Testare unitară în C#

Echipa: Oprea Mihai-Ștefan, Ion Melania-Victorița, Flutur Angelica-Costela, Oprea Tudor, Monete Andreea-Maria

Testare funcțională

Testarea funcțională se bazează pe specificații care includ pre-condiții și postcondiții. Acestea stabilesc condițiile inițiale și rezultatele așteptate ale unei funcționalități. Pentru a asigura o acoperire completă a funcționalității, datele de test sunt generate în conformitate cu aceste specificații. În mod ideal, datele de intrare sunt împărțite în partiții, astfel încât să fie reprezentative pentru diversele comportamente posibile ale sistemului. Această abordare asigură că fiecare scenariu de testare acoperă toate aspectele relevante ale funcționalității, contribuind la o testare completă și eficientă a aplicației.

- Partitionarea in clase de echivalență
- Analiza valorilor de frontieră

Partiționarea în clase de echivalență

```
[Test]
0 references
public void CreateProductBoundryValueAnalysis()
{
    //Clase de echivalenta:
    //Domeniul de intrari:
    //name length < 3
    //name length >= 3 =< 50
    //name length > 50
    //description length < 3
    //description length >= 3 =< 50
    //description length > 50
    //stock < 1
    //stock >= 1 =< double.MaxValue
    //stock > double.MaxValue
    //price < 1
    //price >= 1 =< dobule.MaxValue
    //price > double.MaxValue
    //discount < 0
    //discount >= 0 =< 100
    //discount > 100

    //Domeniul de iesiri:
    //produs creat => CreatedAtActionResult status code 201
    //produs invalid => BadRequestObjectResult status code 400

    string nameGoodValue = "TestProduct";
    string descriptionGoodValue = "TestDescription";
    int stockGoodValue = 1;
    float priceGoodValue = 1;
    int discountGoodValue = 1;
    int categoryGoodValue = 1;

    ProductRequest product = new ProductRequest
    {
        Name = nameGoodValue,
        Description = descriptionGoodValue,
        Stock = stockGoodValue,
        Price = priceGoodValue,
        Discount = discountGoodValue,
        CategoryId = categoryGoodValue
    };
```

În testarea funcțională, domeniul de intrare este împărțit în clase de echivalență, sau partiții, unde toate valorile dintr-o clasă sunt tratate în mod identic conform specificației. Acest lucru permite selectarea unei singure valori reprezentative din fiecare clasă pentru testare. După identificarea claselor, se alege o valoare reprezentativă din fiecare clasă pentru testare, împreună cu date invalide care nu aparțin niciunei clase.

Partiționarea în clase de echivalență

Domeniul de intrări: name length (nl), description length (dl), stock (s), price (p), discount (d)

Se disting astfel următoarele clase de echivalență:

- NL_1 = 3...50 ; NL_2 = { nl | nl<3 }; NL_3 = { nl | nl>50 };
- DL_1 = 3...50 ; DL_2 = { dl | dl<3 }; DL_3 = { dl | dl>50 };
- S_1 = 1...DMV; S_2 = { s | s<1 };
- P_1 = 1...DMV; P_2 = { p | p<1 };
- D_1 = 0...100; D_2 = { d | d<0 }; D_3 = { d | d>100 };

Domeniul de ieșiri: produsul este sau nu creat

- PC_1 = { x | x este valid și creat (CreatedAtActionResult - status code 201) }
- PC_2 = { x | x este invalid (BadRequestObjectResult - status code 400) }

DMV=double.MaxValue ~= 1.7976931348623157E+308

Clasele de echivalență globale

F1 1 1 1 1 1 = { (nl,dl,s,p,d,x) | nl \in NL_1, dl \in DL_1, s \in S_1, p \in P_1, d \in D_1, x \in PC_1 }

