

Raport: Utilizarea unui Tool AI

În cadrul acestui raport, analizăm utilizarea unui tool de AI (ChatGPT) în procesul de scriere și îmbunătățire a testelor unitare. Scopul este de a evidenția modul în care inteligența artificială poate sprijini procesul de testare, prin generarea automată de teste după un model, de cazuri de test neidentificate de noi, de moduri prin care să ajungem la cât mai mulți mutanți killed, extragere a informațiilor asupra anumitor framework-uri de testare etc.

Exemple de prompt + răspuns

I)

Prompt:

Am urmatoarea functie:

```
public (bool success, string? error) CreateComment(Comment comment, int pinId, string
userId)
{
    if (string.IsNullOrEmpty(comment.Text))
        return (false, "Text cannot be empty.");

    if (_pinRepo == null)
        throw new InvalidOperationException("PinRepository is required for
CreateComment.");

    var pinExists = _pinRepo.GetPinById(pinId) != null;
    if (!pinExists)
        return (false, "Invalid pin ID.");

    if (string.IsNullOrEmpty(userId))
        return (false, "User ID cannot be empty.");

    comment.PinId = pinId;
    comment.AppUserId = userId;
    comment.Date = DateTime.Now;

    _repo.Add(comment);
    return (true, null);
}
```

Am scris urmatorul unit test pentru cazul "text" (valid) - t1:

```
[Fact]
```

```

public void CreateComment_ValidInputs_t1()
{
    var comment = new Comment { Text = "Hello world" };
    var repoMock = new Mock<CommentRepository>(null);
    var pinRepoMock = new Mock<PinRepository>(null);
    pinRepoMock.Setup(p => p.GetPinById(1)).Returns(new Pin { Id = 1 });

    var service = new CommentService(repoMock.Object, pinRepoMock.Object);

    var result = service.CreateComment(comment, 1, "user123");

    Assert.True(result.success);
    Assert.Null(result.error);
    repoMock.Verify(r => r.Add(It.IsAny<Comment>()), Times.Once);
}

```

Am nevoie de unit teste pentru cazurile: "" (invalid) - t2 null (invalid) - t3.

Răspuns:

Desigur! Iată cele două teste unitare care acoperă acele scenarii de input invalid:

```

[Fact]
public void CreateComment_EmptyText_t2()
{
    var comment = new Comment { Text = "" };
    var repoMock = new Mock<CommentRepository>(null);
    var pinRepoMock = new Mock<PinRepository>(null);
    pinRepoMock.Setup(p => p.GetPinById(1)).Returns(new Pin { Id = 1 });

    var service = new CommentService(repoMock.Object, pinRepoMock.Object);

    var result = service.CreateComment(comment, 1, "user123");

    Assert.False(result.success);
    Assert.Equal("Text cannot be empty.", result.error);
    repoMock.Verify(r => r.Add(It.IsAny<Comment>()), Times.Never);
}

[Fact]
public void CreateComment_NullText_t3()
{
    var comment = new Comment { Text = null };

```

```

var repoMock = new Mock<CommentRepository>(null);
var pinRepoMock = new Mock<PinRepository>(null);
pinRepoMock.Setup(p => p.GetPinById(1)).Returns(new Pin { Id = 1 });

var service = new CommentService(repoMock.Object, pinRepoMock.Object);

var result = service.CreateComment(comment, 1, "user123");

Assert.False(result.success);
Assert.Equal("Text cannot be empty.", result.error);
}

```

II)

Prompt:

Care sunt framework-urile de testare unitara in c#? Avantaje și dezavantaje.

Răspuns:

Iată o listă cu cele mai populare framework-uri de testare unitară în C#, împreună cu avantajele și dezavantajele lor:

1. xUnit.net (Cel mai folosit în proiecte noi)

- Avantaje:
 - Ușor de folosit și configurat.
 - Integrare excelentă cu Visual Studio, GitHub Actions, Azure Pipelines etc.
 - Scris de aceeași echipă care a creat NUnit și MSTest (mai modern).
 - Suportă testare paralelă *by default* (rulează testele mai rapid).
 - Aduce concepte moderne precum [Theory] și [InlineData] pentru testare parametrică.
- Dezavantaje:
 - Nu este la fel de intuitiv pentru începători ca MSTest.
 - Anumite funcționalități sunt deliberate simplificate, ceea ce uneori cere cod suplimentar (de exemplu, setup/teardown global).

2. NUnit (Foarte popular în proiecte mai vechi sau enterprise)

- Avantaje:
 - Foarte matur și stabil.

- Suportă multe tipuri de aserțiuni și setup-uri personalizabile.
- Teste parametrizate foarte ușor de scris cu [`TestCase`].
- Mare compatibilitate cu tool-uri de CI/CD și test coverage.
- Dezavantaje:
 - Testele rulează secvențial (nu paralel) implicit.
 - Mai greu de configurat pentru paralelism și performanță optimă.
 - Codul poate deveni mai "verbal" (mai mult decorat cu attribute).

