

# Puțină filozofie. Epistemologia

---

“Epistemology is the study of  
*how we know what we know.*”



*“The study or a theory of the nature and grounds of knowledge  
especially with reference to its limits and validity.”*

*Merriam Webster*

*“The study of knowledge and justified belief.”*

*Stanford Encyclopedia of Philosophy*

# Un testor bun practică Epistemologia

---

- capacitatea de a ridica **întrebări utile**
- capacitatea de a **observa**
- capacitatea de a **descrie ce percepe**
- capacitatea de a fi **critic** cu sine însuși
- capacitatea de a **recunoaște** și a gestiona **prejudecățile**
- capacitatea de a produce și a **testa conjecturi**
- capacitatea de **analiză**



# De ce aptitudini are nevoie un testor? [Bach]

---

## **Precauție**

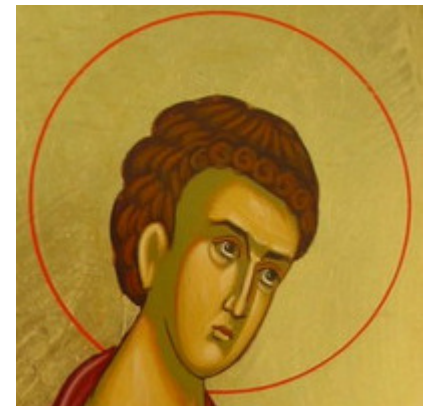
- Jump to conjectures, not conclusions.
- Practice admitting “I don’t know.”
- Have someone check your work.

## **Curiozitate**

- *What* would happen if...?
- *How* does that work?
- *Why* did that happen?

## **Critică**

- Proceed by conjecture and refutation.
- Actively seek counter-evidence.

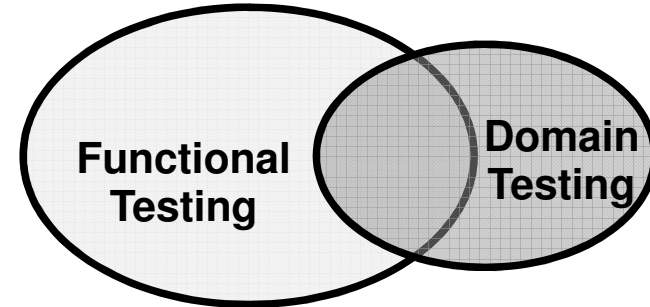


[Toma necredinciosul]

# Domain testing

---

Un fel de testare funcțională



Equivalence  
partitioning

Boundary  
analysis

Category  
partitioning

*“The essence of domain testing is that we partition a domain into subdomains (equivalence classes) and then select representatives of each subdomain for our tests.”*

*Cem Kaner*

*“Equivalence partitioning is ... intuitively used by virtually every tester we’ve ever met.”*

*Richard Craig*

# Ce sunt si ce nu sunt clasele de echivalență

## Prezidențiale 1992 - Turul 2

Candidați:

Emil Constantinescu  
Ion Iliescu

Rezultat pe țară:

