Scénarios, tactiques et vues architecturales

Testabilité

[**1 On veut pouvoir effectuer des tests sans affecter le système existant**](#_70vqd8s3j71f) **3**

[1.1 Tactique 1 : Sandbox](#_zihur43y8761) 3

[1.2 Vue architecturale (à l’extérieur de la classe)](#_ienegxu6yjdr) 3

[1.2.1 Description du diagramme](#_1fob9te) 3

[1.2.2 Description des éléments](#_3znysh7) 4

[1.2.3 Relation entre éléments et tactiques](#_2et92p0) 4

[1.2.4 Tactique 1 : Sandbox](#_tyjcwt) 4

[**2 Lorsqu’on change ou ajoute un nouveau composant du système, nous voulons aussitôt savoir si tout est fonctionnel.**](#_k3kzu5h44tzf) **5**

[2.1 Tactique 1 : Specialized interfaces](#_monku4n79ble) 5

[2.2 Tactique 2 : Executable Assertions](#_99b3f2wg2s98) 5

[2.3 Vue architecturale (à l’extérieur de la classe)](#_c30po3jxv2p5) 6

[2.3.1 Diagramme 1.1](#_3dy6vkm) 6

[2.3.2 Diagramme 1.2](#_2s8eyo1) 7

[2.3.3 Explication du diagramme d’état](#_trwotd5e8ysa) 8

[2.3.4 Table de description des éléments du diagramme](#_3rdcrjn) 9

[2.3.5 Texte décrivant la relation entre les éléments et les tactiques](#_26in1rg) 10

[**3 Scénario : Manquant!!**](#_ciu9fn1rdx6d) **11**

[3.1 Tactique 1 : Control and Observe System State](#_y6dfqblvenqj) 11

[3.2 Vue architecturale (à l’extérieur de la classe)](#_88yaszwwwarl) 12

[3.2.1 Diagramme](#_44fb2esf7tgc) 12

[3.2.2 Texte de description du diagramme](#_9z29c8ay82md) 12

[3.2.3 Table de description des éléments du diagramme](#_5pdebk7uzo8a) 13

[3.2.4 Description de la relation entre les éléments et les tactiques](#_tqmtslkdphvk) 13

[**4 Le développeur désire effectuer des tests sur les fonctionnalités de base de l’application du guichet à l’aide de l’interface**](#_lnxbz9) **14**

[4.1 Tactique 1 : nterface spécialisée](#_jp838oarh7pu) 14

[4.2 Vue architecturale (à l’extérieur de la classe)](#_44sinio) 15

[**5 Réduire le temps d’exécution des tests à 10 minutes, lors de mise à jour du système.**](#_3co8sqf3iqr8) **16**

[5.1 Tactique 1 : Abstract data sources](#_ocxkne50nnl5) 16

[5.2 Tactique 2 : Executable assertions](#_748l2c7zq8p) 16

[5.3 Tactique 3 : Limit structural complexity](#_p6hoq3ahlop3) 16

[5.4 Vue architecturale (à l’extérieur de la classe)](#_ii6mmccr8444) 17

[5.4.1 Diagramme](#_z337ya) 17

[5.4.2 Table de description des éléments du diagramme](#_ymg20b5w9izi) 17

[5.4.3 Table de description des relations des tactiques entre ces éléments](#_bebqijzfii8u) 18

[5.4.4 Description des relations entre les éléments](#_i2ffkrdidho1) 18

[**6 Tester les transactions disponibles au client.**](#_wufq9fckgwn0) **19**

[6.1 Tactique 1 : Sandbox](#_e39iyti2uwk6) 19

[6.2 Tactique 2 : Abstract data source](#_r4hda4h8uid1) 19

[6.3 Vue architecturale (à l’extérieur de la classe)](#_w6ioxa40xyfk) 20

[6.3.1 Description](#_wc0l09hafvh) 21

[6.3.2 Relation avec la tactique](#_3f9e2omyiqky) 21

[**7 Faciliter la création de tests plus rapides pendant le développement.**](#_23didt19fffg) **22**

[7.1 Tactique 1 : Sandbox](#_dy1i3d4ofpmq) 22

[7.2 Tactique 2 : Abstract data sources](#_urciqp4kht14) 22

[7.3 Vue architecturale (à l’extérieur de la classe)](#_1z9x60dnxo5j) 23

[7.3.1 Diagramme](#_4rspcobwgtvr) 23

[7.3.2 Légende](#_838j3wtmtgu8) 23

[7.3.3 Table de description des éléments du diagramme](#_4cbjh6kx2fq) 23

[7.3.4 Texte décrivant la relation entre les éléments et les tactiques](#_ws36vlju5w0x) 24

