Ingineria Programării

Cursul 9 – 18 Aprilie

Cuprins

- Din Cursurile trecute…
  - Design Patterns
    - Creational Patterns
    - Structural Patterns
    - Behavioral Patterns
- Alte tipuri de Design Patterns
- Quality Assurance
  - Software Testing
  - Testing Methodologies
  - Testing process
  - Manual Testing vs Automatic Testing

Din Cursurile Trecute

- GOF = ?
- Creational Patterns
- Structural Patterns
- Behavioral Patterns

Din cursurile trecute – CP

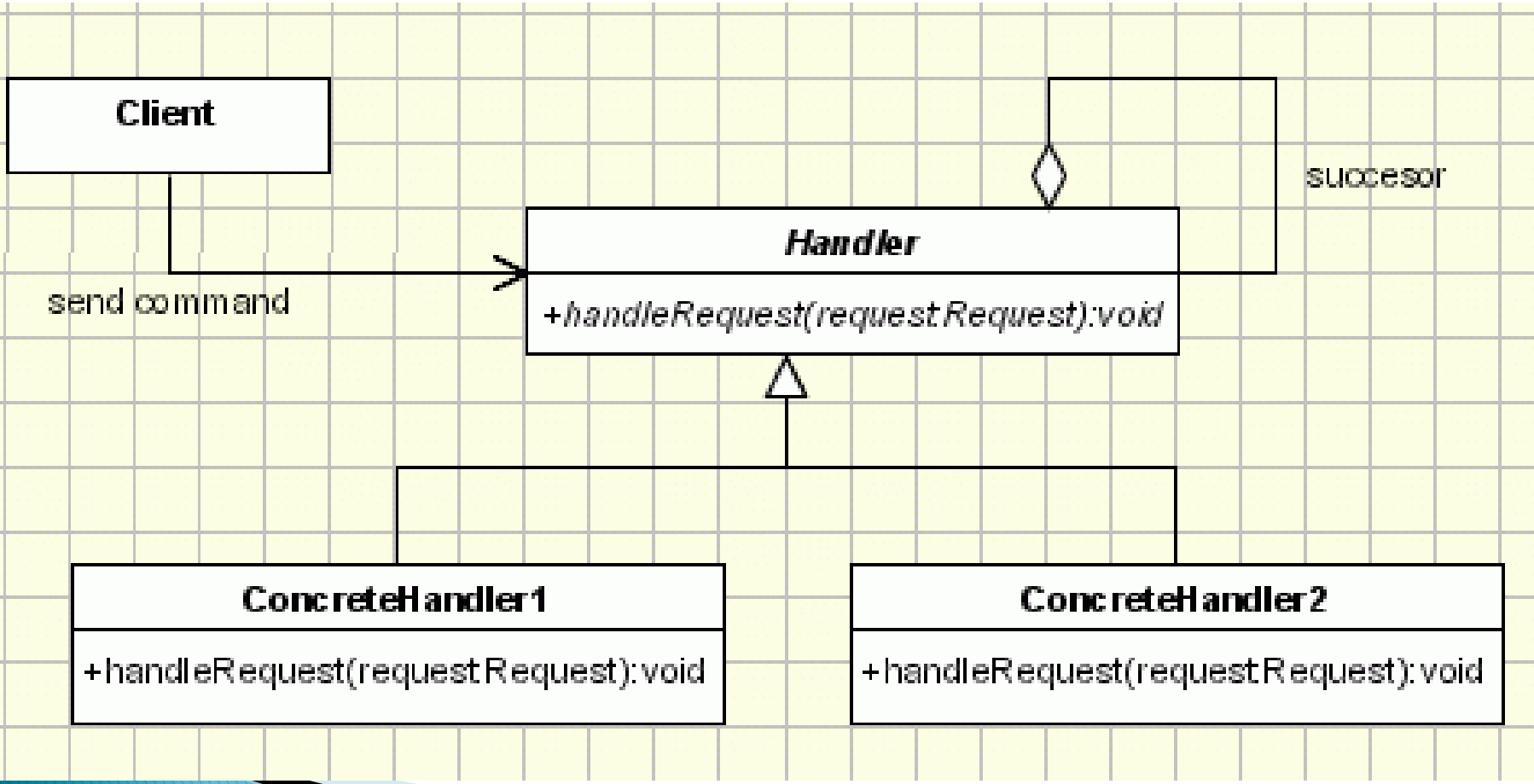
- Abstract Factory
- Builder
- Factory Method
- Prototype = ?
- Singleton

