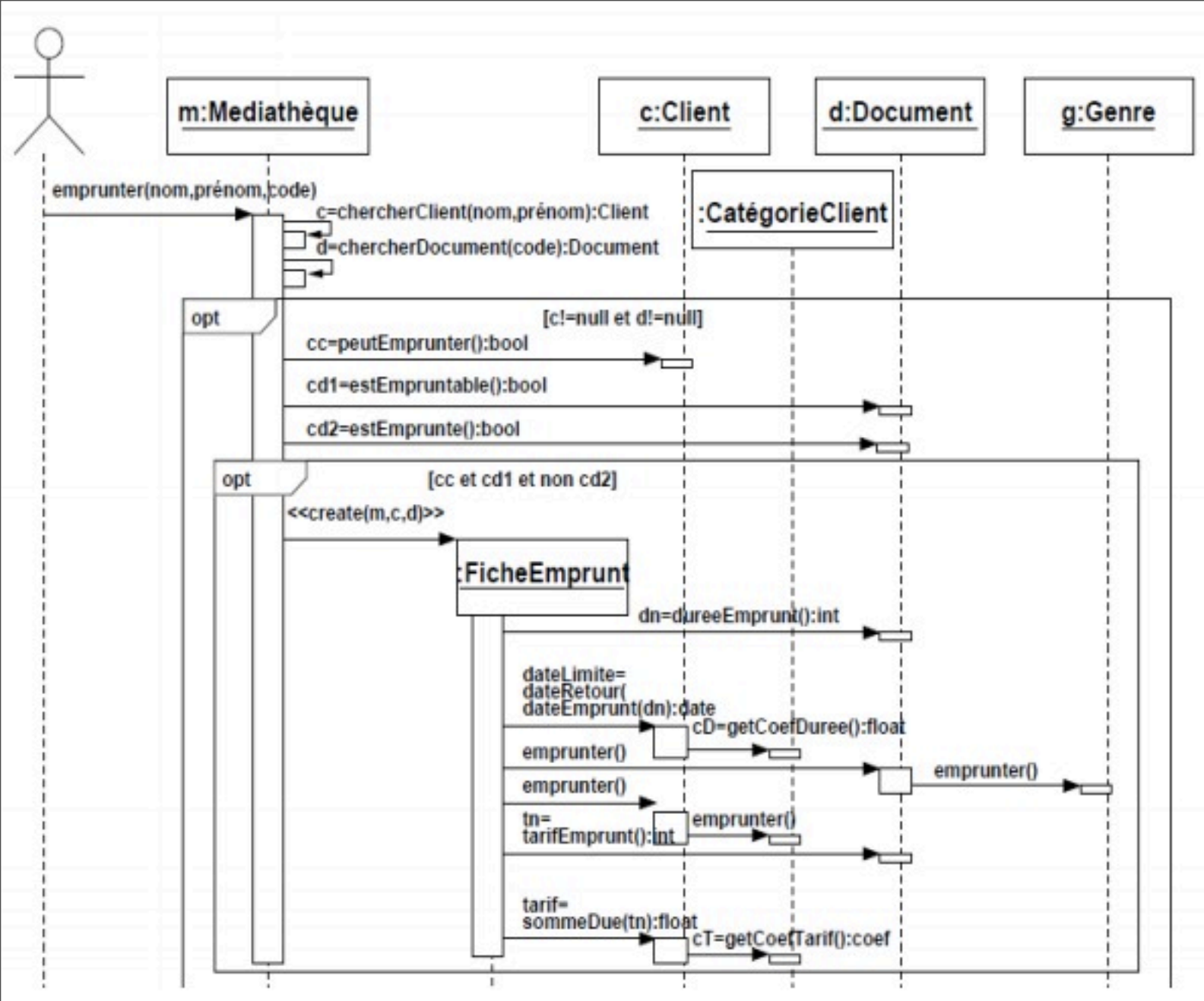


Bibliographie

- Programmation par les tests, ESIREM, Céline ROUDET
- Comment écrire du code testable, Conférence Agile France 2010, Florence CHABANOIS
- Reflexion on Software Quality and Maintenance, Alexandre Bergel, Chili
- An Introduction to Test-Driven Development (TDD), Craig Murphy
- Tests et Validation du logiciel, <http://home.nordnet.fr/~ericleleu>
- Test à partir de modèles : pistes pour le test unitaire de composant, le test d'intégration et le test système, Yves Letraon
- Les tests en orienté objet, J. Paul Gibson <http://www-inf.int-evry.fr/cours/CSC4002/Documents>
- Mocks and Stubs, Martin Fowler
- Introduction au test du logiciel, Premiers pas avec JUnit, Mirabelle Nebut
- Écrire du code testable Par Aurélien Bompard

UML & Tests d'intégration

- ➔ Diagrammes de classes
- ➔ Diagrammes de séquence



Tests d'intégration

L'opération d'emprunter requière une coordination entre les objets : client, médiathèque, document et ficheEmprunt.

On veut vérifier les traces de communications ...
Ces tests peuvent être déduits des diagrammes de séquences.

Integration Test1 : emprunter

- ➔ Un « emprunt » n'est pas autorisé parce que le document n'est pas empruntable
 - Construct **a document**, and make it **not Empruntable**
 - Construct a client
 - Construct a FicheEmprunt for the client and document
 - Check that:
 - 1. the system handles the exceptional case in a meaningful way
 - 2. the client and document states/statistics have not been changed

Integration Test2 : emprunter

- ➔ An « emprunt » is not authorised because the document is « emprunté »
 - Construct a document, which is empruntable and emprunté
 - Construct a client
 - Construct a FicheEmprunt for the client and document
 - Check that:
 - 1.the system handles the exceptional case in a meaningful way
 - 2.the client and document states/statistics have not been changed

Integration Test3 : emprunter

- ➔ Integration Test 3: Emprunt is authorised
 - Construct a **document**, which is **emprunable** and not emprunté
 - Construct a **client**
 - Construct a **FicheEmprunt** for the client and document using a dummy mediatheque
 - Check that:
 - 1. the tarif and duree des emprunts are as required
 - 2. the client and document states/statistics have been updated as required

Integration Test3 : emprunter

[http://www-inf.it-sudparis.eu/cours/CSC4002/
Documents/tests-mediatheque-1pp.pdf](http://www-inf.it-sudparis.eu/cours/CSC4002/Documents/tests-mediatheque-1pp.pdf)

```
public class IntegrationTest {
    Genre g;
    Localisation l;
    Document d1;
    CategorieClient cat;
    Client c1;
    @Before
    public void setUp() throws Exception {
        g = new Genre("Test_nom1");
        l = new Localisation("Test_salle1", "Test_rayon1");
        d1 = new Video("Test_code1", l, "Test_titre1", "Test_auteur1",
            "Test_annee1", g, 120, "Test_mentionLegale1");
        cat = new CategorieClient("TarifReduit", 4, 25.0, 1.0, 1.0, true);
        c1 = new Client("nom1", "prenom1", "adresse1", cat, 0);
    }
    @After
    public void tearDown() throws Exception {
        l = null;
        g = null;
        d1 = null;
        cat = null;
        c1 = null;
    }
}
```


Integration Test3 : emprunter

[http://www-inf.it-sudparis.eu/cours/CSC4002/
Documents/tests-mediatheque-1pp.pdf](http://www-inf.it-sudparis.eu/cours/CSC4002/Documents/tests-mediatheque-1pp.pdf)

@Test

```
public void creationFicheEmpruntDocEmpruntable()  
    throws OperationImpossible, InvariantBroken {  
    // INTEGRATION TEST 3 – Emprunter – Verify correct  
co-ordination by  
    // FicheEmprunt  
    d1.metEmpruntable();  
    FicheEmprunt f1 = new FicheEmprunt(c1, d1);  
    // check that the client, document and genre states  
have been updated correctly  
    Assert.assertEquals(1, d1.getNbEmprunts());  
    Assert.assertEquals(1, c1.getNbEmpruntsEffectues());  
    Assert.assertEquals(1, g.getNbEmprunts());  
    // check that the duration and paiement information  
are ok  
    Assert.assertTrue(f1.getDureeEmprunt() == 14);  
    Assert.assertTrue( f1.getTarifEmprunt() == 1.5);  
}
```

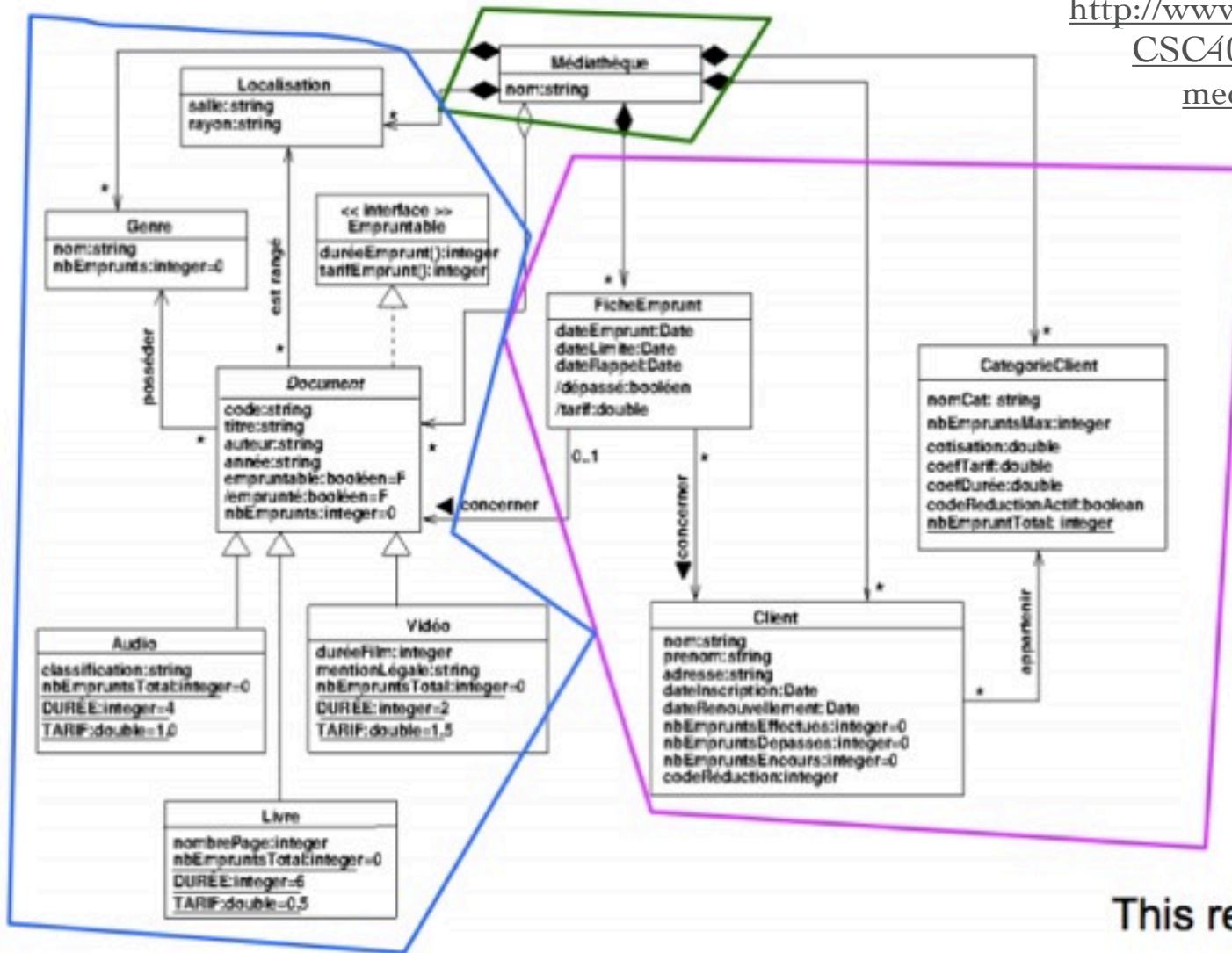
Integration Test3 : emprunter

[http://www-inf.it-sudparis.eu/cours/CSC4002/
Documents/tests-mediatheque-1pp.pdf](http://www-inf.it-sudparis.eu/cours/CSC4002/Documents/tests-mediatheque-1pp.pdf)

```
@Test(expected = OperationImpossible.class)
public void creationFicheEmpruntDocNonEmpruntable()
    throws OperationImpossible, InvariantBroken {
    // INTEGRATION TEST 1 - Emprunter - co-ordination
when Document not
    // empruntable
    new FicheEmprunt(c1, d1);
}
```

Integration Tests: Typical/ Example Development Status

<http://www-inf.it-sudparis.eu/cours/CSC4002/Documents/tests-mediathèque-1pp.pdf>



For example
Completed
Partial
Not yet implemented

This requires further analysis

Tests d'intégration

- ✓ Différents modules d'une application peuvent fonctionner unitairement, leur intégration, entre eux ou avec des services tiers, peut engendrer des dysfonctionnements.
- ✓ Il est souvent impossible de réaliser les tests unitaires dans l'environnement cible avec la totalité des modules à disposition.
- ➔ Les tests d'intégration ont pour objectif de créer une version complète et cohérente du logiciel (avec l'intégralité des modules testés unitairement) et de garantir sa bonne exécution dans l'environnement cible.

