

Компонентно тестване

Добри практики



Учителски екип

Обучение за ИТ кариера

<https://it-kariera.mon.bg/e-learning/>



Разработка
на софтуер



Съдържание

1. Добри практики за компонентно тестване
2. Именуване на тестовете
3. Какво тествахме?
4. Какви съобщения да съставяме?
5. Проверките в тестовия метод



Стандарти за именуване в компонентното тестване

- Името на теста трябва да изразява специфичните изисквания, които се тестват
 - Обикновено **[TestMethod_StateUnderTest_ExpectedBehavior]**
 - Например **TestAccountDepositNegativeSum()** или **AccountShouldNotDepositNegativeSum()**
- Името на теста трябва да включва
 - Очакваното входното състояние
 - Очаквания резултат или състояние
 - Името на тествания метод или клас

Стандарти за именуване в компонентното тестване – Пример

- Даден е метод:

```
public int Sum(params int[] values)
```

с изискване да се игнорират числата, по-големи от 100 в процеса на сумиране

- Името на теста трябва да е:

```
TestSumNumberIgnoredIfGreaterThan100
```


Кога трябва да се промени или премахне тест?

- Най-общо, успешните тестове **никога** не се премахват
- Тези тестове подsigуряват това, че промените в кода не повреждат работещия код
- Успешен тест трябва да се променя само ако се цели да се подобри четимостта му
- Когато тестовете не минават - това обикновено значи, че има конфликтни изисквания – или самият тест е написан дефектно или в кода, който тества има дефект

Кога трябва да се промени или премахне тест? (2)

- Пример:

```
void TestSum_FirstNegativeNumberThrowsException()  
{  
    Assert.Throws<Exception>(()=> Sum(-1, 1, 2));  
}
```

- Този тест ще бъде променен, ако се добави нова функционалност, която позволява отрицателни числа.

Кога трябва да се промени или премахне тест? (3)

- Нов разработчик е написал следния тест:

```
void TestSum_FirstNegativeNumberCalculatesCorrectly()
{
    int sumResult = Sum(-1, 1, 2);
    Assert.AreEqual(2, sumResult);
}
```

- Предишният тест пропада поради промяна на изискванията

Кога трябва да се промени или премахне тест(4)

- Две посоки на действие:
 1. Изтрийте пропаднал тест след потвърждение на неговата невалидност
 2. Промяна на стари тестове:
 - или тестване на новото изискване
 - или тестване на старите изисквани при новите обстоятелства



Тестовите трябва да отразяват реалните изисквания

```
int Sum(int a, int b) -> returns sum of a and b
```

- Какво не е наред в следния тест?

```
public void Sum_AddsOneAndTwo()
{
    int result = Sum(1, 2);
    Assert.AreEqual(4, result, "Bad sum");
}
```

- Пропадането на тест трябва да докаже, че има нещо нередно с кода, който тества, а не с кода на теста

Какво трябва да кажат съобщенията от проверката?

- Съобщението от проверката в теста е едно от най-важните неща!
 - Трябва да ни казва какво **очаква да се случи**, но не и **какво е станало вместо това**
 - Доброто съобщение от проверката ни помага да **проследим грешките** и да разберем по-лесно компонентното тестване
- Пример:
 - *„Изтеглянето е неуспешно: сметките не би трябвало да имат отрицателно салдо“*

Какво трябва да кажат съобщенията от проверката? (2)

- Изразяват какво **трябва да** се случи и какво **не трябва** да се случи
 - *“**Verify()** не хвърли изключение”*
 - *“**Connect()** не отвори връзката преди да завърши”*
- Не:
 - връщат празни или безсмислени съобщения
 - връщат съобщения, които повтарят името на тестовия случай

Избягвайте множество проверки в един компонентен тест

- Избягвайте **множествена проверка** в единичен тестов случай
 - Ако първата проверка пропадне, изпълнението на теста спира за този тестов случай

```
void TestSum_AnyParamBiggerThan1000IsNotSummed()  
{  
    Assert.AreEqual(3, Sum(1001, 1, 2));  
    Assert.AreEqual(3, Sum(1, 1001, 2));  
    Assert.AreEqual(3, Sum(1, 2, 1001));  
}
```

Компонентно тестване – Предизвикателството

- Концепцията за **компонентно тестване** е от много години в общността на разработчиците
- Нови методологии в частност **Scrum** и **XP**, са превърнали компонентното тестване в основа на разработка на софтуер
- Писането на **добри и ефективни** компоненти тестове е важно!
- Съществуват инструменти за генериране на тестове (например **Pex**)
 - Въпреки това тези инструменти не са перфектни и често се налага разработчиците да редактират генерирания код

Обобщение

1. Пишете код, удобен за тестване
2. Следвайте шаблона **ЗА**
3. Пишете компоненти тестове, съгласно спецификациите
 - При смяна на спецификацията трябва да се сменят също и тестовете
4. Когато пишем компонентен тест, следвайте правилата за качествен програмен код
5. Компонентното тестване е добра документация на кода



Компонентно тестване – Добри практики



Въпроси?



Министерство на образованието и науката (МОН)

- Настоящият курс (презентации, примери, задачи, упражнения и др.) е разработен за нуждите на Национална програма "**Обучение за ИТ кариера**" на МОН за подготовка по професия "Приложен програмист"



Министерство
на образованието
и науката



Национална
програма
„Обучение за
ИТ кариера“

- Курсът е базиран на учебно съдържание и методика, предоставени от **фондация "Софтуерен университет"** и се разпространява под свободен лиценз **CC-BY-NC-SA**



SoftUni
Foundation