F_ _ _ _ 2 _ = { (nl,dl,s,p,d,x) | d \in D_2 }

F_ _ _ _ 3 _ = { (nl,dl,s,p,d,x) | d \in D_3 }

F_ _ _ 2 _ _ = { (nl,dl,s,p,d,x) | p \in P_2 }

F_ _ 2 _ _ _ = { (nl,dl,s,p,d,x) | s \in S_2 }

F_ 2 _ _ _ _ = { (nl,dl,s,p,d,x) | dl \in DL_2 }

F_ 3 _ _ _ _ = { (nl,dl,s,p,d,x) | dl \in DL_3 }

F2 _ _ _ _ _ = { (nl,dl,s,p,d,x) | nl \in NL_2 }

F3 _ _ _ _ _ = { (nl,dl,s,p,d,x) | nl \in NL_3 }

Setul de date de test

f1 1 1 1 1 1 : (40, 40, 1, 1, 1, status code 201)

f_ _ _ _ 2 _ : (40, 40, 1, 1, **-2**, status code 400)

f_ _ _ _ 3 _ : (40, 40, 1, 1, **103**, status code 400)

f_ _ _ 2 _ _ : (40, 40, 1, **-2,** 1, status code 400)

f_ _ 2 _ _ _ : (40, 40, **-2**, 1, 1, status code 400)

f_ 2 _ _ _ _ : (40, **51**, 1, 1, 1, status code 400)

f_ 3 _ _ _ _ : (40, **2**, 1, 1, 1, status code 400)

f2 _ _ _ _ _ : (**2**, 40, 1, 1, 1, status code 400)

f3 _ _ _ _ _ : (**51**, 40, 1, 1, 1, status code 400)

Analiza valorilor de frontieră

Metoda analizei valorilor de frontieră este adesea combinată cu partiționarea de echivalență, fiind concentrată pe examinarea valorilor limită ale claselor, care sunt adesea surse importante de erori. Această metodă completează abordarea anterioară prin furnizarea de informații suplimentare pentru generarea setului de date de test. Ea se axează în special pe zonele de frontieră, unde, de obicei, se produc cele mai multe erori, contribuind astfel la identificarea și remedierea lor în mod eficient.

```
//discount = 0
product.Discount = 0;
goodResult = _controller.Post(product) as CreatedAtActionResult;
Assert.AreEqual(201, goodResult.StatusCode);

//discount = 100
product.Discount = 100;
goodResult = _controller.Post(product) as CreatedAtActionResult;
Assert.AreEqual(201, goodResult.StatusCode);
```

```
//name = 3
product.Name = "123";
goodResult = _controller.Post(product) as CreatedAtActionResult;
Assert.AreEqual(201, goodResult.StatusCode);

//name = 50
product.Name = "Lorem Lorem ipsum dolor sit amet, consectetur adip";
goodResult = _controller.Post(product) as CreatedAtActionResult;
Assert.AreEqual(201, goodResult.StatusCode);
```

Valorile de frontieră determinate:

- nl: 2, 3, 50, 51
- dl: 2, 3, 50, 51
- s: 0, 1, DMV
- p: 0, 1, DMV
- d: -1,0,100,101

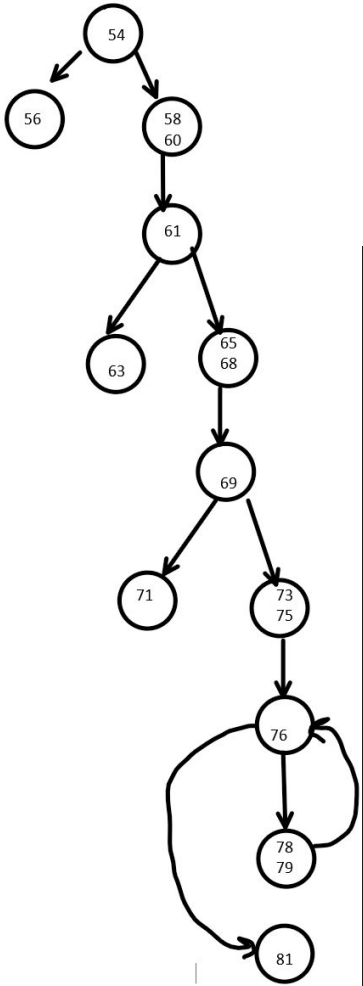
- NL_1 = 3...50 ; NL_2 = 2; NL_3 = 51;
- DL_1 = 3...50 ; DL_2 = 2; DL_3 = 51;
- S_1 = 1...DMV; S_2 = 0;
- P_1 = 1...DMV; P_2 = 0;
- D_1 = 0...100; D_2 = -1; D_3 = 101;