Din cursurile trecute – SP

- Adapter
- Bridge
- Composite
- Decorator
- Façade
- Flyweight = ?
- Proxy

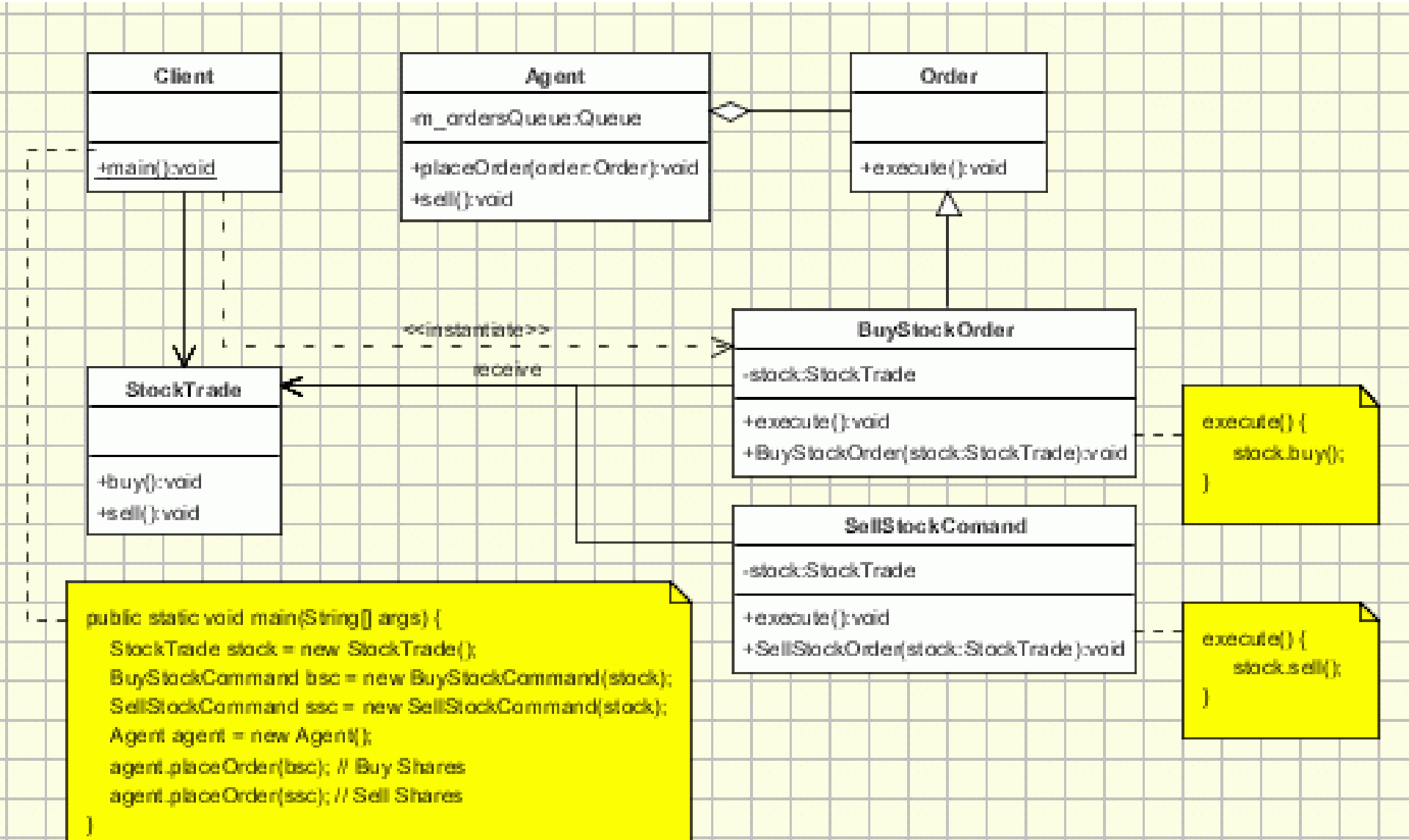
BP – Chain of Responsability

- Help, multi level filter



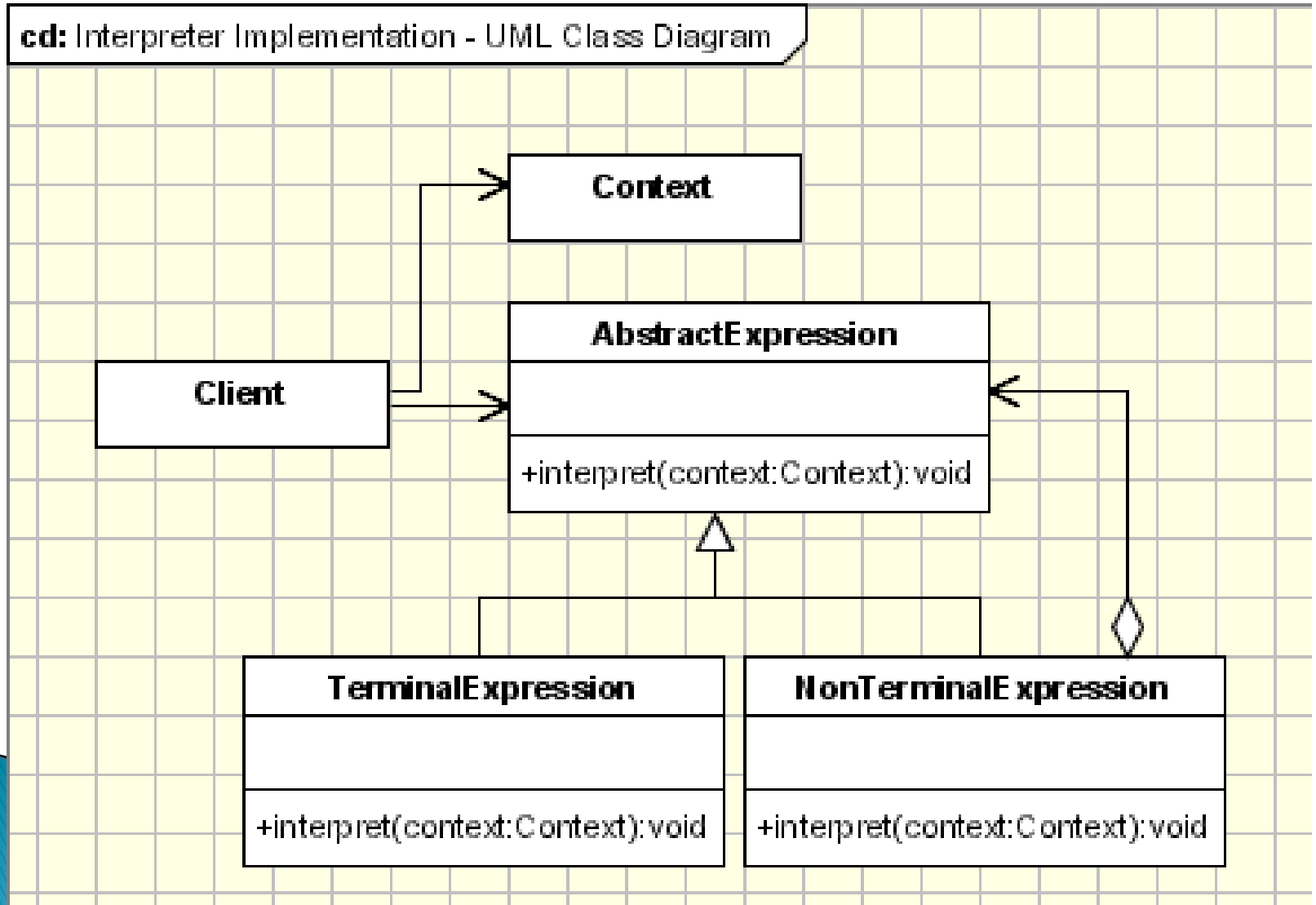
BP – Command

- Customer, waiter, cook



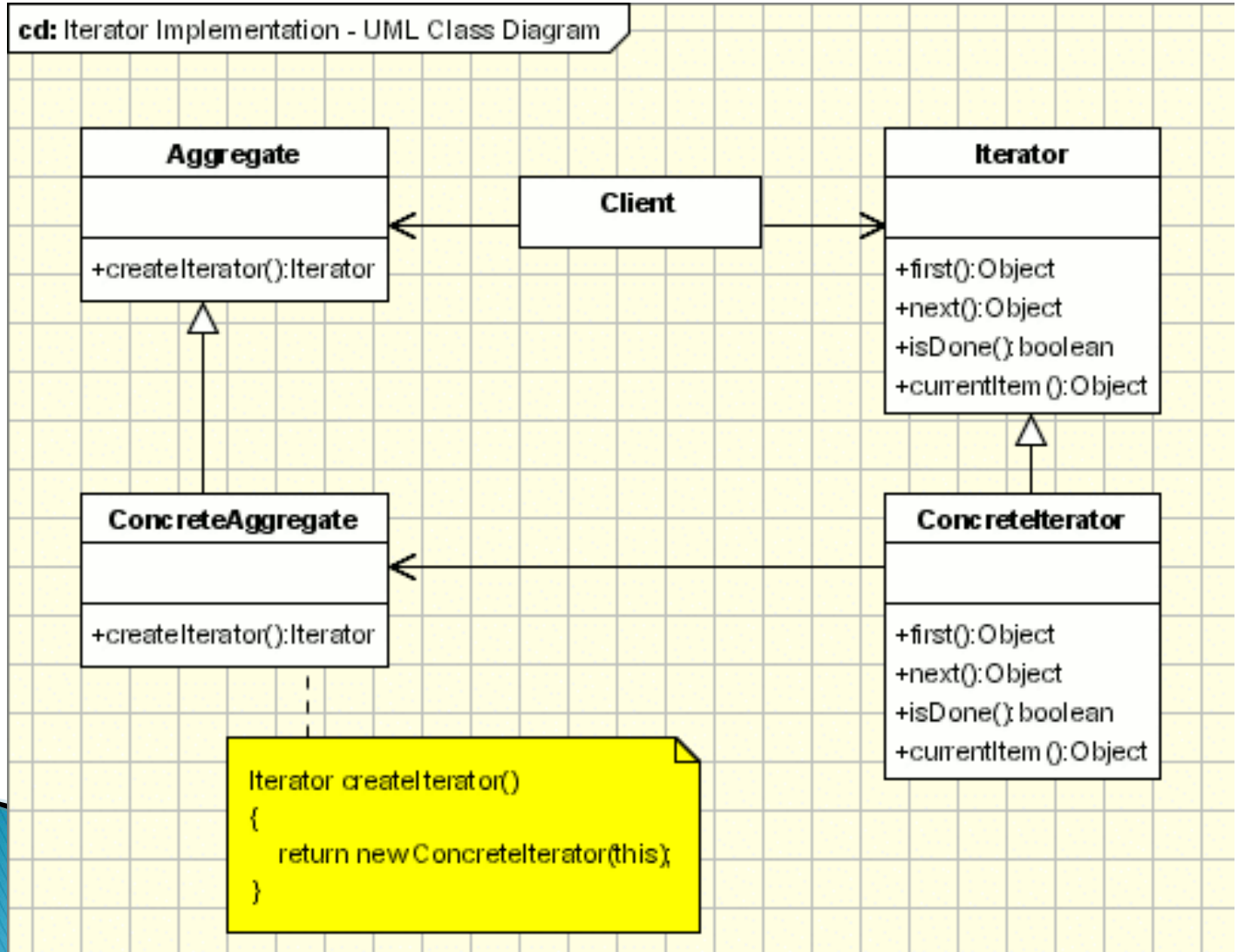
