**LUCRARE ȘTIINȚIFICĂ**

**Testarea programului de gestionare a**

**unei biblioteci**

**Autor: Rogojanu Rebeca-Maria**

**Email:** [**rebeca.rogojanu@student.upt.ro**](mailto:rebeca.rogojanu@student.upt.ro)

**30 Decembrie 2024**

# INTRODUCERE

În această lucrare am detaliat tehnicile de testare folosite pe un sistem de gestionare a unei bibliotecii. Am realizat diverse teste, printre care testarea unitară – Unit Testing asupra clasei principale „Biblioteca” și asupra claselor utilizate în cadrul clasei „Biblioteca” („CardBiblioteca” și „Carte”), punând accent pe interdependențele dintre clase.

Printre testele abordate în testarea software a programului am inclus tehnicile Domain Testing, Test Doubles și am utilizat framework-ul Mocking. Așadar în cele urmează va fi prezentat programul de gestionare a bibliotecii, teste la limita pentru funcțiile din cadrul clasei „Biblioteca”, testarea comportamentului codului la excepții, teste prin injectare de parametrii, teste Stub, teste Mock, acoperirea testelor realizare prin analiza cu „Code Coverage” utilizând comanda „coverlet”.

# PREZENTAREA PROGRAMULUI DE GESTIONARE A UNEI BIBLIOTECI

## Descrierea claselor și a funcționalității

Programul de gestionare a bibliotecii este o aplicație software creată pentru organizarea activităților zilnice desfășurate în cadrul unei biblioteci. Utilă la organizarea resurselor, interacțiunii cu oamenii, oferă servicii de împrumut-retur, achiziționare de cărți și posibilitatea unui card de bibliotecă prin care se ține evidența cărților împrumutate și unor puncte bonus, ce pot fi folosite pentru achiziționarea unei cărți.

* + 1. **Clasa „Biblioteca”**

Clasa este implementată în fisierul „Librarie.cs”, utlizează colecția Dictionary<string, ICarte> pentru a gestiona colecția de cărți și resursele din cadrul bibliotecii. Dictionary<TKey, TValue> este o colecție generică care stochează perechi cheie-valoare fără o anumită ordine. Cheia utilizată în cadrul colecției, de tip string, va fi reprezentată de concatenarea dintre titlul carții și de autor (ex: „string cheie = carte.Titlu + carte.Autor”), deoarece o carte este reprezentată cel mai bine de titlu și autor, existând cărți cu același titlul. Valoarea colecției este un obiect de tip Carte, ce va fi prezentat mai jos.

Clasa are următoarele funcționalității:

* Funcția „AdaugăCarte(ICarte carte)” – permite adăugarea unei cărți noi în cadrul colecției. În cazul în care cantitatea cărții adăugate este zero/negativă sau daca cartea există în colecție atunci funcția va returna „False”, verificând dacă cheia realizata din titlul și autorul cărții există deja în colecție (ex: carti.ContainsKey(cheie));
* Funcția „AdaugăCantitate(string cheie, int cantitate)” – are rolul de a suplimenta numărul de exemplare pentru o carte existentă în bibliotecă. Se verifcă în cadrul funcției daca catitatea e pozitivă și daca cartea există.
* Funcția „ImprumutăCarte(string cheie, ICardBiblioteca card)” – permite unei cărți existente în bibliotecă, dacă numărul de exemplare este pozitiv, sa fie împrumutată.
* Funcția „ReturnezăCarte(string cheie, ICardBiblioteca card)” – permite unei cărți înregistrate în bibliotecă, sa fie returnată.
* Funcția „CumpăraCarte(string cheie, int numarExemplare, ICardBiblioteca card)” – realizează cumpărarea cărții din bibliotecă prin decrementarea numărului de exemplare pentru acea carte și apelarea funcției „AdaugaPuncteBonus()” din cadrul clasei „CardBiblioteca”, în cazul în care condițiile sunt îndeplinite.
  + 1. **Interfața „ICarte”**

Definiția interfeței se găsește în fisierul „ICarte.cs”. Conține trei componente: „Titlu” (titlul cărții, de tip string), „Autor” (autorul cărții, de tip string), „NumarExemplare” (numărul de exemplare pentru cartea respectivă, de tip intreg), „AnPublicare”(anul de publicare al cărții , de tip uint), „Categorie” (o categorie de încadrare a cărții).

* + 1. **Clasa „Carte”**

Clasa „Carte” implementeză interfata „ICarte”. Clasa reprezintă o dependență pentru clasa „Biblioteca”. Obiectul de tip „Carte”, ce conține informațiile despre carte este folosit în cadrul colecției din clasa „Biblioteca”, reprezintă resursa din bibliotecă. Proprietățiile clasei sunt: „Titlu”, „Autor”, „NumarExemplare”, „AnPublicare”, „Categorie”.