3. MSTest (Framework-ul oficial Microsoft)

- Avantaje:
 - Vine direct integrat în Visual Studio (zero setup suplimentar).
 - Bun pentru proiecte interne Microsoft sau aplicații corporate.
 - Foarte simplu de început pentru începători.
- Dezavantaje:
 - Mai limitat comparativ cu xUnit și NUnit în expresivitate.
 - Mai puțin flexibil pentru testare avansată sau configurări speciale.
 - Codul MSTest poate deveni mai "rigid" în comparație cu xUnit.

4. bUnit (Pentru testarea componentelor Blazor)

- Avantaje:
 - Specializat pentru aplicații Blazor.
 - Permite testarea logicii UI într-un mod aproape identic cu realitatea.
- Dezavantaje:
 - Util doar pentru Blazor — nu e potrivit pentru aplicații generale.
 - Încă relativ tânăr comparativ cu xUnit sau NUnit.

III)

Prompt:

Ce cazuri as putea trata prin unit teste pentru functia:

```

public (IEnumerable<Pin> Pins, int LastPage, string PaginationUrl) GetPins(string search, int
page, int perPage)
{
    IQueryable<Pin> pins = string.IsNullOrEmpty(search)
        ? _repo.GetAllPinsOrderedByLikes()
        : _repo.GetPinsBySearch(search);

    int totalItems = pins.Count();
    int lastPage = (int)Math.Ceiling((float)totalItems / perPage);

    if (perPage <= 0 || page <= 0 || page > lastPage)
    {
        return (new List<Pin>(), 0, string.IsNullOrEmpty(search)
            ? "/Pins/Index/?page"
            : $"/Pins/Index/?search={search}&page");
    }

    int offset = (page - 1) * perPage;

    var paginatedPins = pins.Skip(offset).Take(perPage).ToList();
    string url = string.IsNullOrEmpty(search)
        ? "/Pins/Index/?page"
        : $"/Pins/Index/?search={search}&page";

    return (paginatedPins, lastPage, url);
}

```

Raspuns:

Perfect, hai să-ți spun clar și direct ce **cazuri** ai putea testa prin **unit tests** pentru funcția ta GetPins:

Cazuri pentru **search**:

1. **search** = " " (gol) — să verifice că se apelează GetAllPinsOrderedByLikes.
2. **search** = null — comportament identic ca la gol ("").
3. **search** = "test" — să verifice că se apelează GetPinsBySearch("test").

Cazuri pentru **page**:

4. **page** = 1 — pagina validă (normal).
5. **page** = **n** — ultima pagină validă (când **page** == **lastPage**).
6. **page** = 0 — invalid (**page** <= 0) → trebuie să returneze listă goală și **LastPage** = 0.
7. **page** = <0 — invalid (**page** negativ).
8. **page** = **n**+1 — invalid (**page** > **lastPage**) → trebuie să returneze listă goală și **LastPage** = 0.

Cazuri pentru **perPage**:

9. **perPage** = 1 — valid (un pin pe pagină).
10. **perPage** > 1 — valid (mai mulți pins pe pagină).
11. **perPage** = 0 — invalid (**perPage** <= 0).
12. **perPage** < 0 — invalid (**perPage** negativ).

Alte cazuri interesante:

13. **totalItems** = 0 — când nu există niciun Pin.
14. **totalItems** = exact multiplu de **perPage** — verificare corectă pentru **LastPage**.
15. **totalItems** nu este multiplu de **perPage** — verificare rotunjire corectă **Math.Ceiling**.

Interpretarea răspunsurilor

- Răspunsurile oferite de ChatGPT sunt corecte de cele mai multe ori, axate pe ce i s-a cerut;
- Exemplele de teste furnizate sunt funcționale, fără necesitatea unor modificări majore, ceea ce reduce semnificativ timpul de scriere manuală a testelor unitare;
- ChatGPT a identificat corect multe cazuri de testare posibile pentru funcția **GetPins()**: cazuri valide/invalide, cazuri de frontieră etc.
- Cu ajutorul ChatGPT am extras mai ușor informații despre framework-urile de testare, inclusiv comparații între ele, research care ar dura mai mult dacă am căuta pe internet;

- Prezintă răspunsurile bine structurate și ușor de înțeles;
- Am observat că poate da răspunsuri greșite dacă nu primește în prompt toate detaliile specifice aplicației, spre exemplu funcții din codul sursă, structura bazei de date etc.

Concluzie

Utilizarea ChatGPT în testarea software aduce beneficii semnificative: crește viteza de generare a testelor, îmbunătățește acoperirea testării, îți arată cazuri de test la care nu te-ai gândit, ajută la îmbunătățirea calității codului și la rezolvarea rapidă a erorilor.

Cu toate acestea, ChatGPT nu înlocuiește procesul absolut necesar de revizuire a codului de către dezvoltator.