Testarea aplicațiilor software

Tematică proiect

Proiectul poartă denumirea “Cake Maker” și este realizat utilizând limbajul de programare C#. Aplicaţia simulează un aparat de făcut prăjituri automat cu resurse nelimitate. Rolul lui principal este să servească clientul cu prăjitura dorită. Clientul are acces la un meniu principal care îi va oferi anumite posibilităţi de selecţie. Principalele clase ale proiectului sunt cele care urmează:

- Command Panel - Această componentă este interfaţa cu clientul (command line interface). Are ca funcționalități:

  - Afișarea tuturor produselor disponibile pentru cumpărare;

  - Afișarea produselor din depozit;

  - Selectarea produsului dorit din meniu și trimiterea sa mai departe către CommandTaker pentru a fi preluată comanda;

  - Posibilitatea de a comanda mai multe produse de același tip.

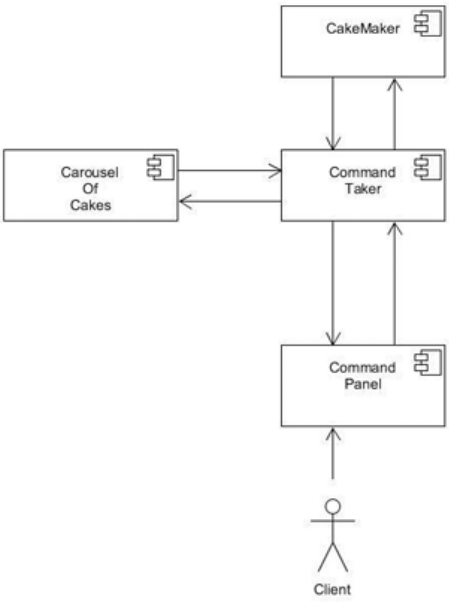
- Command Taker - Această componentă onorează comenzile date de client. Funcționalități:

  - Preia comanda de la CommandPanel. Verifică dacă produsul se află în CarouselOfCakes (mini depozit), dacă acesta există, va fi eliminat din depozit şi îi va fi servit clientului. În cazul în care nu se află în CarouselOfCakes, acesta îl va interoga pe CakeMaker cerându-i să onoreze comanda;

  - Verifică capacitatea curentă din CarouselOfCakes;

  - Dacă capacitatea din Carousel este mică, se va trimita o cerere către CakeMaker pentru a face atâtea prăjituri câte sunt necesare pentru a umple CarouselOfCakes la maxim.

- CarouselOfCakes - Acesta este un depozit de prăjituri. Are o capacitate maximă de 12 prăjituri. El va accepta doar elemente de tipul Cake.
- CakeMaker - Este cel care face prăjitura. El realizează o prăjitură într-un interval specificat în rețetă.
- RecipeCake – Reprezintă rețeta de prăjitură, având ca elemente denumirea prăjiturii și timpul de prepare al acesteia.
- Cake – Clasa prin care se specifică numele prăjiturilor.



(Diagrama de componente a aplicației CakeMaker)

NUnit Testing Framework

NUnit este un cadru de testare unitară pentru toate limbajele .Net. NUnit este un software cu sursă deschisă, iar NUnit 3.0 este lansat sub licență MIT. Acest cadru este foarte ușor de lucrat și are atribute ușor de utilizat pentru lucru.

Caracteristici

- Testele pot fi executate de la un runner de consolă, în Visual Studio printr-un adaptor de testare.
- Testele pot fi efectuate în paralel.
- Suport puternic pentru testele bazate pe date.
- Acceptă mai multe platforme, inclusiv .NET Core, Xamarin Mobile, Compact Framework și Silverlight.
- Fiecare caz de testare poate fi adăugat la una sau mai multe categorii, pentru a permite rularea selectivă.

NUnit oferă un runner de consolă (nunit3-console.exe), care este utilizat pentru executarea loturilor de teste. Runnerul de consolă funcționează prin NUnit Test Engine, care îi oferă capacitatea de a încărca, explora și executa teste. Când testele urmează să fie executate într-un proces separat, motorul folosește programul nunit-agent pentru a le rula.

Aserțiuni

NUnit oferă un set bogat de aserțiuni ca metode statice ale clasei Assert. Dacă o afirmație eșuează, apelul de metodă nu revine și este raportată o eroare. Dacă un test conține mai multe aserțiuni, oricare care urmează pe cea care a eșuat nu va fi executată. Din acest motiv, este de obicei recomandată o afirmație per test.