[**8 Tests des transactions de l’ATM**](#_3j2qqm3) **25**

[8.1 Tactique 1 : Record/Playback](#_j8r8myax9woj) 25

[8.2 Tactique 2 : Specialized interfaces](#_fnomdy7f2mzr) 25

[8.3 Diagramme 1.0 Modélisation de la sécurité du système d'ATM.](#_e5garubv0s8w) 26

[8.3.1 Description du diagramme:](#_6dm8g9d05o8k) 26

[8.3.2 Table de description des éléments du diagramme :](#_7nli5pn0m06b) 27

[8.3.3 Relations :](#_wh9v3wbi7iz1) 27

# 1 On veut pouvoir effectuer des tests sans affecter le système existant

| **objectifs d'affaires** | On veut faciliter le développement des tests. |
| --- | --- |
| **Source** | Testeurs d’intégration |
| **Stimulus** | Tests d’intégration |
| **Artéfact** | Système GAB entier |
| **Environnement** | Intégration |
| **Réponse** | État du système |
| **Mesure de la réponse** | 2 heures pour l’écriture de nouveaux test d’intégration |
| **Questions** |  |

## **1.1 Tactique 1 : Sandbox**

**Description**: Un environnement Sandbox est utilisé lors des tests au lieu d’un système bancaire en production, fournissant le même interface de programmation.

**Justification**: Puisqu’il existe une sandbox il est possible de tester les transactions sans utiliser de l'argent réel et sans affecter le système bancaire en production.

## 1.2 Vue architecturale (à l’extérieur de la classe)

### 

### **1.2.1 Description du diagramme**

Le système du GAB utilise la connexion au serveur de production ou la connexion du serveur bac à sable pour accéder au serveur nécessaire à la tâche en cours. Le serveur bancaire bac à sable est utilisé comme base de données de développement. Le GAB en production n’a qu’à changer son adresse IP de call configurée pour avoir accès aux données bidons du bac à sable..

### **1.2.2 Description des éléments**

**GAB** : Représente le système du GAB.

**Connexion - Live.Bank.Server** : Représente la connexion au serveur de production.

**Connexion - Sandbox.Bank.Server** : Représente la connexion au serveur bac a sable.

**Serveur bancaire en production** : Représente le serveur de production.

**Serveur bancaire bac à sable** : Représente le serveur Sandbox.

### **1.2.3 Relation entre éléments et tactiques**

### **1.2.4 Tactique 1 : Sandbox**

Le diagramme démontre bien la tactique Sandbox, représenté par la connexion au serveur bac à sable ainsi et le serveur lui-même. Il montre bien la possibilité de changer le serveur pour ne pas utiliser les données bancaires réelles.

# 2 Lorsqu’on change ou ajoute un nouveau composant du système, nous voulons aussitôt savoir si tout est fonctionnel.

| **Objectifs d'affaires** | Réduire le temps de lancement des tests automatisés du système de guichet. |
| --- | --- |
| **Source** | Le système du guichet |
| **Stimulus** | Une fois le composant installé~~, les tests d’intégrations sont lancés par le guichet.~~ |
| **Artéfact** | Le système au complet |
| **Environnement** | Run Time |
| **Réponse** | D~~étecte qu’un composant a changé le système,~~ on exécute les tests. |
| **Mesure de la réponse** | Temps nécessaire pour exécuter les tests. |
| **Questions** | 1. Combien d’opérations par secondes le processeur peut-il exécuter ? 2. Combien de tests doit-on exécuter ? 3. Que doit-on configurer ou exécuter pour lancer les tests automatisées sur le guichet ? |

## 2.1 Tactique 1 : Specialized interfaces

**Description**: Une interface pour automatiser la testabilité des composants que l’on veut changer. Ces composants peuvent être soient logiciel ou matériels.

**Justification**:Nous aurons donc une seule méthode à tester pour chaques nouveaux composants qui seront ajoutés. De cette manière, nous pourrons réduire le temps de lancement des tests.

## 2.2 Tactique 2 : Executable Assertions

**Description**: Ajout de d’assertion pour vérifier si le système est dans un état de faute.

**Justification**: Dans le cas qui nous concerne, nous allons utiliser l'Exécutable Assertion afin que cette dernière trouve si un composant a été enlevé et si elle a été remplacé par un autre composant.