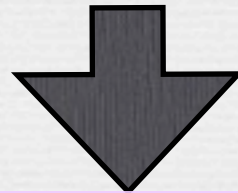
Tests d'intégration

Objectif

Vérifier les interactions entre composants unitaires

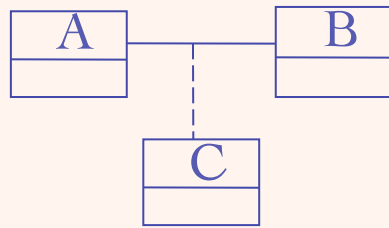
Difficultés principales de l'intégration

- Interfaces floues
- Implantation non conforme à la spécification
- Réutilisation de composants

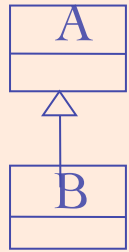
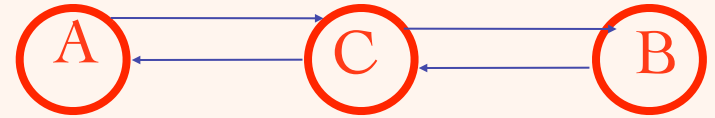


- 1) modéliser la structure de dépendances entre chaque composant et son environnement (graphe de dépendance des tests)
- 2) Choisir un ordre pour intégrer (assembler)

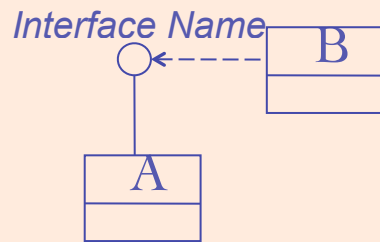
Graphe de dépendance : construction



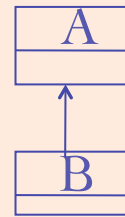
association class



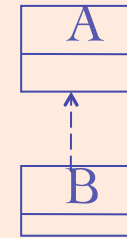
inheritance



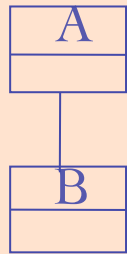
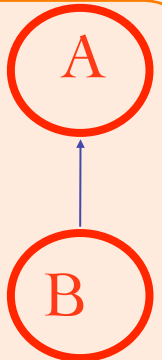
Interfaces



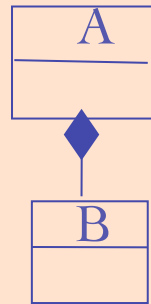
navigability



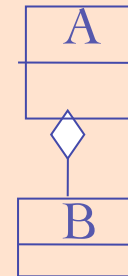
dependency



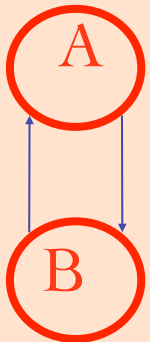
association



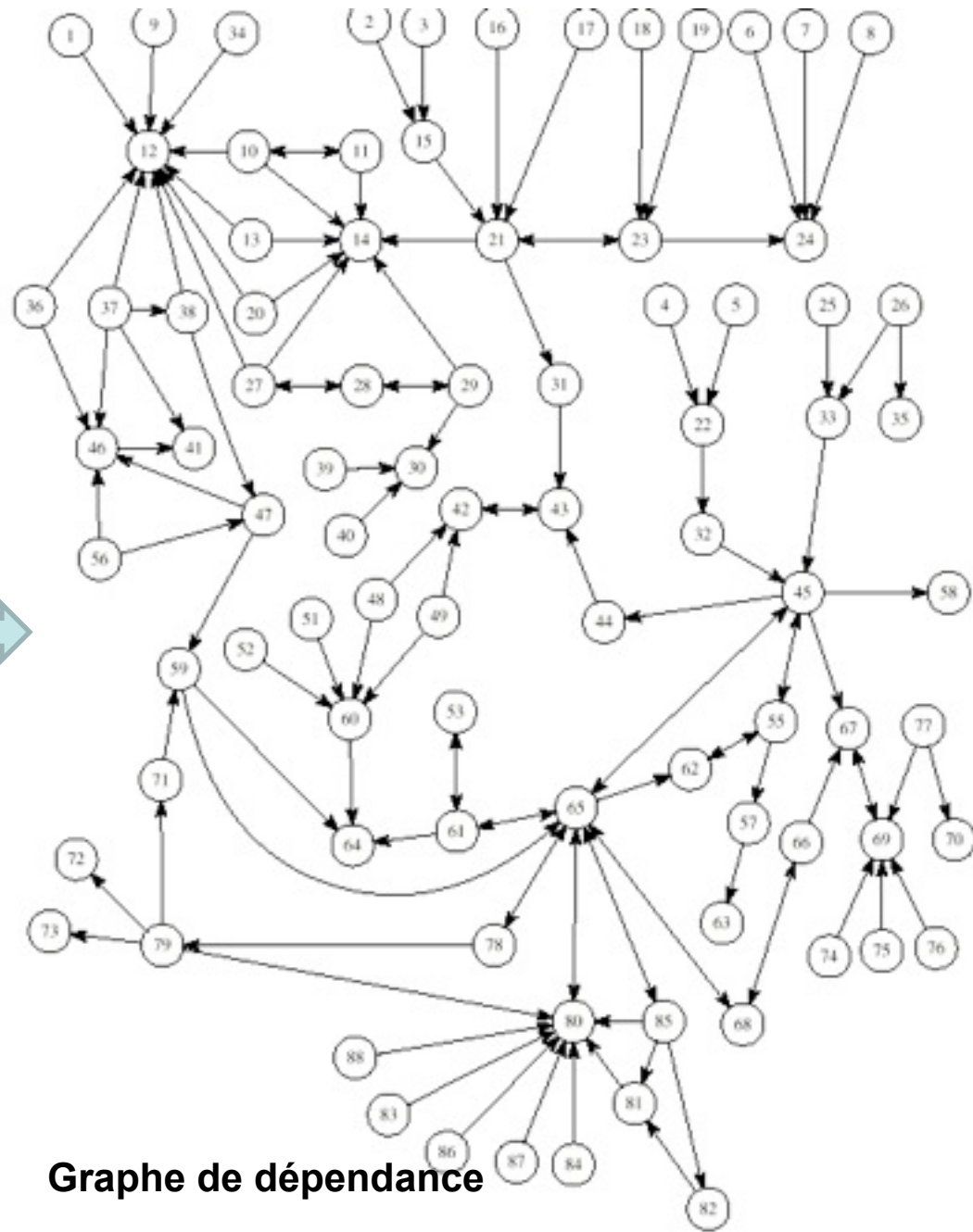
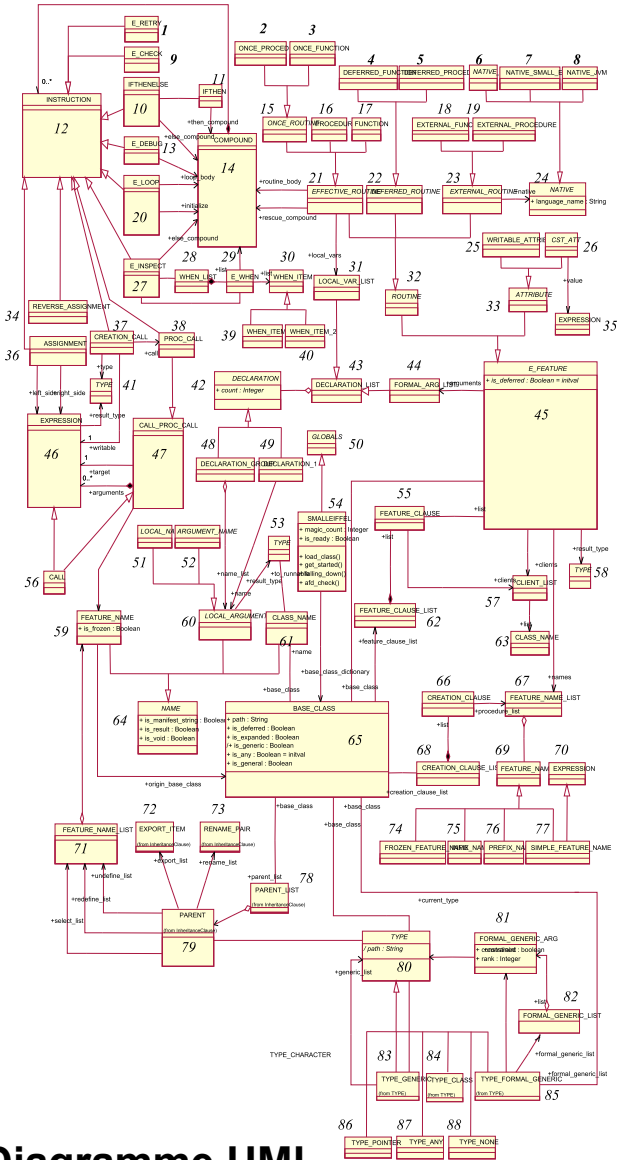
composition



aggregation



Compilateur GNU pour Eiffel :



Graphe de dépendance

Diagramme UML

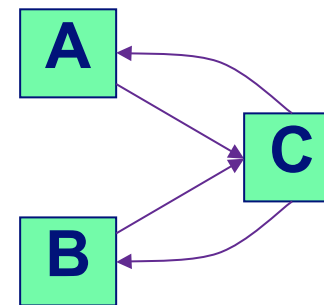
Test d'intégration : les interdépendances

- ➔ Une solution simple consiste à contraindre le concepteur
 - pas de boucle dans l'architecture
 - c'est souvent possible
 - **mais** les optimisations locales ne sont pas toujours optimales globalement
 - **mais** concevoir des composants interdépendants est souvent naturel



Bouchon de test

- Bouchon : une unité qui simule le comportement d'une unité



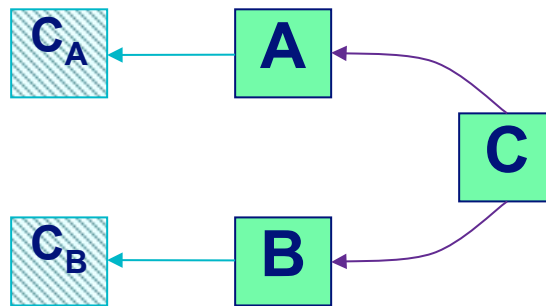
CFC

(Diffusion
Libre)

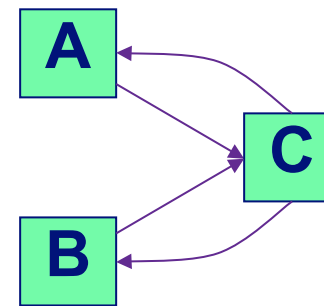


Bouchon de test

- Bouchon : une unité qui simule le comportement d'une unité



Bouchon spécifique



CFC

(Diffusion
Libre)

Tests d'intégration

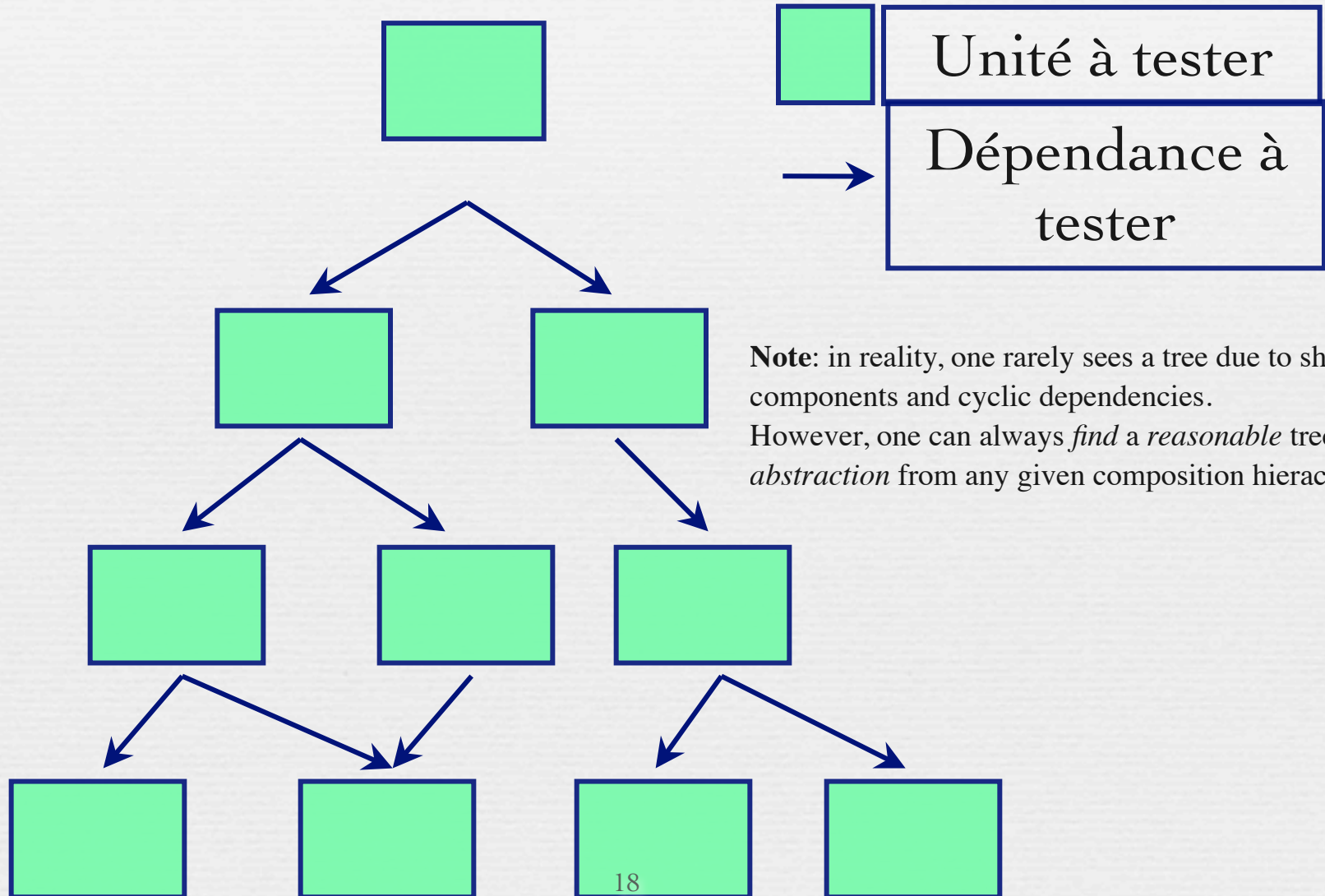
→ Architecture des dépendances

Note: in reality, one rarely sees a tree due to shared components and cyclic dependencies.