Testare unitară

În programarea computerelor, testarea unitară este o metodă de testare a software-ului prin care unitățile individuale de cod sursă - seturi de unul sau mai multe module de programe de calculator împreună cu datele de control asociate, procedurile de utilizare și procedurile de operare - sunt testate pentru a determina dacă sunt adecvate pentru utilizare.

Unitatea ar fi cea mai mică componentă care poate fi izolată în structura complexă a unei aplicații. Ar putea fi o funcție, o subrutină, o metodă sau o proprietate.

Testele unitare sunt teste automate sau teste manuale, scrise și rulate de dezvoltatorii de software pentru a se asigura că o secțiune a unei aplicații (cunoscută sub denumirea de „unitate") îndeplinește designul acesteia și se comportă conform intenției. Acestea sunt adesea efectuate de dezvoltatorul care a scris inițial codul, ca primă linie de apărare înainte de a efectua teste suplimentare.

În programarea orientată pe obiecte, o unitate este adesea o interfață întreagă, cum ar fi o clasă sau o metodă individuală. Scriind teste mai întâi pentru cele mai mici unități testabile, apoi comportamentele compuse dintre acestea, se pot construi teste cuprinzătoare pentru aplicații complexe.

Primul exemplu de test unitar (din fișierul UnitTests.cs):

```
[Test]
0 references
public void Cake_SetName_Succeeded()
{
    // Arrange
    Cake cake = new Cake();
    // Act
    cake.SetName("ChocolateCake");
    // Assert
    Assert.That(cake.GetName(), Is.EqualTo("ChocolateCake"));
}
```

(Test unitar 1)

Acest test unitar este conceput pentru a verifica dacă metoda SetName a clasei Cake funcționează așa cum este de așteptat. Se urmează pașii:

1. Arrange (pregătire):
   - Se crează o instanță a clasei Cake utilizând constructorul implicit. Această instanță, cake, va fi utilizată pentru a efectua testul.
2. Act (acțiune):
   - Se invocă metoda SetName a obiectului cake, stabilind astfel numele prăjiturii. În acest caz, numele "ChocolateCake" este utilizat pentru a seta numele prăjiturii.

3. Assert (asertare):

   - Se verifică rezultatul așteptat după ce s-a efectuat acțiunea
   - Se utilizează Assert.That și Is.EqualTo pentru a asigura că numele prăjiturii (cake.GetName()) este egal cu șirul așteptat "ChocolateCake".

Al doilea exemplu de test unitar (din fișierul UnitTests.cs):

```
[Test]
| 0 references
public void CakeMaker_TimeInTakeCommand_Succeeded()
{
    // Arrange
    RecipeCake recipe = new RecipeCake("Strawberry Cheesecake", 6);
    CakeMaker cakeMaker = new CakeMaker();
    var start = DateTime.Now;
    // Act
    Cake orderedCake = cakeMaker.TakeCommand(recipe);
    var end = DateTime.Now;
    var duration = (end - start).Seconds;
    // Assert
    Assert.That(duration, Is.EqualTo(6));
}
```

(Test unitar 2)

Acest test unitar este conceput pentru a verifica dacă metoda TakeCommand a clasei CakeMaker gestionează corect timpul de preparare al prăjiturii. Testul dispune de:

1. Arrange (pregătire):
   - Se definește o rețetă (RecipeCake) pentru o prăjitură de căpșuni care necesită 6 secunde pentru preparare.
   - Se creează o instanță a clasei CakeMaker pentru a efectua comanda.
2. Act (acțiune):
   - Se înregistrează momentul de început al procesului de comandă.
   - Se invocă metoda TakeCommand a obiectului cakeMaker pentru a pregăti prăjitura în conformitate cu rețeta specificată.
   - Se înregistrează momentul de final al procesului de comandă.
   - Se calculează durata totală a procesului în secunde.

3. Assert (asertare):
   - Se verifică dacă durata totală a procesului de comandă este egală cu durata specificată în rețetă (6 secunde).

Prin utilizarea Arrange, Act, și Assert, testele respectă structura tipică a testelor unitare. Dacă condițiile sunt îndeplinite, testele trec cu succes.