BP – Interpreter

- Musical notes



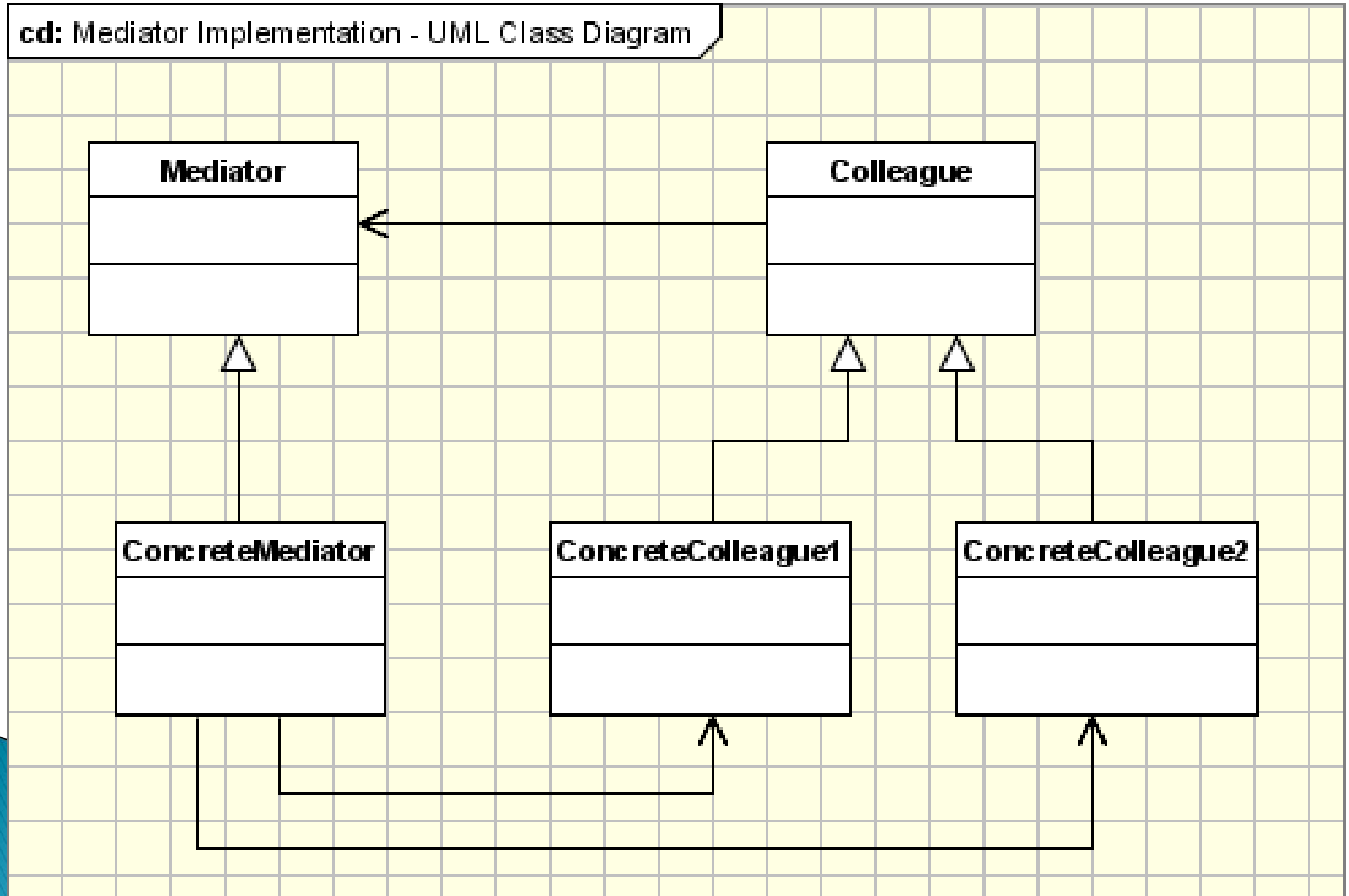
BP – Iterator

- Remote control



BP – Mediator

- Control tower

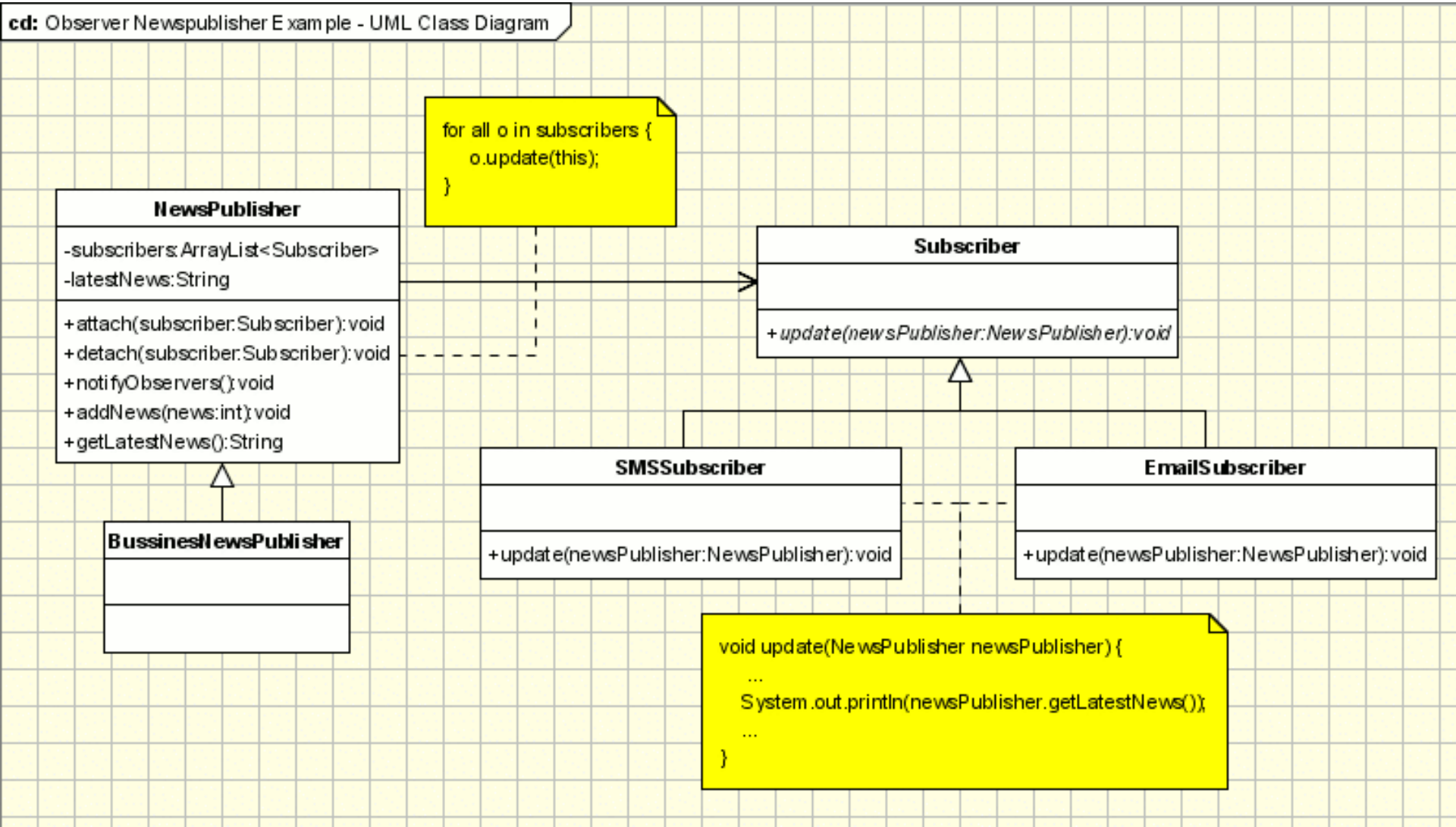


BP – Memento & State

- Memento:
  - Undo and restore operations in most software
  - Database transactions
- State
  - Network connection