## 2.3 Vue architecturale (à l’extérieur de la classe)

### 2.3.1 Diagramme 1.1

Untitled Diagram.png

On sépare ici les deux types d’interface puisque ces dernières sont assez différentes et qu’elles regroupent plusieurs sous interfaces sous elles. Les interfaces logicielles seront

### 

### 

### 2.3.2 Diagramme 1.2

### Untitled Diagram.png

Ici dans le diagramme 1.2 nous illustrons comment les interfaces matérielles seront reliées à l’interface logicielle qui portera le nom d’ ”interface matérielle”. Cette interface sera en quelque sorte une manière plus facile de rendre le code testable et par le fait même plus facilement modifiable. Le nombre de tests sera par le fait même diminué car nous aurons toujours les même méthodes à tester, et en théorie, elles devraient donner le même résultat. Le nombre d’interface matérielle sera dans ce cas-ci plus élevé que ce que démontre le diagramme.

**Diagramme d’état**

Diagramme de package.png

### 2.3.3 Explication du diagramme d’état

Tant et aussi longtemps qu’il n’y a pas de changement détecté, on reste dans l’état du guichet. Aussitôt qu’un mouvement est détecté, on notifie le système d’un changement. Le système est donc en attente que le changement se termine, incluant l'installation et les configurations nécessaire pour le fonctionnement du composant. Une fois que tout est terminé, Le système envoie une demande de validation du fonctionnement en exécutant les tests automatisés. Le système affiche ensuite le résultat des tests et revient à l’état initiale du guichet.

### 2.3.4 Table de description des éléments du diagramme

| Élément | Description |
| --- | --- |
| Interface Matérielle | Fonctionnalités pour faire fonctionner le logiciel avec les éléments matériels du guichet. |
| Interface Logicielle | Fonctionnalités qui sont directement dans le software du guichet. Ces fonctionnalités permettent tous les transactions entre le guichet et le serveur central. |
| Guichet | Le guichet contenant tous ses interfaces. |
| Accepteur de carte/Écran/Accepteur d’enveloppe/Clavier/etc. | Ce quelques éléments du guichet qui peuvent être utilisés afin de faire le pont entre l’objet physique et le logiciel du guichet automatique. |
| État guichet | C’est en quelque sort l’état dans lequel se trouve le guichet lorsqu’il est mis en production |
| État Attendre l’installation d’un composant | C'est l’état dans lequel se trouve le guichet lorsqu’un composant est enlevé et qu’il n’y a pas de composant inséré dans le compartiment. |
| État Effectuer les tests automatisés | C’est l’état dans lequel le guichet vérifie l’authenticité du nouveau composant et qu’il vérifie que ce composant retourne les bonne valeurs à un appel quelconque du guichet. |

### 2.3.5 Texte décrivant la relation entre les éléments et les tactiques

Dans un premier temps, nous avons montré un lien entre la tactique des interfaces spécialisées afin que les tests soient toujours les même entre le guichet et ses éléments matériels. En effet, le diagramme 1.1 nous montre l’exemple d’interface entre le guichet et les composants logiciels et les composants matériels du guichet. Dans le diagramme 1.2, on montre l’interface entre les composants matériels et le guichet qui auront aussi des tests afin de pouvoir plus facilement tester l’installation des composants.

De plus, nous avons un diagramme d’état qui énumère les différents états que le guichet peut se retrouver. Ces états sont les étapes qui surviennent lorsqu’on change un composant, ces états sont communiqués au système via une architecture publisher subscriber, et ils sont mis à jour à travers ce système.

# 3 Scénario : Manquant!!

| **objectifs d'affaires** |  |
| --- | --- |
| **Source** | Unit tester |
| **Stimulus** | Exécute les tests unitaires |
| **Artéfact** | Les transactions sur un compte. |
| **Environnement** | Run time |
| **Réponse** | Capture du résultat avec les composants qui utilisent des interfaces qui permettent de logger les évènements. |
| **Mesure de la réponse** | Le temps que cela prend entre le début des tests et la fin des tests. |
| **Questions** |  |

## 3.1 Tactique 1 : Control and Observe System State

**Description**: Implémenter des interfaces pour enregistrer les réponses des transactions.

**Justification**: En implémentant des interfaces pour enregistrer les réponses, on peut catégoriser les réponses correctes et connues et celle qui sont inconnues et critiques, provoquée par une erreur.