However, one can always *find a reasonable tree abstraction* from any given composition hierarchy.

Tests d'intégration

→ Architecture des dépendances

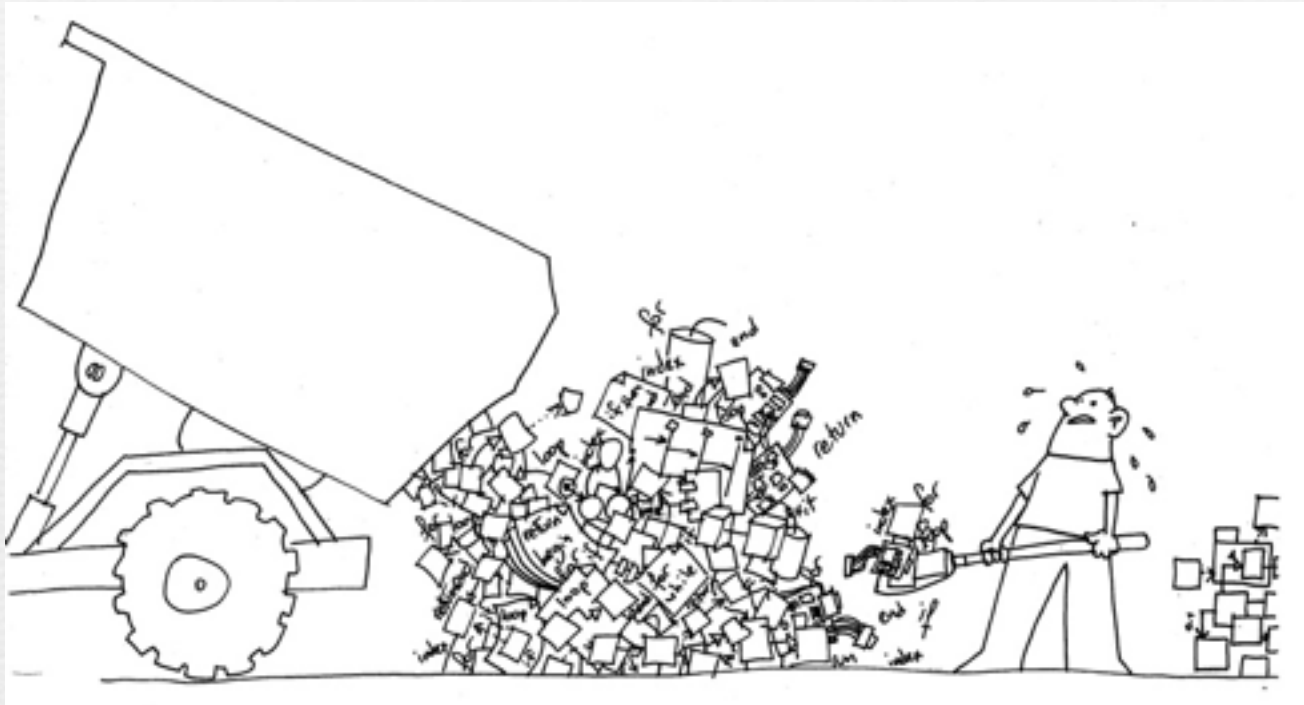


Stratégies

- ➔ Big-bang : tout est testé ensemble (peu recommandé)
- ➔ Top-down (peu courant)
- ➔ Bottom-up (la plus classique)

Intégration - Big-Bang

→ *Big Bang* – Validation du système –

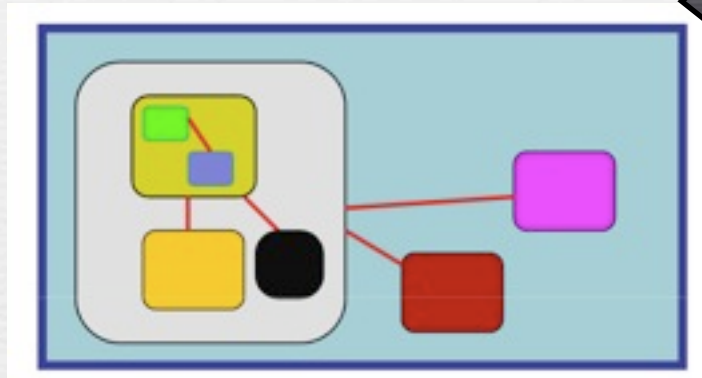


<http://emmanuelchenu.blogspot.com/>

Intégration - Big-Bang

Intégration de tous les composants à tester en une seule étape. (intégration massive)

Tests à l'interface du système

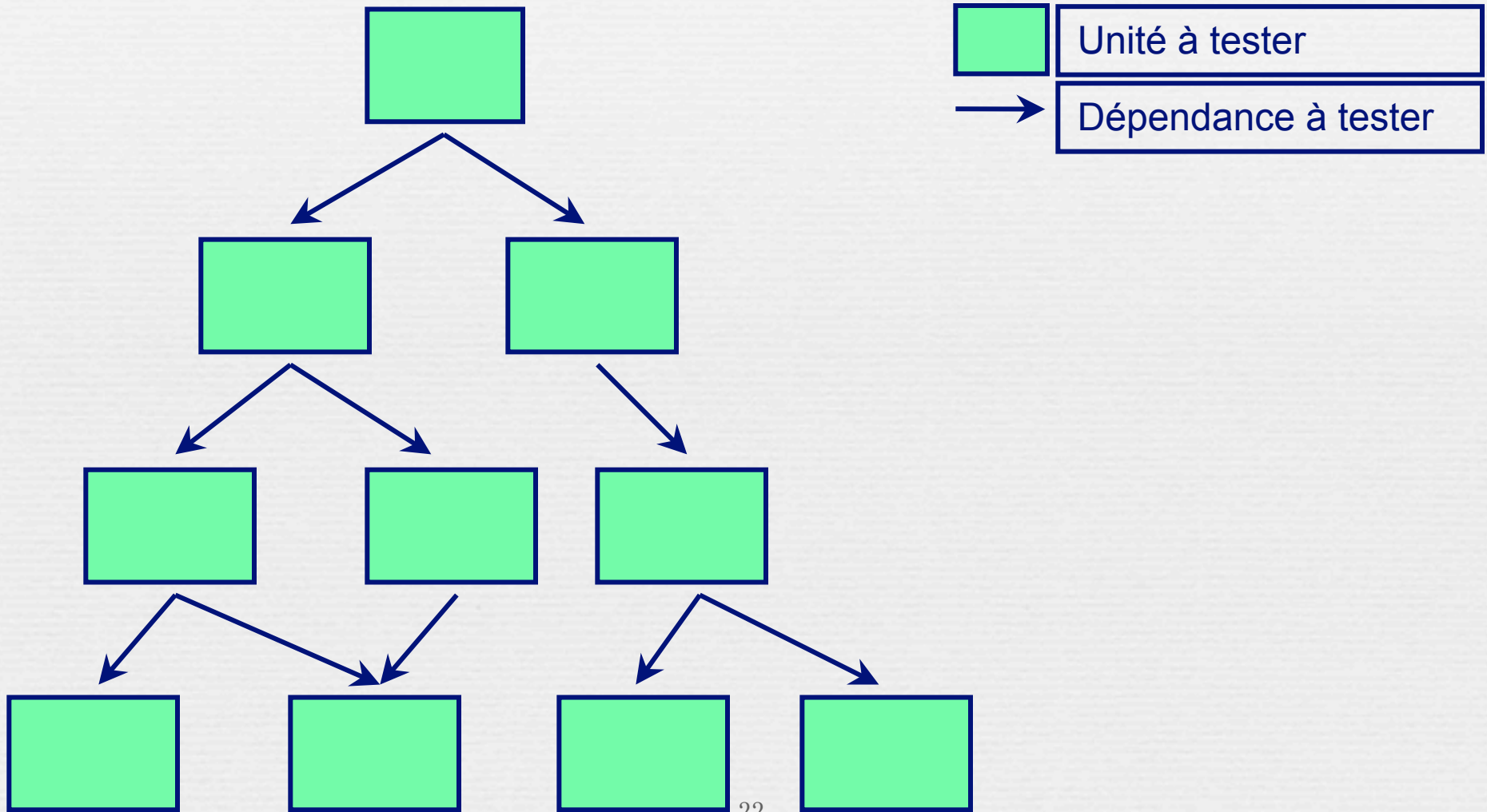


Usage Model testing

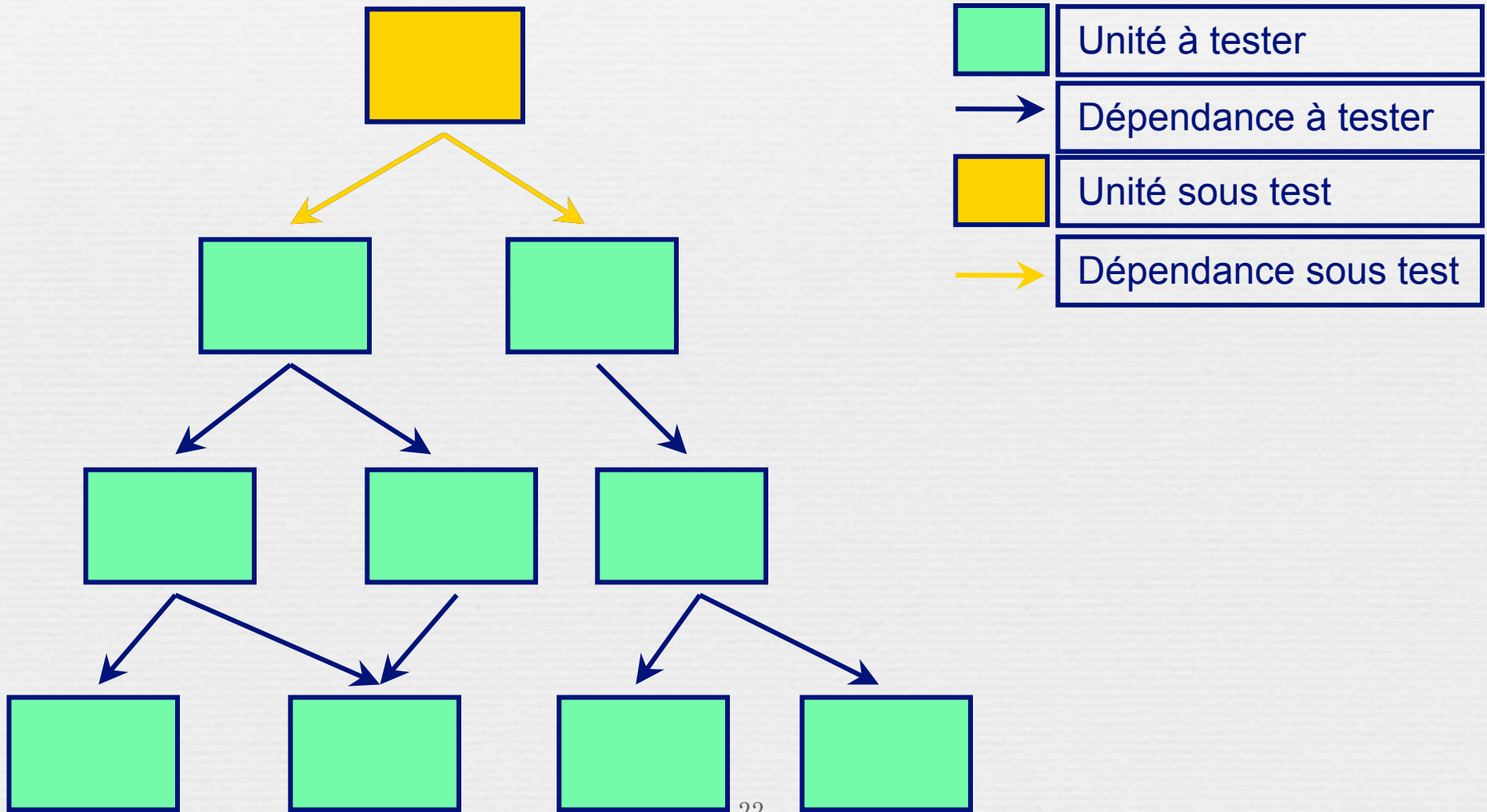
Principaux Problèmes:

- Les tests produisent des erreurs : Quelle en est la cause?
- La complexité induit des tests manquants
- Les tests ne commencent que lorsque tous les composants ont été «codés».

