**LOG430 – Architecture logicielle**

Scénarios de qualité, tactiques et justifications des attributs produits par les équipes en classe

[Proposition 1](#_3znysh7)

[Testabilité – Justification](#_7t2w0yckz352)

[Avant le déploiement d’un guichet, les testeurs d’acceptation exécutent une batterie de tests sur le guichet dans le but de vérifier le fonctionnement de ses périphériques d’entrée.](#_1t3h5sf)

[Suite au développement du module de traitement des transactions d’un guichet, les testeurs lancent une batterie de tests unitaires afin de valider le fonctionnement du module.](#_2s8eyo1)

[Testabilité – Tactiques pour le scénario 1](#_3rdcrjn)

[Testabilité – Tactiques pour le scénario 2](#_26in1rg)

[Testabilité – Vue architecturale](#_1ksv4uv)

[Diagramme](#_44sinio)

[Description du diagramme:](#_2jxsxqh)

[Description des éléments](#_z337ya)

[Relations entre éléments et tactiques](#_3j2qqm3)

[Relations entre tactiques et scénarios de qualité](#_1y810tw)

[Proposition 2](#_4i7ojhp)

[Implémentation d’une logique d’affaires efficacement testable.](#_1ci93xb)

[Scénario 2](#_3as4poj)

[Détecter lorsque des modifications au code provoquent des régressions](#_1pxezwc)

[Tactique - Scénario 1](#_49x2ik5)

[Tactique - Scénario 2](#_2p2csry)

[Vue architecturale](#_147n2zr)

[Description des éléments du diagramme](#_u829zs9xqx1a)

[Proposition 3](#_3o7alnk)

[Testabilité – Scénario 1](#_23ckvvd)

[Ajout d’un nouveau module](#_ihv636)

[Testabilité – Scénario 2](#_epf3fmq6bu3o)

[Modification d’un module existant](#_32hioqz)

[Testabilité – Tactique 1](#_odekqvf9uyrg)

[Testabilité – Tactique 2](#_ym7ujf6lolw9)

[Testabilité – Vue Architecturale](#_kk2i0vf4tbsg)

[Description du diagramme](#_41mghml)

[Table de description des éléments des diagramme](#_2grqrue)

[Proposition 4](#_vx1227)

[<Testabilité> – Justification](#_3fwokq0)

[<Executable assertions> – Scénario 1](#_1v1yuxt)

[S’assurer que le tous les opérations nécessaires à un processus sont exécutés](#_4f1mdlm)

[<Sandbox> – Scénario 2](#_3tbugp1)

[S’assurer du fonctionnement complet des cas d’utilisations principaux](#_28h4qwu)

[<Executable assertions> – Tactique 1](#_nmf14n)

[<Sandbox> – Vue Architecturale](#_46r0co2)

[Proposition 5](#_2lwamvv)

[Testabilité – Justification](#_111kx3o)

[Testabilité – Scénario 1](#_bjdau4s8c8a6)

[Un nouveau type de billet est disponible dans l’ATM](#_3l18frh)

[Testabilité – Scénario 2](#_r7mnvquipuhf)

[Validation du fonctionnement d’un dépôt bancaire ATM-serveur suite à une modification du chiffrement.](#_206ipza)

[Testabilité – Limit structural complexity – Scénario 1](#_c4gty7njkt1)

[Testabilité – Sandbox – Scénario 2](#_nwji1okn8mwu)

[Testabilité – Vue architecturale – Scénario 1](#_hud80lpzdvqp)

[Diagramme et légende](#_2zbgiuw)

[Description du diagramme et des relations](#_1egqt2p)

[Description des éléments du diagramme](#_nny6to6n3ild)

[Tactiques répond aux scénarios](#_2dlolyb)

[Testabilité – Vue architecturale – Scénario 2](#_sqyw64)

[Description du diagramme et des relations](#_1rvwp1q)

[Description des éléments du diagramme](#_4bvk7pj)

[Tactiques répond aux scénarios](#_2r0uhxc)

[Proposition 6](#_1664s55)

[Testabilité – Justification (en classe)](#_neliprpkaa71)

[Testabilité – Scénario 1 (en classe)](#_b4f57hlxyvpy)

[Changement à l’api du serveur](#_3q5sasy)

[Testabilité – Scénario 2 (en classe)](#_nuq1he4mjgn8)

[Exécution de test de régression sur l’ensemble du code en intégration continue.](#_25b2l0r)

[Testabilité – Tactique 1 (en classe)](#_836lwcm8tzfs)

[Testabilité – Tactique 2 (en classe)](#_7xjiz65iowo)

[Testabilité – Vue Architecturale (à l’extérieur de la classe)](#_kgcv8k)

[Tableau des éléments](#_34g0dwd)

[Description de la relation entre les éléments et les tactiques](#_1jlao46)

[Proposition 7](#_43ky6rz)

[Testabilité – Justification (en classe)](#_fb6eo5yztk3d)

[Testabilité – Scénario 1 (en classe)](#_v2qyx5h7hqqj)

[Vérification qu’une transaction s’effectue correctement](#_2iq8gzs)

[Testabilité – Scénario 2 (en classe)](#_red94pqbfkfx)

[Vérifier que l’authentification s’effectue correctement](#_xvir7l)

[Testabilité– Scénario 1 : Sandbox](#_j6es46r8iwb)

[Testabilité– Scénario 2 : Sandbox](#_3hv69ve)

[Testabilité – Vue Architecturale](#_2ldtfawn4z4v)

[Proposition 8](#_1x0gk37)

[Scénario I – Tester l’authentification d’un usager par sa carte bancaire](#_hw3vnmt1ldj7)

[Tester l’authentification d’un usager avec sa carte bancaire](#_4h042r0)

[Scénario II – Tester la validité d’une transaction (dépôt)](#_jn8o6jcinxuz)

[Tester la validité d’une transaction (dépôt)](#_2w5ecyt)

[Tactique - Scénario I](#_fpey08vgua3)

[Tactique - Scénario II](#_qg0czxkv9az2)

[Vue Architecturale](#_8yhtypdzcoqm)

# Proposition 1

## Testabilité – Justification

Les règles d’affaires qui régissent un guichet automatique sont susceptibles de changer pour s’adapter aux nouveaux processus du domaine bancaire. L’enjeu de modificabilité a été géré précédemment. Toutefois, afin de valider ces modifications constantes et d’acquérir un certain niveau de confiance dès la première itération du projet la testabilité devient un enjeu important pour les intervenants:

Pour les développeurs et les mainteneurs, il s’agit d’optimiser la détection de bogues et, à terme, d’améliorer la qualité du code produit. Une bonne testabilité facilite grandement l’écriture de tests unitaires et d’intégration automatisés, ce qui facilite d’autant plus le contrôle de qualité et la régression.

