

# **Plan de tests**

## **Projet de streaming vidéo interactif**



## Auteur(s) et contributeur(s)

Nom & Coordonnées	Qualité & Rôle	Société
Gérald ATTARD	Architecte logiciel	Gibberish

## Historique des modifications et des révisions

N° version	Date	Description et circonstance de la modification	Auteur
0.1	08/12/20--	Création du document	
1.0	29/09/2022	Complément du document	Gérald ATTARD
1.1	01/11/2022	<ul style="list-style-type: none"><li>• Ajout des listes de contrôle</li><li>• Modification des tests d'acceptation – déclinaison et décomposition des tests d'acceptation au niveau <i>cas de tests</i></li><li>• Ajout de l'identification des composants impactés par les différents tests</li></ul>	Gérald ATTARD

## Validation

N° version	Nom & Qualité	Date & Signature	Commentaires & Réserves
1.0	Alex Z Directeur technique		

## Tableau des abréviations

Abr.	Sémantique
BPMN	Business Process Model and Notation (trad. <i>modèle de processus métier et notation</i> )
CI/CD	Continuous Integration / Continuous Delivery / Continuous Deployment (trad. <i>intégration continue / livraison continue / déploiement continu</i> )
BDD	Business Driven Development (trad. <i>développement axé sur le métier</i> ) / Behavior Driven Development (trad. <i>développement dirigé par le comportement</i> )
BPM	Business Process Management (trad. <i>gestion de processus métier</i> )
DDA	Document de Définition d'Architecture
Dev Team	Development Team (trad. <i>équipe de développement</i> )
ETP	Équivalent Temps Plein
IDE	Integrated Development Environment (trad. <i>environnement de développement intégré</i> )
KPI	Key Performance Indicator (trad. <i>indicateur de performance clé</i> )
MDA	Model Driven Architecture (trad. <i>architecture dirigé par modèle</i> )
QA	Quality Assurance (trad. <i>assurance qualité</i> )
ROI	Return Of Investment (trad. <i>retour sur investissement</i> )
SOA	Services Oriented Architecture (trad. <i>architecture orientée services</i> )
TDD	Test Driven Development (trad. <i>développement dirigé par le test</i> )
UML	Unified Modeling Language (trad. <i>langage de modélisation universel</i> )

## Table des matières

I. Introduction.....	5
II. Rappels.....	6
II.A. Architecture cible.....	6
II.B. Phases du projet.....	7
II.C. Notions de tests.....	8
II.C.1. Test unitaire.....	9
II.C.2. Test fonctionnel.....	11
III. Ressources.....	13
III.A. Ressources matériels.....	13
III.B. Ressources intellectuelles.....	14
IV. Politique de tests.....	15
IV.A. Jeu de données.....	17
IV.A.1. Copie de la production.....	17
IV.A.2. Génération de données.....	17
IV.A.3. Création d'un sous-ensemble à partir de la production.....	18
IV.B. Listes de contrôle.....	19
V. Fonctionnalités et composants testés.....	25
VI. Tests de composants.....	33
VII. Infrastructure préconisée.....	43
VII.A. Le pipeline DevOps.....	45
VII.B. L'intégration continue.....	46
VII.B.1. Développement.....	46
VII.B.1.a. Phase d'analyse et de conception.....	46
VII.B.1.b. Méthode d'analyse et de conception.....	46
VII.B.1.b.i. Outil d'analyse.....	48
VII.B.1.b.ii. Outil de développement.....	48
VII.B.2. Suivi de version.....	49
VII.B.2.a. Les branches de développement.....	49
VII.B.2.a.i. Un mouvement en deux temps.....	49
VII.B.2.a.ii. Choisir son tempo.....	49
VII.B.2.b. Le branchement par <i>release</i> .....	50
VII.B.2.c. Outil.....	51
VII.B.3. Package.....	52
VIII. Hypothèses et risques.....	53
VIII.A. Hypothèse en Ressources Humaines.....	53
VIII.B. Hypothèse temporelle.....	53
VIII.C. Risques.....	54
VIII.C.1. Cartographie des risques.....	56



# I. Introduction

Le présent document complète le cahier des charges et le document de définition associés à ce projet de streaming de vidéos interactives.

En ce sens, la méthode utilisée jusqu'à présent pour rédiger les autres documentations était le BDD (*Business Driven Development*). Cette méthode de développement axée sur l'entreprise est un paradigme qui se concentre sur les besoins globaux d'une entreprise, et qui permet de développer des applications spécifiques destinées aux consommateurs.

Or, dans le contexte de *Gibberish*, le besoin et la causalité de ce projet répondent aux critères d'utilisation de cette méthode.

Ainsi, au sein de ce document, cette étude utilisera à nouveau le BDD pour réaliser l'écriture des tests nécessaires à la vérification de la sémantique de chaque fonctionnalité développée.

En appliquant la politique de modernisme continue de *Gibberish*, il s'avérera que l'un des problèmes inhérents, au processus de développement d'une solutions logicielle pour cette entreprise, sera l'incapacité de suivre le rythme auquel elle devra changer son SI, en réponse aux tendances émergentes.

Pour que les différents services informatiques de *Gibberish* survivent, l'entreprise devra s'aligner sur les demandes commerciales émergentes, tout en appliquant les derniers technologies innovantes de la période en question, répondant ainsi à sa propre politique technique.

Cette politique devra alors savoir faire la différence entre la conception de solutions résolvant un ou plusieurs problèmes de processus métier et la création de solutions centrées sur l'informatique, c'est à dire sur la technique elle-même.