Testare structurala

În testarea structurală, datele de test sunt generate pe baza codului implementat, fără a ține cont de specificațiile programului. Pentru a aplica metode structurale de testare, programul este reprezentat ca un graf orientat, denumit graful de flux de control (CFG). Setul de date de test este ales astfel încât să acopere toate elementele grafului, fie că este vorba de o instrucțiune, o ramură sau o cale, cel puțin o singură dată. În funcție de tipul de element selectat, sunt definite diferite metrici pentru acoperirea grafului, incluzând:

- Acoperire la nivel de instrucțiune
- Acoperire la nivel de condiție
- Acoperire la nivel de decizie

Control Flow Graph (CFG)

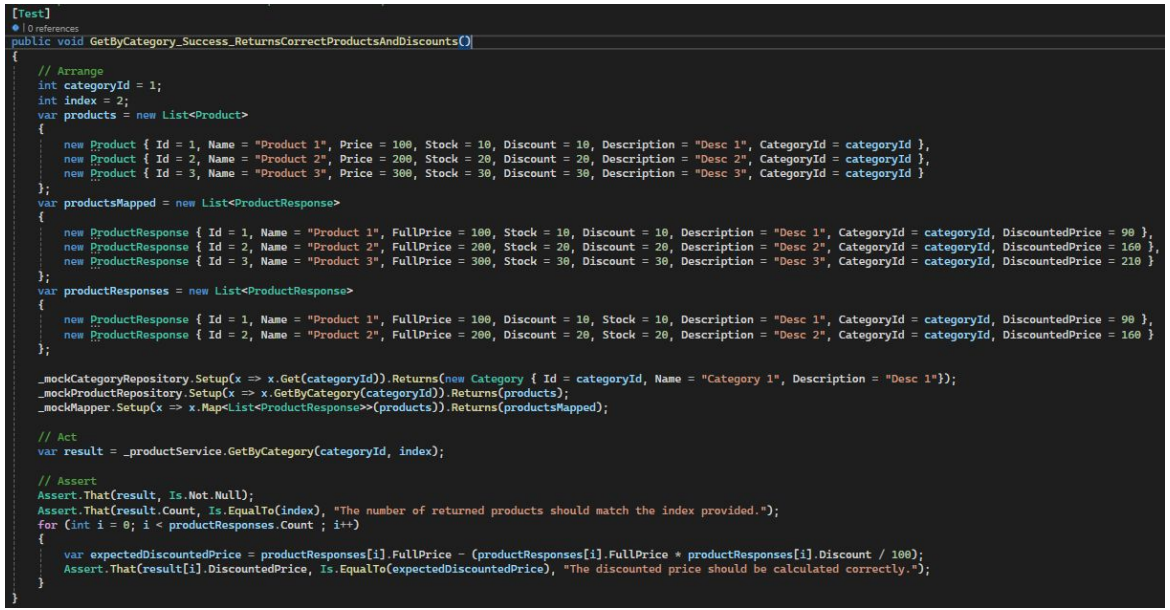


- Pentru Products avem funcția **GetByCategoryId**(int id, int index) => returnează un nr *index* de produse din categoria *id*.

```
50      public List<ProductResponse> GetByCategory(int categoryId,int index)
51      {
52          Category category = _categoryRepository.Get(categoryId);
53
54          if (category == null || index < 0)
55          {
56              throw new KeyNotFoundException($"The category with the specified ID ({categoryId}) was not found or the index value was less than 0.");
57          }
58          else
59          {
60              List<Product> products = _productRepository.GetByCategory(categoryId);
61              if (products.Count == 0)
62              {
63                  throw new KeyNotFoundException($"No products were found for the specified category ID ({categoryId}).");
64              }
65              else
66              {
67
68                  List<ProductResponse> productsDto = _mapper.Map<List<ProductResponse>>(products);
69                  if (productsDto.Count < index)
70                  {
71                      throw new KeyNotFoundException($"There are not ({index}) products in this category ({categoryId}).");
72                  }
73                  else
74                  {List<ProductResponse> displayedProducts = new List<ProductResponse>();
75                      int i = 0;
76                      while(i < index)
77                      {
78                          productsDto[i].DiscountedPrice = productsDto[i].FullPrice - (productsDto[i].FullPrice * productsDto[i].Discount / 100);
79                          displayedProducts.Add(productsDto[i]);i++;
80                      }
81                      return displayedProducts;
82                  }
83              }
84          }
85      }
86
```