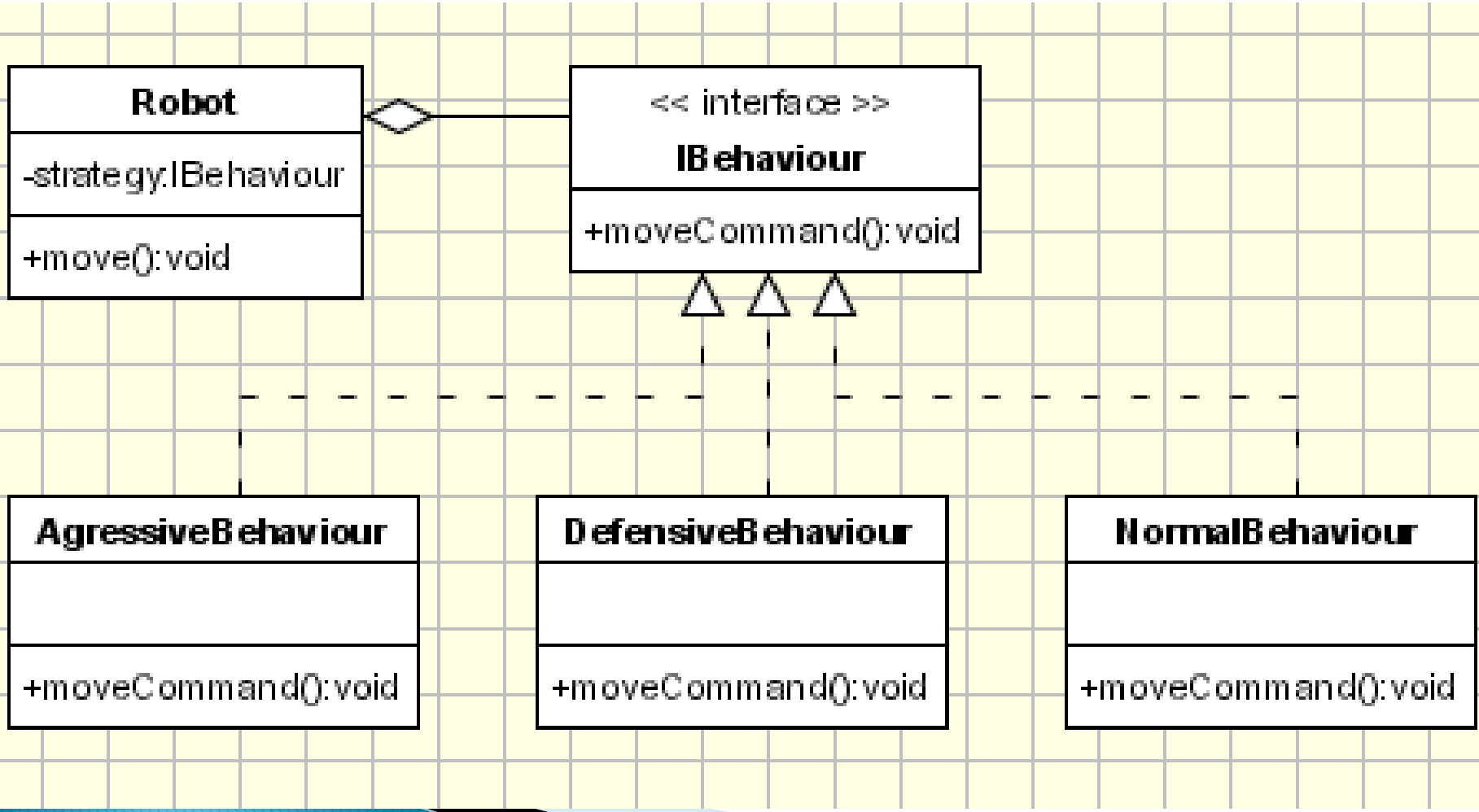
BP – Observer

- Excel graphs, News Agency



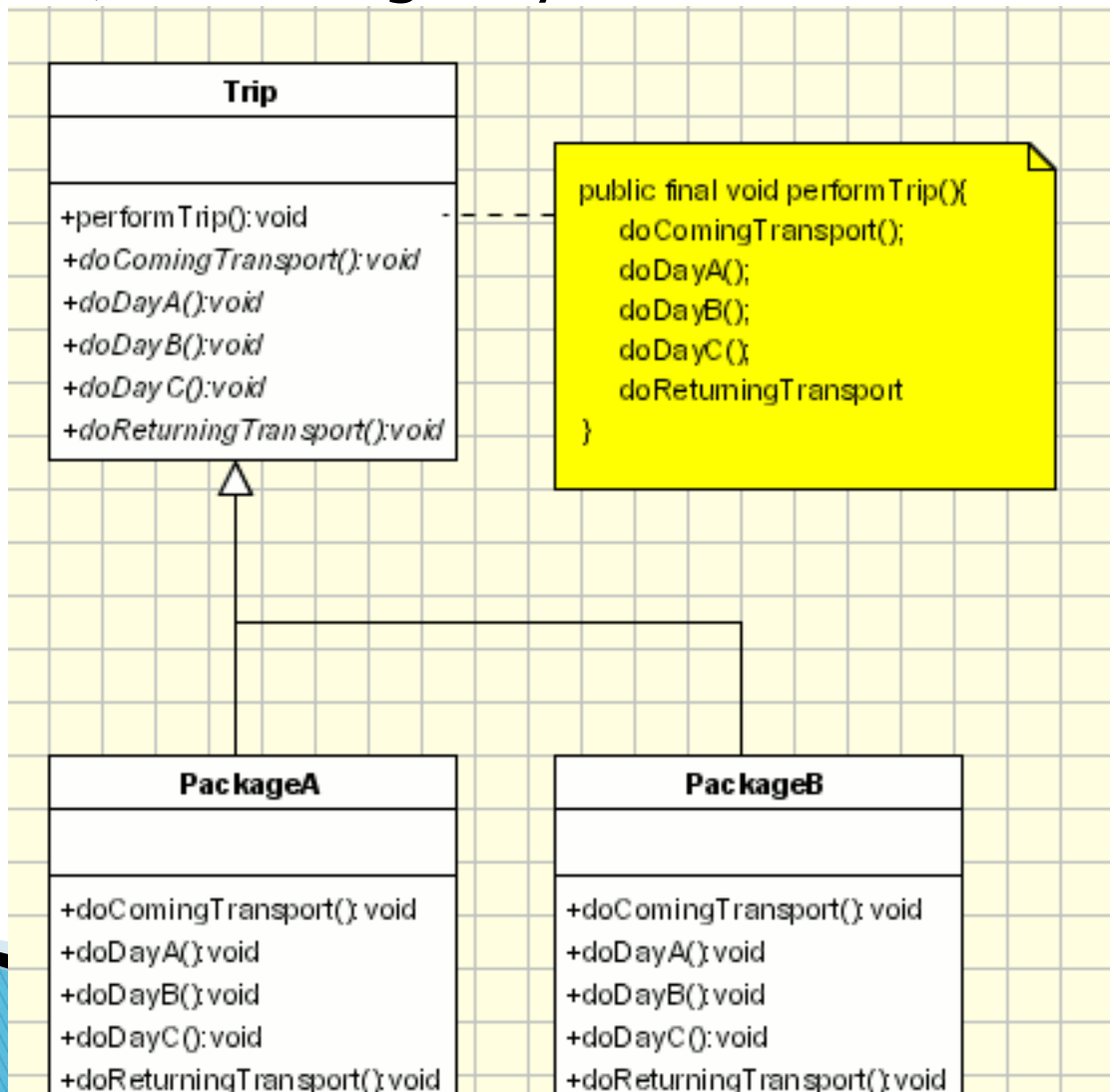
BP – Strategy

- Standard calculator, Robot



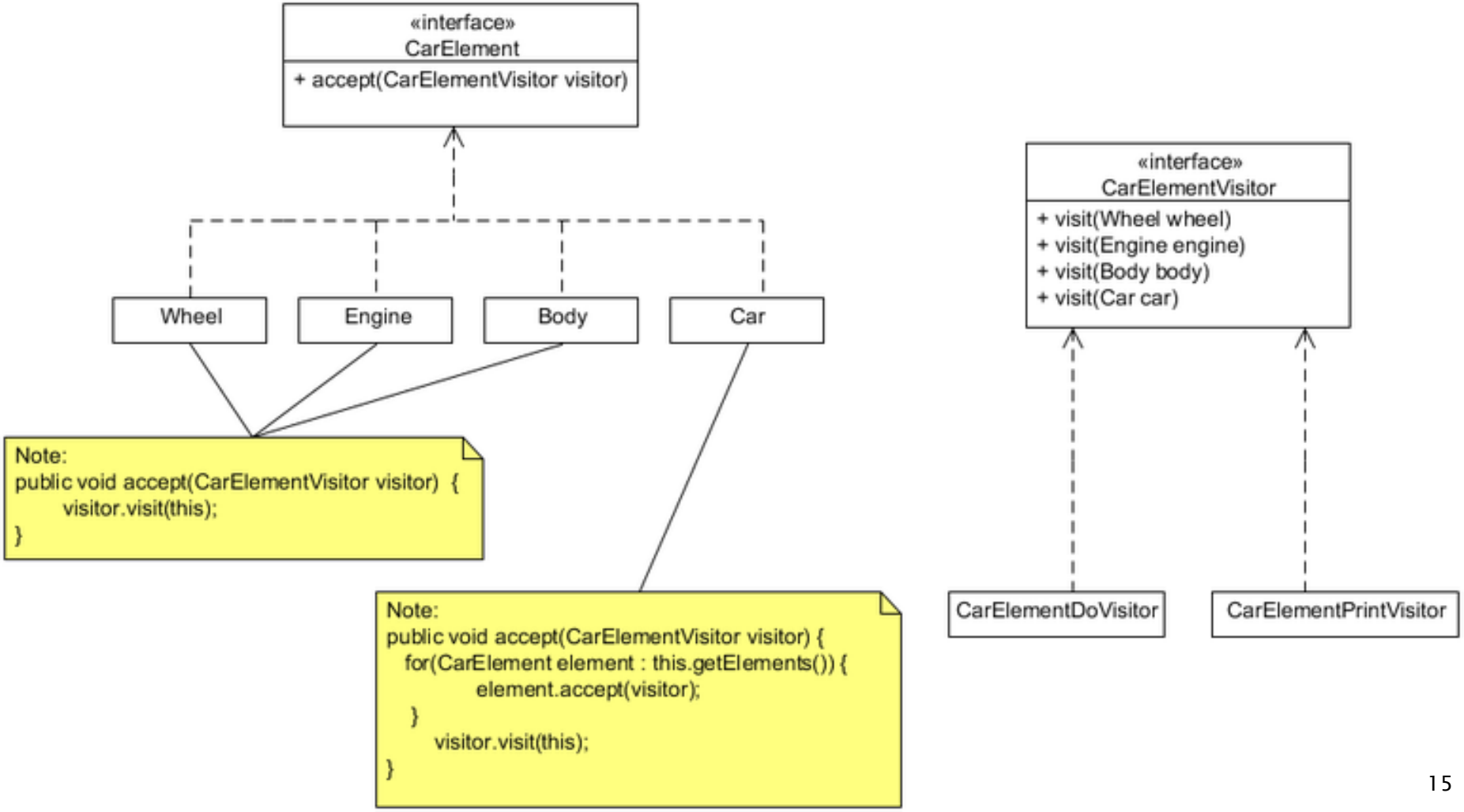
BP – Template Method

- Games, Travel Agency



BP – Visitor

- Car elements visitor



Alte tipuri de DP 1

- **Concurrency Patterns** – deal with multi-threaded programming paradigm
  - **Single Threaded Execution** – Prevent concurrent calls to the method from resulting in concurrent executions of the method
  - **Scheduler** – Control the order in which threads are scheduled to execute single threaded code using an object that explicitly sequences waiting threads
  - **Producer-Consumer** – Coordinate the asynchronous production and consumption of information or objects

Alte tipuri de DP 2

- **Testing Patterns 1**
  - **Black Box Testing** – Ensure that software satisfies requirements
  - **White Box Testing** – Design a suite of test cases to exhaustively test software by testing it in all meaningful situations
  - **Unit Testing** – Test individual classes
  - **Integration Testing** – Test individually developed classes together for the first time
  - **System Testing** – Test a program as a whole entity

Alte tipuri de DP 3

- **Testing Patterns 2**
  - **Regression Testing** – Keep track of the outcomes of testing software with a suite of tests over time
  - **Acceptance Testing** – Is done to ensure that delivered software meets the needs of the customer or organization that the software was developed for
  - **Clean Room Testing** – People designing software should not discuss specifications or their implementation with people designing tests for the software

Alte tipuri de DP 4

- **Distributed Architecture Patterns**
  - **Mobile Agent** – An object needs to access very large volume of remote data => move the object to the data
  - **Demilitarized Zone** – You don't want hackers to be able to gain access to servers
  - **Object Replication** – You need to improve the throughput or availability of a distributed computation

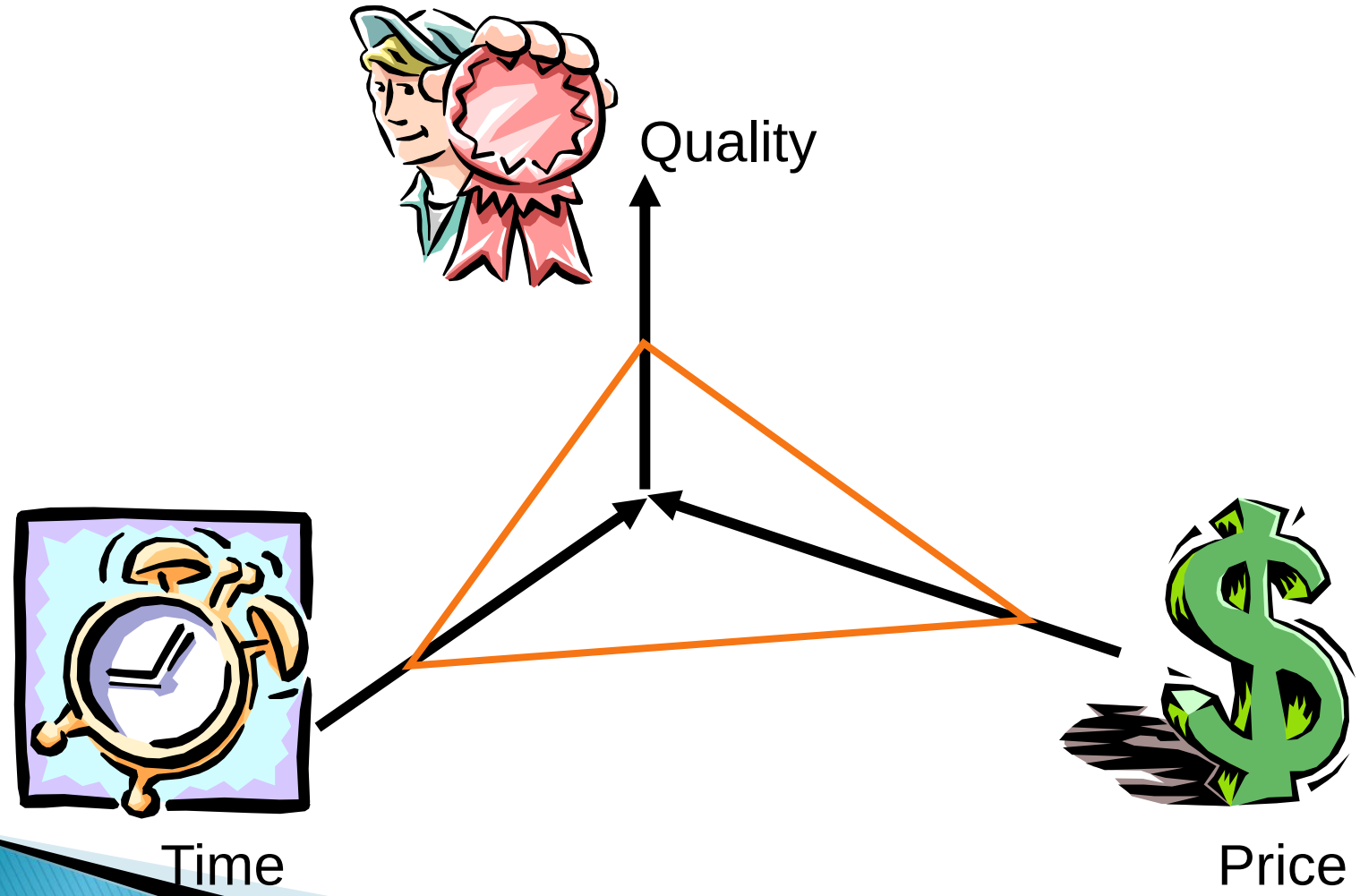
Alte clase de DP

- **Transaction patterns** – Ensure that a transaction will never have any unexpected or inconsistent outcome. Design and implement transactions correctly and with a minimum of effort
- **Distributed computing patterns**
- **Temporal patterns** – distributed applications to function correctly, the clocks on the computers they run on must be synchronized. You may need to access pervious or future states of an object. The values of an object's attributes may change over time
- **Database patterns**

