی که باید قبول می شوند. TDDهمچنین مزایای بسیاری دارد، مانند کمک به طراحی، کاهش پیچیدگی و کمک به شما برای مقابله با مشکلات سخت گام به گام. اما نمی‌توانید در طول زمان TDDرا با موفقیت انجام دهید بدون اینکه بدانید چگونه تست‌های خوب بنویسید.



ممنون از توجه شما !

ما پاسخگوی سوالات شما هستیم!

kiansahafi@gmail.com
mohammadwow24@gmail.com