(parameter) int categoryId

Acoperire la nivel de instrucțiune



```
[Test]
0 references
public void GetByCategory_Success_ReturnsCorrectProductsAndDiscounts()
{
    // Arrange
    int categoryId = 1;
    int index = 2;
    var products = new List<Product>
    {
        new Product { Id = 1, Name = "Product 1", Price = 100, Stock = 10, Discount = 10, Description = "Desc 1", CategoryId = categoryId },
        new Product { Id = 2, Name = "Product 2", Price = 200, Stock = 20, Discount = 20, Description = "Desc 2", CategoryId = categoryId },
        new Product { Id = 3, Name = "Product 3", Price = 300, Stock = 30, Discount = 30, Description = "Desc 3", CategoryId = categoryId }
    };
    var productsMapped = new List<ProductResponse>
    {
        new ProductResponse { Id = 1, Name = "Product 1", FullPrice = 100, Stock = 10, Discount = 10, Description = "Desc 1", CategoryId = categoryId, DiscountedPrice = 90 },
        new ProductResponse { Id = 2, Name = "Product 2", FullPrice = 200, Stock = 20, Discount = 20, Description = "Desc 2", CategoryId = categoryId, DiscountedPrice = 160 },
        new ProductResponse { Id = 3, Name = "Product 3", FullPrice = 300, Stock = 30, Discount = 30, Description = "Desc 3", CategoryId = categoryId, DiscountedPrice = 210 }
    };
    var productResponses = new List<ProductResponse>
    {
        new ProductResponse { Id = 1, Name = "Product 1", FullPrice = 100, Discount = 10, Stock = 10, Description = "Desc 1", CategoryId = categoryId, DiscountedPrice = 90 },
        new ProductResponse { Id = 2, Name = "Product 2", FullPrice = 200, Discount = 20, Stock = 20, Description = "Desc 2", CategoryId = categoryId, DiscountedPrice = 160 }
    };

    _mockCategoryRepository.Setup(x => x.Get(categoryId)).Returns(new Category { Id = categoryId, Name = "Category 1", Description = "Desc 1"});
    _mockProductRepository.Setup(x => x.GetByCategory(categoryId)).Returns(products);
    _mockMapper.Setup(x => x.Map<List<ProductResponse>>(products)).Returns(productsMapped);

    // Act
    var result = _productService.GetByCategory(categoryId, index);

    // Assert
    Assert.That(result, Is.Not.Null);
    Assert.That(result.Count, Is.EqualTo(index), "The number of returned products should match the index provided.");
    for (int i = 0; i < productResponses.Count ; i++)
    {
        var expectedDiscountedPrice = productResponses[i].FullPrice - (productResponses[i].FullPrice * productResponses[i].Discount / 100);
        Assert.That(result[i].DiscountedPrice, Is.EqualTo(expectedDiscountedPrice), "The discounted price should be calculated correctly.");
    }
}
```

Asigură că fiecare instrucțiune (sau nod) din graf este executată cel puțin o dată, verifică fiecare linie de cod pentru a asigura execuția corectă a fiecărei instrucțiuni.

Date de test:

category	index	nr products	nr productsDto	Rezultat	Instructiuni parcurse
null	-1			Category not found	54, 56
1	2	0		No products were found	54, 56..60,61,63
1	2	1	1	There are not index products	54,56..60,61, 65..68,69,71
1	2	3	3	displayedProducts	54, 56..60,61,65. .68,69,73..75 ,76,78 79,81