Clasa conține doar un constructor ce primește argumentele necesare pentru a inițializa obiectul carte, atunci când acesta este creat în program.

* + 1. **Interfata „ICardBiblioteca”**

Definiția interfeței „ICardBiblioteca” se găsește în fișierul „ICardBiblioteca.cs”. Aceasta e necesară pentru gestionarea cardului de membru al bibliotecii, prin intermediul căruia se realizează împrumutul cărțiilor și strângerea punctelor bonus. Interfața conține următoarele proprietăți: „NumeDetinator” de tip string, „PuncteBonus” de tip intreg și o listă cu cărțiile imprumutate „CartiImprumutate”.

Metodele definite în cadrul interfeței „ICardBiblioteca” sunt: „ContineCarte(string titlu, string autor)”, „CarteImprumutata(ICarte carte)”, „ReturneazaCarte(ICarte carte)”, „Puncte()”, „AdaugaPuncteBonus(int puncte)”, „ScadePuncteBonus(int puncte)”.

* + 1. **Clasa „CardBiblioteca”**

Clasa „CardBiblioteca” implementează interfața „ICardBiblioteca”, prin care putem crea pentru fiecare membru al bibliotecii un card, utilizat în activitatea de împrumut/retur a unei cărți și pentru a monitoriza punctele bonus, acumulate în urma achiziționării unei cărți, punctele bonus putând fi folosite pentru a cumpăra cărții.

Metodele implementate în cadrul clasei au următorele funcționalități:

* Funcția „ContineCarte(string titlu, string autor)” – returnează True dacă se găsește cartea dată ca argument, după titlu și autor, în lista de cărți împrumutate („List<ICarte> CartiImprumutate”);
* Funcțiile „CarteImprumutata(ICarte carte)” și „ReturneazaCarte(ICarte carte)” – au rolul de a adăuga, respectiv elimina cartea, dată ca argument, din lista de cărți împrumutate.
* Funcțiile „AdaugaPuncteBonus(int puncte)” și „ScadePuncteBonus(int puncte) – permit adăugarea, respectiv scăderea, punctelor bonus pentru respectivul card de bibliotecă cu valoarea dată ca argument funcțiilor.

## Dependența claselor

In prgramul nostru de gestionarea a unei biblioteci, relațiile de dependență între clase sunt esențiale. „Biblioteca” gestionează colecția de cărți, deci depinde direct de clasa „Carte”, utilizând instanțe ale clasei „Carte”. Clasa „Biblioteca” interecționează cu interfața „ICardBiblioteca”, implementata prin clasa „CardBiblioteca”, pentru a gestiona împrumuturile, returirile și punctele bonus acumulate. În momentul în care un membru returneză o carte, „Biblioteca” va actuliza informațiile (lista de cărți împrumutate) din cadrul obiectului de tip „CardBiblioteca”.

Interațiunea dintre interfața „ICarte” și „Carte”, la fel ca și în cazul interfeței „ICardBiblioteca” și „CardBiblioteca” este evidentă, interfața defindește funcționalitățiile si proprietățiile pentru o carte, respectiv cu card de biblioteca, iar clasele oferă implementări ale acestor interfețe.

# TEHNICIILE DE TESTARE UTILIZATE

Unit Testing sau testarea unitară presupune scrierea unui cod de testare, care validează codul aplicației noastre, testarea devenind automată. Aceasta presupune testarea părțiilor mici din codul aplicației, cum ar fi metode si funcții, în izolare de restul codului, fiecare test validează un comportament al aplicației. Scopul este de a asigura că fiecare bucată de cod, componenta (clasa, metoda, functie) funcționează cum este de așteptat și de a reduce numărul de greșeli pe care le putem face la scrierea codului.

Există mai multe software-uri automate de testare unitară disponibile pentru a ajuta la testarea unitară în testarea software-ului, în cazul aplicației am utilizat NUnit. NUnit este utilizat pe scară largă în cadrul de testare unitară pentru toate limbile .net. Este un instrument open source care permite scrierea manuală a scripturilor. Acceptă teste bazate pe date care pot rula în paralel.

Testele unitare folosesc bucăți de cod care seamănă cu codul din aplicație, în cazul nostru funcțiile din clasa „CardBiblioteca” și clasa „Carte”, dar în realitate folosesc doar la teste. Stub-urile și mock-urile sunt cele mai întânite duble de testare.

Am utilizat în cadrul testelor modelul AAA (Arrange-Act-Assert), un model folosit pentru aranjarea codului în metodele de test unitar. Modelul Arrange-Act-Assert, este consider un model bun în practica, pentru crearea testelor într-un mod mai natural. Ideea este de a dezvolta un test unitar în trei pașii: Arrange (configurarea obiectelor de testare, pregătirea cerințelor pentru test), Act (apelează funcția ce se dorește testată, realizează acțiunea), Assert (verifică rezultatul).