Pour les clients et l’administration des banques, les avantages découlent du travail des développeurs et des mainteneurs: l’application qu’il utilisent/distribuent sera de meilleure qualité, aura moins de bogues ce qui générera moins d’appel de support.

# 

# 

### 

| Scénario | Avant le déploiement d’un guichet, les testeurs d’acceptation exécutent une batterie de tests sur le guichet dans le but de vérifier le fonctionnement de ses périphériques d’entrée. |
| --- | --- |
| Objectifs d'affaires | Vérifier et valider le fonctionnement des périphériques du guichet |
| Attribut de qualité | Testabilité |
| Source | Testeurs d’acceptation |
| Stimulus | Batterie de tests effectués sur les périphériques d’entrée du guichet automatique |
| Artefact | Périphériques d’entrée du guichet;  Code embarqué de ces périphériques |
| Environnement | Période de déploiement |
| Réponse | Les tests sont exécutés et les résultats sont enregistrés. |
| Mesure de la réponse | Probabilité qu’une faute soit détectée lors de la prochaine période de tests;  Temps nécessaire pour effectuer les tests;  Temps nécessaire pour préparer l’environnement de tests |
| Questions |  |

### 

| Scénario | Suite au développement du module de traitement des transactions d’un guichet, les testeurs lancent une batterie de tests unitaires afin de valider le fonctionnement du module. |
| --- | --- |
| Objectifs d'affaires | L’ensemble des tests unitaires valide le fonctionnement de tous les modules du système en moins de 3 minutes. La création des tests de nouveaux modules ne prend pas plus du quart du temps nécessaire à leur développement. |
| Attribut de qualité | Testabilité |
| Source | Testeurs unitaires |
| Stimulus | Batterie de tests effectués sur le module de traitement des transactions |
| Artefact | Code du guichet automatique |
| Environnement | Période de développement |
| Réponse | Les tests sont exécutés et les résultats sont capturés. |
| Mesure de la réponse | Effort requis pour détecter une faute dans le module transactionnel;  Effort requis pour obtenir une couverture de tests supérieure à 75%;  Durée requise pour préparer l’environnement de tests |
| Questions |  |

# 

## Testabilité – Tactiques pour le scénario 1

Description: *Specialized interfaces*

Justification: La tactique d’emploi d’interfaces spécialisées est pertinente pour le premier scénario de qualité. Il s’agit de créer des interfaces spécialisées conçues pour obtenir des renseignements et contrôler l’état des composants lors de l’exécution de tests. Il serait donc possible d’avoir une meilleure visibilité sur l’état interne du logiciel embarqué des périphériques d’entrée lors de l’exécution de la batterie de tests diagnostic. Il est à noter que ces interfaces utilitaires devraient être isolées des méthodes liées à l’utilisation normale des périphériques afin de ne pas contrevenir au bon fonctionnement des périphériques.

## Testabilité – Tactiques pour le scénario 2

Description: *Abstract data sources*

Justification: La tactique d’abstraction des sources de données est toute indiquée pour le deuxième scénario de qualité. Il s’agit donc d’abstraire l’accès aux données du module de transaction de façon à pouvoir les remplacer facilement. De ce fait, il serait possible de remplacer la source de données du module par des modules spécialement créés pour les tests et ainsi substituer contrôler les données de l’environnement de tests.

Description: *Sandbox*

Justification: La tactique du bac à sable est une tactique intéressante pour le scénario 2. Il s’agit d’isoler le module qui est sujet de son environnement externe et ainsi éliminer le souci des conséquences des opérations du module sur l’état du système. Pour ce faire, il serait possible de virtualiser les ressources qu’exploite le module de transactions, comme les sources de données (voir tactique *Abstract data sources*) et les sources d’entrée. Cette tactique permettrait ultimement de tester le composant et son fonctionnement sans que l’état des composants environnants de vienne influencer les résultats des tests.

# 

## Testabilité – Vue architecturale

## Diagramme





## Description du diagramme:

## Description des éléments

**(Nouveaux éléments en gras)**

| Élément | Description |
| --- | --- |
| **Sous-Module d’affichage** | Module gérant l’affichage du guichet, le guichet permet de sélectionner l’affichage mode Test, plus détaillé pour effectuer les batteries de tests ou le mode normal. Il n’est pas possible de changer le mode du guichet sans avoir accès à clé. |
| **Simulateur de communication** | De manière similaire, le Sous-Module de communication du guichet peut être remplacé par le Simulateur de communication qui permet d’effectuer des tests sur le guichet en générant les réponses du serveur sur le guichet lui-même. |
| Guichet | Représente une classe contenant l’état et informations éphémères contenues sur le guichet. Contrôleur de session. |
| LecteurCarte | Représente le lecteur de carte, est responsable d’effectuer toutes les validations automatiques effectuées lors de l’insertion de la carte dans le guichet. |
| ValidateurCarte | Classe abstraite effectuant une vérification sur la carte insérée et pouvant demander à un autre VérificateurCarte de faire de même et ainsi de suite (patron composite). |
| ValidateurBandeMagnétique | Classe concrète effectuant des vérifications sur la carte à l’aide de la bande magnétique. |
| ValidateurPuce | Classe concrète effectuant des vérifications sur la carte à l’aide de la puce. |
| CréateurTransaction | Classe responsable de créer les transactions que le client veut réaliser en utilisant l’ATM. |
| ChiffreurTransaction | Une interface permettant de chiffrer une transaction de façon quelconque. Elle permet de changer le module ou classe chiffrant la transaction de façon transparente au guichet. |
| ChiffreurTransactionRSA | Une classe concrète permettant de chiffrer une transaction grâce à l’algorithme RSA. (Patron stratégie) |

## Relations entre éléments et tactiques

Scénario 1:

Specialized interfaces:

Les batteries de tests effectuée sur l’ATM nécessiteront davantage de détails qui seront normalement cachés à l’utilisateur normal. Il est donc utile d’avoir un module d’affichage agissant différemment selon si le système est en mode de test ou de production. Cette option est disponible grâce au Sous-module d’affichage contenant plusieurs classes interfaces utilisateurs différentes selon les besoins.