## 3.2 Vue architecturale (à l’extérieur de la classe)

### 3.2.1 Diagramme

### 3.2.2 Texte de description du diagramme

Le diagramme est une représentation exhaustive de la tactique #1 pour l’utilisation d’une interface qui permet de logger les réponses des transactions. Lorsqu’une exception non-gérée survient, c’est l’interface UnknownException qui est utilisé pour traiter la réponse, sinon c’est l’interface TransactionLogger qui enregistre toutes les réponses.

### 3.2.3 Table de description des éléments du diagramme

| Transaction | Il s’agit d’une classe abstraite qui représente le concepte des différentes transactions qui pourraient être faites sur l’ATM. |
| --- | --- |
| Depot | Une classe concrète d’une transaction qui peut être faite sur l’ATM. |
| Retrait | Une classe concrète d’une transaction qui peut être faite sur l’ATM. |
| ILogger | Une interface qui fournit les méthodes de base pour enregistrer les réponses. |
| TransactionLogger | Une interface qui hérite de ILogger qui permet d’enregistrer toutes les réponses d’une quelconque transaction. |
| UnknownLogger | Une interface qui hérite de ILogger qui permet d’enregistrer toute réponse non-conforme aux transaction ou provenant d’une exception non-gérée. |

### 3.2.4 Description de la relation entre les éléments et les tactiques

Dans notre cas, nous avons implémenté trois interfaces dont deux spécialisées que nos classes de transactions pourront utiliser pour garder la traces des différentes réponses faites pendant une transaction et permettre de détecter les possibles erreurs sans faire planter l’ATM.

# 4 Le développeur désire effectuer des tests sur les fonctionnalités de base de l’application du guichet à l’aide de l’interface

| **Objectifs d'affaires** | Permettre aux développeurs de tester les fonctionnalités de base à l’aide de test automatisés lancés lors de la compilation. |
| --- | --- |
| **Source** | Le développeur lançant les tests |
| **Stimulus** | Exécution des tests à partir de l’interface spécialisée |
| **Artéfact** | Les fonctionnalités de base du GAB |
| **Environnement** | Exécution, environnement de test |
| **Réponse** | Chacun des test se termine sans erreur. |
| **Mesure de la réponse** | Les tests s’effectuent chacun en moins de 30 secondes, avec la batterie se terminant en moins de 2 minutes. Les résultats des tests sont visibles et des messages d’erreurs sont disponibles pour les tests ayant échoués. |
| **Questions** | 1. Est-ce que les modifications effectuées par les développeurs peuvent changer drastiquement le comportement des fonctionnalités? 2. Est-ce que les développeurs ont l’habitude de compiler le code fréquemment? 3. Quel est le temps acceptable pour que les tests soient effectués sur tout le système? 4. Quel est le temps acceptable pour que les tests soient effectués sur les fonctionnalités de base uniquement? 5. Y-a-t’il d’autres mesures à entreprendre lorsqu’un test échoue ou est-ce que la simple notification du développeur suffit? |

## 4.1 Tactique 1 : Interface spécialisée

**Description**: Interface spécialisée

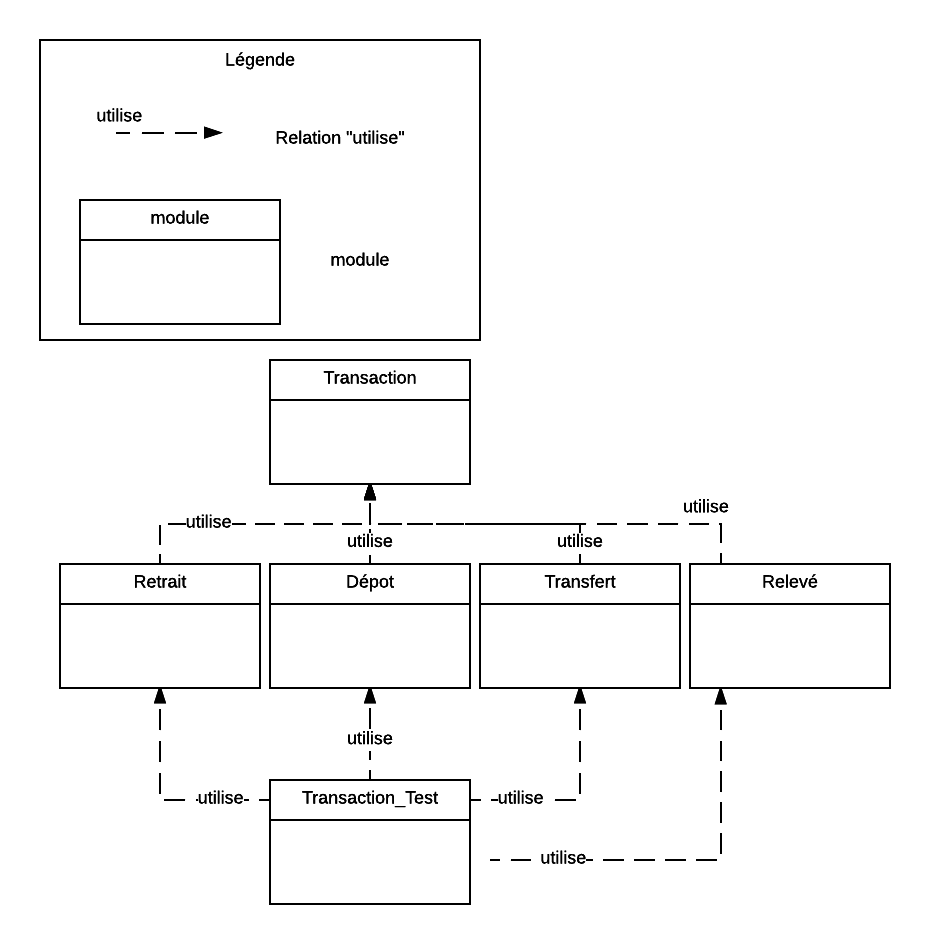
**Justification**: Afin d’augmenter la testabilité du système en entier, une interface spécialisée conçue pour tester individuellement les fonctionnalités ou encore de lancer une batterie de test incluant tous les modules de l’application. Cela présente une solution de test automatique et les rends facilement utilisable et fournit des réponses lisibles pour faciliter la détection d’erreur.