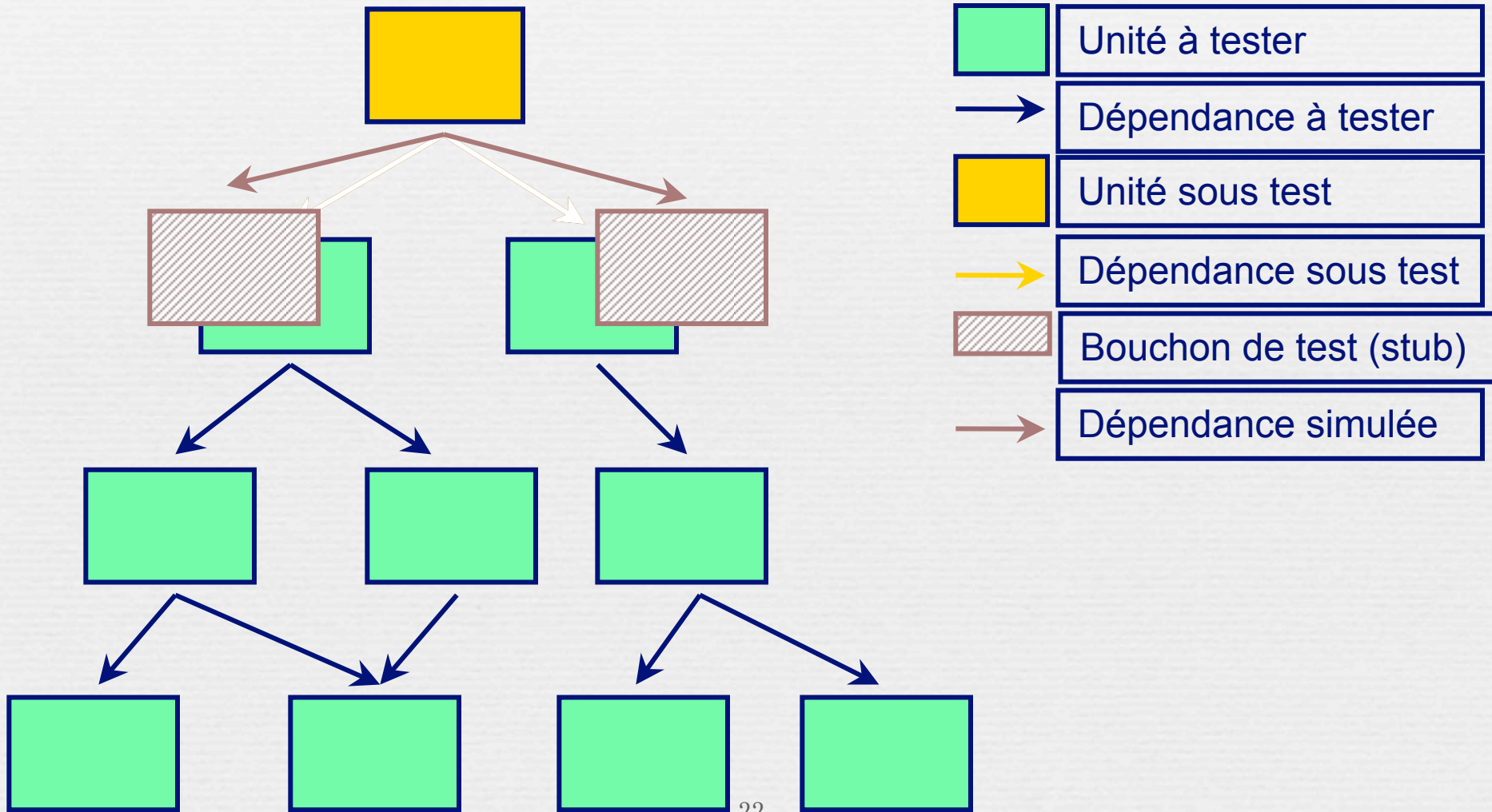
Approche descendante



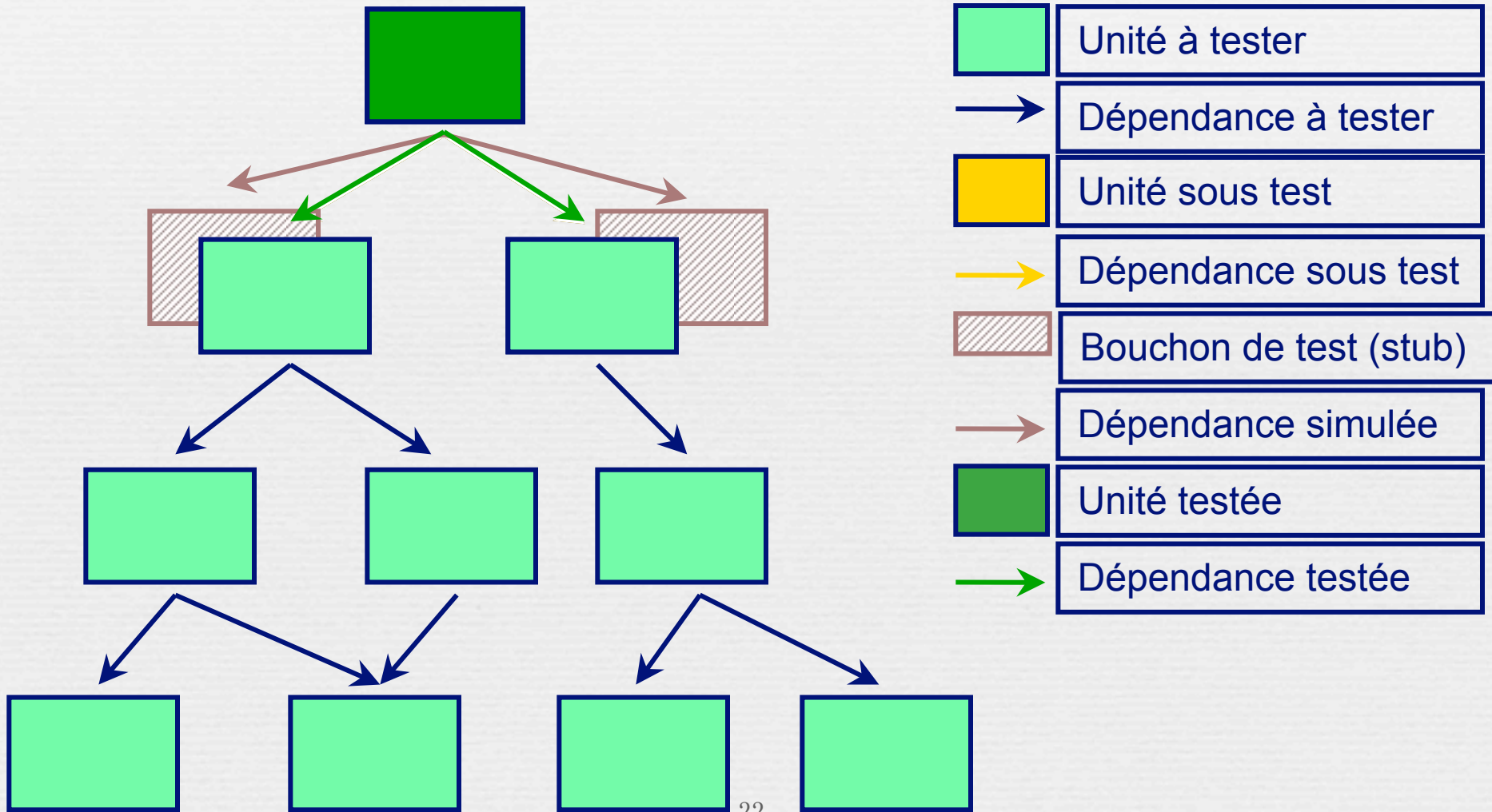
Approche descendante



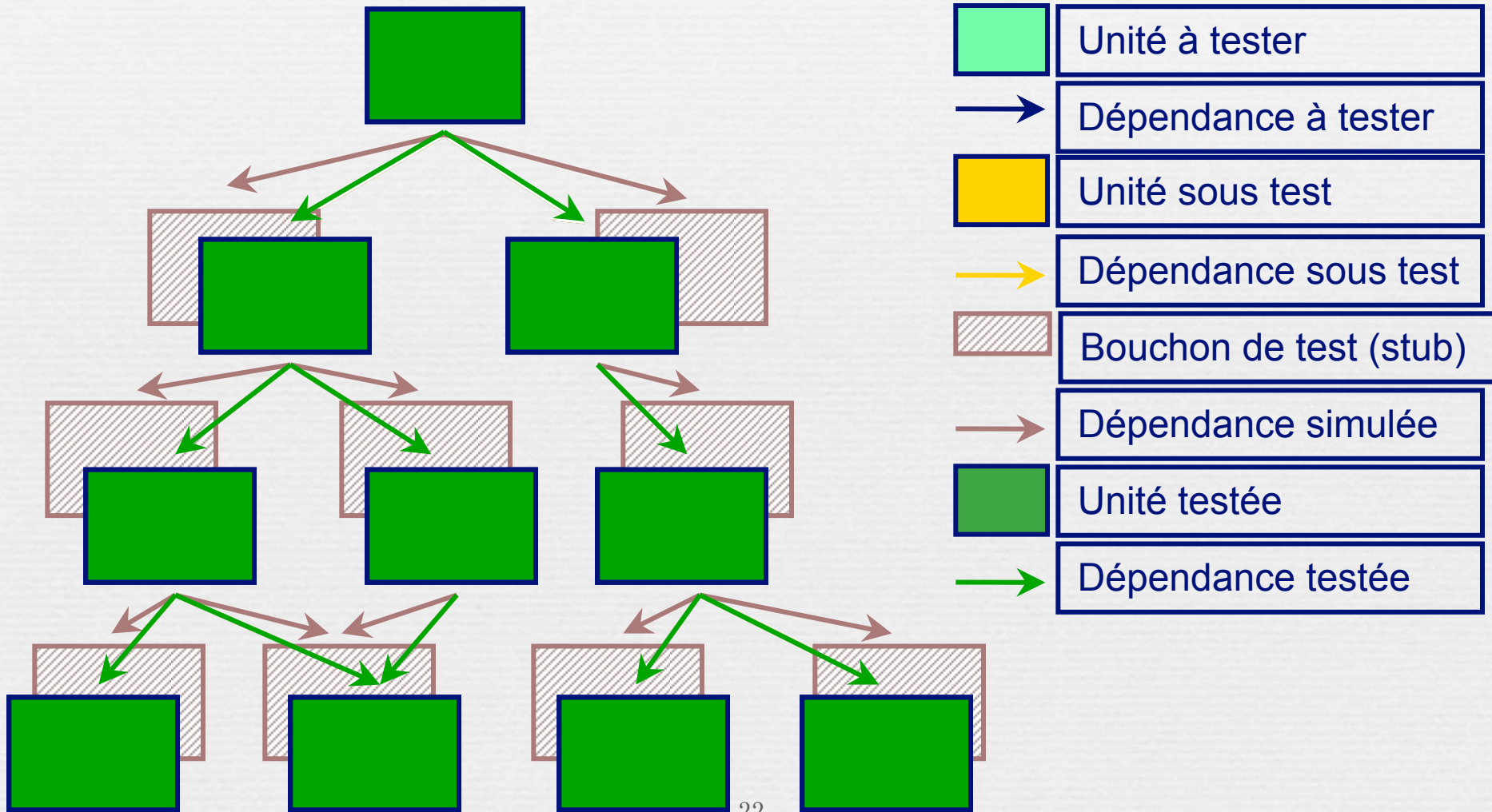
Approche descendante



Approche descendante



Approche descendante

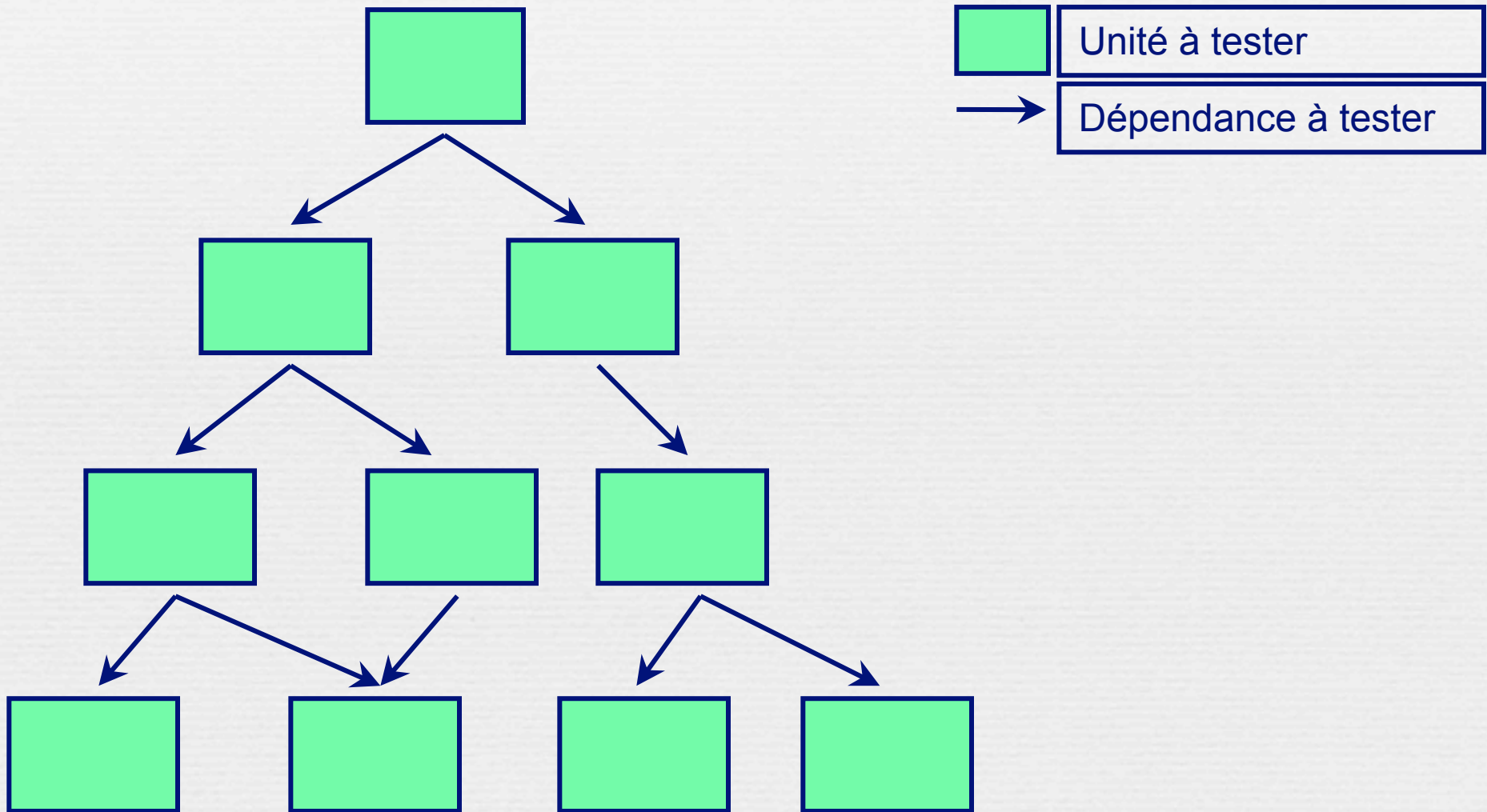


Approche descendante

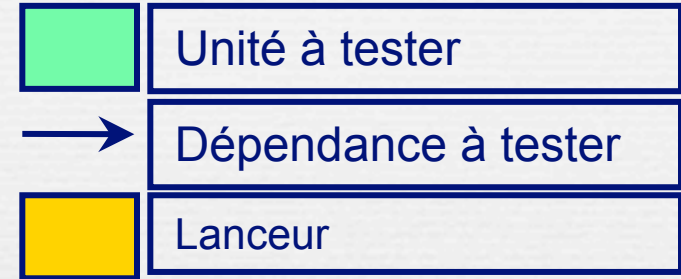
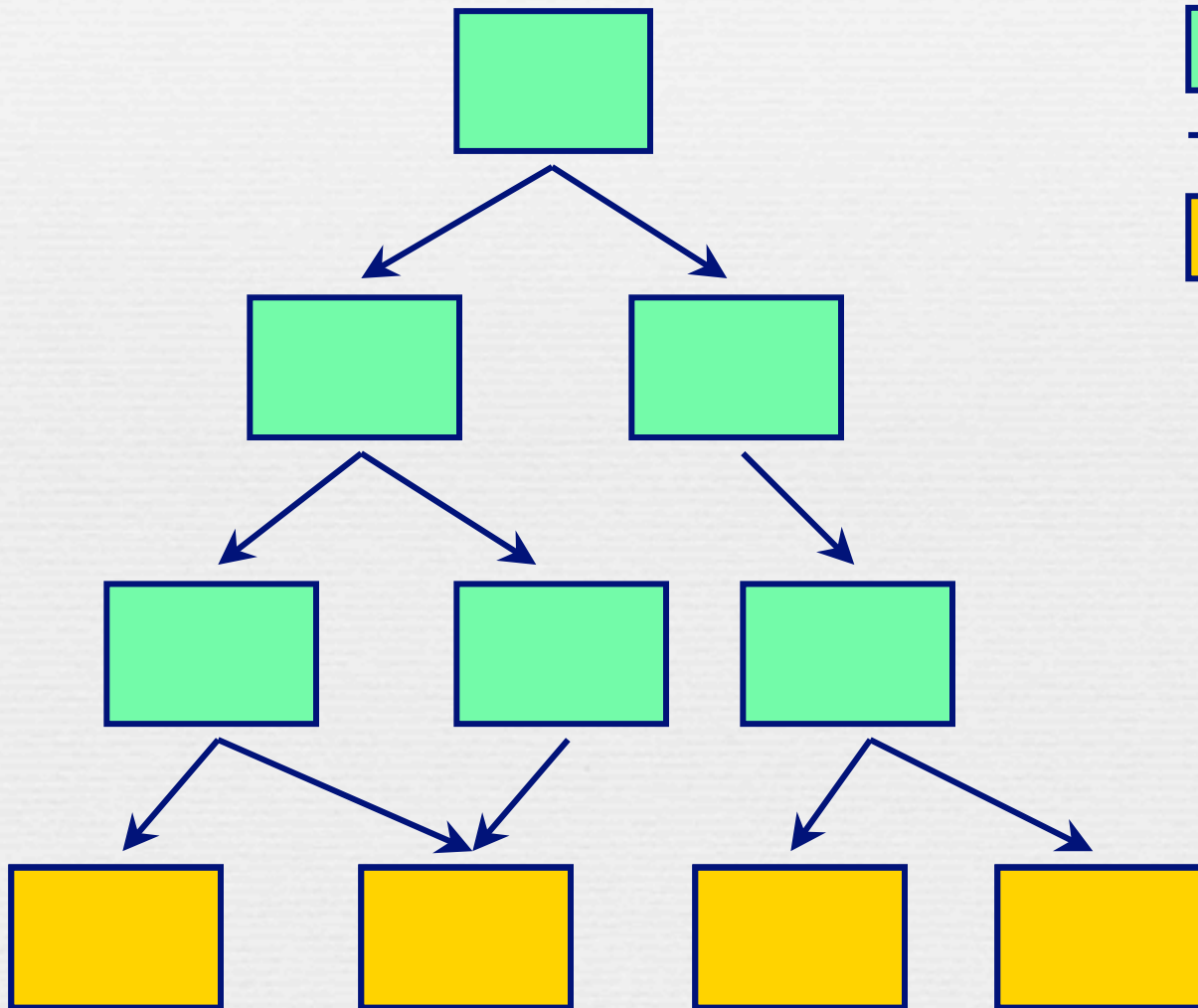