Quality Assurance

- Refers to **planned and systematic production processes** that provide confidence in a product's suitability for its intended purpose.
- A set of activities intended to ensure that **products satisfy customer requirements**
- *QA cannot absolutely guarantee the production of quality products but makes this more likely*
- Two key principles characterize QA:
  - "fit for purpose" – the product should be suitable for the intended purpose, and
  - "right first time" – mistakes should be eliminated

Quality Assurance Dilemma



Quality Assurance – Definition

"The process of exercising or evaluating a system by manual or automated means to verify that it satisfies specified <u>requirements</u> or to identify differences between expected and actual results."

(IEEE Standard Glossary, 1983)

Software Quality Assurance

- (SQA) consists of a means of **monitoring the software engineering processes and methods used to ensure quality**
- May include ensuring conformance to one or more standards, such as ISO 9000 or CMMI
- SQA encompasses the entire software development process, which includes processes such as *software design, coding, source code control, code reviews, change management, configuration management, and release management*

ISO 9000

- ISO 9000 is a family of standards for quality management systems
- Some of the requirements in ISO 9001 (from ISO 9000 family) include
  - a set of procedures;
  - monitoring processes;
  - keeping adequate records;
  - checking output for defects;
  - regularly reviewing individual processes;
  - facilitating continual improvement

Software Testing – Introduction

- An empirical investigation conducted to provide information about the *quality of the product or service under test, with respect to the context in which it is intended to operate*.
- Allow the business to *appreciate and understand* the risks at implementation of the software
- Test techniques include the process of executing a program or application with **the intent of finding software bugs**
- The process of **validating and verifying** that a software program/application/product meets the business and technical requirements that guided its design and development

Software Testing – When?

- Can be implemented at any time in the development process
- However the most test effort is employed after the requirements have been defined and coding process has been completed
- In XP…

Testare software

- Testarea Software NU este o fază
- Este un proces care trebuie integrat în toate fazele construcţiei produsului software
- Există documente de testare asociate la fiecare fază a dezvoltării

Care sunt scopurile testării?

- De a localiza şi **preveni** erorile cât mai curând posibil
- De a efectua toate Testele corespunzător Cerinţelor, într-un mod cât mai eficient şi mai economic
- De a aduce produsul software la un nivel de calitate cât mai ridicat (pentru client)
- Testarea nu e doar pentru Software! Este pentru toate componentele ce vor fi livrate clientului

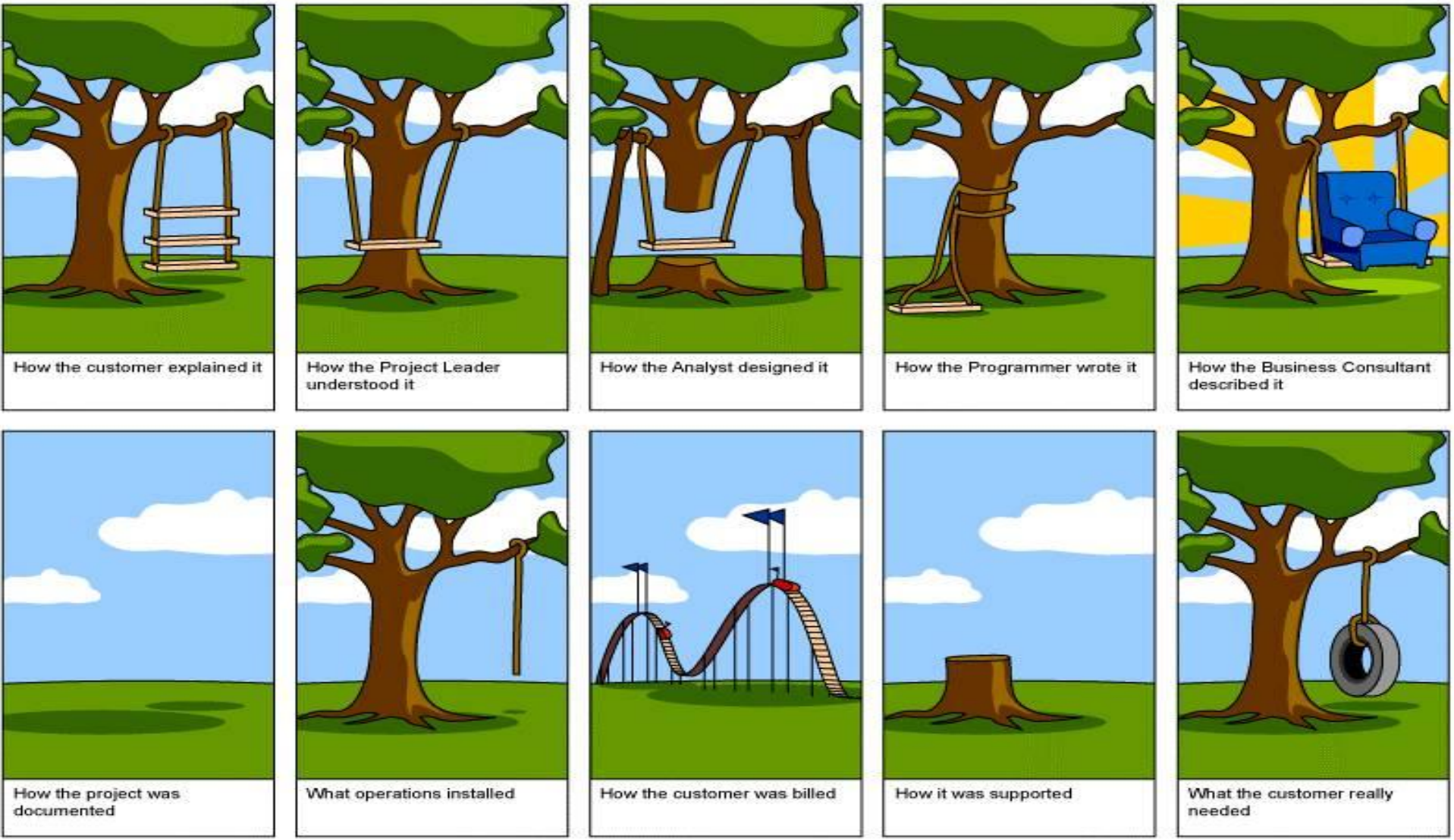
De ce avem erori în software?

- Comunicarea deficitară sau Blocajele de comunicare
- Înţelegerea deficitară
- Presiunea Timpului
- Nivelul Programatorului este Scăzut

Comunicare deficitară

"Didn't you get my e-mail?"

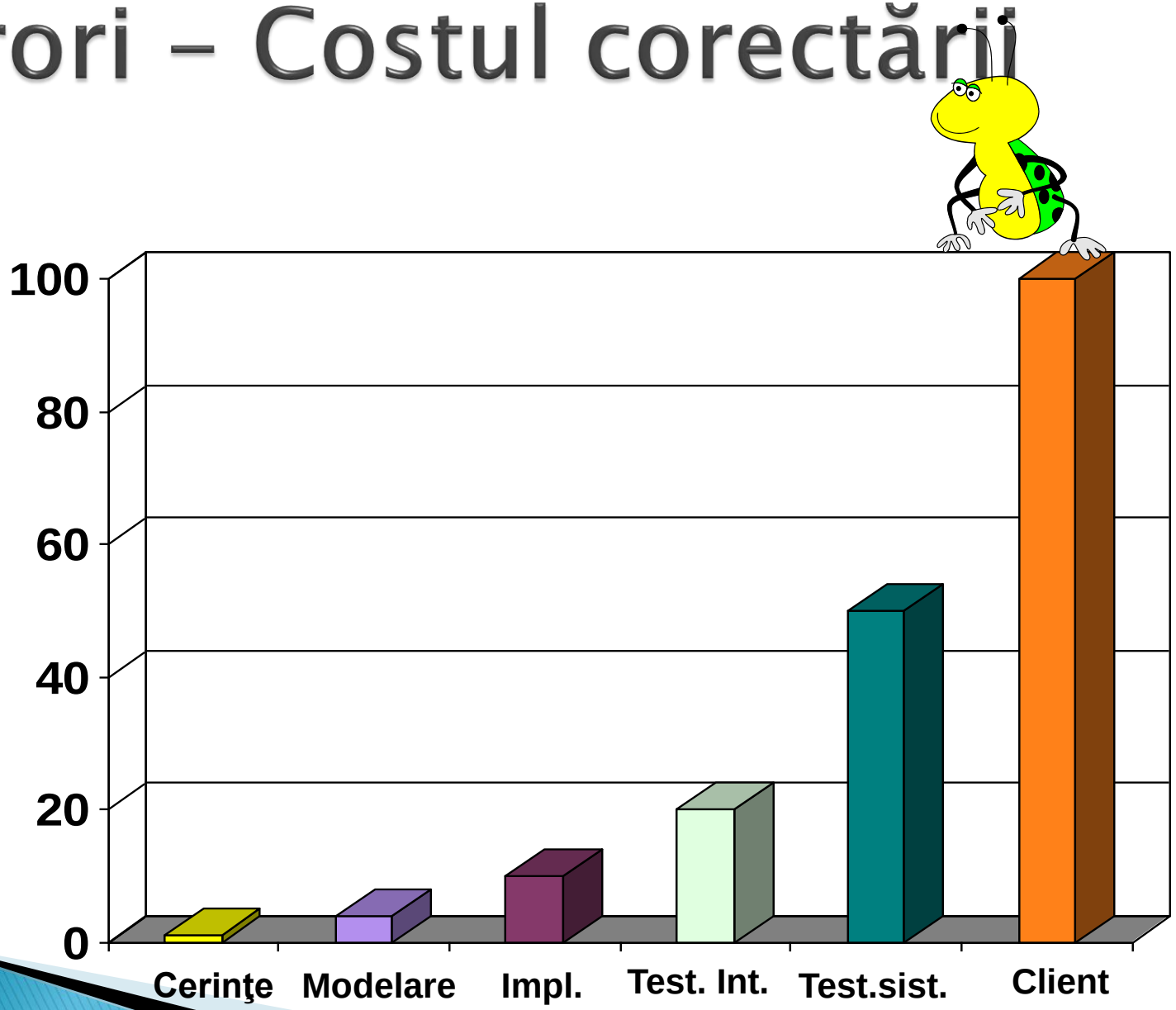
Comunicare deficitară – În tratarea cerinţelor