În cazul testelor folosite pe parcursul aplicației am realizat un Arrange în funcția Setup(), în care inițializam un obiect carte, cardbiblioteca, câteva obiecte de tip carte și adăugam cărțiile în lista de cărți din bibliotecă.

.

## Domain Testing

Testarea domeniului este un tip de testare în care aplicația este testată pentru a se asigura că nu se acceptă valori nevalide sau în afara intervalului. Este testată ieșirea cu un număr minim de intrări pentru a vedea dacă sistemul acceptă intrarea în intervalul necesar sau nu. White Box Testing este un exemplu perfect de Domain Testing. Limitele pentru valorii, obiecte, metode, definesc un domeniu.

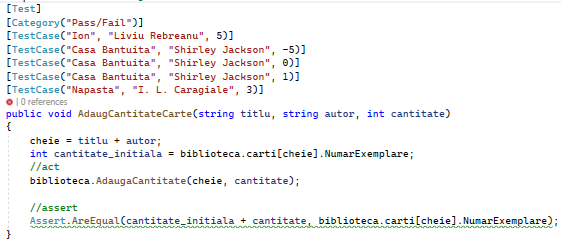
Prin aceste teste analizăm fiecare punct din apropierea acestor limite, atât valorile de intrare valide, cât și cele invalide și un punct din interiorul intervalului. Tesăm orice domeniu care conține funcționalități de intrare și ieșire, introducem valorea de intrare (de testare) și verificăm rezultatul.

În fișierul „Domain\_UnitTesting.cs” se găsesc în total treizeci și unu de teste. Teste care verifică limitele și excepțiile fiecărei funcții.

Inițial am inițializat obiectul bibliotecă și restul obiectelor de care avem nevoie, cum am precizat mai sus. Urmând să creăm teste pentru fiecare funcție.

În cazul funcției „AdaugaCarte()” am realizat patru teste. Primul test, verifică cu valori valide dacă funcția adaugă cartea în lista de cărți și dacă cantitatea introdusă de noi pentru cartea testată este aceeași cu cea memorată pentru obiectul adăugat în listă. Al doilea test, verifică dacă la introducerea unei cărți deja existente în bibliotecă primim eroare, deoarece acea carte există. Al treilea test și al patrulea test verifică dacă în cazul unei carți noi care are cantitatea zero, respectiv o valoare negativă, va returna false, adică că cartea nu a fost introdusă in bibliotecă, din cauza că nu avem cantitate mai mare ca zero. După aceste teste putem trage concluzia că funcția funcționează cum este așteptat.

Pentru funcția „AdaugăCantitate” am realizat 6 teste. Primul test, verifică cu valori valide, care nu sunt la limită, dacă funcția adaugă la cantiatatea initială, a unei anumite cărți existe în bibliotecă, numărul de cărți cu care a fost suplimentat stocul. Al doilea și al treilea test verifică dacă pentru zero sau pentru cantitate negativă, nu se modifică cantitatea și se returnează excepția „Cantitatea trebuie să fie pozitivă”. Al patrulea verifică o valorea validă la limită, iar al cincelea test verifică daca în cazul în care cartea dată ca input nu există, nu se modifică nimic și se returneză excepția „Cartea specificată nu există în bibliotecă”.



Pentru funcția „CarteaExista()”, care are ca input doar cheia, formată din titlu si autor, avem doar două teste. Un test în care cheia există în bilbiotecă și returnează – true și un test în care cheia testată e invalidă și se returnează – false, funcția având comportamentul așteptat.

O imagine care conține text, Font, captură de ecran, linie

Descriere generată automat

Pentru funcția „ImprumutăCarte()” a fost nevoie de 4 teste. Un test în care se verifică dacă se poate împrumuta o cartea existentă în bibliotecă, pentru care cantitatea nu era la limită și verificăm daca cantiatea a fost decrementată și dacă cartea împrumutată a fost adăugată în lista de cărți împrumutate pentru acel card de membru. Al doilea test verifică același comportament pentru o valoare validă la limită. Al treilea test verifică pentru o carte neînregistrată în bibliotecă și se așteaptă sa nu se facă modificări, iar la patrulea test verifică dacă o carte validă cu stocul zero, se poate împrumuta. După aceste teste s-a observat că funcția are coportamentul așteptat.

Funcția „ReturneazăCarte()” are o implementare asemănătoare cu funcția „ImprumutaCarte()”, dar necesită mai puține teste. Testăm dacă pentru o carte existentă în bibliotecă se poare realiza returul, numărul de exemplare pentru acea carte se incrementează și cartea se elimină din lista de cărți imprumutate. Al doilea test verifică că nu se poate realiza returul pentru o carte neînregistrată în bibliotecă.