- ➔Création de **bouchons**
- ➔Test tardif des couches basses
- ➔Détection précoce des défauts d'architecture

- ➔Effort important de simulation des composants absents et multiplie le risque d'erreurs lors du remplacement des bouchons.
- ➔La simulation par « couches » n'est pas obligatoire

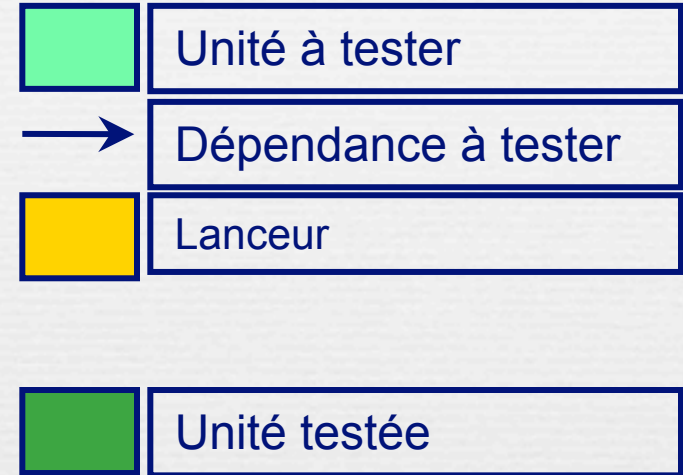
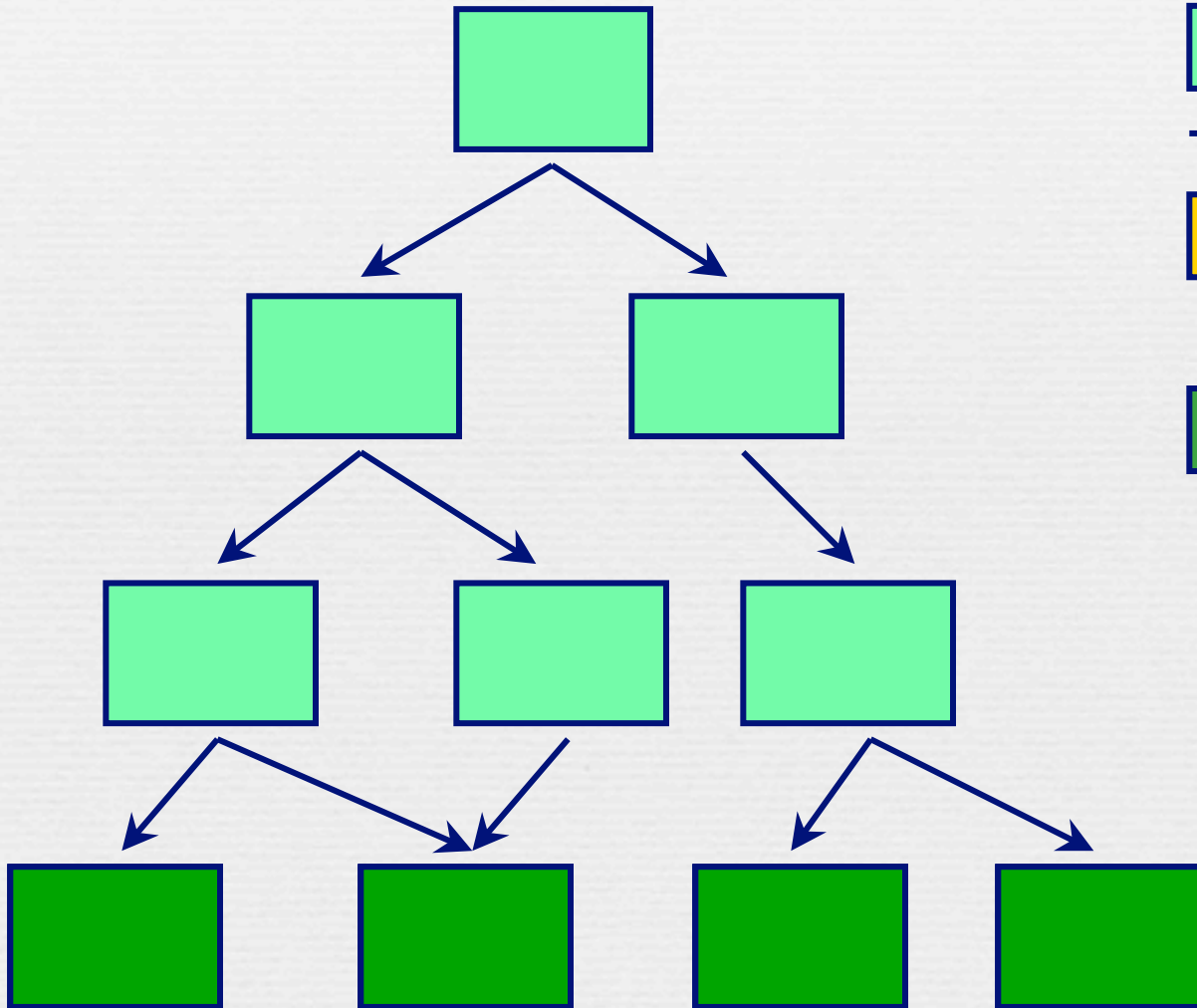
Approche Ascendante