Testare Stub

Stub-urile, în testarea software-ului, sunt folosite pentru a furniza răspunsuri predefinite la apeluri de metode specifice. Stub-urile sunt folosite atunci când se dorește izolarea unității testată de o anumită dependență, dar nu este necesară verificarea interacțiunilor cu acea dependență. Stub-urile sunt concepute pentru a returna valori predefinite sau pentru a declanșa un comportament specific pentru a ajuta la testarea unității în diferite condiții.

Într-un ecosistem software vast, izolarea componentelor individuale poate fi o provocare. Stub-urile simplifică acest lucru. Prin înlocuirea componentelor dependente testerii se pot concentra numai pe componenta pe care intenționează să o testeze. Această izolare asigură acuratețea testelor, deoarece componentele externe, nedezvoltate, nu le influențează. În plus, stub-urile oferă un mediu controlat.

Metoda de pregătire (setup) pentru testele următoare:

```
[SetUp]
0 references
public void SetUp()
{
    Cake[] cakes1 = new Cake[12];
    Cake[] cakes2 = new Cake[12];
    for (int i = 0; i < 12; i++)
    {
        cakes1[i] = new Cake("empty");
        cakes2[i] = new Cake("Cheesecake");
    }
    carouselEmptyStub = new CarouselOfCakesStub(cakes1);
    carouselFullStub = new CarouselOfCakesStub(cakes2);
}
```

(Metoda cu atributul [SetUp])

Primul exemplu de test stub (din fișierul StubAndMockTests.cs):

```
[Test]
0 references
public void CommandTaker_CheckCarouselOfCakesIsNotEmpty_Succeeded()
{
    // Arrange
    CommandTaker commandTaker = new CommandTaker();
    commandTaker.SetCarouselOfCakes(carouselFullStub);
    // Act
    bool result = commandTaker.CheckCarouselOfCakes();
    // Assert
    Assert.IsTrue(result);
}
```

(Test stub 1)

Al doilea exemplu de test stub (din fișierul StubAndMockTests.cs):

```
[Test]
0 references
public void CommandTaker_CheckCarouselOfCakesIsEmpty_Succeeded()
{
    // Arrange
    CommandTaker commandTaker = new CommandTaker();
    commandTaker.SetCarouselOfCakes(carouselEmptyStub);
    // Act
    bool result = commandTaker.CheckCarouselOfCakes();
    // Assert
    Assert.IsFalse(result);
}
```

(Test stub 2)

În acest exemplu prin folosirea stub-urilor carouselFullStub și carouselEmptyStub (CarouselOfCakesStub), se poate controla comportamentul caruselului de prăjituri în timpul testelor, oferind un mediu controlat pentru verificarea unei funcționalități din CommandTaker: dacă este caruselul de prăjituri gol sau nu. Au fost urmați pașii:

1. SetUp (pregătirea testelor):

   În metoda SetUp, se pregătesc două obiecte de tip stub pentru a simula două situații diferite ale caruselului de prăjituri:

- carouselEmptyStub este un stub pentru un carusel gol, având un array de prăjituri cu denumirea "empty".

- carouselFullStub este un stub pentru un carusel plin, având un array de prăjituri cu denumirea "Cheesecake".

2. Arrange (pregătire):
   - În testul CommandTaker_CheckCarouselOfCakesIsNotEmpty_Succeeded, se crează o instanță a clasei CommandTaker.
   - Se setează caruselul de prăjituri al CommandTaker cu unul dintre obiectele stub create anterior.
3. Act (acțiune):
   - Se invocă metoda CheckCarouselOfCakes a CommandTaker pentru a verifica dacă caruselul este gol sau nu.
4. Assert (asertare):
   - În primul test se utilizează Assert.IsTrue pentru a verifica că rezultatul întors de CheckCarouselOfCakes este true, ceea ce indică faptul că caruselul este plin în acest caz.
   - În al doilea test se utilizează Assert.IsFalse pentru a verifica că rezultatul întors de CheckCarouselOfCakes este false, ceea ce indică faptul că caruselul este gol în acest caz.

Testare Mock

În testarea Mock, înlocuim obiectele dependente cu obiecte simulate. Aceste obiecte simulate simulează sau „ironizează" comportamentul obiectelor reale într-un mod controlat și prezintă caracteristicile exacte ale celor autentice.

Ele sunt de obicei folosite pentru a înlocui obiecte reale care sunt dificil de instanțiat sau de operat într-un mediu de testare. Obiectele simulate le permit testorilor să specifice interacțiunile așteptate cu obiectul, permițându-le să verifice dacă unitatea testată se comportă conform așteptărilor.