Scénario 2:

*Abstract data sources* et *sandbox:*

Afin de pouvoir tester le guichet indépendemment du fonctionnement et du contenu du serveur, il est utile d’avoir un module pouvant remplacer le module de communication : Le simulateur de communication. Ce module permet au guichet de répondre aux requêtes qu’il enverrait normalement au serveur afin de le tester de façon indépendante et contrôlée.

## Relations entre tactiques et scénarios de qualité

Scénario 1:

Specialized interfaces:

Afin de pouvoir tester en profondeur les fonctionnalités de l’ATM en développement et effectuer des batteries de test rigoureuses, il est utile d’avoir une version de l’affichage de l’ATM propre aux tests. Cela permet aussi de modifier l’interface utilisateur pour être davantage agréable/ornementée sans impacter l’efficacité des tests.

Scénario 2:

Abstract data sources:

Afin de pouvoir tester le bon fonctionnement de l’ATM sans risquer d’envoyer des requêtes compromettant le serveur, il est utile de permettre à l’ATM de répondre à ses propres requêtes indépendemment (en mode test uniquement bien sûr!)

Sandbox:

De plus, afin de permettre aux testeurs de tester une bonne couverture de code il faut s’assurer que l’ATM puisse répondre à ses propres requêtes de façon aussi variée que pourrait le faire le serveur.

# Proposition 2

| Scénario | Implémentation d’une logique d’affaires efficacement testable. |
| --- | --- |
| Objectifs d’affaires | Réduire l’effort de développement et de maintenance. Objectif de temps d’exécution des tests unitaires de max. 5min. |
| Attribut de qualité | Testabilité |
| Source | Développeur |
| Stimulus | Les développeurs désirent implémenter une logique d’affaires efficacement testable. |
| Artéfact | Le module de règles d’affaires |
| Environnement | Développement |
| Réponse | Un module de logique d’affaires efficacement testable est implémenté. |
| Mesure de la réponse | * Temps pour préparer un environnement testable * Temps pour performer le test |
| Questions | Aucune question |

## 

## 

## 

## Scénario 2

| Scénario | Détecter lorsque des modifications au code provoquent des régressions |
| --- | --- |
| Objectifs d’affaires | Réduire l’effort de développement et de maintenance |
| Attribut de qualité | Testabilité |
| Source | Intégration testée |
| Stimulus | Les développeurs veulent s’assurer de ne rien briser lorsqu’ils modifient du code. |
| Artéfact | Tests |
| Environnement | Développement |
| Réponse | Exécution des tests et capturer les résultats |
| Mesure de la réponse | Effort pour atteindre un pourcentage donné de la couverture de l'espace d'état |
| Questions | Quel niveau de couverture est nécessaire? |

## Tactique - Scénario 1

**Description**: Limit structural complexity

**Justification**: En limitant la complexité de différentes façons (encapsuler, éviter les dépendances cycliques, etc.), le code devient plus facile à tester. Il devient plus simple d’utiliser des “fakes” et de tester un seul module à la fois, sans se soucier de ses dépendances.

## Tactique - Scénario 2

**Description**: Executable assertions

**Justification**: Cette tactique permettre de détecter l’endroit de la faute dans le code ou le logiciel.

## Vue architecturale

Diagramme et légende



Description du diagramme et description des relations au scénario

Pour la première tactique «Limit structural complexity» les différents modules A,B et C ont été séparés pour éliminer les dépendances cycliques, de sorte qu’ils deviennent testables de façon indépendante. Ils héritent de l’interface IBussinessLogic, ce qui permet au module *test* d’effectuer des tests des règles d’affaires sur l’interface. D’un autre côté, un module d’intégration continue a pour responsabilité de lancer les tests unitaires et d’intégration à chaque modification du code source afin de s’assurer qu’aucune régression n’a eu lieu. Si c’est le cas, une erreur sera lancée par le système d’intégration continue ceci représente l’implémentation de la tactique “Executable assertion” qui nous permet dynamiquement de savoir lorsqu’un défaut est introduit dans le code.

## Description des éléments du diagramme

| Business Layer | Représente la couche de la logique d’affaires |
| --- | --- |
| IBusinessLogic | Interface utilisée pour gérer les différents modules de la logique d’affaires |
| A,B,C | Différent module dans la logique d’affaires |
| Test | Module externe utilisé pour les tests |
| Data Access | Cet élément représente l’accès aux données(DB, data structure, etc). |

# Proposition 3

La testabilité apporte :

* Une forte stabilité puisque des défaillances peuvent être détectées au fil du temps.
* Une assurance que le logiciel respecte les exigences d’affaires.
* Une assurance que le logiciel respecte les standards de performance.
* Une assurance que le logiciel interagis correctement avec d’autres composantes internes ou externes.
* Une assurance que le logiciel est verbose, puisque les tests permettent facilement de comprendre la logique d’affaire.

## Testabilité – Scénario 1

| **Scénario** | Ajout d’un nouveau module |
| --- | --- |
| **Objectifs d'affaires** | S’assurer que le logiciel est encore stable et n’a pas d’impact sur les autres modules et que l’ensemble du code est couvert. |
| **Attribut de qualité** | Testabilité |
| **Source** | Développeur |
| **Stimulus** | Exécution des harnais de test du système |
| **Artéfact** | * Harnais de tests du système * Nouveau module à tester |
| **Environnement** | Environnement de développement |
| **Réponse** | Tous les tests sont exécutés avec succès. |
| **Mesure de la réponse** | * Pourcentage de lignes couvertes * Nombre de tests ayant échoués et réussis |
| **Questions** | 1. Quel est le temps d’exécution du harnais de test? 2. Quand faut-il rouler les tests? 3. Dans quel environnement se dérouleront les tests? |

## Testabilité – Scénario 2

| **Scénario** | Modification d’un module existant |
| --- | --- |
| **Objectifs d'affaires** | S’assurer que les tests du module s’exécutent encore avec succès et que les modifications sont également testées. De plus, les modification n’ont pas d’impacts sur les autres modules. |
| **Attribut de qualité** | Testabilité. |
| **Source** | Développeur |
| **Stimulus** | Exécution des harnais de test du système |
| **Artéfact** | * Harnais de tests du système * Module existant |
| **Environnement** | Environnement de développement |
| **Réponse** | Tous les tests sont exécutés avec succès. |
| **Mesure de la réponse** | * Pourcentage de lignes couvertes * Nombre de tests ayant échoués et réussis |
| **Questions** | 1. Quel outil utilisera-t-on pour trouver le pourcentage de lignes couvertes? 2. Quel cadriciel utilisera-t-on pour développer les tests? 3. Combien de temps prendra le développement des tests? |