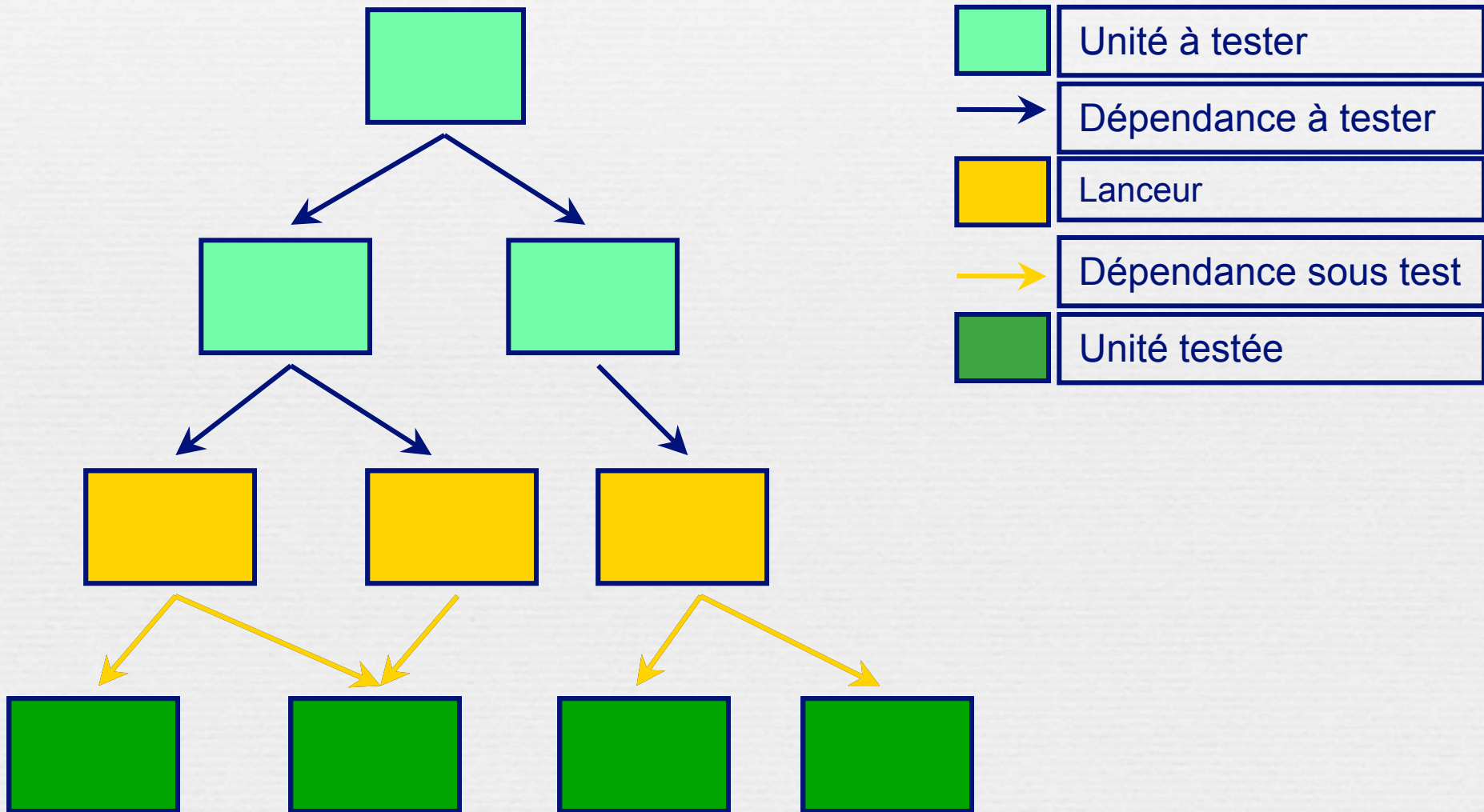
Approche Ascendante



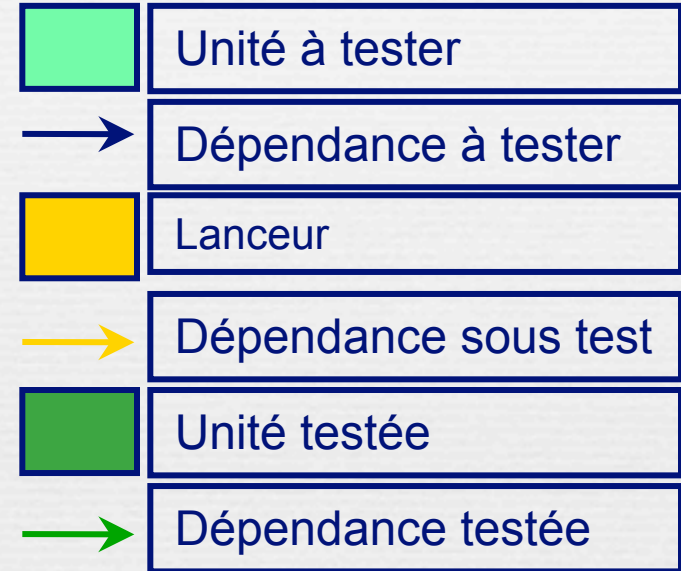
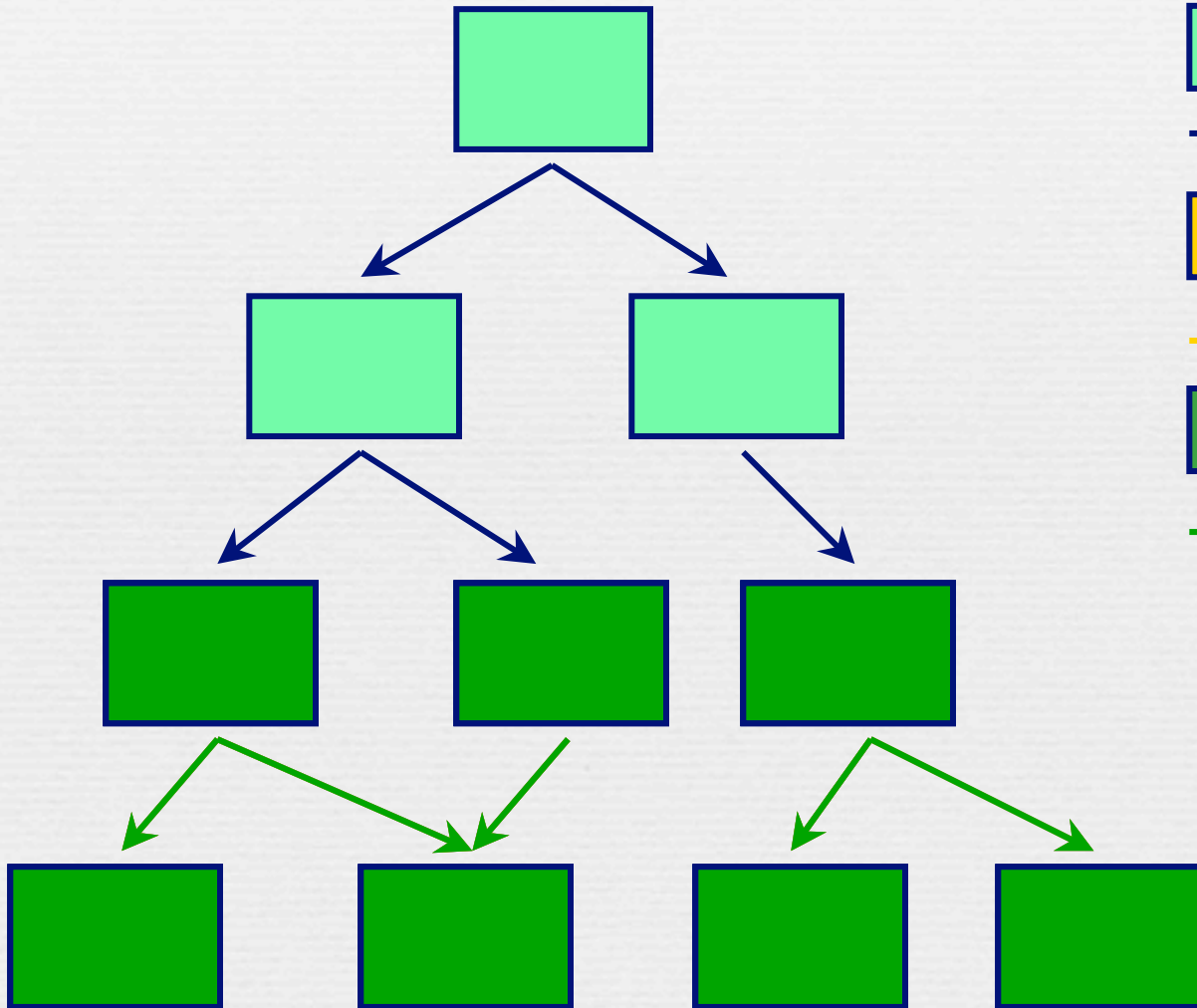
Approche Ascendante



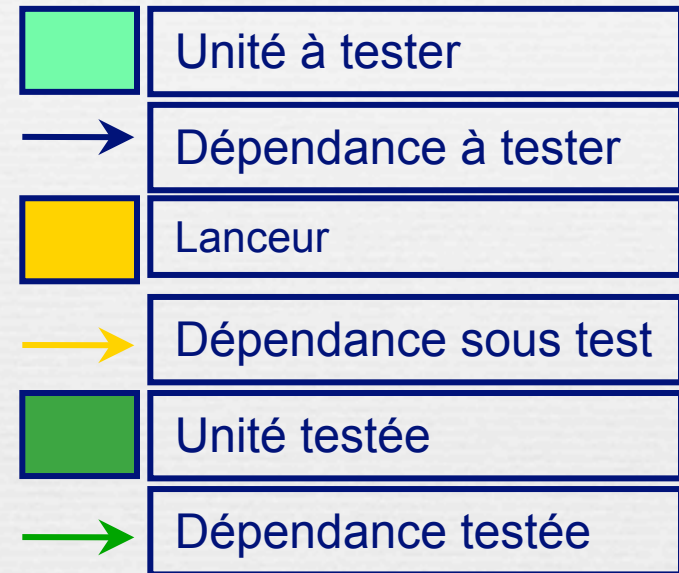
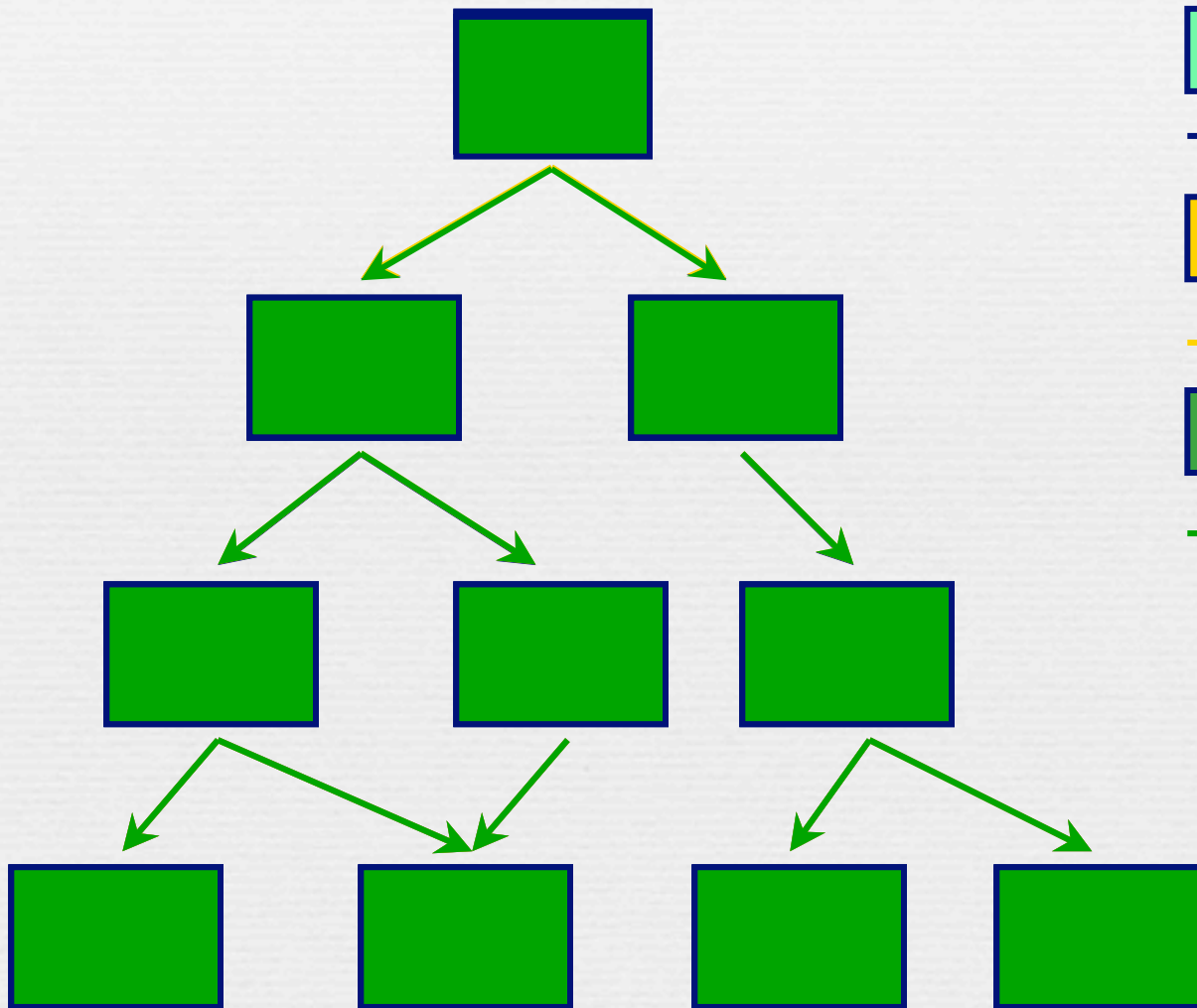
Approche Ascendante



Approche Ascendante



Approche Ascendante



Approche ascendante

→ Avantages

- Faible effort de simulation
- Construction progressive de l'application s'appuie sur les modules réels. Pas de version provisoire du logiciel
- Les composants de bas niveau sont les plus testés,
- Définition des jeux d'essais plus aisée
- Démarche est naturelle.