Il sera donc nécessaire de demander aux parties prenantes si elles :

- souhaitent consacrer une part importante de leur budget à l'amélioration et à la maintenance de leurs applications existantes ;
- préfèrent faire du « *neuf avec du vieux* » ;
- s'orientent plutôt vers un consensus des deux précédentes options.

En effet, au fur et à mesure que *Gibberish* progressera face aux dernières améliorations de ses processus, les applications existantes inflexibles pourraient ne plus être capables d'honorer les changements nécessaires. Dans de tel scénario, le besoin d'un nouveau mécanisme alignant efforts informatiques des exigences et de la stratégie d'entreprise apparaîtra forcément.

C'est dans ce contexte que l'utilisation du BDD facilitera cette transition, peu importe l'option choisie, au travers d'un cadre compris, standardisé et pouvant être réalisé de manière efficace et répétée.

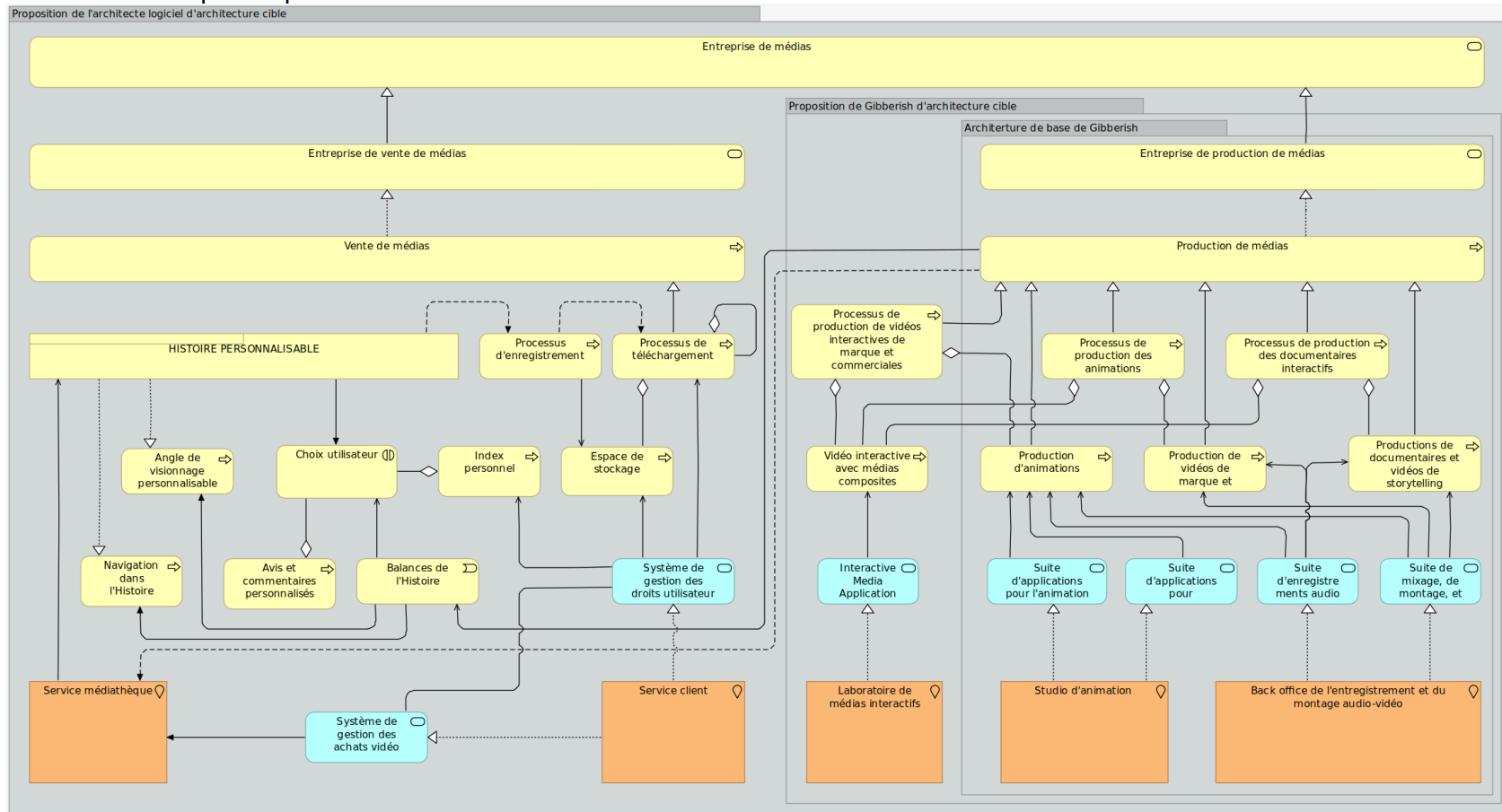
La première étape consistera à créer un modèle de processus métier (BPM) et à le mesurer à l'aide d'indicateurs de performance clés (KPI), de retour sur investissement (ROI) ou d'autres mesures. Enfin, l'entreprise pourra utiliser ces BPM comme un mécanisme crucial pour communiquer les exigences de l'entreprise au domaine informatique.

C'est donc, en suivant cette méthode au sein de ce contexte que ce plan de test sera rédigé, AVANT la rédaction des spécifications techniques, et sera mis en œuvre APRÈS la finalisation de chaque brique fonctionnelle.