## Testabilité – Tactique 1

Description: Sandbox

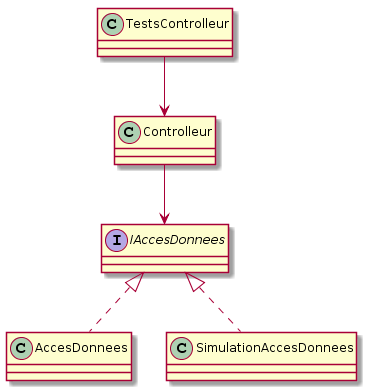
Justification: Lors de la création d’un nouveau module, il sera important d’utiliser un environnement de simulation de type « sandbox » pour vérifier sont comportement dans un environnement de production. Le sandbox simulera l’environnement de production et permettra donc de détecter des anomalies.

## Testabilité – Tactique 2

Description: Executable assertions

Justification: Le harnais de tests du système comportera une série d’assertions pouvant être exécutés à tout moment. Ce harnais sera modifié pour inclure les tests de la nouvelle fonctionnalité et donc assurer qu’elle respecte les standards de performance, de sécurité, et les exigences d’affaires.

## Testabilité – Vue Architecturale

**Légende: <Notation UML>**

## Description du diagramme

Le contrôleur représente un module typique de l’application. Il accède à ses dépendances à travers un interface (*IAccesDonnees*) pour permettre l’injection des dépendances. La suite de tests *TestsControlleur* appelle directement les méthodes de *Controlleur* pour valider leurs bon fonctionnement.

La tactique 1 est représenté dans ce diagramme par l’injection des dépendances. Le module *Controlleur* ne connais que l’interface *IAccesDonnees*, mais pas son implémentation. Il est donc facile de simuler les dépendances du module *Controlleur* lors des tests du systèmes.

La tactique 2 est représenté dans ce diagramme par le module TestsControlleur. En effet, il s’agit de la suite de tests du harnais qui appelle directement les méthodes de *Controlleur* pour valider leurs bon fonctionnement.

## Table de description des éléments des diagramme

Voici un tableau qui décrit les caractéristiques des éléments des vues architecturales présentées à la page précédente.

| **Élément** | **Description** |
| --- | --- |
| TestsControlleur | Ensemble de tests couvrant toutes les fonctionnalités offerte par *Controlleur* |
| Controlleur | Module typique de l’application. Il gère le lien entre les vues et les modèles. |
| IAccesDonnees | Interface d’accès aux données |
| AccesDonnees | Accès aux données réelles |
| SimulationAccesDonnees | Accès aux données fictives utilisées par le harnais de tests |
| Liens | Les liens entre les classes sont des liens de dépendance et des liens d’héritage selon la notation UML |

# Proposition 4

## <Testabilité> – Justification

La testabilité est un attribut de qualité important pour facilité la détection de fautes dans un système et permet, lorsque les tactiques de testabilité sont appliquées adéquatement, de réduire le coût de développement des tests. Il est particulièrement important de s’assurer qu’un logiciel gérant un guichet automatique contienne le moins de fautes et que ces fautes soient détectées rapidement.

## <Executable assertions> – Scénario 1

| Scénario | S’assurer que le tous les opérations nécessaires à un processus sont exécutés |
| --- | --- |
| Objectifs d'affaires | Éviter de déployer le système incomplet |
| Attribut de qualité | Testabilité |
| Source | Outil de test automatisé |
| Stimulus | Implémentation complète du système |
| Artéfact | L’ensemble du système |
| Environnement | Période de développement et d’intégration |
| Réponse | Exécution d’une suite de test avec pourcentage de couverture totale |
| Mesure de la réponse | -Couverture des tests de plus de 95%  -Temps nécessaire pour tester le entièrement le système |
| Questions | 1. Quel est le ratio du temps de développement des tests par rapport au temps de développement total? 2. Le pourcentage de couverture permet il de tirer des conclusions sur la résilience du système? 3. À quelle fréquence le système devrait il être complètement testé? 4. Quelle est le temps nécessaire pour rouler la suite de test. |

# 

# 

## <Sandbox> – Scénario 2

| Scénario | S’assurer du fonctionnement complet des cas d’utilisations principaux |
| --- | --- |
| Objectifs d'affaires | Diminuer les risques de bugs pour le client |
| Attribut de qualité | Testabilité |
| Source | Testeur QA |
| Stimulus | Préparation de la livraison au client |
| Artéfact | Ensemble du système |
| Environnement | Période d’exécution |
| Réponse | -Capture de l’activité responsable d’une faute |
| Mesure de la réponse | -Temps nécessaire pour soulever une faute  -Nombre d’essai par le QA pour découvrir une faute |
| Questions | 1. Est-il possible d’automatiser un test complet du système plutôt que d’utiliser du personnel de QA? |

## <Executable assertions> – Tactique 1

Description:

La tactique *Executable assertions* implique de développer des tests qui couvrent idéalement l’ensemble des scénarios prévisibles en utilisant différentes valeurs d’entrée d’une procédure ou d’une fonction et de valider les valeurs de sortie pour assurer que le traitement s’opère toujours comme prévu.

Justification: En développant une suite de tests, la validation peut être effectuée à chaque fois que le système est modifié, permettant de détecter que le système est aussi fonctionnel qu’avant la modification et réduit par le fait même le risque de déployer un logiciel au comportement imprévisible.

# 

# 

## <Sandbox> – Vue Architecturale

Vue-architecturale.png

Description du diagramme:

Un serveur “Sandbox” est utilisé pour que les testeurs QA puissent utiliser le système sans restriction pour couvrir l’ensemble des actions qu’un client pourrait effectuer, sans risque d’affecter négativement le système réel et les données sauvegardées sur les bases de données bancaires.

Les connecteurs reliant le logiciel aux serveurs bancaires redirigent plutôt les informations envoyées vers un serveur bancaire simulé. De plus, un logiciel imite le comportement des composants physiques pour contrôler leur état et tester la façon dont le système réagit dans des situations de défectuosité.

# Proposition 5

## Testabilité – Justification

Les tests sont une partie importante dans le développement d’applications et de systèmes. Afin que le système soit fiable et sécuritaire, il est nécessaire de tester les différentes parties de celui-ci. Dès lors, cela requiert de vérifier de multiples composants.