Pentru funcția „CumparaCarte()” am realizat opt teste, care verifică excepțiile și limitele. Primul test verifică excepția în cazul în care cartea dată ca input e invalidă, nu se găsește în bibliotecă. Al doilea și al treilea test verifică dacă pentru un număr negativ, respectiv zero, de cărți ce se dorește a fi cumpărat nu se realizează cumpărarea și se aruncă excepția „Numărul de exemplare specificat este invalid”. Al patrulea test vertifică pentru o cantitate introdusă ca input, mai mare decât numărul de exemplare pentru cartea respectivă, nu se realizează modificarea și se aruncă exceptia „Nu sunt destule produse in stoc”. Iar al cincelea test verifică valoarea nevalidă, la limita superioară.

O imagine care conține text, captură de ecran, Font, număr

Descriere generată automat

Următoarele trei teste verifică valori valide, la limită pentru o carte existentă: „1” limita inferioară acceptată, valoarea superioară acceptata și am verficat valorarea superioră validă minus unu, testând astefel o valoare din interval. După aceste teste putem concluziona că funcția are comportamentul așteptat.

O imagine care conține text, captură de ecran, Font

Descriere generată automat

Putem observa că în urma testelor de mai sus, a fost testată funcționalitatea, comportamentul fiecărei funcții din clasa „Biblioteca” luându-se în considerare limitele, testarea valorilor valide din domeniu și testarea excepțiilor.

## Test Doubles

Test Doubles presupune înlocuirea dependențelor pe care le are System Under Test-ul nostru, atunci când dorim să testăm în izolare. System Under Test este soft-ul pe care-l testăm. Printre tipurile de teste doubles în aplicația noastră am utilizat Stub și Mock.

Aceste teste sunt importante în testarea unitare pentru izolarea bucățiilor de cod testate și pentru verificarea că acestea interacționează corect cu dependențele sale.

## Stub Testing

Prin Stub Testing, putem testa SUT-ul în izolare, simulând dependințele. O interfață stub simulează un obiect real folosind câteva metode. Este un obiect cu date preexistente care furnizează o valoare constantă, indiferent de intrare.

În aplicația noastră folosim Stub care implementează interfețele „ICarte” și „ICardBiblioteca”, așadar folosim o altă clasă care seamănă cu clasa originală (dependința).

Testele Stub se găsesc în fișierul „TestStub.cs”, unde am implementat și clasele stub pentru teste:

O imagine care conține text, captură de ecran, Font

Descriere generată automat

O imagine care conține text, captură de ecran, document, Font

Descriere generată automat

Pentru a utiliza aceste clase, a fost nevoie sa modific Arrang-ul astfel încat la inițializarea carțiilor și cardului să utilizez noile funcții:

O imagine care conține text, captură de ecran, Font

Descriere generată automat

În cele ce urmează o să prezint câteva teste Stub din cadrul aplicației, pentru a se putea observa cum se utlizează funcțiile din clase.

## Mock Testing

Un Mock este o dublă de testare care validează colaborarea între clase. Mock-ul validează apeluri de metode, cu anumiți parametri, de un anumit număr de ori. Din această cauză, un mock poate fi folosit și la validarea apelurilor de metode care nu întorc valori.

# 4. CONCLUZII

În **interprinderii**, Unit testing-ul poate reduce semnificativ timpul petrecut cu repararea bug-urilor din aplicații, reducând încărcarea colegilor care se ocupă de suport și testare și permitând introducerea de noi funcționalități mai repede, ceea ce conduce la competivitate crescută

# BIBLIOGRAFIE

1. Ovidiu Banias, Domain Testing & Unit Testing, Universitatea Politehnica Timișoara, 2023.
2. Ovidiu Banias, Test Doubles, Universitatea Politehnica Timișoara, 2023
3. Ovidiu Banias, Code Coverage, Universitatea Politehnica Timișoara, 2023
4. <https://www.tutorialsteacher.com/csharp/csharp-dictionary>, accesat în data de 29.12.2023
5. <https://www.todaysoftmag.ro/article/377/din-uneltele-artizanului-software-unit-testing>, accsat în data de 30.12.2023
6. <https://www.tutorialspoint.com/what-is-domain-testing-in-software-testing> , accesat în data de 30.12.2023
7. <https://docs.telerik.com/devtools/justmock/basic-usage/arrange-act-assert>, accesat în data de 02.01.2024
8. <https://learn.microsoft.com/en-us/visualstudio/test/using-code-coverage-to-determine-how-much-code-is-being-tested?view=vs-2022&tabs=csharp>, accesat în data de 01.01.2024