## II. Rappels

### II.A. Architecture cible

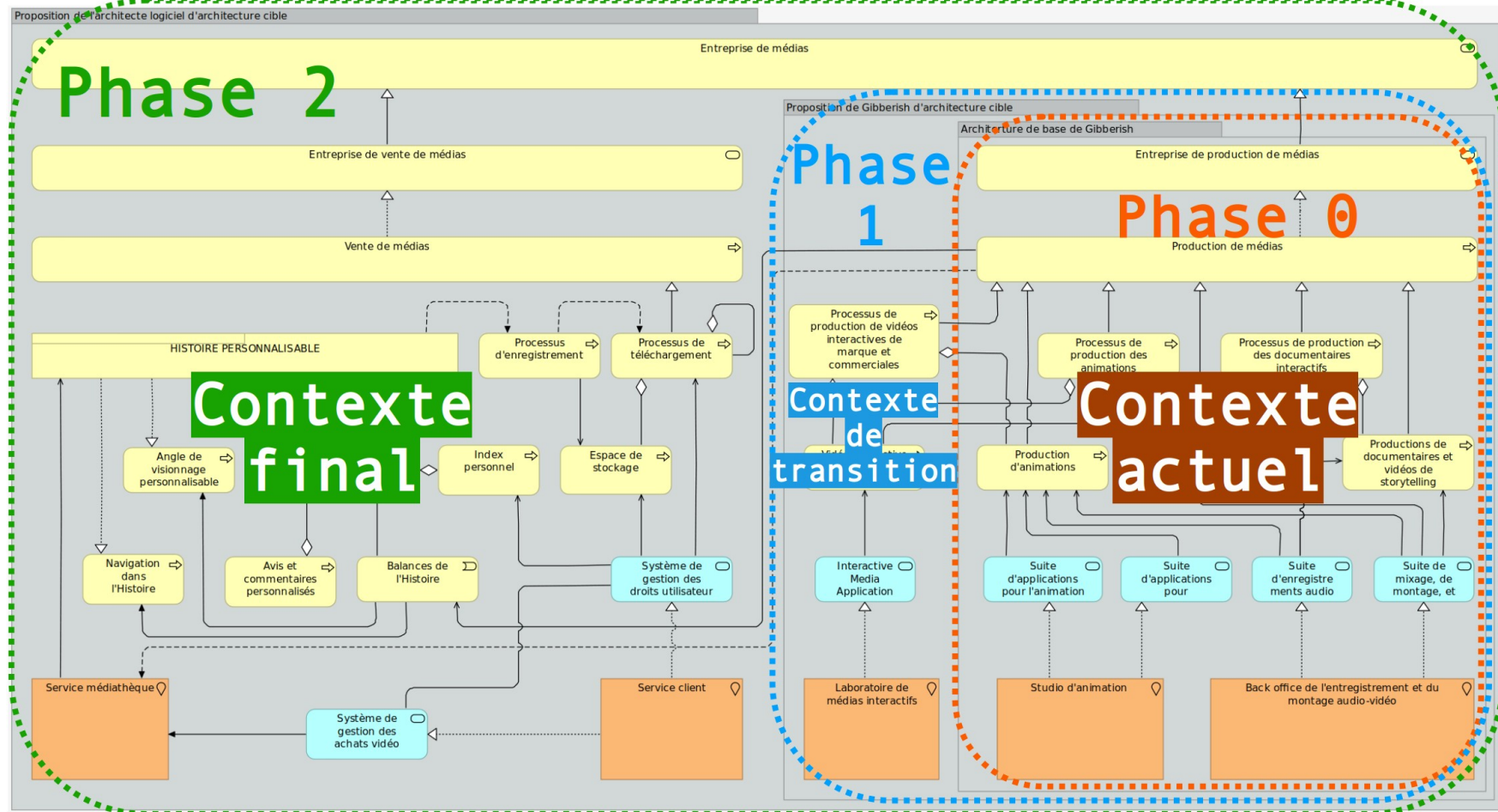
La représentation graphique ci-dessous rappelle la solution proposée au sein du document de définition d'architecture, découpée par thématique de proposition des différentes parties prenantes :





## II.B. Phases du projet

Associée à la représentation du précédent paragraphe, il est également nécessaire de rappeler les phases annoncées au sein du document de définition d'architecture et relative aux différentes architectures à mettre en place :



## II.C. Notions de tests

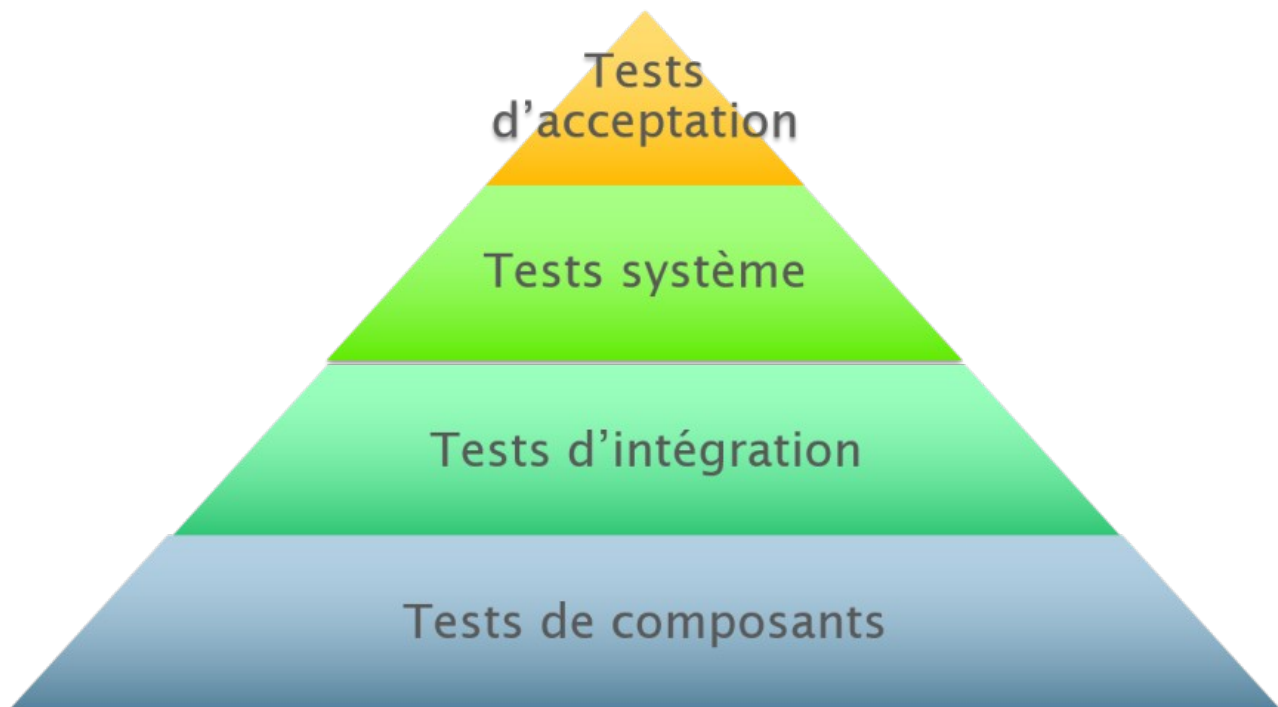
Il existe de nombreux types de tests aux objectifs et stratégies spécifiques, dont **la validation des exigences de base est critique**, tels que :