De unde vin problemele software?

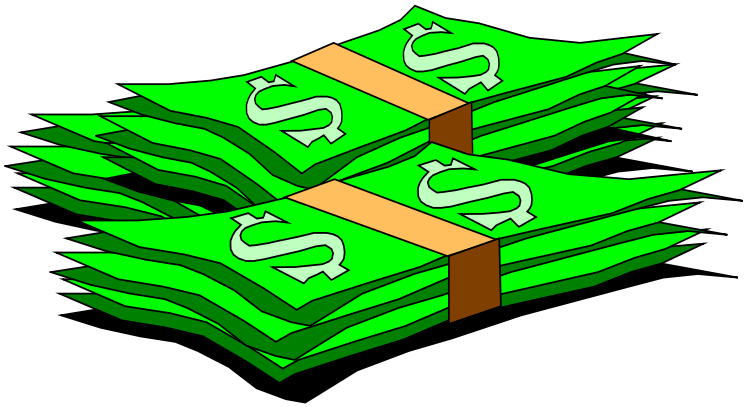
- Cerinţe definite Incomplet 50%
- Modelare Ambiguă sau Insuficientă 30%
- Erori de Programare 20%

Erori – Costul corectării

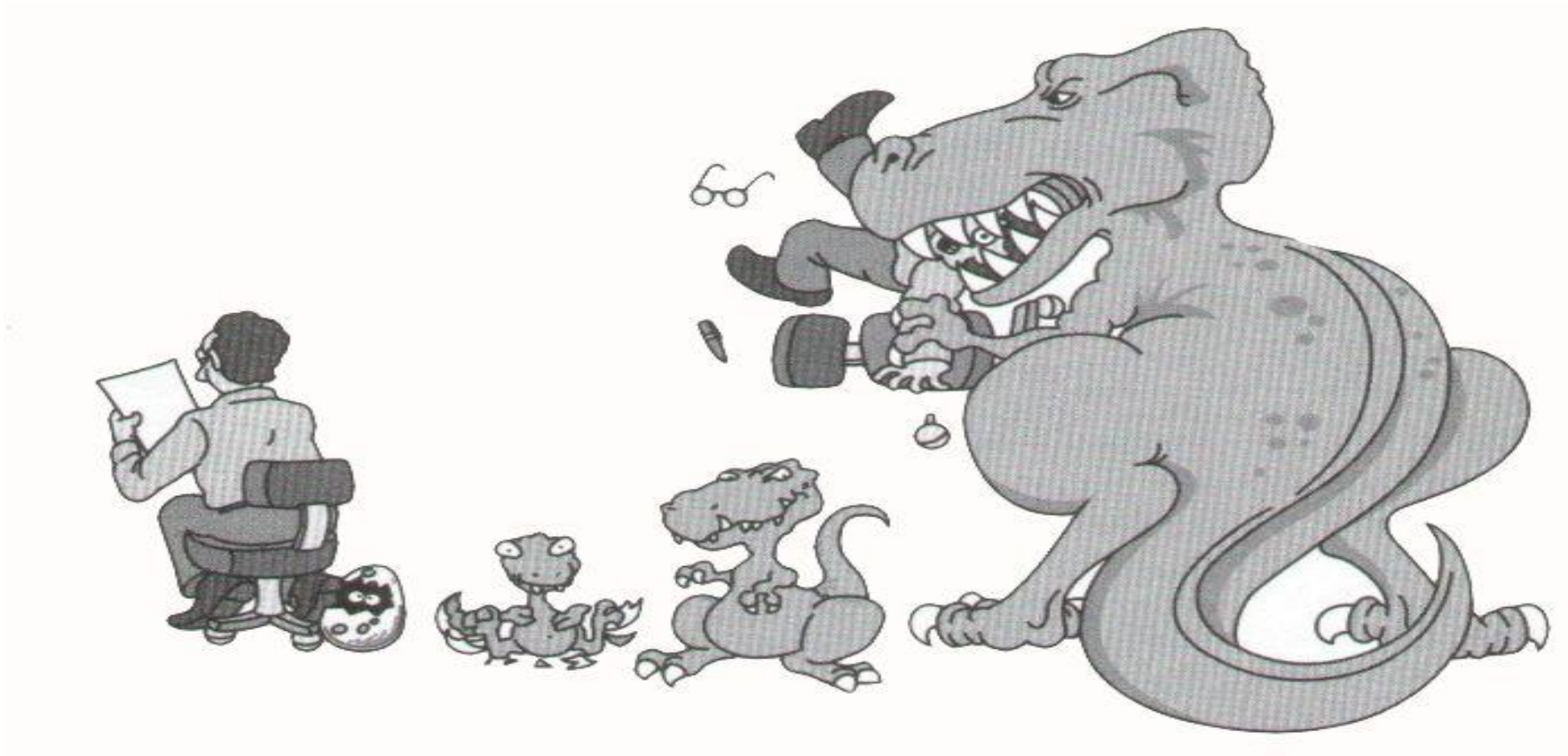


Atenţie

Găsirea târzie a erorilor ⇒ un cost mai mare pentru a le repara



Erori? Trebuie reparate cât mai devreme posibil



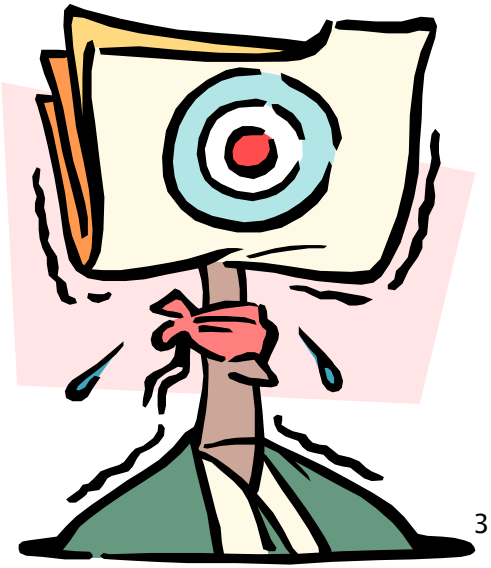
CERINŢE MODELARE IMPLEM. TESTARE CLIENT

Testare profesională

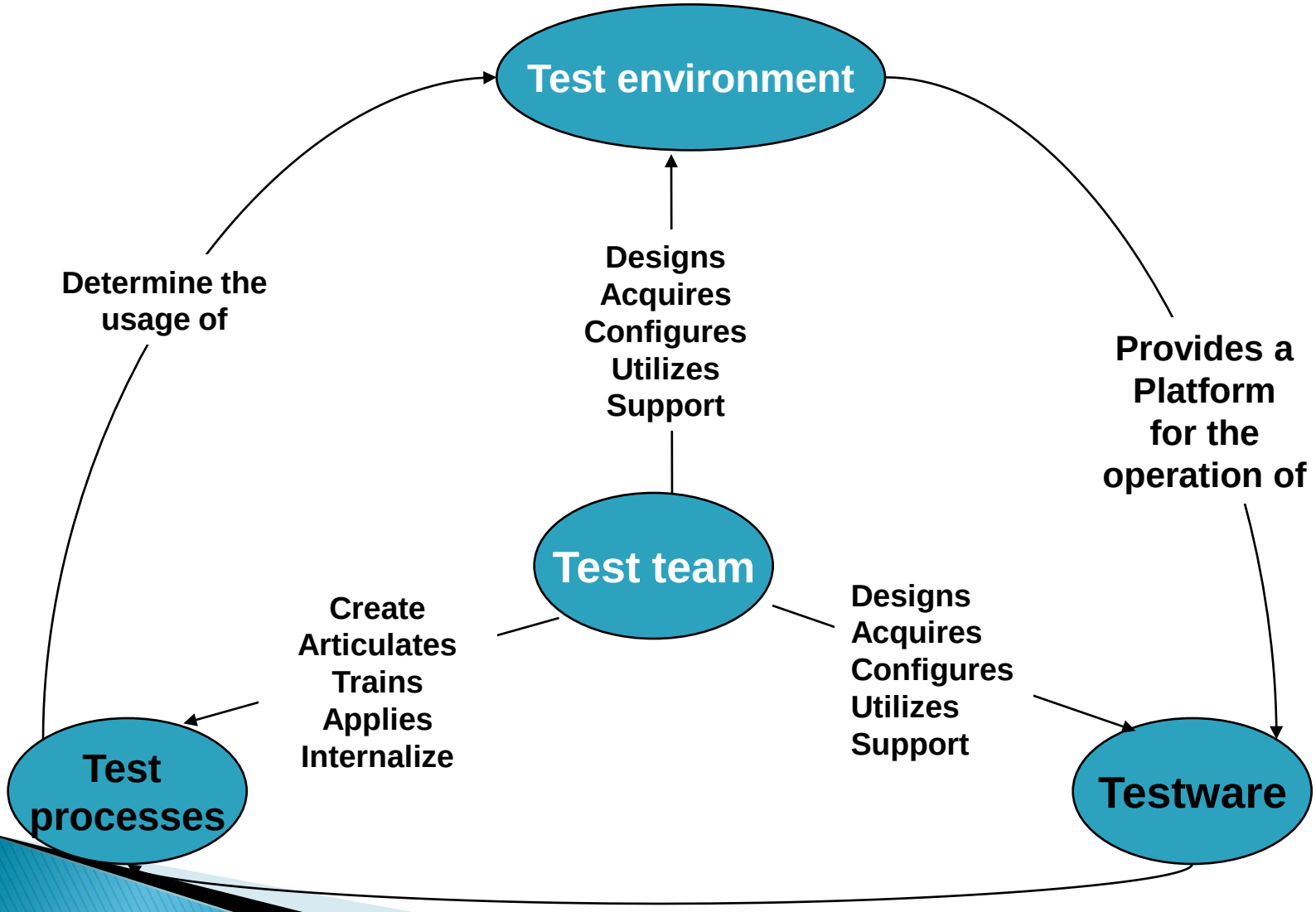
Profesionalismul în testare constă în abilitatea de a selecta numărul minim de cazuri de testare eficientă ce va fi capabil să verifice numărul maxim de funcţii ale sistemului

