

# TESTARE UNITARĂ ÎN C#

**Burlacu Mircea-Florian**

**Niță Andreea-Diana**

**Testarea Sistemelor Software | 2024**

# FRAMEWORK

1

În acest proiect am ales să folosim NUnit.

NUnit este un framework popular pentru testarea unitară în .NET, inclusiv pentru proiectele scrise în C#. Acesta oferă o serie de caracteristici utile pentru scrierea și rularea testelor unitare.

Elemente cheie:

- atributul SetUp -> marchează metodele care trebuie executate înainte de fiecare test;
- atributul TestFixture -> marchează o clasă care conține teste;
- atributul Test -> marchează metodele care reprezintă teste individuale;
- metodele pot fi rulate automat cu comanda dotnet test, care construiește proiectul de test și rulează toate testele definite;
- metoda Assert.That -> permite verificarea condițiilor sau a relațiilor între valori în cadrul testelor.

# IMPLEMENTARE

## ● Punct

Definit de coordonatele 3D ( $x, y, z$ ).

Utilitare pentru două puncte date: adunare, comparație, distanță, produs scalar.

## ● Cub

Definit fie de vârfuri (8), fie de centru și de lungimea muchiei.

Utilitare: bounding box, verificare dacă un punct dat se află în interiorul bBox-ului.

## ● Triunghi

Definit de trei puncte date.

Utilitare: calcularea lungimii muchiilor, verificări pentru tip (echilateral, isoscel, dreptunghic, obtuzunghic).

## ● Patrulater

Definit de patru puncte date.

Utilitare: calcularea lungimii muchiilor, verificări pentru tip (trapez, paralelogram, dreptunghi, pătrat, romb).

# TESTE GENERALE

- PointCreation: Verifică dacă un punct poate fi creat corect și dacă proprietățile acestuia sunt setate corect.
- DotProductTest: Testează calculul produsului scalar dintre două puncte.
- NullCubeCreation: Verifică crearea unui cub cu parametri implicit, asigurându-se că nu există valori neașteptate.
- CubeCreationVertexList: Testează crearea unui cub folosind o listă de puncte ca argument, verificând dacă cubul este creat corect și dacă proprietățile sale sunt setate corect.
- CubeCreationCenter: Verifică crearea unui cub centrat la origine, asigurându-se că toate vârfurile sunt setate corect.
- UtilitiesTestDotProductTest: Testează o funcție utilitară pentru calculul produsului scalar, comparând rezultatul cu un calcul manual.

# TESTE DE PARTIȚIONARE ȘI ANALIZA VALORILOR DE FRONTIERĂ

## Bounding Box

- EquivalencePartitioningBoundingBox: Testează funcția `IsPointInsideCubeBB` folosind partiționarea echivalentă, verificând dacă punctele sunt în interiorul cubului sau nu.
- BoundaryTestingBoundingBox: Similar cu testul anterior, dar folosește analiza valorilor de frontieră pentru a testa funcția `IsPointInsideCubeBB`.

## Distanță

- EquivalencePartitioningDistance: Testează funcția `ComputeDistance` folosind partiționarea echivalentă, verificând distanțele calculate între puncte pentru diferite tipuri de distanță.
- BoundaryTestingDistance: Similar cu testul anterior, dar folosește analiza valorilor de frontieră pentru a testa funcția `ComputeDistance`.

# CLASE DE ECHIVALENȚĂ

## Bounding Box

Intrări:

- un punct;
- o valoare a toleranței;
- o listă de vârfuri, care formează un cub;
- opțional, lungimea laturii.

Lungimea listei,  $n$ , trebuie să fie 1 sau 8. Astfel, putem distinge mai multe clase de echivalență:

- $N_1 = \{n \mid n < 1\}$ ;
- $N_2 = 1$ ;
- $N_3 = 2..7$ ;
- $N_4 = 8$ ;
- $N_5 = \{n \mid n > 8\}$ .

Toleranța nu determină alte clase de echivalență.

## Distanță

Intrări:

- Punctul de plecare;
- Punctul de sosire;
- Tipul distanței (Euclidean, Manhattan, Chebyshev).