- **les tests d'acceptation** permettant de vérifier si l'ensemble du système fonctionne comme prévu ;
- **les tests d'intégration** permettant de s'assurer que les composants logiciels ou les fonctions fonctionnent ensemble ;
- **les tests unitaires** permettant de valider que chaque unité logicielle, représentant le plus petit composant testable d'une application, fonctionne comme prévu ;
- **les tests fonctionnels** permettant de vérifier les fonctions en émulant des scénarios métiers, en fonction des besoins fonctionnels - les tests en boîte noire sont un moyen courant de vérifier les fonctions ;
- **les tests de performance** permettant d'évaluer les performances du logiciel sous différentes charges de travail, tels que, par exemple, les tests de charge utilisés pour évaluer les performances dans des conditions de charge réelles ;
- **les tests de régression** permettant de vérifier si de nouvelles fonctionnalités brisent ou dégradent la fonctionnalité. En outre, les tests d'intégrité peuvent être utilisés pour vérifier les menus, les fonctions et les commandes au niveau de la surface, lorsqu'il n'y a pas suffisamment de temps pour effectuer un test de régression complet ;
- **les tests de charge** permettant de tester la quantité de contraintes que le système peut supporter avant qu'il ne tombe en panne – ces tests sont considérés comme un type de test non fonctionnel.
- **les tests d'utilisation** permettant de valider la capacité d'un client à utiliser un système, telle qu'une application Web, pour effectuer une tâche spécifique.

De plus, tout aussi important, **les tests exploratoires aident un testeur** ou une équipe de test à **découvrir des scénarios et des situations difficiles à prévoir** pouvant entraîner des erreurs logicielles.



Ainsi, l'ensemble de ces tests peut se représenter selon la pyramide ci-dessous :



Même une application simple peut faire l'objet d'un grand nombre et d'une variété de tests. Un plan de gestion de tests permet de hiérarchiser les types de tests qui apportent le plus de valeur, compte tenu du temps et des ressources disponibles. L'efficacité des tests est optimisée en exécutant le moins de tests possible pour trouver le plus grand nombre de défauts.

Bien que cette étude ait vocation à se concentrer sur les tests fonctionnels, elle va néanmoins aborder sommairement les tests unitaires qui seront à prévoir lors du développement de l'architecture.

### **II.C.1. Test unitaire**

Le test unitaire consiste à isoler une partie du code et à vérifier qu'il fonctionne parfaitement.

Il s'agit de petits tests qui valident l'attitude d'un objet et la logique du code.


Les tests unitaires sont effectués pendant la phase de développement des applications logicielles.

Ces tests sont effectués par les développeurs ; ils peuvent également être effectués par les responsables en assurance Qualité (QA) afin de s'assurer particulièrement sur telle ou partie du code.

En termes de bonne pratique Qualité, cette étude préconisera à *Gibberish* de favoriser fortement l'utilisation du TDD et/ou du BDD. Néanmoins, ce document privilégiera le BDD et fera abstraction du TDD. En effet, ce dernier est une pratique AGILE orienté vers l'opérationnel et le technique, alors que ce document se veut être un ensemble de recommandations fonctionnelles.

Néanmoins pour décrire quelques peu l'utilisation du TDD, cette étude énoncera les raisons suivantes :

- le test unitaire révèle si la logique derrière le code est appropriée et si elle fonctionnera dans tous les cas ;
- le test unitaire améliore la lisibilité du code et aide les développeurs à comprendre le code de base, ce qui facilite grandement la mise en œuvre des modifications et cela, plus rapidement ;
- des tests unitaires bien conduits sont également de bons outils pour la documentation du projet ;
- en termes de performance, les tests sont effectués en un peu plus de quelques millisecondes, ce qui vous permet d'en réaliser des centaines en très peu de temps ;
- le test unitaire permet au développeur de remanier le code ultérieurement et de s'assurer que le module continue à fonctionner correctement. Des cas de test sont écrits à cet effet pour toutes les fonctions et méthodes afin que les erreurs puissent être rapidement identifiées et réparées, chaque fois que l'une d'elles est créée par l'introduction d'un changement dans le code ;
- la qualité finale du code s'améliorera parce qu'il s'agira en fin de compte d'un code propre et de haute qualité grâce à ces essais continus ;
- puisque le test unitaire divise le code en petits fragments, il est possible de tester différentes parties du projet sans avoir à attendre que d'autres parties soient terminées. Ici l'utilisation de *Mock* prend tout son sens et sa dimension.