Pour les banques, il est nécessaire de limiter les problèmes de ces nouvelles technologies. Ainsi, l’ajout de ces nouvelles technologies ne doit pas affecter les présentes composantes déjà en place. Pour les clients, il est nécessaire d'effectuer ses transactions sans problèmes et le plus sécuritaire possible.

Afin d’effectuer l’ajout de nouvelles parties ou composantes, il est nécessaire pour les développeurs de pouvoir tester le système sans obligatoirement posséder l’ATM physiquement ou bien valider certains tests d'intégrité physiquement sur celui-ci. Dès lors, il faut que le processus d’affaire prenne en compte la modifiabilité de l’ensemble et d’ajouter des tactiques applicables à celui-ci.

## Testabilité – Scénario 1

| **Scénario** | Un nouveau type de billet est disponible dans l’ATM |
| --- | --- |
| **objectifs d'affaires** | Permettre au consommateur de minimiser le nombre de billets retirés |
| **Attribut de qualité** | Testabilité |
| **Source** | Tests manuels |
| **Stimulus** | Introduction d’un nouveau type de billets dans le distributeur |
| **Artéfact** | ATM, billets |
| **Environnement** | Intergration time |
| **Réponse** | Control et monitor l’état du système |
| **Mesure de la réponse** | Les billets distribués sont bien répartis entre les différents types en fonction de la somme requise. |
| **Questions** | 1. Comment généraliser les tests pour un nombre variable de type de billets ? 2. Comment simplifier l’ajout ou la modification d’un type de billet ? |

## Testabilité – Scénario 2

| **Scénario** | Validation du fonctionnement d’un dépôt bancaire ATM-serveur suite à une modification du chiffrement. |
| --- | --- |
| **objectifs d'affaires** | Assurer la sécurité des communications bancaires |
| **Attribut de qualité** | Testabilité |
| **Source** | Outils de tests automatiques |
| **Stimulus** | Les tests sont exécutés suite à la complétion de nouveaux codes de classes ou services. |
| **Artéfact** | Protocoles de communication, serveur, sécurité, transaction |
| **Environnement** | Deployment time |
| **Réponse** | Les tests sont exécutés et les résultats sont retenus |
| **Mesure de la réponse** | La transaction doit être effectuée en moins de 10 secondes. |
| **Questions** | 1. Comment générer les résultats de référence des tests ? 2. Comment mettre en commun la méthode de chiffrement entre l’ATM et les tests? 3. Comment vérifier la communication ATM-serveur sans devoir mobiliser un serveur ? |

## Testabilité – Limit structural complexity – Scénario 1

Description: Sachant qu’il est probable qu’un nouveau type de billet doit être supporté, la présente implémentassions doit prévoir cette éventualité. Ainsi l’accès au type de billets devrait permettre cette modularité.

Justification: Les tests étant fait de façon manuelle, le système devrait être en mesure de traiter les différents types de billets automatiquement sans nécessiter l’ajout de classes objets représentant ces types de billets. Dans le présent contexte, cette stratégie est plus effective qu’un “sandbox” sachant que la présente situation test l’état physique de l’ATM.

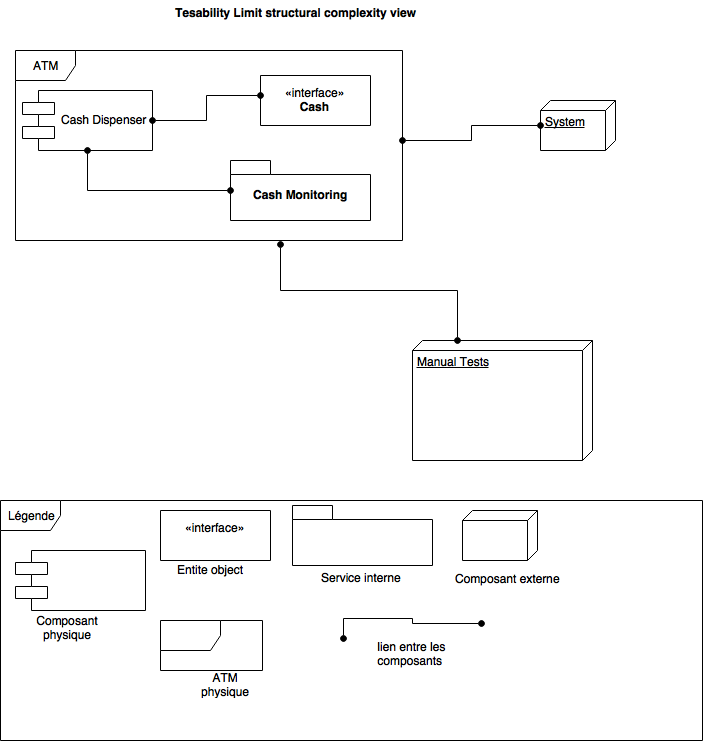
## Testabilité – Sandbox – Scénario 2

Description: Une fois la modification effectuée, la suite de tests est lancé en utilisant un environnement de type “sandbox” afin d’effectuer un dépôt bancaire et ainsi valider l’intégrité du chiffrement de la transaction.

Justification: Considérant le fait que le processus de la transaction n’est pas affecté par la modification, il est important de tester l’efficacité du chiffrement en suivant des normes établies. Contrairement à un “Abstract data source” qui serrais optimal pour validé l’état d’une transaction déjà effectué. Cependant le cas présent suggère une nouvelle transaction selon le dépôt bancaire.

## Testabilité – Vue architecturale – Scénario 1

#### Diagramme et légende



#### Description du diagramme et des relations

Il est possible de remarquer que l’interface “Cash” représente une abstraction du type de billets devant être servis par le distributeur automatique. L’importance de cela est de limité la complexité structurelle en permettant de ne pas fixer un type de billet, mais bien de pouvoir en ajouter de multiples.