61,4%

38,6%

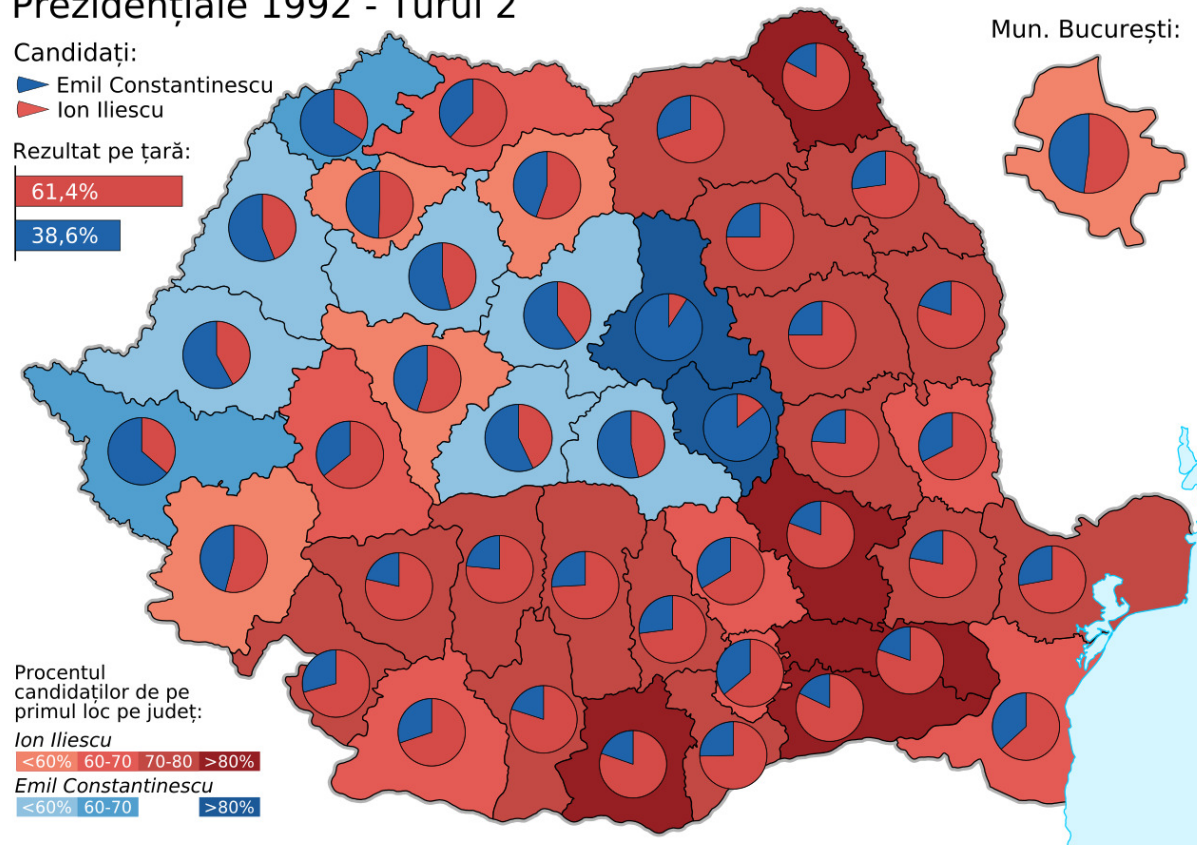
Procentul  
candidaților de pe  
primul loc pe județ:

Ion Iliescu

<60% 60-70 70-80 >80%

Emil Constantinescu

<60% 60-70 >80%



Persoana X este majoră?

$\forall \text{ varsta}(X) \geq 18$

19, 102, 45,...?

$a + b > 0 ? \forall a, b \in \mathbb{N}^*$

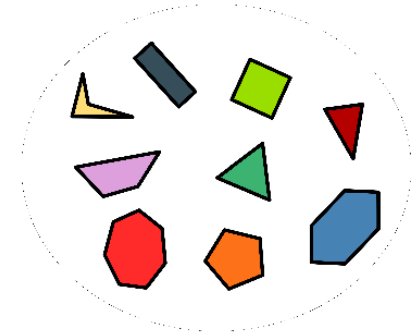
2+3, 242+5, 52+64,...?

# Clase de echivalență

---

“All elements within an equivalence class are essentially the same for the purpose of testing.”

Cem Kaner



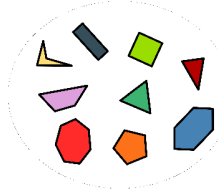
- › Echivalență **intuitivă** – valori diferite produc același rezultat/ rezultate asemănătoare – nu există regulă de împărțire – adaptabilitate în funcție de cerințe
- › Echivalență definită prin **specificații** – rareori utilizată – de obicei este sarcina testorului să aleagă clasele de echivalență, nu a celor care scriu specificații
- › Echivalență prin **analiza variabilelor** – white box
- › Echivalență **subiectivă** – doi testori vor realiza probabil clase de echivalență diferite
- › Echivalență bazată pe **risc** – valorile sunt grupate în funcție de asemănarea în ceea ce privește tipul de defect ce s-ar putea produce (s-ar putea sa nu existe limite ale domeniului, doar riscuri!). (ex. Problema triunghiului).

# Domain testing/Equivalence classes/Best representatives

---

## SCOP

- **reducerea** considerabilă a numărului mare de **cazuri de test** necesare în vederea testării unei funcționalități/ metode/ componente/ ...



## PAȘI

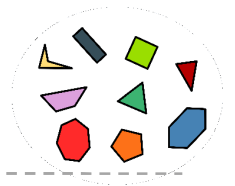
- identificarea tuturor variabilelor de intrare aferente sistemului testat
- împărțirea domeniului de valori aferent fiecărei variabile în clase de echivalență
- pentru fiecare clasă de echivalență alegerea câtorva reprezentanți

## REZULTAT

- **număr minimal de cazuri de test**
- variabilele de intrare în loc sa ia toate valorile posibile aferente întregului domeniu de valori vor lua doar valorile reprezentanților aleși
- produs cartezian între mulțimile reprezentanților fiecărei variabile

*“A **best representative** of an equivalence class is a value that is at least as likely as any other value in the class to expose an error.” [Kaner]*