En termes de framework d'utilisation, sur un projet à dominante POO, il sera conseillé de travailler avec *xUnit* , dont le framework *TestNG* - basé sur *xUnit* – est possible.

## II.C.2. Test fonctionnel

Les tests fonctionnels sont les tests définis par l'ISO-25010. Ils couvrent les tests d'exactitude, de complétude et d'aptitude à l'usage.

Ils n'ont aucun lien particulier avec les niveaux de tests et peuvent être effectués par les développeurs, les parties prenantes « métiers », les testeurs et tous les intervenants étant amenés à faire du test.

Cependant, une constante domine : les tests fonctionnels sont les tests réalisés par des testeurs...même si cela peut paraître évident à première vue, le métier de QA est un métier à part entière ; cette définition réalise alors une liaison avec les niveaux de test en assimilant les tests fonctionnels aux différents tests d'intégration ou systèmes.

Néanmoins, en fonction des contextes de test, cette définition peut faire apparaître certains contre-arguments à son utilisation, dont notamment :

- le fonctionnel ne peut pas se faire au niveau unitaire (pas de fonctionnel pour les développeurs) ;
- les testeurs ne font que du test fonctionnel alors qu'il serait également possible de leur demander des tests de performances, d'utilisabilité, de portabilité ou tout autre type de test...

Comme vous pourrez le lire, ces deux axiomes ne peuvent être considérés comme tel dans certains contextes d'entreprise. Il sera donc nécessaire d'analyser le contexte de *Gibberish* pour décider « *qui teste quoi ?* ».

Une autre facette à prendre en considération est que les tests fonctionnels sont les cas *passants*, les autres sont du « *Negative testing* ». Cette façon de percevoir les tests offre une autre vision tendant à définir le fonctionnel comme les cas d'usages principaux. Or, les cas non désirés ou d'erreurs sont souvent des usages fonctionnels qu'un utilisateur détourne volontairement ou involontairement, et dont il est important de tenir compte lors du développement.

En outre, les tests fonctionnels sont les tests effectués pendant l'exécution de l'application, c'est à dire que les tests fonctionnels seraient des tests dynamiques et que les tests statiques (comme les revues) ne pourraient pas proposer du test fonctionnel. Cette définition est un contre-sens du point de vue de la norme ISO-25010. Prenons l'exemple d'une revue d'une *User Story* qui met en avant un chemin alternatif manquant. La revue est un test statique et contribue à l'enrichissement fonctionnel de l'application avec l'ajout du chemin alternatif manquant.

De plus, il est bon de préciser que les tests fonctionnels ne sont pas que des tests d'IHM. Cette définition existe également et est réductrice. Elle revient un peu à celle des tests fait par les testeurs mais peut également englober le niveau des tests d'acceptation et certains types de tests comme les tests d'utilisabilité. Par définition de l'ISO-25010, les tests d'utilisabilité sont du non-fonctionnel et il serait réducteur de considérer la testabilité de l'IHM qu'à travers le prisme du fonctionnel...

Dans le contexte de *Gibberish*, il ne sera également pas souhaitable de considérer les tests fonctionnels comme des tests de bout en bout (*End to End*). De par la complexité de l'architecture préconisée pour *Gibberish*, les logiciels utilisés au sein de cette architecture complexe, où les tests composants pourraient être des tests sur des logiciels tiers, chaque composant testé sera à considérer comme un système embarqué ou tout simplement un composant parmi de nombreux partenaires. Cependant, il sera nécessaire de garder à l'esprit qu'un test fonctionnel ne saurait se voir assimiler à un test d'intégration.

Enfin, cette étude insistera sur le fait qu'un test fonctionnel n'est pas forcément un test manuel. L'équipe de QA peut faire appel à des outils de tests automatisés pour les effectués de manière périodiques et pérenne. Cette automatisation verra tout son sens lors des tests fonctionnels de non-régression permettant aux responsables QA de garantir que le système ne tombe pas en panne lors d'une mise à jour logicielle, même si celle-ci est mineure.



## III. Ressources

### III.A. Ressources matériels



Les ressources associées à ce plan de test découleront directement des fonctionnalités définies dans le cahier des charges et de l'infrastructure cible décrite au sein du document de définition de l'architecture.

De plus, ce paragraphe n'assumera que les ressources matérielles nécessaires à ce projet. Les différents éléments abstraits, tels que des événements contextuels ou des choix procéduraux, seront traités ultérieurement.





En outre, en considérant les trois phases décrites dans le document de définition d'architecture, il ne sera pas nécessaire de prendre en compte la **Phase 0**, puisqu'elle correspond à l'organisation et l'infrastructure actuelle de *Gibberish*.

Cette étude prendra donc cet état de fait comme hypothèse de départ sur laquelle les deux prochaines phases viendront s'appuyer.

Ainsi, en considérant spécifiquement la **Phase 1** décrite dans le DDA, la réalisation de cette phase nécessitera l'acquisition de deux ressources matérielles supplémentaires, à savoir :

- *step 1.1*, un local dédié au  **Laboratoire de médias interactifs** au sein duquel tous les processus métier, nécessaires à la production de vidéos interactives, seront opérés ;
- *step 1.2*, une suite logicielle  **Interactive Media Application** correspondant aux outils nécessaires à la production de vidéos interactives.