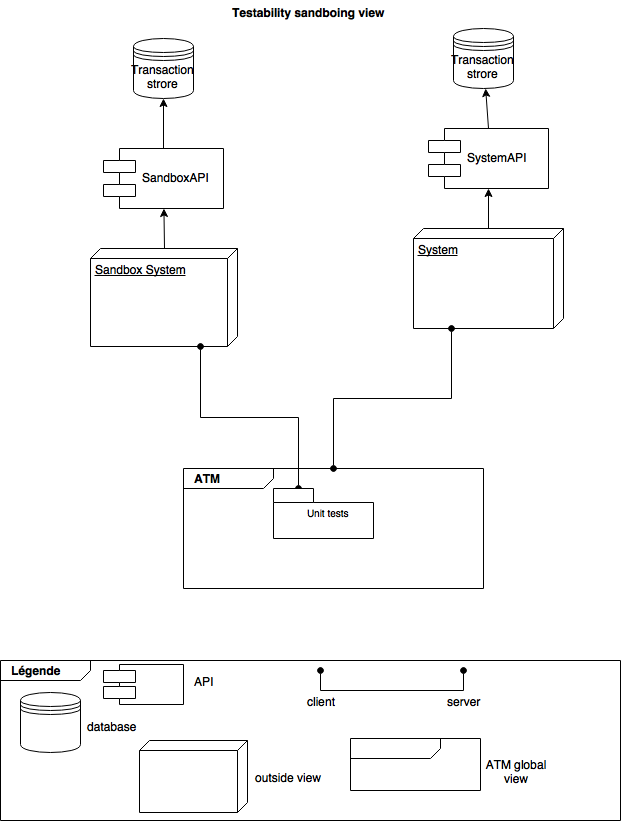
#### 

## Description des éléments du diagramme

| ATM | Représente un appareil physique et son système  embarqué sur le site d’opération (banque) |
| --- | --- |
|  |
| Cash dispenser | Représente le composant physique servant à distribuer l’argent |
|  |  |
| Cash monitoring | Représente le service s’occupant de monitor l’argent. |
|  |  |
| Manuel Tests | Suite de tests manuels permettant de déterminer la validité du traitement de nouveaux billets |
|  |  |
| System | Système de communication de la banque |
|  |  |
| Cash | Interface de représentation du traitement de l’argent dans le distributeur |
|  |  |
| Tactiques répond aux scénarios | |

La tactique “Limit structural complexity” a été implémentée en utilisant une interface de “Cash” afin de permettre la modularité du traitement des types d’argents possibles en ne prenant pas en compte leurs différences.

## Testabilité – Vue architecturale – Scénario 2



## Description du diagramme et des relations

Il est possible de remarquer que le package “Unit tests” se connecte à un point différent. L'importance est de mettre en évidence que les tests lancés, simulent des transactions sur l'environnement “Sandbox” afin de ne pas polluer la base de données principale du système.

## Description des éléments du diagramme

| ATM | Représente un appareil physique et son système  embarqué sur le site d’opération (banque) |
| --- | --- |
|  |
| Unit tests | Suites de tests contenus sur le système embarqué ATM avertissant des problèmes rencontré. |
|  |  |
| Sandbox System | Duplication du fonctionnement du système normal |
|  |  |
| SandboxAPI | API de connections utilisé dans le module de Sandbox |
|  |  |
| System | Système de communication de la banque |
|  |  |
| SystemAPI | API de connections utilisé dans le système de communication bancaire. |
|  |  |
| Transaction store | Base de donnée ou est contenu l’ensemble des transactions effectué sur le système. |
|  |  |
| Tactiques répond aux scénarios | |

La tactique “Sandbox” a été implémentée en utilisant un environnement de “Sandbox” afin de testé le chiffrement des transactions une fois la modification de celle-ci faite sur le système de communication bancaire.

# Proposition 6

## Testabilité – Justification (en classe)

Il est important qu’un guichet automatique n’est pas de bogue majeur, puisque celui si manipule des sommes d’argent importantes et est utilisé aux quotidiens par la plus des gens. De plus un bogue pourrait facilement engendrer des coûts énormes.

## Testabilité – Scénario 1 (en classe)

| **Scénario** | Changement à l’api du serveur |
| --- | --- |
| **Objectifs d'affaires** | Tester l’api du serveur pour s’assurer qu’il n’ait pas de bogue majeur. |
| **Attribut de qualité** | Testabilité |
| **Source** | Tester unitaire |
| **Stimulus** | Implémentation du nouvel api terminée |
| **Artéfact** | Le nouveau module de l’api |
| **Environnement** | Development time |
| **Réponse** | Exécution des tests de non-régression ainsi que les tests unitaire et les smokes tests |
| **Mesure de la réponse** | Temps d’exécution de l’ensemble des tests.  Le nombre de bogues en production.  La couverture de code des tests. |
| **Questions** |  |

## Testabilité – Scénario 2 (en classe)

| **Scénario** | Exécution de test de régression sur l’ensemble du code en intégration continue. |
| --- | --- |
| **Objectifs d'affaires** | Réduire le nombre de bogues en production. |
| **Attribut de qualité** | Testabilité |
| **Source** | Gestionnaire d’intégration continue |
| **Stimulus** | Exécution des tests lancée automatiquement par le gestionnaire |
| **Artéfact** | Le code de l’ATM. |
| **Environnement** | Integration time |
| **Réponse** | Exécution des tests de non-régression ainsi que les tests unitaire et les smokes tests |
| **Mesure de la réponse** | Temps d’exécution de l’ensemble des tests.  Le nombre de bogues en production.  La couverture de code des tests de régression. |
| **Questions** | 1. Est-ce que l’exécution des tests par l’intégration continue diffère de l’exécution manuelle? |