# O primă alegere empirică a reprezentanților

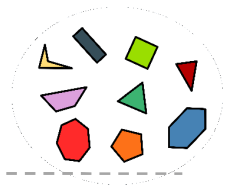


Tip anunț *	<input type="text" value="Ofertă"/>	
Tip tranzacție *	<input type="text" value="Vânzare"/>	
Tip imobil	<input type="text" value="Apartamente"/>	
Nume *	<input type="text"/>	
Localitate *	<input type="text"/>	
Telefon *	<input type="text"/>	
E-mail *	<input type="text"/>	
Preț (Eur) *	<input type="text" value="minim"/>	<input type="text" value="maxim"/>

- variabila *Localitate*:  
{ “Timisoara”, “Sibiu”, “Un nume de localitate lung”, “T”, “123”, .. }
- variabila *Telefon*:  
{ “0732673386”, “0724123456”  
“+40723123456”, “32552”,  
“0040256403211”, “x743”, “Numar de telefon”, ... }
- variabila *Preț minim*:  
{ 100, 150, 200, 1000, -10,  
100000000,  $x$  }
- variabila *Preț maxim*:  
{ 100, 150, 200, 1000, -10,  
100000000,  $x$  }



# O altă alegere empirică a reprezentanților

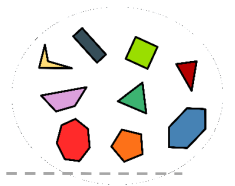


Ipoteză: Se dă o metodă care calculează produsul a 3 parametri (a,b,c) de 2, 3 și respectiv 4 cifre. Să se verifice corectitudinea implementării / să se testeze

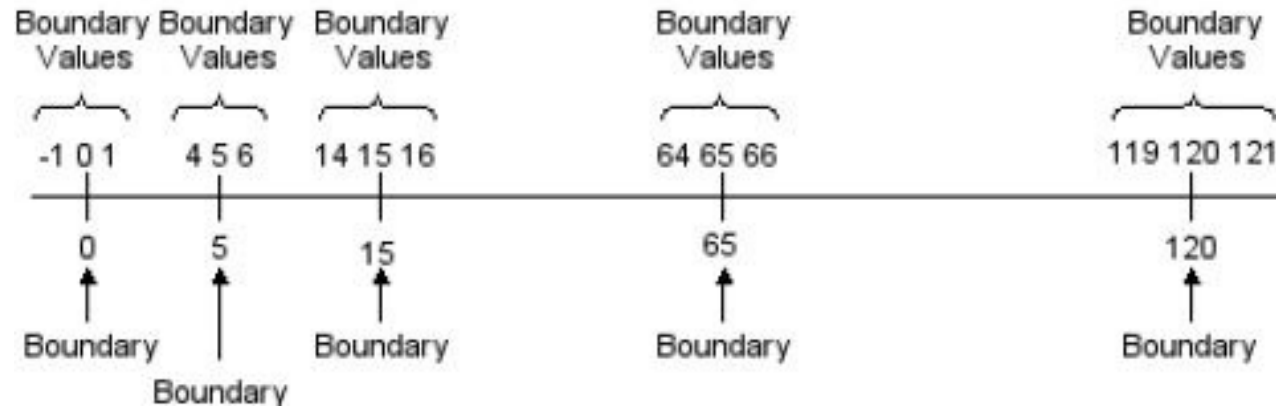
- testare **pozitivă** (ar trebui să treacă testul)  
 $90 * 900 * 9000 = 729 * 10^6$  combinații / teste
- testare **negativă** (ar trebui să pice pestul)  
*o infinitate*
- Împărțirea în clase de echivalență  
variabila a:  $(-\infty, 9)$ ,  $[10, 99]$ ,  $[100, \infty)$ , NaN  
variabila b:  $(-\infty, 99)$ ,  $[100, 999]$ ,  $[1000, \infty)$ , NaN  
variabila c:  $(-\infty, 999)$ ,  $[1000, 9999]$ ,  $[10000, \infty)$ , NaN
  - Alegerea reprezentanților pentru testarea pozitivă  
 $(9, 10, 34, 99, 100) \times (99, 100, 546, 999, 1000) \times$   
 $(999, 1000, 8635, 9999, 10000)$

Ce observăm? Am ales valori la limitele intervalelor claselor de echivalență

Câte cazuri de test?