În funcție de parametrul `type`, putem distinge două clase de echivalență:

- $T_1 = \{x \mid x \in \{\text{euclidean, manhattan, chebyshev}\}\}$ ;
- $T_2 = \{x \mid x \notin \{\text{euclidean, manhattan, chebyshev}\}\}$ .

Coordonatele punctelor nu determină alte clase de echivalență.

# VALORI DE FRONTIERĂ

## ● Bounding Box

Având clasele de echivalență identificate, putem distinge valorile de frontieră:

- $N_1 = n(|\text{vertices}|) = 1$
- $N_2 = n(|\text{vertices}|) = 8$
- $N_3 = n(|\text{vertices}|) = 0$
- $N_4 = n(|\text{vertices}|) = 9$
- $N_5 = n(|\text{vertices}|) = 5$

Pentru restul argumentelor putem lua valori arbitrare.

## ● Distanță

Pentru valorile de frontieră, putem introduce clasele:

- $P_1 = \{a, b \mid a = b\}$
- $P_2 = \{a, b \mid ax < 10^{-5}, by < 10^{-6}\}$

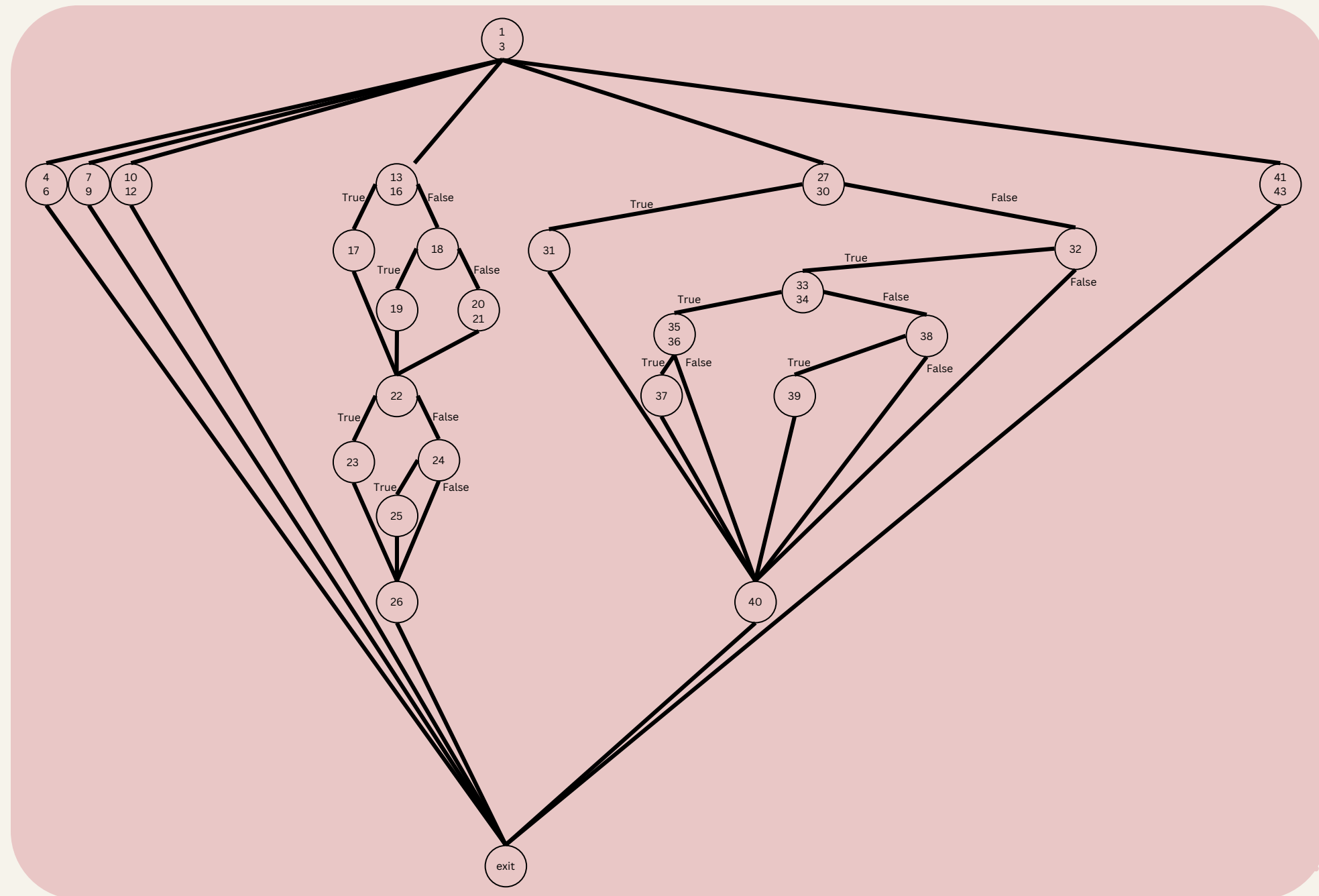
Restul argumentelor pot lua orice valoare.