## Testabilité – Tactique 1 (en classe)

Description: **Abstract Data Sources**

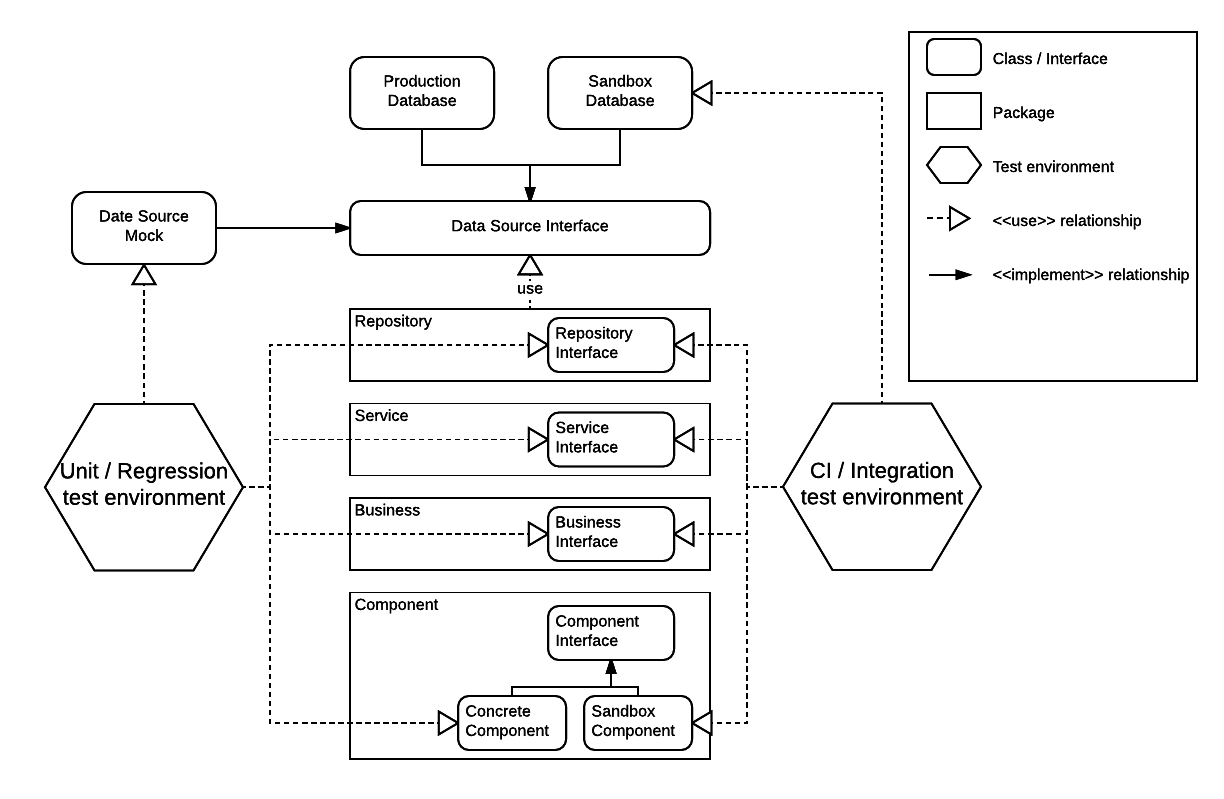
Justification: Un obstacle quand vient le temps de tester un API manipulant des données importantes est de pouvoir rouler une suite de tests vérifiant la logique d’affaires sans affecter les données présentes dans les bases de données de développement, production ou intégration. Afin de rouler les tests sans affecter les données potentiellement importantes, tout en étant en mesure de tester les données enregistrées plus facilement, les sources de données seront interfacées. Ces interfaces seraient par la suite implémentées par les classes de production, affectant les données réelles du système, ainsi que par des classes de tests, isolées des données physiques et offrant des façons de vérifier ce qui aurait été enregistré.

## Testabilité – Tactique 2 (en classe)

Description: **Sandbox**

Justification: Pour bien tester l’ensemble du code en intégration, le sandbox nous permet d’isoler cette partie du système de l’ensemble des composants et nous permet de faire tous les tests possibles sans se soucier des conséquences sur les composants avec lequel il communique.

# Testabilité – Vue Architecturale (à l’extérieur de la classe)



## Tableau des éléments

| **Éléments** | **Description** |
| --- | --- |
| Data Source Interface | Source de données abstraite, permettant l’utilisation d’une source différente dépendamment de l’environnement. |
| Production / Sandbox Database | Sources de données concrètes, utilisées dans des environnements de production et sandbox, respectivement. |
| Data Source Mock | Source de données concrète, utilisée dans un environnement de tests unitaires. |
| Unit / Regression test environment | Environnement de tests unitaires et de régression. |
| CI / Integration test environment | Environnement de tests d’intégration et d’intégration continue. |
| Repository / Service / Business / Component interface | Interfaces vers les classes de Repository, Service, Business et Component. |
| Concrete / Sandbox component | Implémentations concrètes des classes de Component, dans des environnement de production et sandbox, respectivement. |

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

* L’élément “Data Source Interface” représente l’interface commune aux différentes sources de données dans le contexte de la tactique 1 (Abstract Data Sources).
* Les éléments “Data Source Mock”, “Production Database” et “Sandbox Database” représentent les sources de données concrètes dans le contexte de la tactique 1 (Abstract Data Sources)
  + L’élément “Data Source Mock”, plus précisément, est la source de données concrètes permettant les tests unitaires et les tests de régression sans affecter les données réelles du système. Il offre des méthodes de validation des données supplémentaires.
* Les éléments “Data Source Interface”, “Repository Interface”, “Service Interface”, “Business Interface” et “Component Interface” représentent les abstraction de chacune des couches de l’application, permettant ainsi de remplacer chacune au besoin dans un environnement de type Sandbox, pour la tactique 2 (Sandbox).
* Les éléments “Sandbox Database” et “Sandbox Component” représentent les implémentations concrètes de la Source de données et des composants, respectivement, pour la tactique 2 (Sandbox). Ils permettant la séparation de l’environnement de tests avec des données / composants réels du systèmes.

# Proposition 7

## Testabilité – Justification (en classe)

La testabilité représente 50% du temps investi dans le développement logiciel. Cet attribut permet d’éviter des bogues logiciels, des anomalies. Ce qui permet de garder la confiance du client et limiter les pertes d’argent lié à des défauts du système.

## Testabilité – Scénario 1 (en classe)

| **Scénario** | Vérification qu’une transaction s’effectue correctement |
| --- | --- |
| **objectifs d'affaires** | Éviter les bugs, les erreurs. Confiance du client. Limiter les pertes d’argent |
| **Attribut de qualité** | Testabilité |
| **Source** | Unit testers |
| **Stimulus** | Test du module de transaction |
| **Artéfact** | Le module de transaction |
| **Environnement** | Design time |
| **Réponse** | Exécution du test et recueillir les résultats |
| **Mesure de la réponse** | % de succès des test  Éffort nécessaire pour réaliser |
| **Questions** | 1. Qu’est qu’on a besoin pour réaliser le test ? (Serveur, DB, Réseau) 2. Combien de temps pour réaliser le test, combien de testeurs ? 3. Quelle devrais être le % de succès pour accepter le résultat de test 4. Quelle type de test pour valider le succès du test (Unitaire,système..) |

## Testabilité – Scénario 2 (en classe)

| **Scénario** | Vérifier que l’authentification s’effectue correctement |
| --- | --- |
| **objectifs d'affaires** | Valider l’identité du client et limiter les fraudes |
| **Attribut de qualité** | Testabilité |
| **Source** | Unit testers |
| **Stimulus** | Test du module d’authentification |
| **Artéfact** | Le module d’authentification |
| **Environnement** | Design time |
| **Réponse** | Exécution du test et recueillir les résultats |
| **Mesure de la réponse** | % de succès des test  Éffort nécessaire pour réaliser |
| **Questions** | 1. Qu’est qu’on a besoin pour réaliser le test ? (Serveur, DB, Réseau) 2. Combien de temps pour réaliser le test, combien de testeurs ? 3. Quelle devrais être le % de succès pour accepter le résultat de test 4. Quelle type de test pour valider le succès du test (Unitaire,système..) |