# Valori la limită. Boundary representatives



**In/On/Off**  
points

- dacă spațiul intrărilor poate fi mapat pe axa numerelor, atunci clasele de echivalență vor fi delimitate de către valorile limită [Myers]
- clasele de echivalență și valorile limită sunt stabilite de către testor în funcție de experiență și intuiție
- din fiecare clasă de echivalență numerică (spații ordonate) se aleg de obicei 5 reprezentanți:
  - O valoare arbitrară din interiorul intervalului
  - LimitaInferioară și LimitaInferioară - 1
  - LimitaSuperioară și LimitaSuperioară + 1
- pentru spații neordonate se aleg alte criterii de testare la limitelor (ex. testarea tipăririi la sute de imprimante – se aleg acele imprimante cu memorie, viteză, costuri mici/mari)

# Testarea domeniului de valori. Greșeli frecvente

---



- generalizare slabă – alegerea neadecvată a valorilor
- Lipsa unor valori importante – lipsa manipulării erorilor
- Prea multe valori – se testează inutil valori (testarea valorilor limită 97, 98, 99, 100, pentru un număr de 2 cifre)
- Nedetectarea unei limite
- Nedetectarea dimensiunii – de exemplu dimensiunea unui câmp de text în care se introduc numere de 1,2,3,4 cifre
- Neformularea riscului – orice test porneste de la un risc pe care testorul și-l imaginează
- Neexplicarea riscului – motivarea legăturii între cazul de test și risc

# Primii pași în testarea domeniului de valori

---



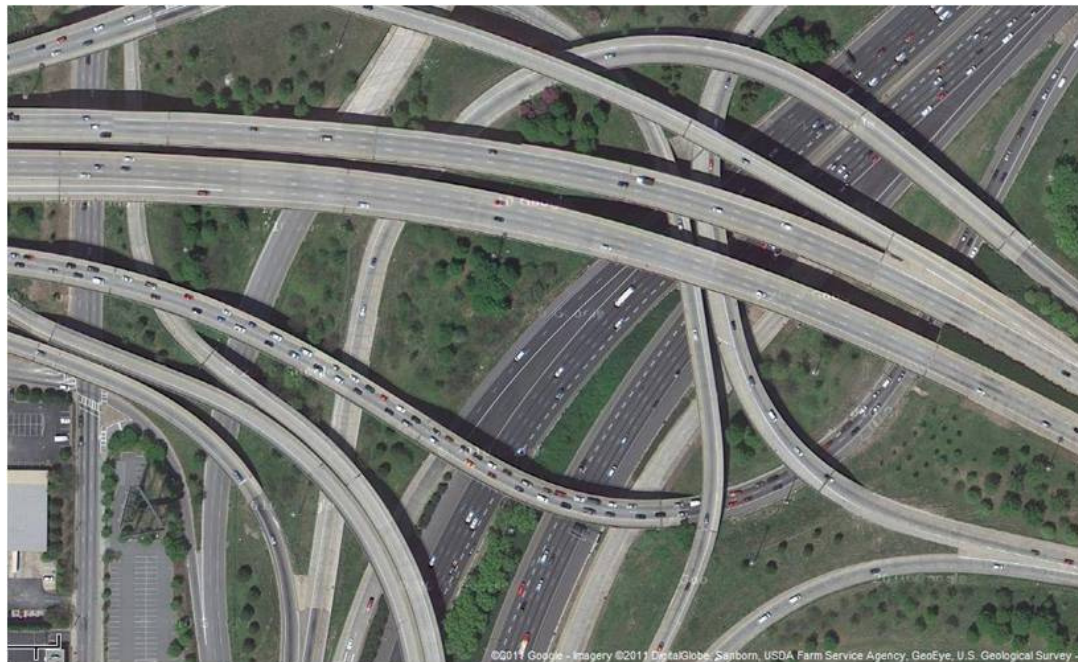
1. Primele teste sunt **simple** și **evidente**
  - dacă rezultă defect, s-a detectat o problemă gravă -> reproiectare sau rezolvarea timpurie evitând efectele secundare ce pot apărea
  - Modalitate de învățare a programului, a funcționalității
2. testarea **simpatetică** – raportarea defectelor se face cunoscând produsul, iar implicațiile pe care le pot aduce defectele sunt descrise pentru a fi înțelese de programator
3. testarea **succintă** a tuturor modulelor – înainte de testarea amănunțită
4. alegerea testelor mai puternice – împărțirea valorilor în **clase de echivalență** și testarea **valorilor limită**
5. imaginarea posibilelor defecte bazate pe **riscul** pe care acestea ar putea să îl implice
6. testarea **subiectivă** în funcție de presupunerile testorului

# Testarea domeniul de valori. Orientare spre risc

---

*“Partitioning should be revealing.”*

*Weyuker & Ostrand. Kaner*



- Testare subiectivă, bazată pe risc
- Două valori (reprezentanți) pot fi **echivalente** relativ la un anumit risc / eroare
- Două valori pot fi **neechivalente** relativ la un alt risc / eroare

# Exemple

---



## 1. Problema triunghiului

Se dau trei numere. Să se verifice dacă formează un triunghi scalen, isoscel, echilateral sau nici un triunghi.