Acoperire la nivel de decizie

```
//Acoperire de Decizie
[Test]
0 references
public void GetByCategory_NoProductsInCategory_ThrowsKeyNotFoundException()
{
    // Arrange
    int categoryId = 1;
    int index = 0;
    _mockCategoryRepository.Setup(repo => repo.Get(categoryId)).Returns(new Category());
    _mockProductRepository.Setup(repo => repo.GetByCategory(categoryId)).Returns(new List<Product>());

    // Act & Assert
    var ex = Assert.Throws<KeyNotFoundException>(() => _productService.GetByCategory(categoryId, index));
    Assert.That(ex.Message, Is.EqualTo($"No products were found for the specified category ID ({categoryId})."));
}

//Acoperire de Decizie
[Test]
0 references
public void GetByCategory_IndexGreaterThanProductCount_ThrowsKeyNotFoundException()
{
    // Arrange
    int categoryId = 1;
    int index = 3;
    var products = new List<Product> { new Product(), new Product() }; // Only 2 products
    _mockCategoryRepository.Setup(repo => repo.Get(categoryId)).Returns(new Category());
    _mockProductRepository.Setup(repo => repo.GetByCategory(categoryId)).Returns(products);
    _mockMapper.Setup(m => m.Map<List<ProductResponse>>(It.IsAny<List<Product>>())).Returns(new List<ProductResponse> { new ProductResponse(), new ProductResponse() });

    // Act & Assert
    var ex = Assert.Throws<KeyNotFoundException>(() => _productService.GetByCategory(categoryId, index));
    Assert.That(ex.Message, Is.EqualTo($"There are not ({index}) products in this category ({categoryId})."));
}
```

Definiție: Asigură că toate ramurile posibile de decizie din cadrul blocurilor if, while, for, etc., sunt testate.

Verifică toate combinațiile posibile ale condițiilor pentru a asigura corectitudinea fluxului de decizie.

Decizii:

-if (category == null || index < 0) (54)

-if (products.Count == 0) (61)

-if (productsDto.Count < index) (69)

- while(i < index) (am initializat i=0 deci daca index >0 se respecta mereu)

Date test:

category	**index**	**nr products**	**nr productsDto**	**Rezultat**	**Decizii acoperite**
null	-1			Category not found	if (category == null || index < 0)
1	2	0		No products were found	if (products.Count == 0)
1	2	1	1	There are not index products	if (productsDto.Count < index)

Acoperire la nivel de condiție

```
//Acoperire de Condiție
[Test]
0 references
public void GetByCategory_CategoryNotFound_ThrowsKeyNotFoundException()
{
    // Arrange
    int categoryId = 1;
    int index = 0;
    _mockCategoryRepository.Setup(repo => repo.Get(categoryId)).Returns((Category)null);

    // Act & Assert
    var ex = Assert.Throws<KeyNotFoundException>(() => _productService.GetByCategory(categoryId, index));
    Assert.That(ex.Message, Is.EqualTo($"The category with the specified ID ({categoryId}) was not found or the index value was less than 0."));
}

//Acoperire de Condiție
[Test]
0 references
public void GetByCategory_NegativeIndex_ThrowsKeyNotFoundException()
{
    // Arrange
    int categoryId = 1;
    int index = -1;
    _mockCategoryRepository.Setup(repo => repo.Get(categoryId)).Returns(new Category());

    // Act & Assert
    var ex = Assert.Throws<KeyNotFoundException>(() => _productService.GetByCategory(categoryId, index));
    Assert.That(ex.Message, Is.EqualTo($"The category with the specified ID ({categoryId}) was not found or the index value was less than 0."));
}
```

Asigură că fiecare condiție individuală (sau muchie de decizie) din cadrul unui bloc if, while, for, etc., este testată atât pentru adevărat, cât și pentru fals.

Verifică toate posibilele valori ale condițiilor pentru a asigura corectitudinea ramurilor de decizie.

Decizii:

-if (category == null || index < 0)(54)

-if (products.Count == 0) (61)

-if (productsDto.Count < index) (69)

- while(i < index) (76)

Conditii individuale:

-category==null, index<0

-products.Count==0

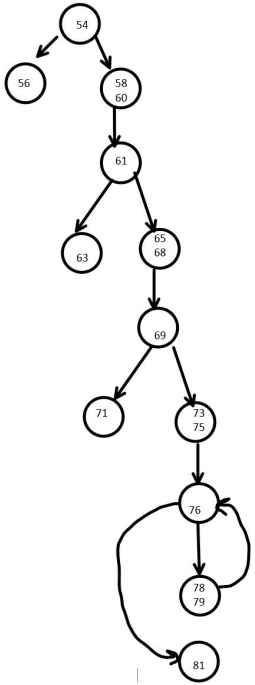
-productsDto.Count<index

-i<index (e setat i=0 deci daca index trece de primul if e ok mereu)

Date test:

category	**index**	**nr products**	**nr productsDto**	**Rezultat**	**Conditii individuale acoperite**
null				Category not found	category==null
1	-1			Category not found	index<0
1	2	0		No products were found	products.Count==0
1	2	1	1	There are not index products	productsDto.Count<index