## 

## 

## 4.2 Vue architecturale (à l’extérieur de la classe)

### 



Cette vue démontre le module de test de transaction (Transaction\_Test) ainsi que son utilisation de chacune des transactions pour pouvoir effectuer les tests nécessaires sur chacune d’entre elles, à travers l’interface prévu à cet effet.

# 5 Réduire le temps d’exécution des tests à 10 minutes, lors de mise à jour du système.

| **Objectifs d'affaires** | Réduire le temps d’exécution des tests à 10 minutes, lors de mise à jour du système. |
| --- | --- |
| **Source** | Unit test |
| **Stimulus** | Mise à jour. |
| **Artéfact** | Un log des tests effectués. |
| **Environnement** | Compile time |
| **Réponse** | Tester les fonctionnalités du guichet.  Vérifier le besoin d’une mise à jour. |
| **Mesure de la réponse** | Temps de réalisation des tests.  Coûts de réalisation des tests.  État de la mise à jour. (Success, a refaire ,etc.) |
| **Questions** | 1. Quel est la tactique pour améliorer le temps d’exécution des tests ? 2. Quel est la limite des ressources physiques ? 3. Quel est le coût de réalisation des tests ? |

## **5.1 Tactique 1 : Abstract data sources**

**Description**: Cette tactique permet d’utiliser des valeurs contrôlées et de les fournir à notre système pour faire les tests. Des abstractions de l’interface sont utilisées pour avoir un meilleur contrôle sur les tests, car la création d'entrées n’est pas nécessaire et utile lors des processus de tests à répétition.

**Justification**: Cette tactique s’applique bien, car il faut tester des opérations précises peu importe leurs valeurs, alors en contrôlant les valeurs lors des tests, il est possible de savoir la valeur de sortie et de la comparer avec les résultats obtenus. Par ailleurs, il est possible d’avoir une batterie de test et de valeurs préenregistrées (base de données, fichier, etc.) qui seront utilisées pour chacun des tests des différents systèmes.

## **5.2 Tactique 2 : Executable assertions**

**Description:** Lors de la programmation du système, des tests avec des ‘assert’ sont disposés à des endroits justifiés et optimaux.

**Justification:** Lors des tests, il sera possible de vérifier le résultat de certaines opérations à des endroits précis et d’éviter de passer par l’interface pour faire les tests. Aussi, ce type de test permet de faire une vérification plus rapide, car on sait quelles opérations ont échoué sans avoir à creuser dans le code testé.

## **5.3 Tactique 3 : Limit structural complexity**

**Description:** Cette tactique va permettre de réduire des dépendances cycliques entre un ou plusieurs composants de tests. Les composants seront séparés en différents composants plus indépendants.

**Justification:** Pendant la mise à jour, on pourrait exécuter seulement des composants de tests reliés à la mise à jour sans avoir à exécuter tous les autres tests, ceci est dû à la réduction du couplage entre les classes de tests.