## Testabilité– Scénario 1 : Sandbox

**Description**:L’utilisation de la tactique Sandbox

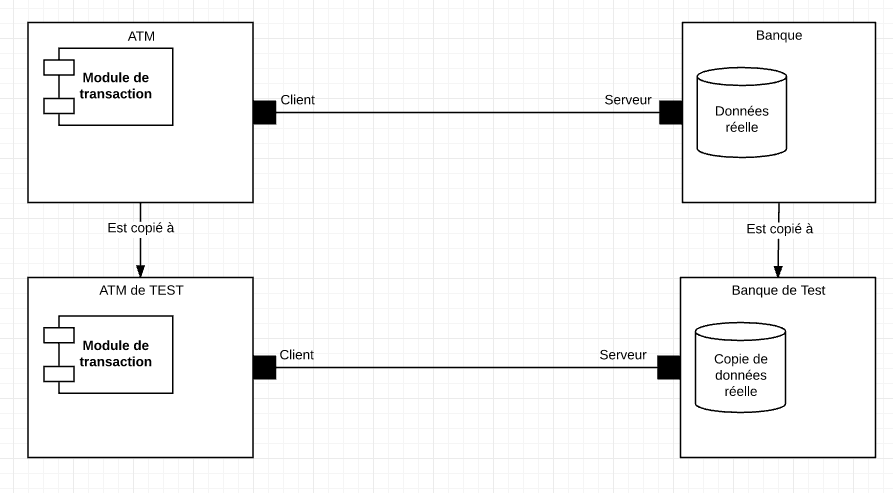
**Justification**: La nécessité d’avoir un environnement de test isolée pour effectuer des tests sans changer les donnés du système réel.

## Testabilité– Scénario 2 : Sandbox

Description: L’utilisation de la tactique Sandbox

Justification: Idem

## Testabilité – Vue Architecturale





# Proposition 8

## Scénario I – Tester l’authentification d’un usager par sa carte bancaire

| **Scénario** | Tester l’authentification d’un usager avec sa carte bancaire |
| --- | --- |
| **objectifs d'affaires** | Assurer la confidentialité des clients (que les données soient protégées). |
| **Attribut de qualité** | Testabilité |
| **Source** | Testeur unitaire, testeur d’intégration |
| **Stimulus** | Exécution des tests unitaires et d’intégration |
| **Artéfact** | Module d’authentification des clients, lecteur de carte bancaire |
| **Environnement** | Environnement de développement |
| **Réponse** | Exécution des tests, capture des résultats des tests |
| **Mesure de la réponse** | * Temps d’exécution des tests * Succès ou non de l’authentification du client |
| **Questions** | 1. Quelles sont les stratégies des tests pour assurer que les données confidentielles soient bien protégées? 2. Quelle est la configuration pour l’environnement de test? |

## Scénario II – Tester la validité d’une transaction (dépôt)

| **Scénario** | Tester la validité d’une transaction (dépôt) |
| --- | --- |
| **objectifs d'affaires** | Assurer que la transaction soit effectuée correctement et que les clients soient assurés que leur argent est entre de bonnes mains! |
| **Attribut de qualité** | Testabilité |
| **Source** | Testeur unitaire, testeur d’intégration |
| **Stimulus** | Exécution des tests unitaires et d’intégration |
| **Artéfact** | Module(s) de transactions bancaires |
| **Environnement** | Environnement de développement |
| **Réponse** | Exécution des tests, capture de résultats des tests |
| **Mesure de la réponse** | * Temps d’exécution des tests * Succès ou non de la transaction |
| **Questions** | 1. Quels sont les stratégies de test? 2. Quels sont les outils de test? 3. Est-ce qu’il est possible de contrôler et d’observer tous les composants importants? |

## Tactique - Scénario I

**Description**: Record-Playback

**Justification**: Dans le cas du scénario 1, la confidentialité est important. On veut connaître tous les manipulations et les données possibles qui peut causer des défaillances à système de l’ATM. Cette tactique va permettre d’enregistrer ces informations importantes afin d’améliorer les stratégies de tests. On peut donc utiliser ces données pour tester les futurs modifications du système.

## Tactique - Scénario II

**Description**: Executable assertions

**Justification**: Vérifier que la bonne transaction a été effectuée est simple, puisque les résultats attendus sont connus par exemple, un utilisation va déposer de l’argent et qu’après cette transaction, le montant du compte du client doit augmenter. Cela signifie qu’avec cette tactique, on va vérifier les résultats retournés des tests pour confirmer que chaque transaction fait ce qu’il faut faire.

## Vue Architecturale

**Diagramme - Scénario I & II:**



Ce diagramme consiste une petite partie du système de l’ATM contenant les modules affectés par les scénarios. Il montre les relations et l’héritage entre les modules.

| **Éléments** | **Descriptions** |
| --- | --- |
| **GUI** | Le module qui représente l’interface d’utilisateur de la machine ATM. |
| **ATM** | Le module qui représente le coeur de l’ATM. Il permet de rediriger les requêtes à la bonne service. |
| **Authentication** | Le module qui offre le service pour s’authentifier |
| **Transaction** | Le module abstrait qui représente une transaction générique |
| **TransactionType1, 2, 3** | Les modules qui offrent de différents services de transaction |
| **AnthenticationTest** | Le module qui test les fonctionnalités principales du module de l’authentification |
| **Record/Playback** | Le module qui enregistre les données de tests et qui peuvent être utilisée pour les futurs changements |
| **TransactionTest** | Le module qui teste les fonctionnalités principales des différents type de transaction |

Pour faire le lien avec les scénarios et le choix des tactiques, le module Record/Playback a été ajouté afin d’enregistrer les données que le module de test, AuthenticationTest, envoie à sa cible, le module Authentication, durant les exécutions de tests. Ces données enregistrées peuvent être utiliser pour les futurs tests s’il y a des changements sur le module Authentication. De plus, pour les tests sur les transactions, le module TransactionTest est présent pour exécuter des tests unitaires pour chaque type de transaction en utilisant la méthode d’assertion.