# GRAF DE DEPENDENȚE

7

Am creat CFG pentru funcția care decide tipul de figură construit pe baza unei liste de puncte date. Aceasta are la bază un switch cu 6 cazuri:

- 0 - figura nu există;
- 1 - punct;
- 2 - linie;
- 3 - triunghi, cu tipul determinat nested;
- 4 - patrulater, cu tipul determinat nested;
- default - figură necunoscută.





# GENERARE DE MUTANȚI

Pentru a genera mutanți am folosit utilitarul Stryker. După rularea testelor mutațiilor, Stryker generează un raport care arată care mutații au supraviețuit (adică testele nu au detectat mutația) și care nu. Un scor de mutație de 100% indică faptul că toate mutațiile au fost detectate de suita de teste, ceea ce sugerează că testele acoperă corespunzător funcționalitatea codului.

## 1 Primul raport

All files											
10521173											
File / Directory	Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	35.12	105	21	0	173	30	0	6	105	194	335
Cube.cs	76.32	87	17	0	10	9	0	5	87	27	128
FigureDecision.cs	0.00	0	0	0	29	1	0	1	0	29	31
Point.cs	43.90	18	4	0	19	6	0	0	18	23	47
Quadrilateral.cs	0.00	0	0	0	76	8	0	0	0	76	84
Triangle.cs	0.00	0	0	0	26	4	0	0	0	26	30
Utilities.cs	0.00	0	0	0	13	2	0	0	0	13	15

## 2 Raport după consolidarea testelor

All files											
1569350											
File / Directory	Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
All files	52.17	156	93	1	50	30	0	6	156	143	335
Cube.cs	76.32	87	17	0	10	9	0	5	87	27	128
FigureDecision.cs	0.00	0	0	0	29	1	0	1	0	29	31
Point.cs	73.17	29	6	1	5	6	0	0	30	11	47
Quadrilateral.cs	10.53	8	62	0	6	8	0	0	8	68	84
Triangle.cs	73.08	19	7	0	0	4	0	0	19	7	30
Utilities.cs	92.31	12	1	0	0	2	0	0	12	1	15



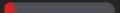

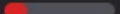

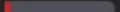

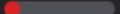

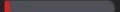

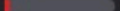

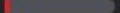
# RAPORT MICROSOFT COPILOT

Query-urile date sunt referitoare la generarea unor teste de partiționare în clase de echivalență și de analiză a valorilor de frontieră pentru bounding box și distanță (detaliat a se vedea în screenshot-urile atașate în folder-ul intitulat "Copilot" al proiectului).