→ Inconvénients

- Détection tardive des erreurs majeures
- Planification dépendante de la disponibilité des composants

Approche Mixte

→ Combinaison des approches descendante et ascendante.

→ **Avantages :**

- Suivre le planning de développement de sorte que les premiers composants terminés soient intégrés en premier ,
- Prise en compte du risque lié à un composant de sorte que les composants les plus critiques puissent être intégrés en premier.

→ La principale difficulté d'une intégration mixte réside dans sa complexité car il faut alors gérer intelligemment sa stratégie de test afin de concilier les deux modes d'intégration : ascendante et descendante.

Mocks and Stubs

d'après Martin Fowler –

<http://www.martinfowler.com/articles/mocksArentStubs.html>

Légèrement incrémenté par
M. Blay-Fornarino

Example – Electronic Store

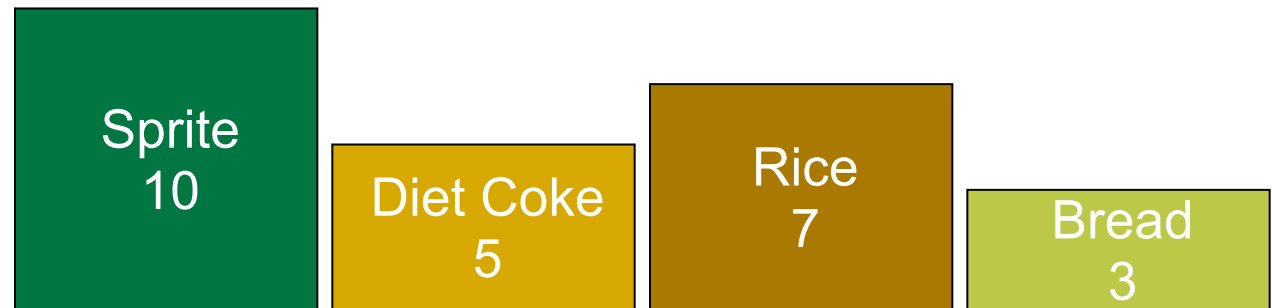
■ Orders and a Warehouse

Order1: Diet Coke - 5

Order2: Diet Coke - 2

Order3: Sprite - 3

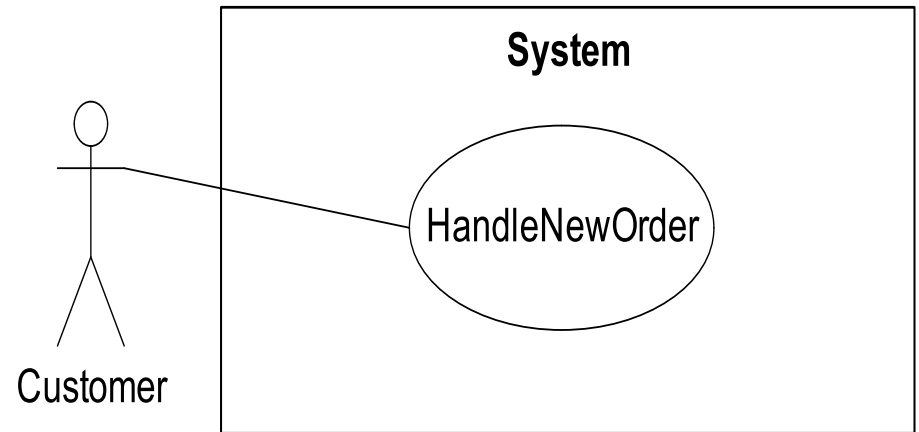
Order4: Bread - 1



28

Example – Electronic Store

■ Use Case Model



■ System Sequence

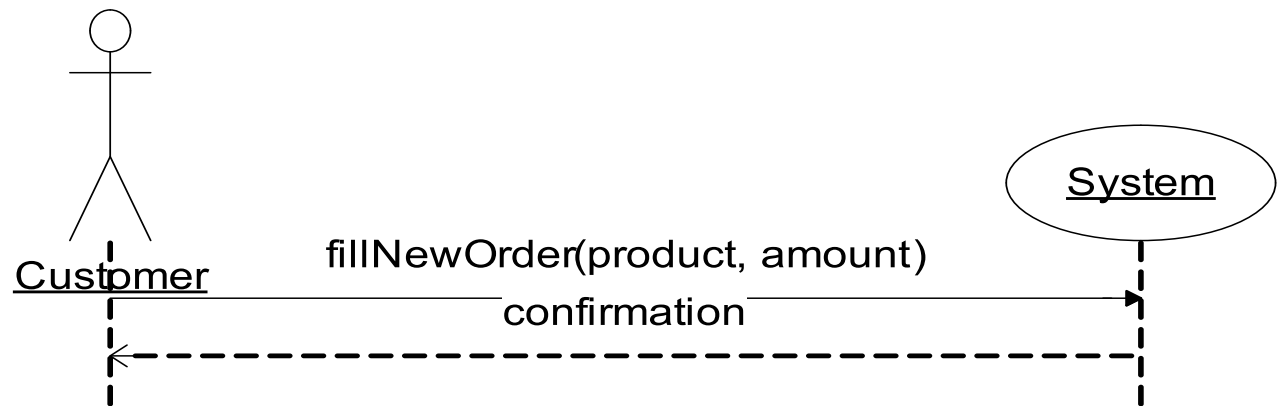


Diagramme de séquence (Conception)

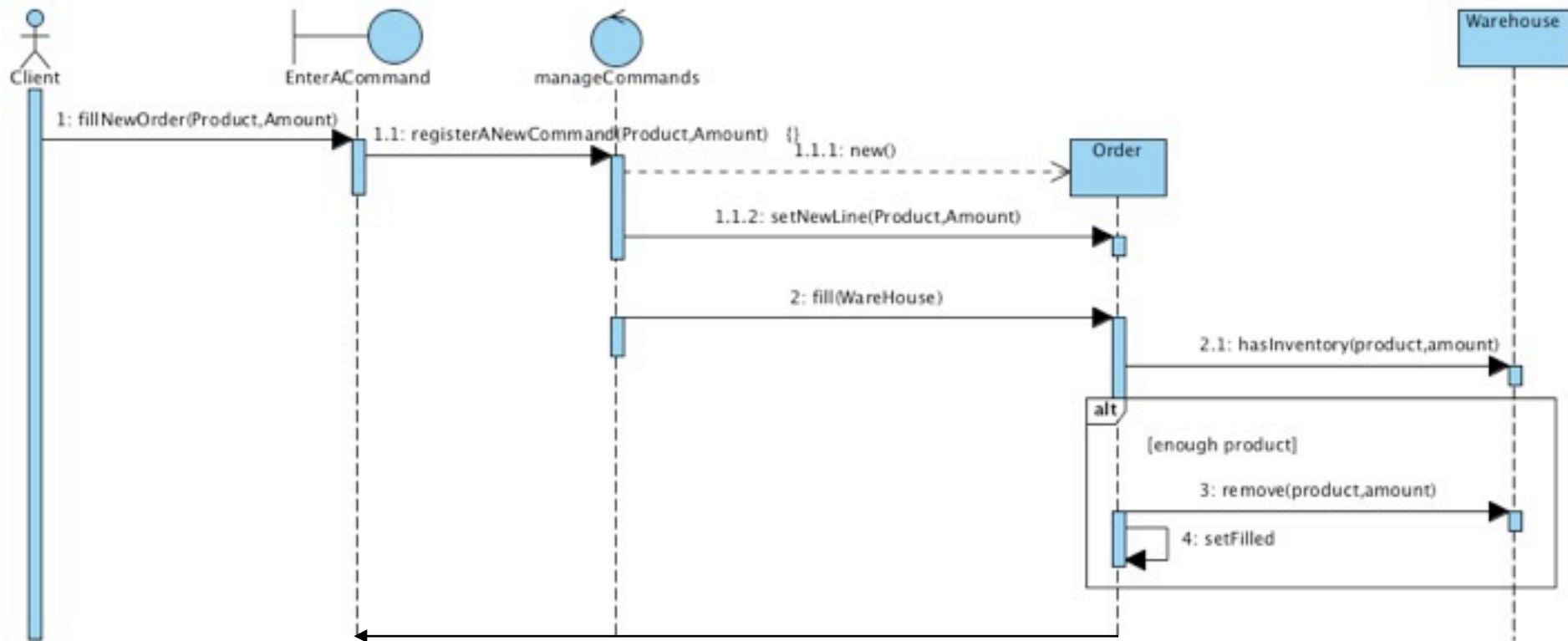
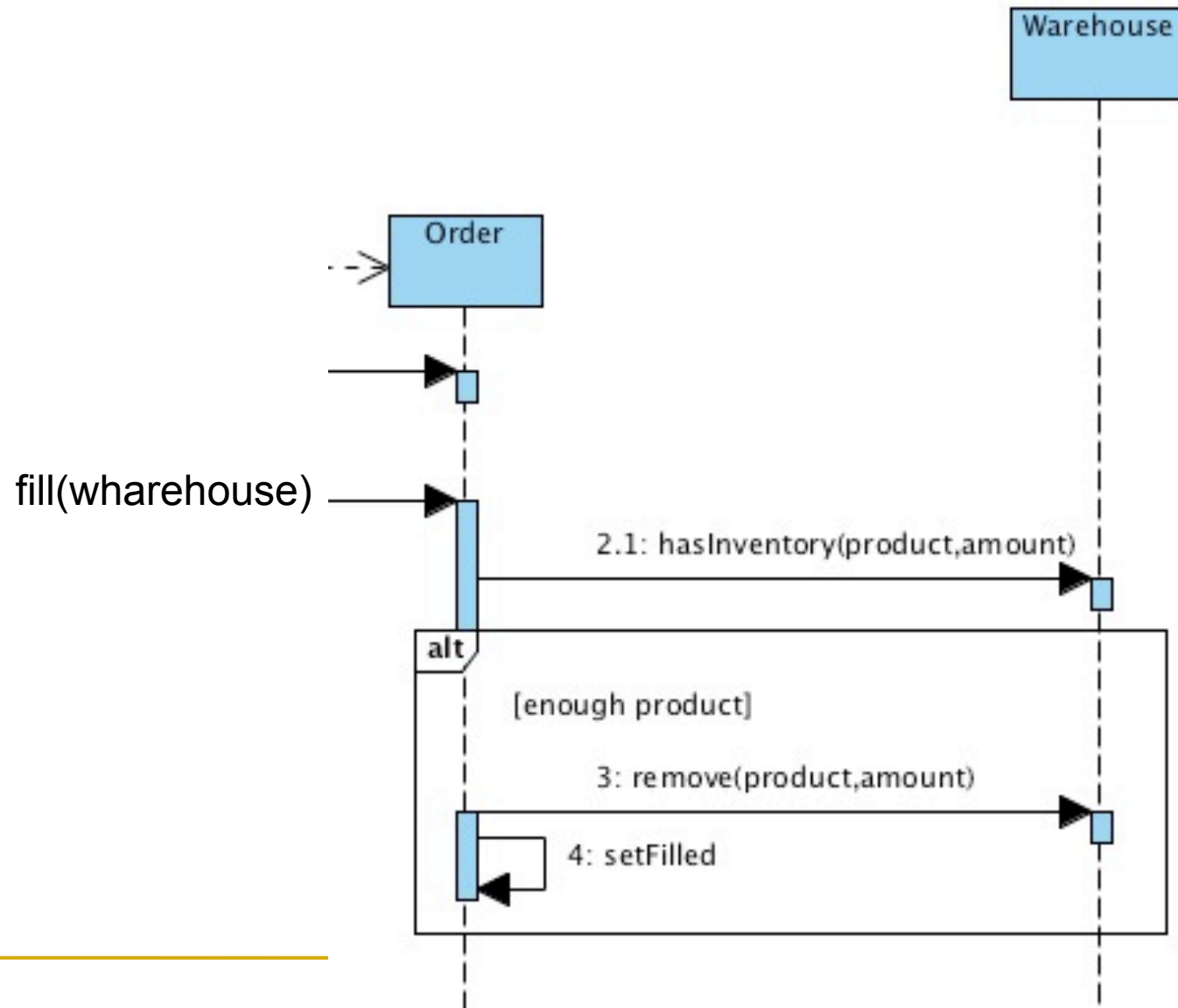
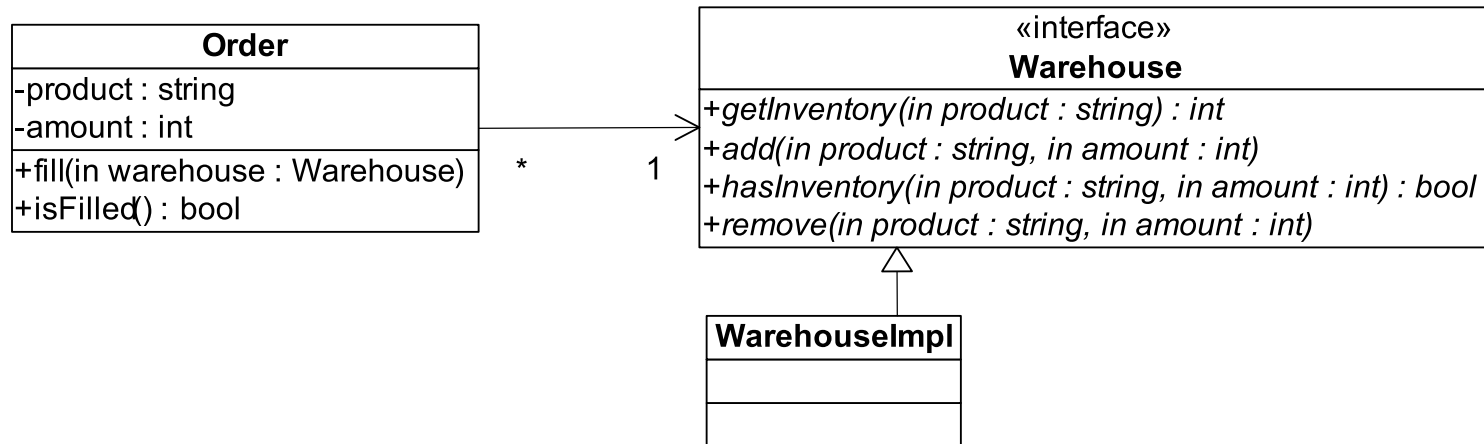