## 2. Suma a două numere de două cifre

➤ Binder's domain test matrix

# Testarea Componentelor. Unit testing

---

*In computer programming, unit testing is a software verification and validation method in which a programmer tests if individual units of source code are fit for use.*

*A software testing methodology in which individual tests (unit tests) are developed for each small part of a program.*

*In computer programming, a unit test is a procedure used to validate that a particular module of source code is working properly.*

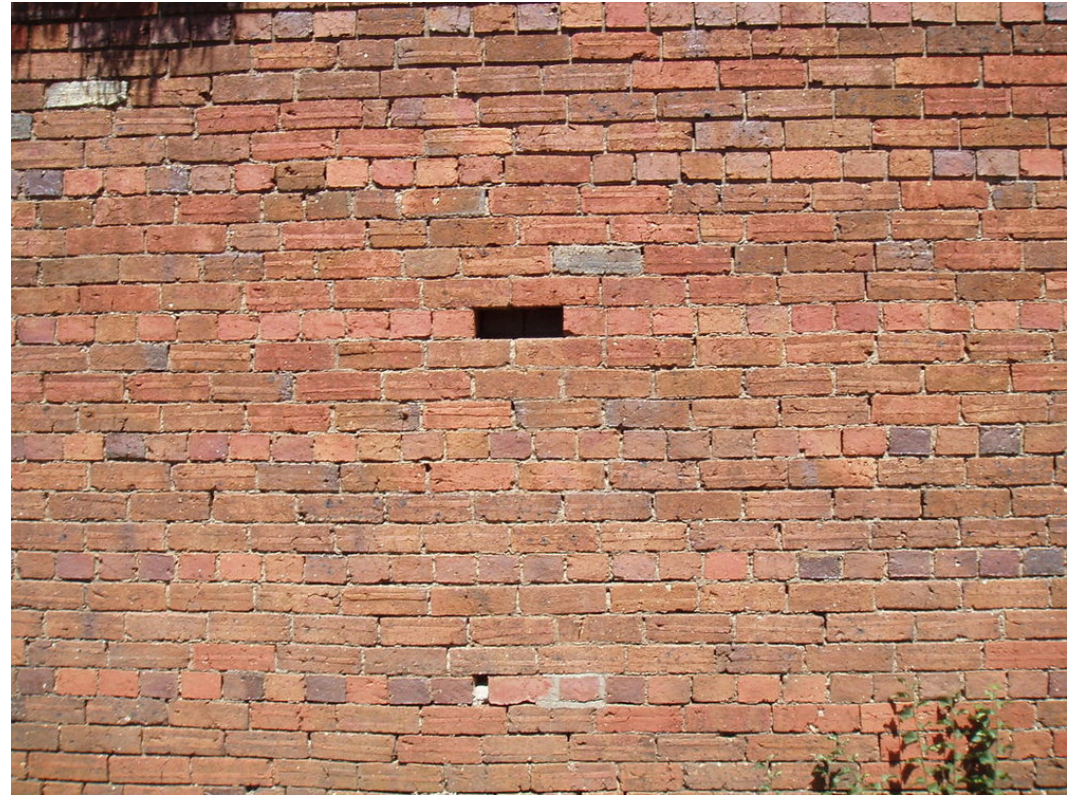
*The most 'micro' scale of testing; to test particular functions or code modules. Typically done by the programmer and not by testers, as it requires detailed knowledge of the internal program design and code.*



# Ce este o componentă?

---

*Clasă*  
*Metodă*  
*Funcție*  
*Bucată de cod*



- Nu mai mult de 4 nivele de imbricare
- Complexitatea ciclomatică  $\leq 10$  [McCabe] ( $CC=M-N+1$ , controlflow graph, ex.)
- Dacă conține cod comun cu altă componentă -> nu este o componentă
- Este testarea de nivel cel mai jos