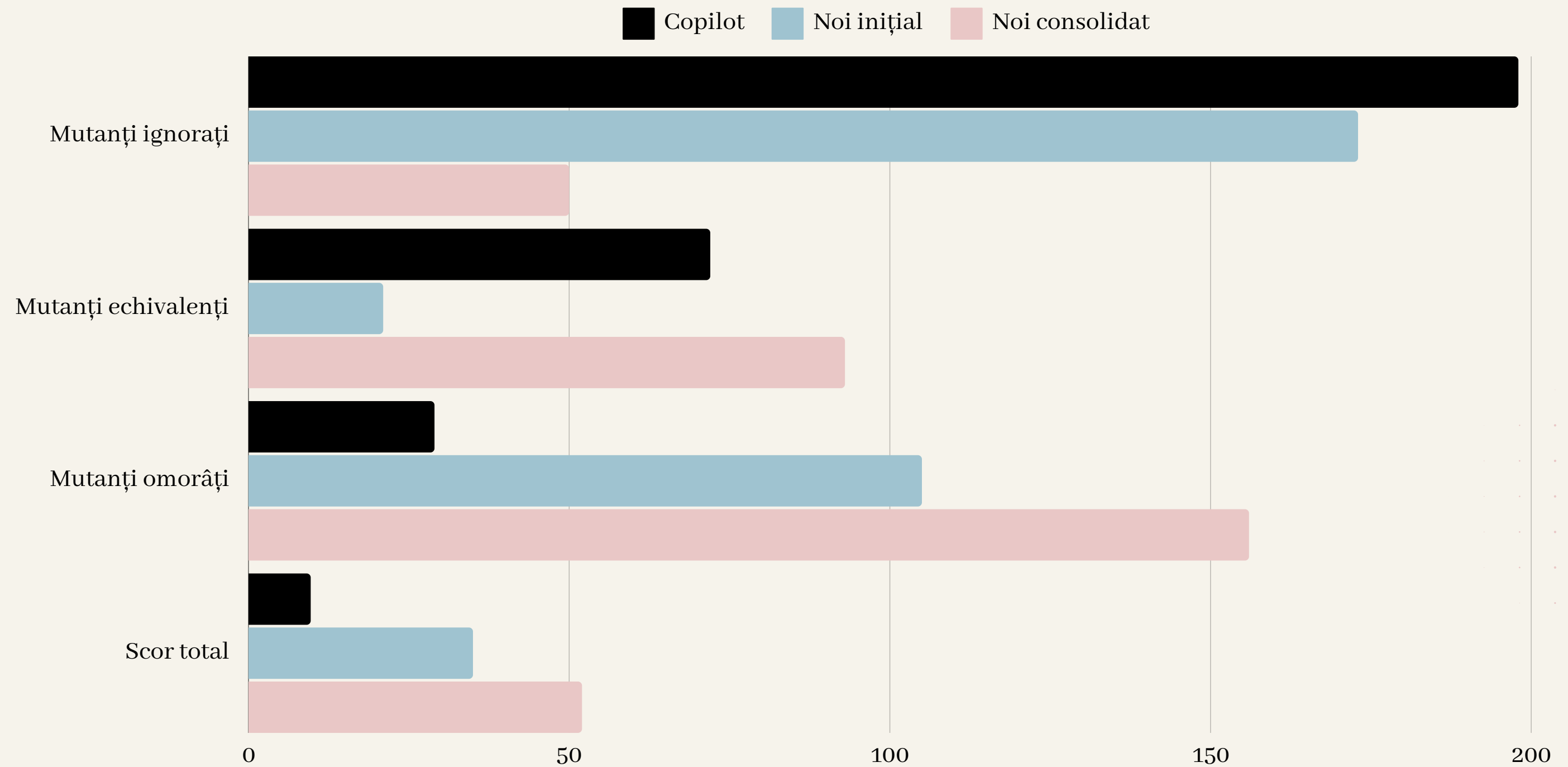
Testele generate de AI au o eficiență scăzută, comparativ cu testele scrise de noi.

Rată de acoperire:

- AI - 9,7%
- noi, variantă inițială - 35,12%
- noi, variantă consolidată - 52,17%

All files												
29		72				198						
File / Directory		Mutation score	Killed	Survived	Timeout	No coverage	Ignored	Runtime errors	Compile errors	Detected	Undetected	Total
 All files		 9.70	29	72	0	198	30	0	6	29	270	335
 Cube.cs		 20.18	23	69	0	22	9	0	5	23	91	128
 FigureDecision.cs		 0.00	0	0	0	29	1	0	1	0	29	31
 Point.cs		 14.63	6	3	0	32	6	0	0	6	35	47
 Quadrilateral.cs		 0.00	0	0	0	76	8	0	0	0	76	84
 Triangle.cs		 0.00	0	0	0	26	4	0	0	0	26	30
 Utilities.cs		 0.00	0	0	0	13	2	0	0	0	13	15

# REZULTATE



Testarea Sistemelor Software | 2024

**MULTUMIM!**