Când oprim testarea?

- Niciodată
- Când numărul de erori găsite într-un ciclu de testare este mai mic decât un număr stabilit
- Când nu mai sunt găsite defecte critice şi majore
- Când timpul a expirat



Schema unui Sistem de Testare



Metodologii de testare

- Diferenţa dintre testare SW şi debug SW
- Nivele de Test
- Metode de Testare
- Conţinutul Testării
- Testare Manuală vs Testare Automată

Diferența dintre testare SW & debug

Testare	Debug
• Verificarea respectării cerințelor	• Verificarea validității secțiunilor
• De regulă e făcută de o entitate externă şi neutră	• E făcută de programator
• Este un proces planificat şi controlat	• E un proces aleator

Nivele de test

- Unitate sau Debug
- Modul/Sub-Sistem
- Integrare
- Sistem
- Acceptare

Unit Testing

- Testarea unei funcţii, a unui program, a unui ecran, a unei funcţionalităţi
- Se face de către programatori
- Predefinită.
- Rezultatele trebuie documentate
- Se folosesc simulatoare pentru Input şi Output

Testare la integrare

- Testarea funcţionării unor module în acelaşi timp
- Testarea coexistenţei
- Se execută de către programatori sau de către testeri analişti
- Testare pre-planificată
- Rezultatele se documentează

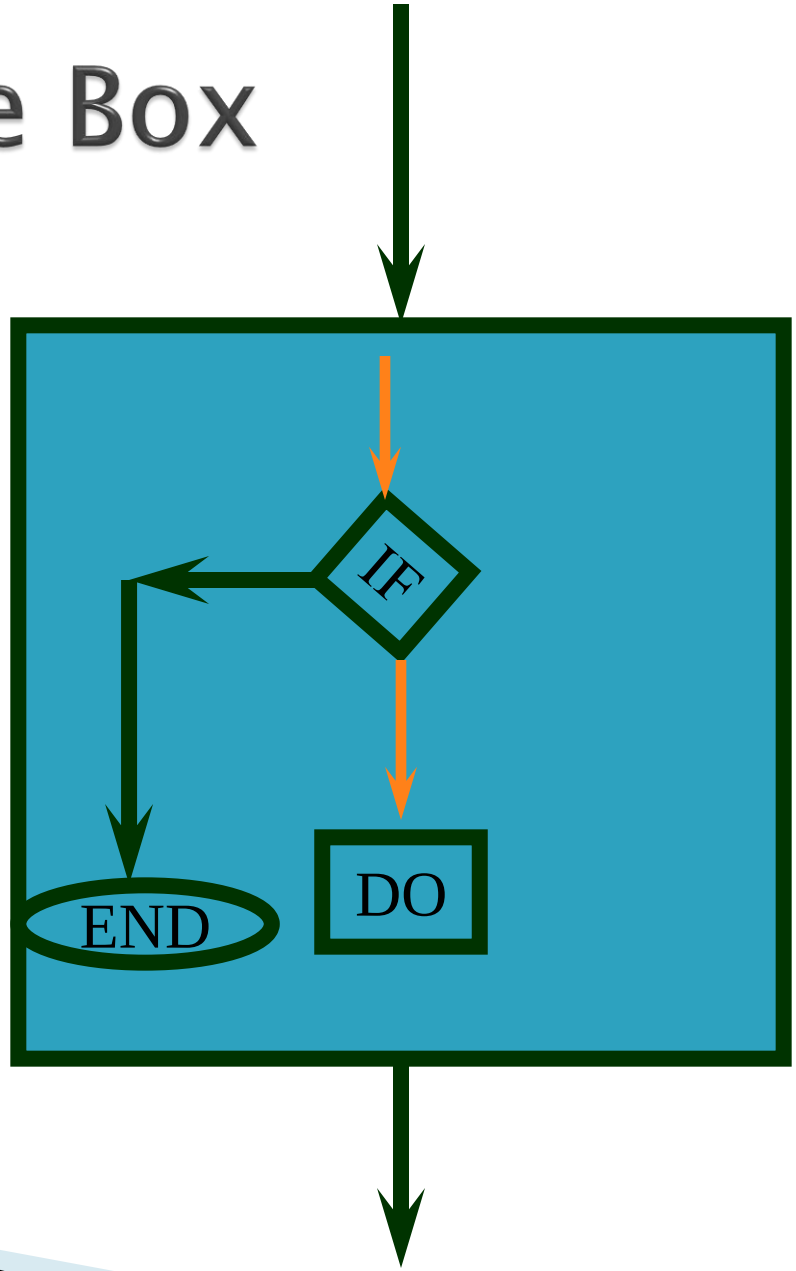
System Testing

- System testing of software or hardware is testing conducted **on a complete, integrated system** to evaluate the system's compliance with its specified requirements.
- System testing falls within the scope of **black box testing**
- System testing is a more limiting type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole.

Metode de testare

- White Box
- Black Box
- Gray Box
- Graphical user Interface Testing
- Acceptance Testing
- Regression Testing

White Box

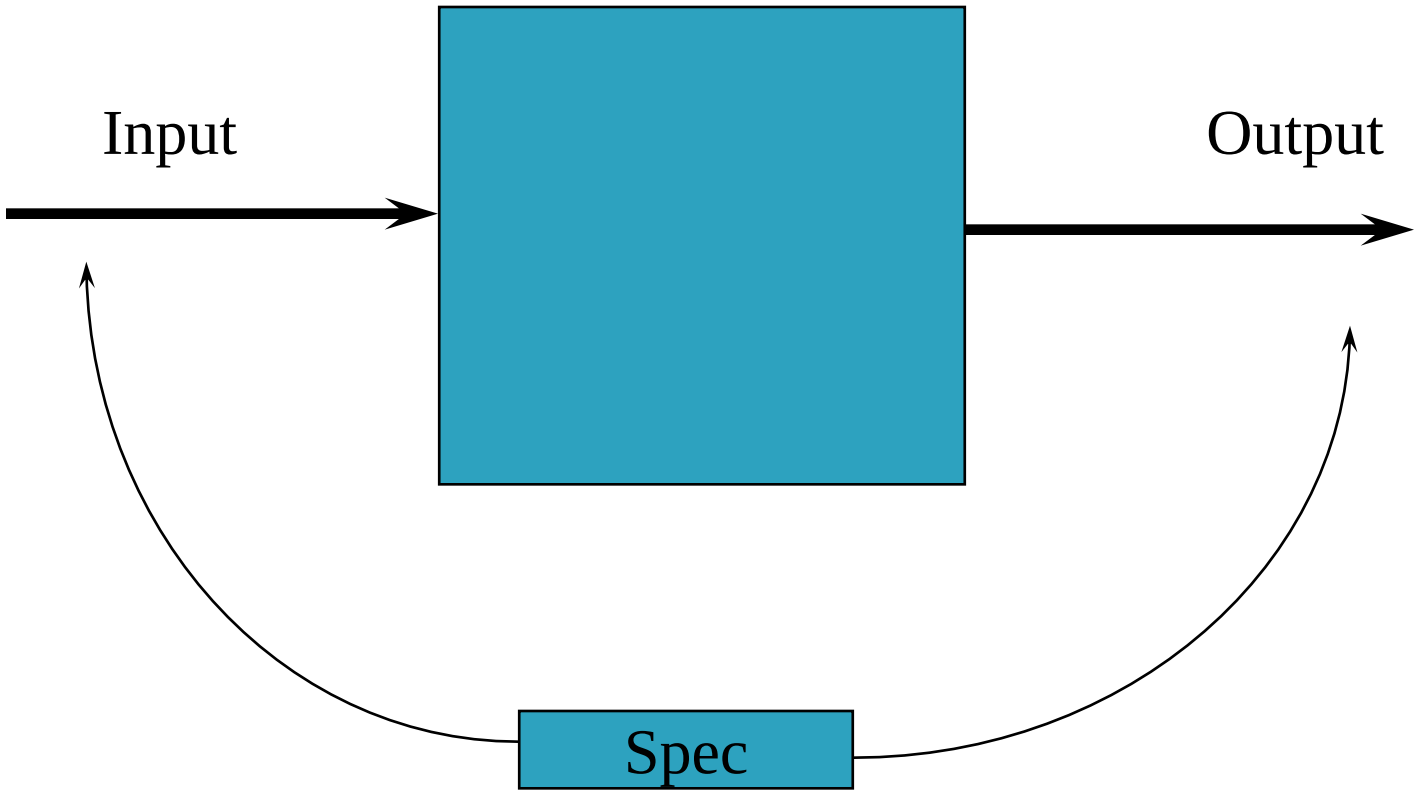


White Box (2)

- The tester has access to the internal data structures and algorithms

- Types of white box testing
  - api testing – Testing of the application using Public and Private APIs
  - code coverage – creating tests to satisfy some criteria of code coverage
  - fault injection methods
  - mutation testing methods
  - static testing – White box testing includes all static testing

Black Box



- "like a walk in a dark labyrinth without a flashlight,"

Black Box (2)

- Specification-based testing

- Black box testing methods include: equivalence partitioning, boundary value analysis, all-pairs testing, fuzz testing, model-based testing, traceability matrix, exploratory testing and specification-based testing.