Scopul principal al testării simulate este de a centraliza accentul pe testare fără a fi preocupat de dependențe.

Primul exemplu de test mock (din fișierul StubAndMockTests.cs):

```
[Test]
0 references
public void RecipeCake_NegativeTimeInput_SetsTimeToZero()
{
    // Arrange
    var recipe = new RecipeCake();

    // Mock for Console
    var consoleMock = new Mock<TextWriter>();
    Console.SetOut(consoleMock.Object);

    // Act
    recipe.SetTime(-5);

    // Assert
    Assert.That(recipe.GetTime(), Is.EqualTo(0));

    // Check if Console.WriteLine was called with a specific message
    consoleMock.Verify(w => w.WriteLine("Error: Preparation time cannot be negative. It will be set to 0."), Times.Once);
}
```

(Test mock 1)

Acest test are ca scop verificarea comportamentului clasei RecipeCake în situația în care este setat un timp de pregătire negativ. Se pot identifica următoarele etape:

1. Arrange (pregătirea):
   - Se creează o instanță a clasei RecipeCake.
   - Se utilizează un mock pentru clasa Console pentru a intercepta ieșirea la consolă și a verifica mesajele de eroare.
2. Act (acțiune):
   - Se invocă metoda SetTime a obiectului recipe cu un argument negativ (-5).
3. Assert (asertare):
   - Se verifică dacă timpul de pregătire al rețetei (recipe.GetTime()) a fost setat la 0, în conformitate cu specificația testului.
4. Verificarea apelului la consolă:
   - Se verifică dacă metoda Console.WriteLine a fost apelată exact o dată cu un mesaj specific de eroare.

Testul verifică că, atunci când timpul de pregătire este setat la o valoare negativă, acesta este corect ajustat la 0 conform specificației și se afișează un mesaj de eroare corespunzător în consolă.

Al doilea exemplu de test mock (din fișierul StubAndMockTests.cs):

```
[Test]
| 0 references
public void CommandPanel_ShowProductsInCarousel_PrintsCorrectMessage()
{
    // Arrange
    CommandPanel commandPanel = new CommandPanel();

    // Stub for ICommandTaker
    var commandTakerStub = new Mock<ICommandTaker>();
    commandTakerStub.Setup(taker => taker.GetCakesFromCarousel())
                    .Returns(() => Enumerable.Range(0, 12)
                                             .Select(i => i < 3 ? new Cake($"Cake{i}") : new Cake("empty"))
                                             .ToArray())
                    .Verifiable();
    commandPanel.SetCommandTaker(commandTakerStub.Object);

    // Redirect Console.Out to a StringWriter
    using (StringWriter sw = new StringWriter())
    {
        Console.SetOut(sw);

        // Act
        commandPanel.ShowProductsInCarousel();

        // Assert
        string expectedOutput = "The products in the carousel are:\r\nCake0 | Cake1 | Cake2 | \r\n";
        Assert.That(sw.ToString(), Is.EqualTo(expectedOutput));

        // Check if the GetCakesFromCarousel method has been called exactly once
        commandTakerStub.Verify(taker => taker.GetCakesFromCarousel(), Times.Once);
    }
}
```

(Test mock 2)

În acest test se foloseşte atât stub, cât şi mock pentru a verifica comportamentul metodei ShowProductsInCarousel a clasei CommandPanel. Sunt urmate etapele:

1. Arrange (pregătirea):
   - Se creează o instanță a clasei CommandPanel.
   - Se crează un stub pentru interfața ICommandTaker pentru a simula comportamentul metodei GetCakesFromCarousel (Stub-ul commandTakerStub este configurat să returneze o listă simulată de prăjituri în momentul apelului la GetCakesFromCarousel, anume 3 prăjituri care să nu fie empty și 9 care să fie empty).
   - Console.Out este redirecționat către un obiect StringWriter pentru a captura ieșirea la consolă în timpul testului.
2. Act (acțiune):
   - Se invocă metoda ShowProductsInCarousel a obiectului commandPanel.
3. Assert (asertare):
   - Se verifică dacă ieşirea la consolă corespunde cu un şir aşteptat, care reprezintă produsele din carusel (sunt afișate doar prăjiturile care nu sunt empty).
4. Verificarea apelului la consolă:
   - Se verifică dacă metoda GetCakesFromCarousel a fost apelată exact o dată pe obiectul commandTakerStub.