## **5.4 Vue architecturale (à l’extérieur de la classe)**

### **5.4.1 Diagramme**

Testabilite(2).png

**Figure 1.0 :** Diagramme du scénario de testabilité

Le diagramme illustré dans la figure 1.0 consiste à faire une abstraction des classes pour réduire le couplage et augmenter la cohésion de celles-ci.

### 5.4.2 Table de description des éléments du diagramme

| **Élément** | **Responsabilité** |
| --- | --- |
| AbstractTest | Classes abstraites |
| TestClass01 | Classe concrète avec des fonctionnalités spécialisés |
| TestClass02 | Même chose que TestClass01 |
| AbstractFactoryTest | Fournir des fonctionnalité abstraite |
| ConcreteFactoryTest | Implémenter des fonctionnalités communes pour toutes les classes de tests. |

### 5.4.3 Table de description des relations des tactiques entre ces éléments

| **Élément** | **Description** |
| --- | --- |
| Abstract Data Sources | Abstraction des interfaces de tests |
| Executable Assertions | Vérification de la validité des tests pour chaque fonctionnalité. |
| Limit structural complexity | Séparation des classes de tests pour réduire les dépendances cycliques. |

### 5.4.4 Description des relations entre les éléments

En utilisant la tactique Abstract Data Sources, nous introduisons une abstraction dans le processus de test afin d’améliorer la rapidité d'exécution. Pour l’améliorer, nous utilisons des valeurs prédéfinies et qui sont contrôlées pour pouvoir faire une comparaison avec les valeurs attendues et même celles qui sont dans nos sources de données. De plus, nous ajoutons la tactique Executable Assertions afin de faire la validation et vérifier des opérations critiques à des endroits précis. Finalement, la tactique Limit structural complexity va nous permettre de réduire les dépendances cycliques entre nos classes de tests, en limitant par exemple le polymorphisme et les multiples héritages cela va réduire le couplage et augmenter la cohésion.

# 6 Tester les transactions disponibles au client.

| **Objectifs d’affaires** | Être capable de tester et développer le code source du ATM sans affecter les données clients. |
| --- | --- |
| **Source** | Développeur en assurance de qualité. |
| **Stimulus** | * Des tests aux niveaux de services transactions |
| **Artéfact** | * Des tests unitaires |
| **Environnement** | * Mode développement |
| **Réponse** | * ~~Observer~~ les résultats extrants |
| **Mesure de la réponse** | * Le temps de mesures pour trouver l’erreur * La complexité de l’algorithme pour obtenir l’extrant |
| **Questions** | * Quels sont les services offerts par le système central de la banque en mode normal ? * Le sandbox est-il un miroir parfait du système central de la banque ? * Est-ce que les extrants sont la même chose qu’en mode production et développement ? |

## 6.1 Tactique 1 : Sandbox

**Description**: Une simulation qui remplace l’environnement réel de transactions pour effectuer des tests sans affecter les comptes usagers de la banque.

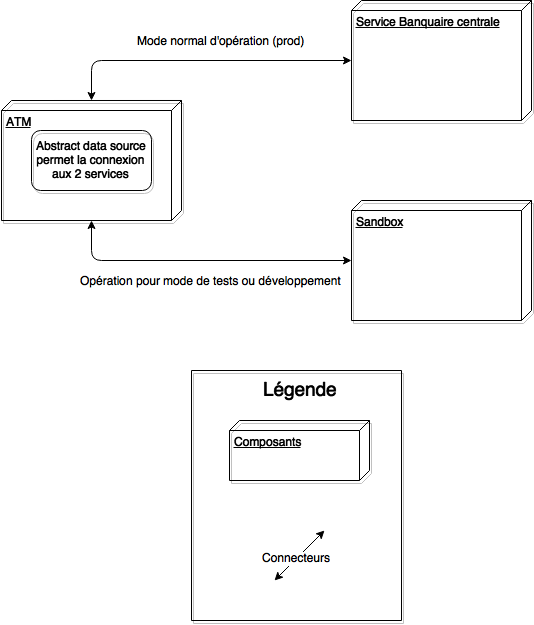
**Justification**: Comme l’environnement comporte des données critiques (argent des clients), il est important d’avoir un environnement de test qui simule exactement toutes les interactions avec le système central de la banque sans vraiment affecter de données critiques.

## 6.2 Tactique 2 : Abstract data source

**Description**: Aiguille le ATM vers quel set de données se connecter.

**Justification :** Comme le ATM devra se connecter à différentes sources de données selon le mode d’exécution, cette tactique est appropriée pour s’assurer de la bonne connexion dans tous les cas.