# Cine face Unit Testing?

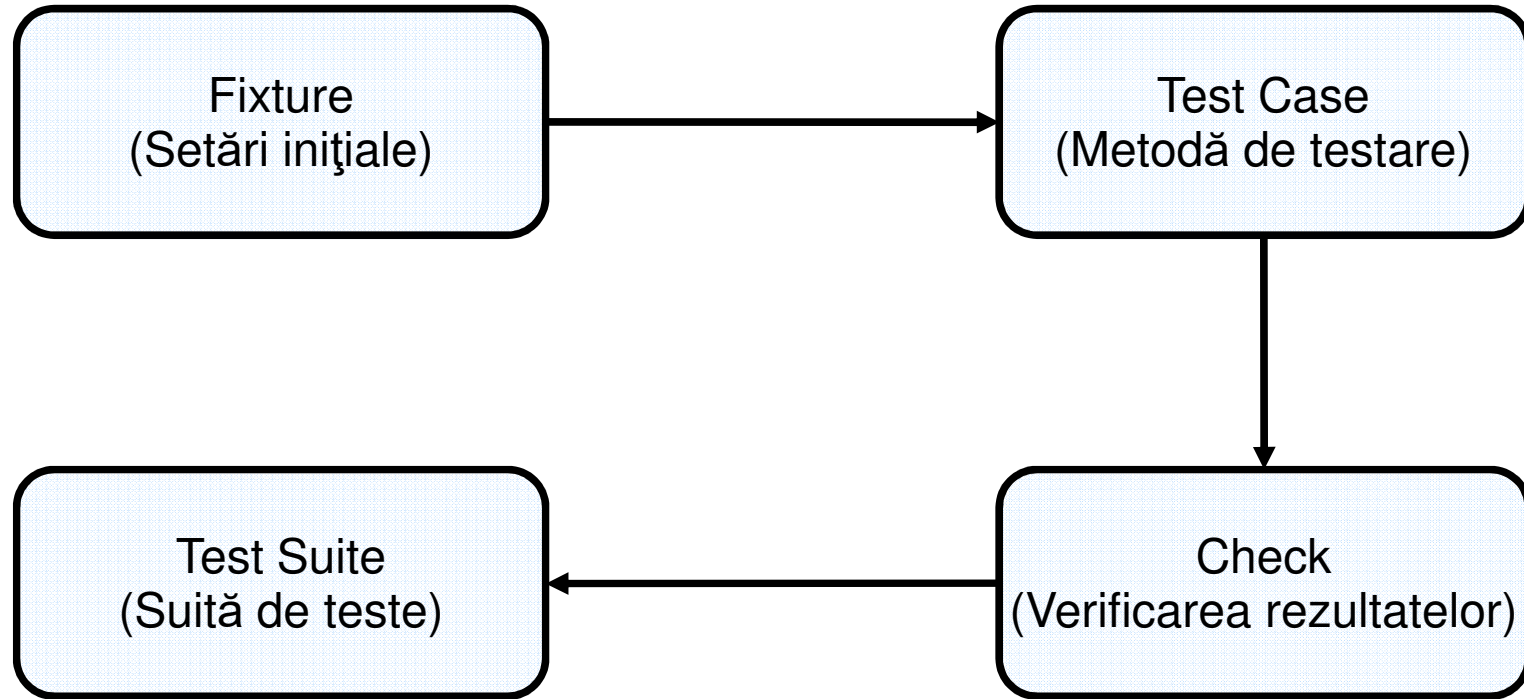
---



***Programatorul***

# xUnit. Test Automation Framework

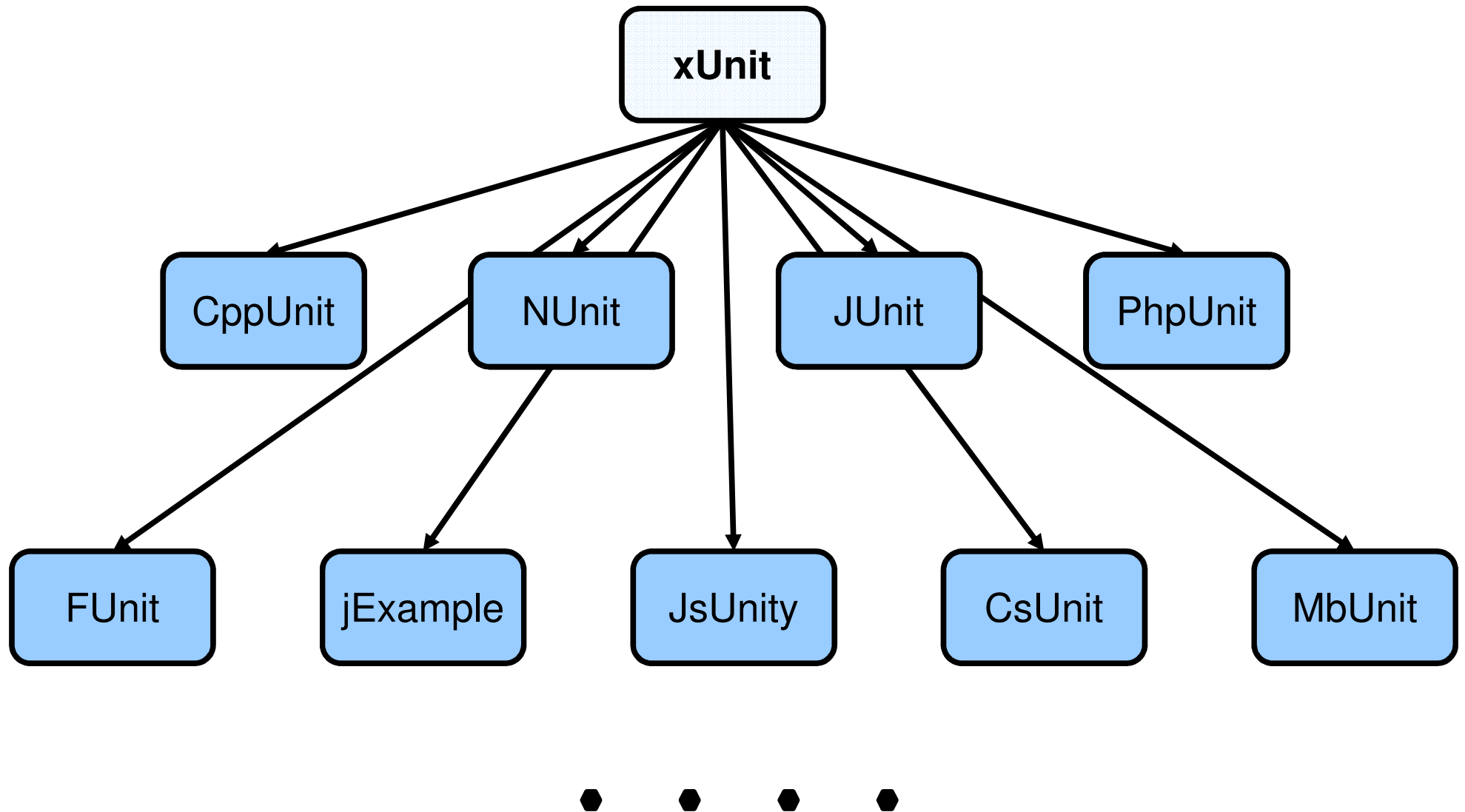
---



[Kent Beck]

# xUnit. Test Automation Framework

---



# De ce xUnit?

---

[Gerard Meszaros]

➤ *Hand-Scripted Tests* automated in the same programming language as the SUT (System Under Test)

➤ Easy for developers to unit test. They are familiar with the language

➤ All members of the xUnit family implement a set of features:

Specify a **test as a method**

Specify the **expected results using assertions**

Aggregate tests into **test suites**

**Run one or more** tests and get **reports**

# Un test in xUnit

---

Patru faze

- *Fixture Setup* - **arrange**
- *SUT exercise* - **act**
- *Verify results with assertions* - **assert**
- *Tear down*