Diagramme de séquence (Conception)

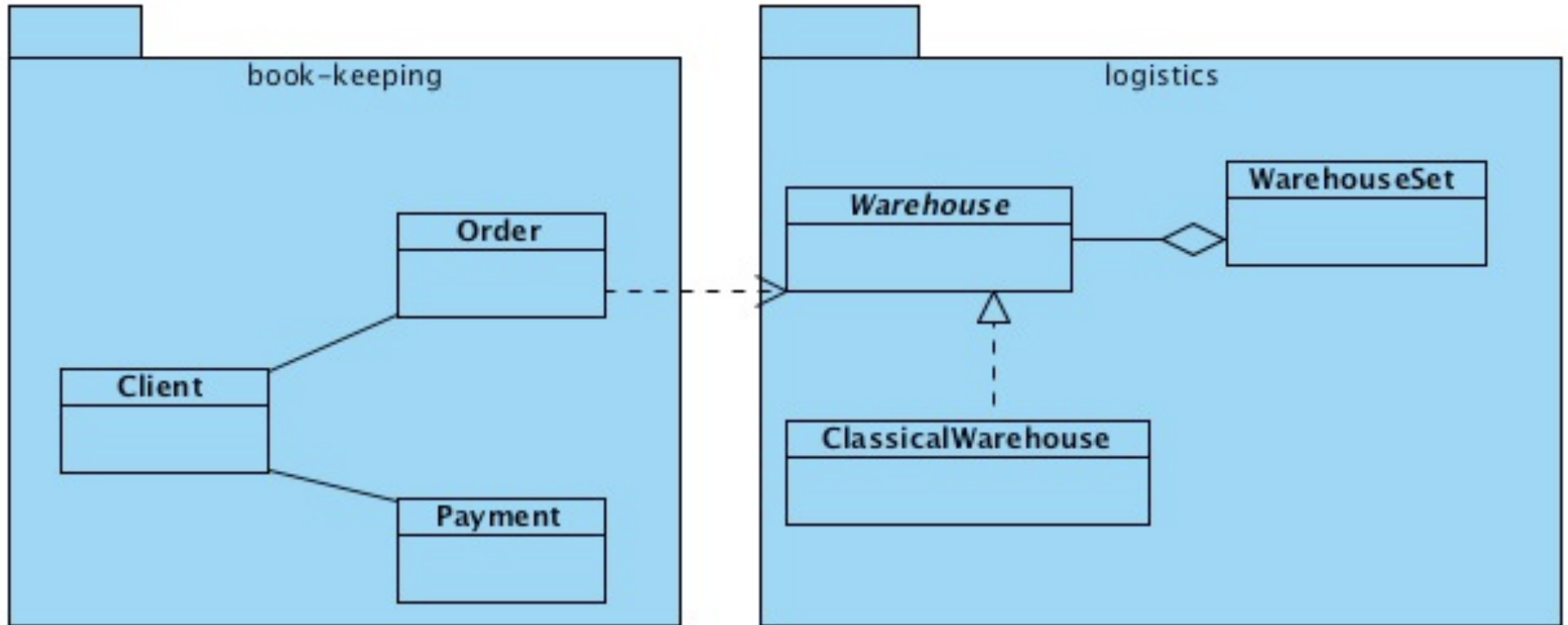


Example – Electronic Store

■ Domain Model



Packages & Séparation des responsabilités



Example – Electronic Store

■ Testing the **Order** class

```
public class Order {
    ...
    public Order(String product, int i) {
        this.product = product;
        this.amount = i;
        this.isFilled = false;
    }

    public void fill(Warehouse warehouse) {
        if (warehouse.hasInventory(product,amount)) {
            warehouse.remove(product,amount);
            isFilled = true;
        }
    }

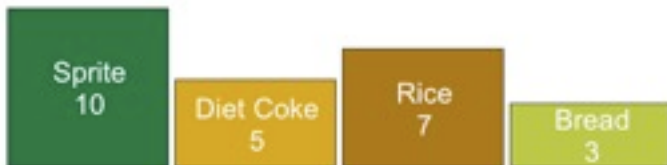
    public boolean isFilled() {
        return isFilled;
    }
}
```

.....

Example – Electronic Store

■ Testing the Order class

```
public class OrderStateTester extends TestCase {  
    ...  
    public void testOrderIsFilledIfEnoughInWarehouse(){  
        Order order = new Order(DIET_COKE,5);  
        order.fill(warehouse);  
        // Primary object test  
        assertTrue(order.isFilled());  
        // Secondary object test(s)  
        assertEquals(0,warehouse.getInventory(DIET_COKE));  
    }  
  
    public void testOrderDoesNotRemovelfNotEnough(){  
        Order order = new Order(SPRITE,11);  
        order.fill(warehouse);  
        // Primary object test  
        assertFalse(order.isFilled());  
        // Secondary object test(s)  
        assertEquals(10, warehouse.getInventory(SPRITE));  
    }  
}
```



Example – Electronic Store

■ Testing the **Order** class:

```
public class OrderStateTester extends TestCase {
    private static String DIET_COKE = "Diet Coke";
    private static String SPRITE = "Sprite";
    Warehouse warehouse;

    protected void setUp() throws Exception {
        //Fixture with secondary object(s)
        warehouse = new WarehouseImpl();
        warehouse.add(DIET_COKE,5);
        warehouse.add(SPRITE,10);
    }
    ...
}
```



Example – Electronic Store

Stub

- Using a *stub* to run the tests -
 - Stubs return canned data to methods calls:

```
public class WarehouseImpl implements Warehouse {  
    public void add(String product, int i) {}  
  
    public int getInventory(String product) {  
        return 0;  
    }  
  
    public boolean hasInventory(String product) {  
        return false;  
    }  
  
    public void remove(String product, int i) {}  
}
```


Java - OrderStateTester.java - Eclipse SDK

File Edit Source Refactor Navigate Search Project Run Window Help

Package Explorer xUnit X 1

Finished after 0.05 seconds

Runs: 2/2 Errors: 0 Failures: 2

Failures Hierarchy

- testOrderIsFilledIfEnoughInWarehouse - unitt
- testOrderDoesNotRemoveIfNotEnough - unitt

Failure Trace

```

junit.framework.AssertionFailedError
at unittests.OrderStateTester.testOrderIsFile
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.NativeMethodAccessorImpl.invo
at sun.reflect.DelegatingMethodAccessorImpl.

```

OrderStateTester.java

```

public class OrderStateTester extends TestCase {
    private static String DIET_COKE = "Diet Coke";
    private static String SPRITE = "Sprite";
    Warehouse warehouse;

    protected void setUp() throws Exception {
        //Fixture with secondary object(s)
        warehouse = new WarehouseImpl();
        warehouse.add(DIET_COKE, 5);
        warehouse.add(SPRITE, 10);
    }

    public void testOrderIsFilledIfEnoughInWarehouse() {
        Order order = new Order(DIET_COKE, 5);
        order.fill(warehouse);
        // Primary object test
        assertTrue(order.isFilled());
        // Secondary object test(s)
        assertEquals(0, warehouse.getInventory(DIET_COKE));
    }

    public void testOrderDoesNotRemoveIfNotEnough() {
        Order order = new Order(SPRITE, 11);
        order.fill(warehouse);
    }
}

```

Outline

- unittests
 - import declarations
 - domainlogic.*
 - junit.framework.TestC
 - OrderStateTester
 - DIET_COKE : String
 - SPRITE : String
 - warehouse : Warehouse
 - setUp()
 - testOrderIsFilledIfEno
 - testOrderDoesNotRem

Problems Javadoc Declaration

0 errors, 0 warnings, 0 infos

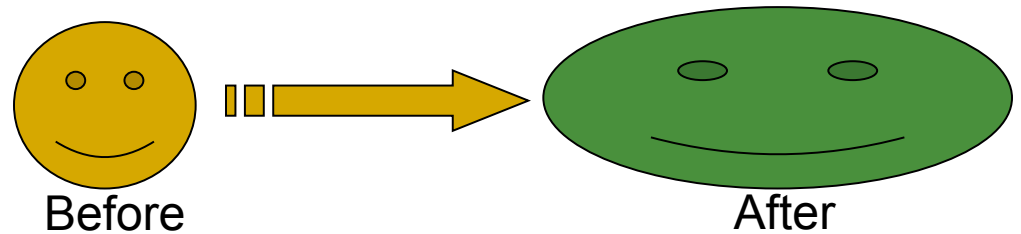
Description	Resource	In Folder	Location

start Java ... Micro... Conn... Mods... EN 99%

Example – Electronic Store

- The tests fail since the stub object – **warehouse** (secondary) misses the required functionality
- Remember: the intension is to test the behavior of the primary object - **Order**, all other objects are tested in their own corresponding tests.
- The test is only *State-Based*
 - E.g., was the *remove()* method invoked? Other methods of the warehouse object?

Example – Electronic Store



- State Based tests:
 - All objects involved must be created – **complex fixture**
 - After the primary object was “kicked” with the behavior that is being tested, the **result** is evaluated against:
 - The primary object
 - All secondary objects
 - If the test fails, its source might be fuzzy between the primary and the secondary objects
 - No interaction is being tested!
- A possible solution – *Mock* objects

Example – Electronic Store

- Tests basés sur les interactions
 - Les tests doivent vérifier quelles méthodes ont été appelées sur les objets secondaires.
 - Tous les objets secondaires sont remplacés par des «mocks»
 - => spécification des interfaces des objets secondaires
 - Test en Isolation: Les Bugs détectés dans les tests sont uniquement liés aux objets primaires
 - Fortement couplés avec la mise en œuvre => peuvent interférer avec la refactorisation

Using EasyMock

- Define only the interface of the Mock object:

```
public interface Warehouse {  
  
    void add(String product, int i);  
    int getInventory(String product);  
    boolean hasInventory(String product,int amount);  
    void remove(String product, int i);  
  
}
```

Using EasyMock

- Create the Mock object:

```
protected void setUp() throws Exception {  
    //Fixture with secondary object(s)  
    mock = createMock(Warehouse.class);  
}
```

You need to :

- Add the EasyMock jar file (easymock.jar) from this directory to your classpath
- *import static org.easymock.EasyMock.*;*

Using EasyMock

```
public void fill(Warehouse warehouse) {  
    if (warehouse.hasInventory(product,amount)) {  
        warehouse.remove(product,amount)  
        isFilled = true;  
    }  
}
```

■ Running tests with expectations:

```
public void testOrderIsFilledIfEnoughInWarehouse(){
```

```
    //Expectations
```

```
    expect(mock.hasInventory(DIET_COKE,5)).andReturn(true);
```

```
    mock.remove(DIET_COKE,5);
```

```
    replay(mock);
```

```
    Order order = new Order(DIET_COKE,5);
```

```
    order.fill(mock);
```

```
    // Primary object test
```

```
    assertTrue(order.isFilled());
```

```
    // Secondary object test(s)
```

```
    verify(mock);
```

```
}
```

Using EasyMock

- Verifying Behavior
 - If the method is not called on the Mock Object

```
public void testDemo(){
    mock.remove("cola",2);
    replay(mock);

    verify(mock);
}
```

```
java.lang.AssertionError:
Expectation failure on verify:
remove("cola", 2): expected: 1, actual: 0
```


Using EasyMock

■ Verifying Behavior

- If the method is not called on the Mock Object

```
public void testDemo(){
    mock.remove("cola",2);
    replay(mock);
    Order order = new Order(SPRITE,11);
    order.fill(mock);
    verify(mock);
}
```

```
java.lang.AssertionError:
Unexpected method call hasInventory("Sprite", 11):
remove("cola", 2): expected: 1, actual: 0
```

"I have not failed, I've just found ten thousand ways that won't work." –
Thomas Edison