Par la suite, en considérant spécifiquement la **Phase 2** décrite dans le DDA, celle-ci nécessitera l'acquisition de quatre ressources matérielles supplémentaires :

- *step 2.2*, un local hébergeant le  **service client** ;
- *step 2.2*, un local hébergeant le  **service médiathèques** ;
- *step 2.3*, un  **Système des gestions des droits utilisateurs** ;
- *step 2.4*, un dépôt d'  **Espace de stockage** pour les vidéos au contenu interactif.





### III.B. Ressources intellectuelles

Les ressources intellectuelles, nécessaires à ce projet de streaming de vidéos au contenu interactif, seront l'ensemble des processus métier, des événements et des choix utilisateurs possibles contenus au sein des séquences vidéo.











Ainsi, ces différents éléments seront présentés, tout comme dans le paragraphe précédent, selon une thématique temporelle selon les différentes phases du projet préconisées par cette étude.

Tout comme relaté précédemment, les éléments abstraits nécessaires à la phase étant déjà en place, ceux-ci ne seront énumérés ici, puisque déjà existant.

En considérant spécifiquement la **phase 1**, les processus métier ajoutés sont :

- *step 1.3*, la production de  **Vidéo interactive avec médias composites** ayant la responsabilité de rajouter des éléments contextuels hétérogènes, telles que des choix d'Histoire possibles à l'utilisateur ou différents angles de visionnage de ladite Histoire ;
- *step 1.3*, l'élaboration d'un  **Processus de production de vidéos interactives de marque et commerciale** ayant la charge de rassembler les différentes séquences produites par le  **Processus de vidéo interactive avec médias composites** et celle issues de la  **Production d'animations**, afin de les rassembler en une seule vidéo exhaustive proposant une Histoire de base.

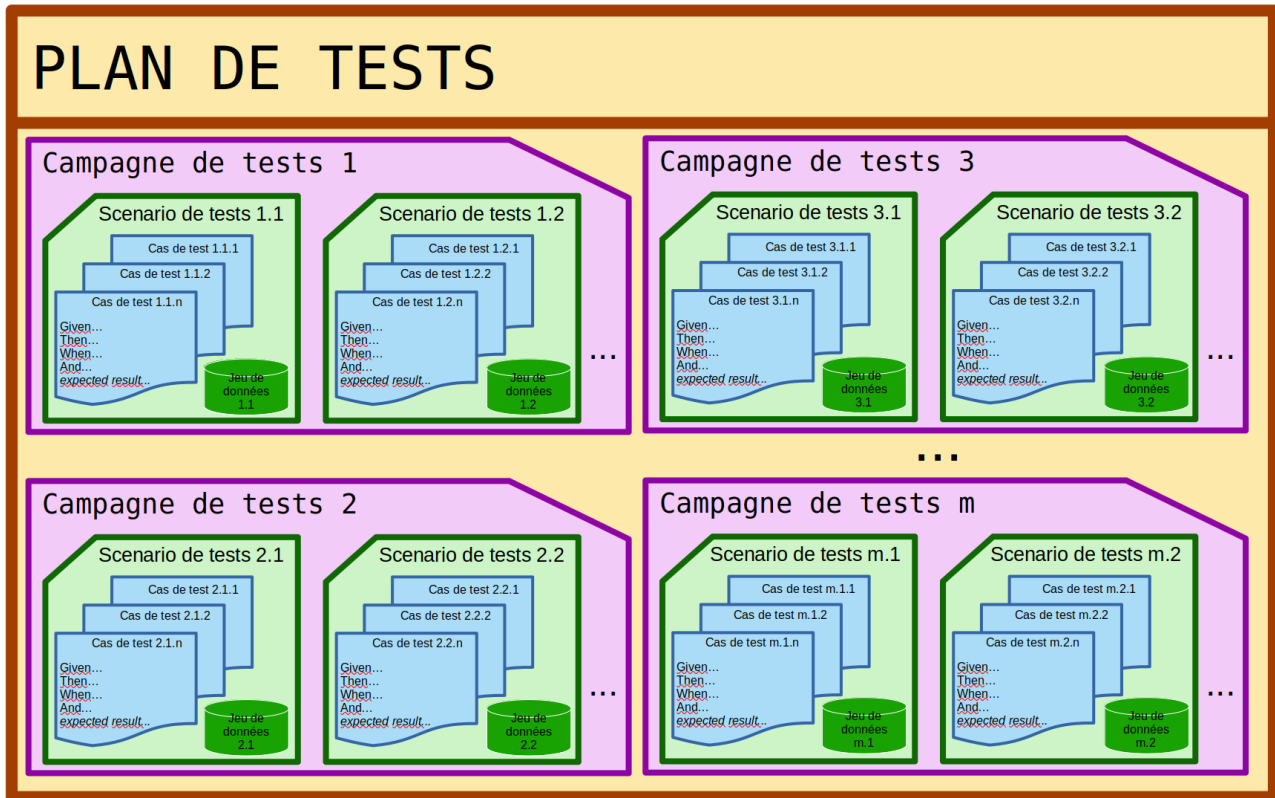
Enfin, en considérant spécifiquement la **Phase 2**, les entités intellectuelles abstraites à élaborer sont :

- *step 2.1*, la création de l' **Entreprise de médias** prenant en charge les  **Entreprise de vente de médias** et l' **Entreprise de production de médias** ;
- *step 2.1*, la création de l' **Entreprise de vente de médias** chargée de commercialiser les vidéos au contenu interactif produites par *Gibberish* ;
- *step 2.4*, la définition des différentes  **Balances de l'Histoire** proposant différentes orientations de l'Histoire à l'utilisateur ;
- *step 2.4*, la mise en place d'une fonctionnalité d'ajout des  **Avis et commentaires personnalisables** par l'utilisateur associés à une vidéo spécifique et contenu dans son profil ;
- *step 2.4*, la création d'un  **Index personnel** à chaque choix de l'utilisateur au sein d'une Histoire ;
- *step 2.4*, la mise en place d'un  **flux valeur** composé d'un  **Processus d'enregistrement** puis d'un  **Processus de téléchargement** pour mettre à disposition des utilisateurs toutes les vidéos proposées par *Gibberish*, selon les abonnements achetés par les clients.