## **6.3 Vue architecturale (à l’extérieur de la classe)**



### 6.3.1 Description

Lors de la mode, test ou développement, dans une « Sandbox », va être utilisé des tests sans affecter les comptes usagers de la banque. Ainsi, on a des différents connecteurs pour connecter aux sources selon les besoins.

### **6.3.2 Relation avec la tactique**

*Sandbox:*

Il servira pour le mode de test et développement sans affecter les comptes usagers de la banque qui est dans le mode production.

*Abstract Data Source :*

*Il permet de connecter à différentes serveuses selon la connexion qui s’est requis.*

**Composantes :**

ATM : Il contient « Abstract Data Source » afin de différencier les deux connecteurs

Sandbox : Serveur de test et développement

Service bancaire Centrale : Serveur pour mode production

# 7 Faciliter la création de tests plus rapides pendant le développement.

| **objectifs d'affaires** | En tant que développeur de services banquiers, je veux améliorer la rapidité de création et d’exécution des tests , pour assurer que le produit est fiable pour les clients. |
| --- | --- |
| **Source** | Testeurs de système |
| **Stimulus** | Avoir un système complètement couvert par des tests. |
| **Artéfact** | Projet de tests unitaires et d'intégration |
| **Environnement** | ~~Exécution des tests lors du~~ développement |
| **Réponse** | L'exécution des test |
| **Mesure de la réponse** | * Les tests réussis et échoués * Le coverage de nos tests   Niveau de connectivité des classes |
| **Questions** | 1. Quel sera le coverage de nos tests. 2. Quel sera le temps moyen pour un développeur créer un nouveau test. |
|  |  |

## 7.1 Tactique 1 : Sandbox

**Description**: Créer un environnement indépendant pour l’exécution des tests, possiblement une copie de la production. Cela permet d’avoir un environnement identique entre chacun des tests, ressemblant le plus possible à l’environnement réel de production~~, mais toutefois avoir d’impact hors de l’environnement de test.~~

**Justification**: Les développeurs pourront exécuter les tests du guichet sur un environnement dédié identique à celui de production. Il est cependant critique que les tests des développeurs n’aient aucun impact sur l’environnement de production.

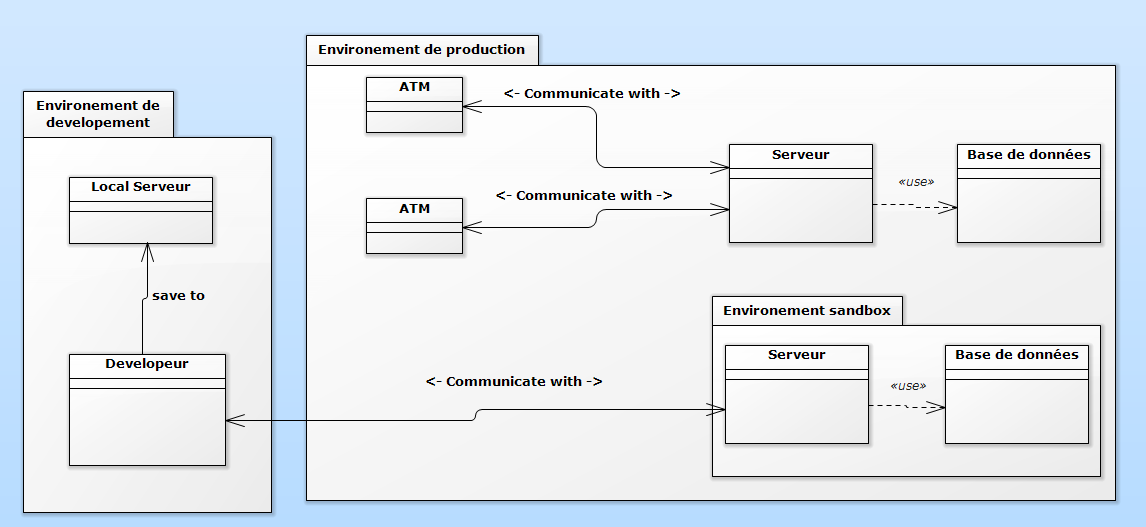
## 7.2 Tactique 2 : Abstract data sources

**Description**: Définir une configuration différente pour l’application selon l’environnement où elle s’exécute (Dev, QA, Prod, ...)

**Justification**: Permet au guichet de se connecter au sandbox lors des tests, et de faire parvenir les logs à des endroits différents lors des tests ou en production.