Testarea circuitelor independente



- identificăm limita superioară pentru numărul de căi necesare obținerii acoperirii la nivel de ramură

V(G) = e - n + 1 (formula i)

e=numărul de muchii = 12;

n=numărul de noduri = 12; => V(G) = 1

Circuitul independent este:

a) 76, 78..79, 76

Mutanți

Aplicăm o serie de modificări ușoare peste funcția noastră **GetByCategoryId(int categoryId, int index).**

Modificări = { ++ înlocuit de +=2 , 0 înlocuit de 1, index înlocuit de index+1, 0 înlocuit de 1 }

Testele vor fi de forma { categoryId, index }

- t1 = {1, 0},
- t2 = {1, 1},
- t3 = {1, 2},
- t4 = {1, 3}.

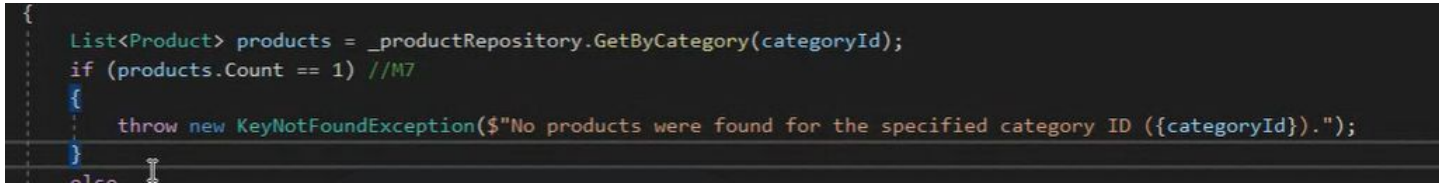
```
public List<ProductResponse> GetByCategoryM5(int categoryId, int index)
{
    Category category = _categoryRepository.Get(categoryId);
    index = index + 1; //M5
    if (category == null || index < 0)
    {
        throw new KeyNotFoundException($"The category with the specified ID ({categoryId}) was not found or the index value was less than 0.");
    }
    else
    {
        List<Product> products = _productRepository.GetByCategory(categoryId);
        if (products.Count == 0)
        {
            throw new KeyNotFoundException($"No products were found for the specified category ID ({categoryId}).");
        }
        else
        {

            List<ProductResponse> productsDto = _mapper.Map<List<ProductResponse>>(products);
            if (productsDto.Count < index)
            {
                throw new KeyNotFoundException($"There are not ({index}) products in this category ({categoryId}).");
            }
            else
            {
                List<ProductResponse> displayedProducts = new List<ProductResponse>();
                int i = 1; //M5
                while (i < index)
                {
                    productsDto[i].DiscountedPrice = productsDto[i].FullPrice - (productsDto[i].FullPrice * productsDto[i].Discount / 100);
                    displayedProducts.Add(productsDto[i]); i++;
                }
                return displayedProducts;
            }
        }
    }
}
```

Mutanți

	t1	t2	t3	t4	Mutant distins
P	List<Product> = []	List<Product> = [{Produs 1}]	List<Product> = [{Produs 1}, {Produs 2}]	List<Product> = [{Produs 1}, {Produs 2}, {Produs 3}]	
M1	List<Product> = []	List<Product> = [{Produs 1}]	**List<Product> = [{Produs 1}]**	-	Y
M2	***NULL***	-	-	-	Y
M3	***NULL***	-	-	-	Y
M4	**List<Product> = [{Produs 1}]**	-	-	-	Y
M5	List<Product> = []	**List<Product> = [{Produs 2}]**	-	-	Y
M6	List<Product> = []	**List<Product> = [{Produs 2}]**	-	-	Y
M7	List<Product> = []	List<Product> = [{Produs 1}]	List<Product> = [{Produs 1}, {Produs 2}]	List<Product> = [{Produs 1}, {Produs 2}, {Produs 3}]	N

Mutanti nedistinși (alive) = { M7 }

M7 nu este distins deoarece noi in categorie avem mereu 3 produse.



MS(T) = D/(L+D)

D=nr de mutanti distinsi = 6

L= nr de mutanti nedistinsi = 1 => **MS(T) = 6/7**