## IV. Politique de tests

La conception du plan de tests associé à ce projet de streaming de vidéo interactive sera organisée selon le schéma ci-dessous :



Comme vous pourrez en prendre connaissance dans le schéma ci-dessus, le plan de tests sera constitué de :

- **plusieurs campagnes de tests**, dont chacune correspondra à une fonctionnalité spécifique ;
- **chaque campagne de tests sera découpée en un ou plusieurs scénarios de tests**, dont chacun représentera une mise en situation pour la campagne considérée ;
- **chaque scénario sera découpé en un ou plusieurs cas de tests**, dont chacun représentera un ou plusieurs cas d'usage précis pour le scénario considéré ;
- **chaque cas de test correspondra à un et un seul jeu de données** associé au cas de test considéré, lui-même associée au scénario de tests considéré, lui-même.

En suivant le découpage proposé ci-dessus, il sera également primordial de garder à l'esprit que la réalisation d'un test n'est pas une finalité : la réalisation d'un test correspond à la vérification qu'une fonction, un processus ou n'importe quelle entité conceptualisée, répond au besoin pour lequel il ou elle a été développé(e) de façon certaine, répétable et reproductible.

Néanmoins, avant d'arriver à ce résultat, l'origine du test débute toujours par la rédaction d'un plan de test. Ce dernier déterminera une base fixe du déroulement du test en question. Ainsi, il facilitera et organisera le déroulement et l'exécution des scénarios de tests lors d'une campagne ; tout comme la construction d'une maison nécessite un plan...

Pour concevoir ce plan, il sera alors nécessaire d'étudier, d'analyser, de poser des limites et des impacts, de déterminer des outils et une ou des technologies adaptées...tout en définissant le temps alloué. En d'autres termes plus succincts, il sera nécessaire d'adopter une méthodologie ET une méthode.

En ce qui concerne ce projet, en se basant sur les analyses réalisées au sein du cahier des charges et du document de définition d'architecture, il s'agira ici de définir les étapes du processus de test et d'identifier les risques afin de détecter le maximum d'anomalies possibles.

Suite, à ces différentes analyses, il s'agira alors de commencer à dessiner le plan en établissant un ordre dans lequel chaque composant sera complété, testé individuellement et, enfin, intégré avec les autres composants du système.

En ce sens, ce plan de test ne sera pas seulement informatif ; **le plan de test aura** aussi, ou plutôt, **une valeur DÉCLARATIVE.**

Ce plan de test permettra alors de définir ce qui sera testé, pourquoi le tester, comment ces tests s'effectueront, quand et qui testera. Il sera alors possible d'appréhender un tel document, comme le menu d'un restaurant ou le sommaire d'un livre.

De plus, le contenu d'un tel plan dépendra alors de la complexité du projet, et du niveau à partir duquel les tests seront initiés : des tests de fonctionnalités hautes pourront se baser sur une méthode *Business Driven Development*, des tests fonctionnels de vérification d'implémentation sur une méthode de *Behavior Driven Development* et des tests de programmation sur le *Test Driven Development*.

En ce qui concerne le contexte de cette étude commanditée par *Gibberish*, le présent document s'alignera sur la méthode utilisée au sein du cahier des charges et du DDA, à savoir le *Business Driven Development*.

En suivant cette méthodologie et sa méthode associée, le présent document s'efforcera d'être générique, fonctionnel et automatisable dans son implémentation, tout en proposant un plan de navigation, un niveau d'exigences et d'importance, permettant une compréhension globale de ses phases de réalisation.

En outre, la structure des différents éléments constituant ce plan sera indépendante, c'est à dire que chaque module sera indépendant l'un de l'autre. Ainsi, après avoir défini toutes les fonctionnalités à tester (cf *cahier des charges*), il s'avérera nécessaire d'organiser des campagnes de tests pour chacune des fonctionnalités définies ; **campagnes de tests** qui **seront** elles-mêmes **découpées** en **scénarios de test**, appelés aussi cas de test.

**Enfin, cette étude apportera, volontairement et sciemment, une contrainte à chaque cas de tests considéré: un cas de test ne testera qu'une et une seule situation ou résultat attendu, sur la base d'un contexte figé et fixé à l'avance. Ultérieurement, tous les cas de tests associées à ces mêmes fonctionnalités, utiliseront alors TOUJOURS le même contexte ou la même situation.**



## IV.A. Jeu de données

Le fait est que dans le contexte de *Gibberish*, des données il en existe déjà : ce projet démarre sur une base déjà existante qui va être enrichie de nouvelles fonctionnalités. Ainsi, face à l'exploitation d'une application en production, il y a (souvent) un besoin récurrent : comment avoir des jeux de données pour développer de nouvelles fonctionnalités et les tester AVANT de les déployer en production ?

La réponse à cette question pourra avoir trois options dans le contexte de ce projet au sein de *Gibberish*, options dont le choix final reviendra à l'ensemble des parties prenantes de la société. Ces trois options, énumérées ci-dessous, seront détaillées dans les paragraphes qui suivent, à savoir par :

- copie de la production,
- génération pure et dure de données,
- création d'un sous-ensemble de données à partir de la production - solution de compromis des deux options précédentes).