Grey Box

- This involves having access to internal data structures and algorithms for purposes of designing the test cases, but testing at the user, or black-box level
- Manipulating input data and formatting output do not qualify as "grey-box," because the input and output are clearly outside of the "black-box" that we are calling "the software under test"

GUI Testing

- In computer science, GUI software testing is the process of testing a product that uses a graphical user interface, to ensure it meets its written specifications.
- The variety of errors found in GUI applications:
  - Data validation, Incorrect field defaults, Mandatory fields, not mandatory, Wrong fields retrieved by queries, Incorrect search criteria
  - Field order, Multiple database rows returned, single row expected
  - Currency of data on screens, Correct window modality?
  - Control state alignment with state of data in window?

Acceptance Testing

- **A black-box testing performed on a system** prior to its delivery
- In software development, acceptance testing by the system provider is often distinguished from acceptance testing by the customer (the user or client) prior to accepting transfer of ownership.
- In such environments, acceptance testing performed by the customer is known as **user acceptance testing** (UAT).
- This is also known as **end-user testing**, site (acceptance) testing, or field (acceptance) testing.

Regression Testing

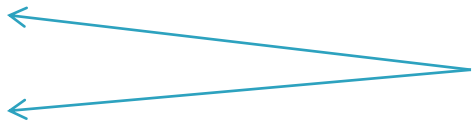
- Regression testing is **any type of software testing** which seeks to uncover software regressions.
- Such regressions occur whenever software functionality that was previously working correctly, stops working as intended.
- Typically regressions **occur as an unintended consequence of program changes**.
- Common methods of regression testing include **re-running previously run tests and checking whether previously fixed faults have re-emerged**

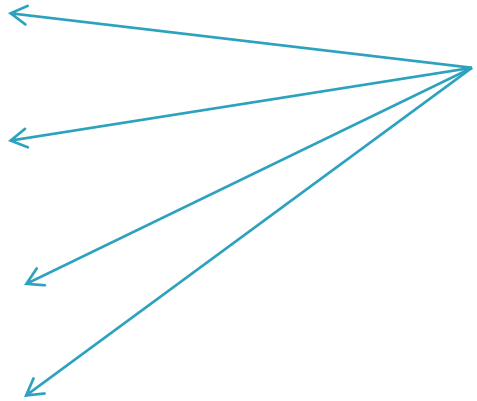
Testare Manuală – Scenariu de Test

- Definirea structurii testării
- Se împarte sistemul într-o structură ierarhică
- Se descriu resursele necesare pentru testare
- Se planifică testarea
- Împărţirea în paşi se face ţinând cont de cerinţe
- Se descrie ce va fi testat pentru componente şi funcţii
- Descrie CUM să testăm sistemul

Testare Automată – Clasa Square

```
public class Square {
        private int side;

        Square(){
            side = 0;
        }

        Square(int s){
            side = s;
        }

        public int getSide(){
            return side;
        }

        public void setSide(int s){
            side = s;
        }

        public int getArea(){
            return side*side;
        }

        public int getPerimeter(){
            return 4*side;
        }

        public String toString(){
            return "Patrat: latura = " + side+"\n\t aria = "+ getArea() +
                    "\n\t perimetru = " + getPerimeter();
        }
}
```





Testare Automată – Clasa SquareTest

```
 */
public class SquareTest {
    Square s;
    Square[] sList = new Square[5];

    @Test
    public void testConstructorImplicit(){
        System.out.println("Testare Constructor Implicit");
        s = new Square();
        try{
                System.out.println(s);
                assertTrue("Test Constructor Implicit: Valoarea implicita a laturii e gresita.",
                        s.getSide() == 0);
                System.out.println("Testarea Constructor Implicit sfarsita cu succes.");
        }
        catch (AssertionError e){
                System.out.println(e.getMessage());
        }
    }

    @Test
    public void testValoriImplicit(){
        System.out.println("Testare Valori Implicite");
        s = new Square();
        try{
            System.out.println(s);
            assertTrue("Test Valori Implicite: Valorile implicte sunt gresite.",
                    s.getSide() == 0 && s.getArea() == 0 && s.getPerimeter() == 0);
            System.out.println("Test Valori Implicite sfarsit cu succes.");
        }
        catch (AssertionError e){
            System.out.println(e.getMessage());
        }
    }
```

Testare Automată – Clasa SquareTest 2

```
@Test
public void testConstructorParametri(){
    System.out.println("Testare Constructor Parametri");
    s = new Square(10);
    try{
        System.out.println(s);
        assertTrue("Test Constructor cu parametri: Valorile initiale sunt gresite.",
                s.getSide() == 10 && s.getArea() == 100 && s.getPerimeter() == 40);
        System.out.println("Test Constructor cu parametri sfarsit cu succes.");
    }
    catch (AssertionError e){
        System.out.println(e.getMessage());
    }
}

@Test
public void testConstructorParametri2(){
    System.out.println("Testare Constructor Parametri 2");
    s = new Square(-10);
    try{
        System.out.println(s);
        assertTrue("Test Constructor cu parametri: Valorile initiale sunt gresite.",
                s.getSide() == 0 && s.getArea() == 0 && s.getPerimeter() == 0);
        System.out.println("Test Constructor cu parametri sfarsit cu succes.");
    }
    catch (AssertionError e){
        System.out.println(e.getMessage());
    }
}
```

Testare Automată – Clasa SquareTest 3

```
@Test
public void testArray(){
    System.out.println("Testare Array");
    try{
        for (int i = 0; i < 5; i++){
            sList[i] = new Square(2*i);
            System.out.println(sList[i]);
            assertTrue("Test Array: Valori gresite pentru i = " + i,
                    sList[i].getSide() == 2*i && sList[i].getArea() == 4*i*i && sList[i].getPerimeter() == 8*i);
        }
        System.out.println("Test Array terminat cu succes.");
    }
    catch (AssertionError e){
        System.out.println(e.getMessage());
    }
}

@Test
public void testSet(){
    System.out.println("Testare Set");
    try{
        for (int i=0;i<5;i++){
            sList[i] = new Square(2*i);
            sList[i].setSide(3*i);
            System.out.println(sList[i]);
            assertTrue("Test Set: Valorile modificate sunt gresite.",
                    sList[i].getSide() == 3*i && sList[i].getArea() == 9*i*i && sList[i].getPerimeter() == 12*i);
        }
        System.out.println("Test Set terminat cu succes.");
    }
    catch (AssertionError e){
        System.out.println(e.getMessage());
    }
}
```

Testare Automată vs Testare Manuală

- *Se găsesc rapid problemele*
- *Se câştigă timp când e nevoie să repetăm testele*
- *Procesul de scriere a codului e mult mai flexibil*
- *Reduce volumul de testare manuală*
- *Dezvoltarea software devine previzibilă şi repetabilă*

- *Rezolvă problemele de interfaţă: scrierea corectă a textelor, mesajelor, aranjarea corectă în pagină, în ordinea care trebuie, sunt vizibile, etc.*
- *Realizarea Scenariilor de test poate fi o treabă de durată şi anevoioasă şi implică o cunoaştere temeinică a întregului sistem*

Coding Style – Motivaţie

- Convenţiile de programare sunt importante deoarece:
- 80% din timpul alocat unei componente software este întreţinere
- Foarte rar un produs software este întreţinut pe toată durata folosirii lui de către aceeaşi persoană
- Convenţiile de cod îmbunătăţesc lizibilitatea produsului, şi permit inginerilor software să înţeleagă rapid un program nou

Coding Style – Cerinţe

- Folosirea fără rezerve a Comentariilor: ce fac procedurile, ce reprezintă variabilele, explicarea paşilor algoritmului, etc.
- Folosirea numelor sugestive pentru variabile si proceduri
- Scrierea modulara a proiectului
- Folosirea perechilor de tip set/get, start/stop, adauga/sterge, salvare/incarcare

Coding Style – Links

- C++:
  - http://www.chris-lott.org/resources/cstyle/
  - http://geosoft.no/development/cppstyle.html
- Java:
  - http://java.sun.com/docs/codeconv/
  - http://geosoft.no/development/javastyle.html

Links

- Behavioral Patterns: http://www.oodesign.com/behavioral-patterns/
- Pattern Synopses2,3: http://www.mindspring.com/~mgrand/pattern_synopses2.htm http://www.mindspring.com/~mgrand/pattern_synopses3.htm
- Software Quality Assurance: http://satc.gsfc.nasa.gov/assure/agbsec3.txt
- Software Testing: http://en.wikipedia.org/wiki/Software_testing
- GUI Software Testing: http://en.wikipedia.org/wiki/GUI_software_testing
- Regression Testing: http://en.wikipedia.org/wiki/Regression_testing
- Junit Test Example: http://www.cs.unc.edu/~weiss/COMP401/s08-27-JUnitTestExample.doc