## 7.3 Vue architecturale (à l’extérieur de la classe)

### 7.3.1 Diagramme



### 7.3.2 Légende

* + Les liens représentent les communications réseaux entre les éléments physiques.

Texte de description du diagramme

### 7.3.3 Table de description des éléments du diagramme

* + Environnement de développement représente la machine du développeur.
  + Environnement de production représente ce qui ne peut pas être touché par les équipes de développement.d

### 7.3.4 Texte décrivant la relation entre les éléments et les tactiques

~~N’oubliez pas que vous pouvez réutiliser des éléments des vues architecturales précédentes pour réaliser la vue courante. Toutes les tactiques peuvent être combinées dans une ou plusieurs vues selon les besoins.~~

# 8 Tests des transactions de l’ATM

| **objectifs d'affaires** | Tester toutes les transactions possibles sur l’ATM |
| --- | --- |
| **Source** | Testeur Système |
| **Stimulus** | Système complet (module de transactions) |
| **Artéfact** | Transaction (modules de transactions) |
| **Environnement** | Développement |
| **Réponse** | 1. Résultat des tests |
| **Mesure de la réponse** | Moins de temps pour programmer les tests  85% de couverture en 3 heures |
| **Questions** | 1. Combien de temps est nécessaire pour l’implémentation d’un test ? 2. Quels sont les types de transactions à tester ? 3. Combien de temps a-t-on pour faire les tests? |

## 8.1 Tactique 1 : Record/Playback

**Description**: Permets de contrôler et récupérer les valeurs de variables des composants du système ainsi que l’état du système qui a causé une faute. Elle nous permet d’enregistrer l’état d’un système afin de pouvoir la rejouer et trouver la cause de la faute. Cela nous permet d’obtenir de l’information sur notre système.

**Justification**: Cette tactique nous permet d’obtenir les méthodes get et set, reset et report, qui permet de retourner l’état complet des objets du système au moment de l’exécution de tests. Elle nous permet donc d’enregistrer l’état d’un système afin de pouvoir la rejouer et trouver la cause des fautes. Cela nous permet d’obtenir de l’information sur notre système. Cela est particulièrement utile quand l’on veut comprendre pourquoi une transaction ne s’est pas déroulée comme prévu.

## 8.2 Tactique 2 : Specialized interfaces

**Description**: La méthode consiste à créer des interfaces de tests spécialisés. Cela nous permet de contrôler et de capturer les variables pour un composant à travers l'exécution normale du test ou d’un cas de tests.

**Justification**: Cette tactique nous permet d’obtenir les méthodes get et set, reset et report, qui permet de retourner l’état complet des objets du système. Il nous est donc possible de faire des tests avec ces interfaces afin de tester le comportement de notre application. Il nous est donc possible de voir si notre système se comporte correctement. À l’aide de cette méthode, on veut détecter un comportement anormal et essayer d’en trouver la cause.

## 8.3 Diagramme 1.0 Modélisation de la sécurité du système d'ATM.

### 

Untitled Diagram.png

Serveur

### 8.3.1 Description du diagramme:

Deux composants du système, l’ATM et le serveur de la Banque ont des relations pour tester le système. Les deux systèmes respecteront la tactique de spécialisation d’interfaces pour les tests. De plus, le système respectera la tactique de Record/Playback pour effectuer les tests sans perte d’information.

### 8.3.2 Table de description des éléments du diagramme :

| **Tactiques** | **Descriptions** |
| --- | --- |
| Specialized interfaces | Permet contrôler et récupérer les valeurs de variables des composants du système à l’aide de diverses classes d’interfaces. |
| Record/Playback | Permets de revenir vers un état initial fonctionnel.Cela permet d’analyser une faute et d’en trouver la cause. |

### 8.3.3 Relations :

Les lignes de l'ATM vers le serveur représentent la tactique de spécialisation d’interfaces et de Record/Playback. Comme mentionné précédemment, la méthode de record/playback nous permet d’enregistrer l’état d’un système afin de pouvoir la rejouer et trouver la cause de la faute. Cela nous permet d’obtenir de l’information sur notre système. De plus, la méthode specialized interfaces nous permet de créer des interfaces de tests nous permettant de récupérer des valeurs de notre système afin de faciliter les tests. En combinant le Record/Playback avec cette technique, il est donc possible d’obtenir encore plus d’information sur l’état d’un système lors d’une transaction. On pourra ainsi obtenir de l’information sur l'état d’une transaction et pouvoir l’étudier à l’aide de la technique Record/Playback.