**¡THREE AMIGOS!**



**arrange**

**act**

**assert**

## xUnit. Fixture

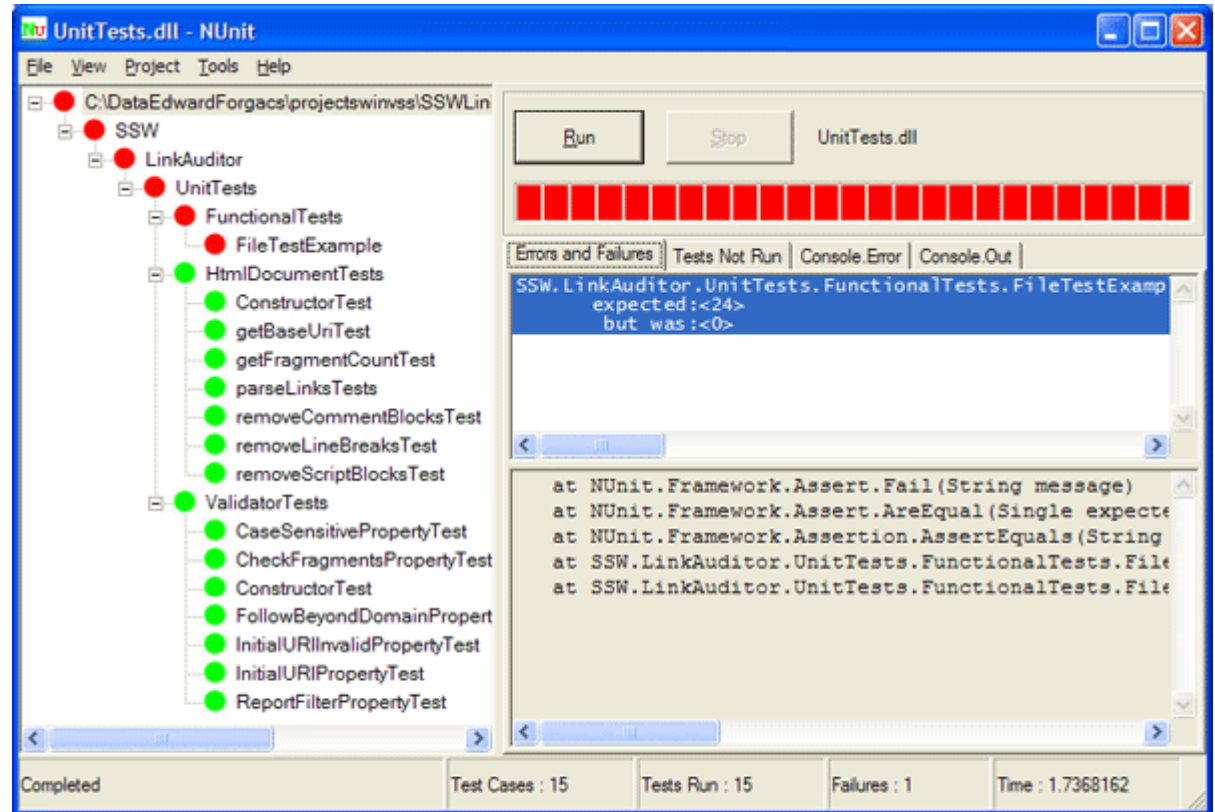
---

- Starting part of a test method or different method(s)
- Everything needed to exercise the SUT
- Instance of a class whose functionality is tested
- May contain or simulate dependencies of SUT



# xUnit. How to run tests? What are the results?

- Command line
- Graphical test runner
- IDE integrated



- **pass**
- **fail** – fail the assertion
- **error** – runtime crash

NUnit → xUnit

---



**2.6.4**

- Framework de testare a componentelor
- Nu este un software de testarea automată GUI
- Attribute .NET (adăugare metadata în cod, oferă informații despre .NET assembly code)
- scris în .NET C#, portat din jUnit
- Nu permite scrierea de scripturi, doar rularea codului compilat .dll



```
namespace NumeNamespace{

    [TestFixture]
    public class NumeClasa{
        //...

        [SetUp]
        public void FunctieInitializare(){
            //...
        }

        [TearDown]
        public void FunctieTerminare(){
            //...
        }

        [Test]
        public void FunctieTestare(){
            //...
        }
    }
}
```

Atribut: **[Category(NumeCategorie)]** – permite selectarea testelor rulate din NUnit

```
[Test]
[Category("Categorie X")]
```

Atribut: **[Combinatorial]** – generează toate combinațiile pt. parametri funcției de test

```
[Test]
[Combinatorial]
```

```
public void FunctieTestare(Values(1,2,3,4) int x,[Values("A","B","c")] string y)
- apelul funcției FunctieTestare se va face de 4*3=12 ori
```

Atribut: **[TestCase(p1,p2,...)]** – atribuie valorile  $p1, p2, \dots$ , parametrilor funcției de test

```
[TestCase(12,3,4)]
[TestCase(12,2,6)]
[TestCase(12,4,3)]
```

```
public void DivideTest(int n, int d, int q)
```

Atribut: **[Description(descriere)]** – descrierea testului

```
[Test]
[Description("....")]
```

Atribut: **[ExpectedException(typeof(tipExceptie))]** – dacă funcția de test înregistrează o excepție *tipExceptie*, atunci testul trece cu success

```
[Test]
[ExpectedException( "System.ArgumentException" ) ]]
```

Atribut: **[Ignore("mesaj")]** – testul va fi ignorat cât timp este folosit acest atribut

`Assert.AreEqual(rezultat_asteptat, rezultat_obtinut, mesaj_eroare);`

- rezultatele pot fi de tipul {int, float, object,...}

`Assert.AreSame( object expected, object actual );`

- același obiect este referit de ambii parametri

`Assert.Contains( object anObject, IList collection );`

- un obiect este conținut într-o listă sau șir de obiecte

`Assert.True( bool condition, string message, object[] parms );`

- dacă condiția este falsă -> Fail + mesaj

`Assert.Greater( int arg1, int arg2 );`

`Assert.Pass( [[string message], object[] parms ] );`

`Assert.Fail( [[string message], object[] parms] );`

`Assert.Ignore( [[string message], object[] parms] );`

- test ignorat la rulare

`Assert.Inconclusive( [[string message], object[] parms] );`

- se consideră ca nu sunt suficiente date

`StringAssert.Contains( string expected, string actual );`

`StringAssert.StartsWith( string expected, string actual );`

`StringAssert.EndsWith( string expected, string actual );`

`StringAssert.AreEqualIgnoringCase( string expected, string actual );`

`StringAssert.IsMatch( string regexPattern, string actual );`

# Nunit. Exemplificare

---