### IV.A.1. Copie de la production

La solution la plus simple et la plus directe pour créer un jeu de données est de simplement copier celles extraites de la production et de l'utilisation d'une application existante.

Néanmoins, bien que copier des données de production soit une solution de facilité, cela n'est pourtant pas toujours possible, ou simplement permis.

En effet, cette approche ne satisfait pas toujours aux exigences de sécurité, apportées aux clients de *Gibberish*, relativement à la RGPD décrite dans le DDA.

En pratique, les différents jeux de données ne doivent comporter aucune donnée qui soit issue de sa mise en production : ces données des clients doivent rester confinées dans un environnement hautement sécurisé.

Il sera alors nécessaire des jeux de données suffisamment exhaustives et réalistes et qui soient anonymisées, c'est à dire qui ne contiennent aucune information client.

Cette approche n'est donc pas toujours la plus « facile » à mettre en place.

### IV.A.2. Génération de données

Cette approche diffère de la précédente puisqu'à chaque fois qu'un jeu de données sera nécessaire pour le déroulement d'un test, celui-ci sera généré avec de fausses données préfabriquées à l'avance à cette intention.

Ces données pourront être décrites dans des fichiers plats ou générées une fois pour toutes à l'initialisation d'un nouveau scénario de tests.

L'avantage de cette approche est la fabrication et l'exécution d'un tel jeu de données peut être relativement rapide.

Néanmoins, il faudra bien prendre en considération que plus le modèle de données attendu sera complexe, plus sa maintenance le sera également de façon exponentielle, et non pas linéaire.

De plus, s'il est prévu que le modèle de données évolue souvent, cela peut devenir tout simplement ingérable, surtout si le jeu de données en contient un volume important.

Concrètement, cette approche nécessite de recréer une base locale de jeux de données à chaque scénario de test ; travail titanesque et chronophage.

### IV.A.3. Création d'un sous-ensemble à partir de la production

En ce qui concerne *Gibberish*, cette étude préconisera la construction d'un processus permettant de créer un sous-ensemble réduit et anonymisé de données à partir de la production.

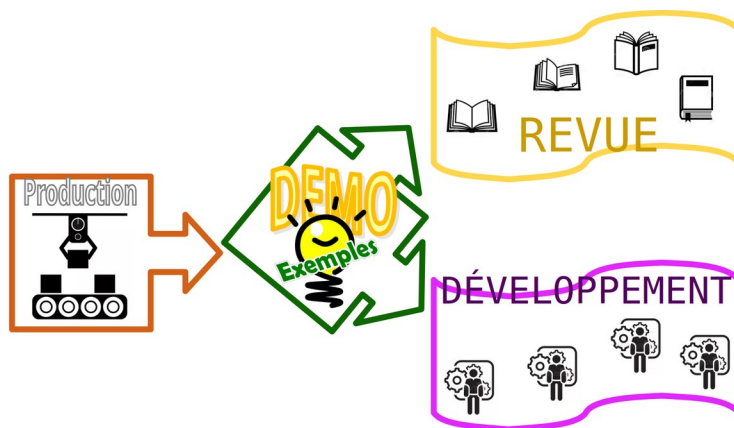
Cette approche permettra d'avoir des données réalistes et riches, en qualité comme en quantité, sans risque pour les données client.

D'une part, l'application chargée de ce processus sera utilisée quotidiennement par les clients, partenaires et équipes opérationnelles, et d'autre part l'équipe technique pourra développer de nouvelles fonctionnalités, à son rythme, en ayant toujours des jeux de données pérennes à disposition pour les tests.

Il faut tout de même être conscient que cette fonction va nécessiter des processus et des outils, en plus de la rigueur et de l'organisation indispensables. En outre, il faudra également ne pas perdre de vue que le cœur de ce système demeurera les **données**.

La mise en œuvre d'une telle solution consistera à maintenir une instance de démonstration, un prototype sommaire, dont le code sera le même que celui de l'instance de production, et dont les données sont différentes. En ce sens, ces dernières seront issues de la production, puis réduites en quantité et enfin anonymisées.

Cette approche, préconisée pour ce projet, présentera l'avantage de produire de nombreux jeux de données, complets, cohérents et réalistes, tout en leur retirant leur sensibilité, tel qu'imaginée dans le schéma ci-dessous :



## IV.B. Listes de contrôle

Les listes de contrôle proposées ci-dessous sont basées sur les architectures proposées au sein du document de définition d'architecture, elles sont donc également force de proposition face à l'absence d'éléments techniques concrets à la rédaction de cette étude. Ainsi, elles seront à compléter et à ajuster durant tout le projet afin d'être les plus exhaustives possibles lors de l'implémentation des composants.

Ces listes de contrôle couvriront les domaines suivants :

- les fonctionnalités et les technologies employées ;
- l'exécution des tests ;
- la convivialité ;
- les licences et les prix des outils logiciels utilisés ;
- le support et le service après-vente.

FONCTIONNALITÉS		✓
Technologies GUI de l'application à tester	<b>Applications Java</b> : Swing, JavaFX. AWT, SWT, Eclipse plug-ins, RCP, Applets, JavaWebStart, RIA, ULC, CaptainCasa pour les composants JavaFX, SubScene, JIDE Common Layer	
	<b>Applications Web</b> : Browser: Chrome, Firefox, Opera, Safari, Edge, Chromium basiert, Microsoft Edge Legacy, Internet Explorer; Versions, Headless Browser pour Chrome, Firefox et Edge (basé sur Chromium) Les outils devront supportés des environnements HTML 5, AJAX, Angular, React et Vue.js, mais aussi des toolkits concrets comme Smart, (GWT), ExtGWT, ExtJS, ICEfaces, jQuery UI, jQueryEasyUI, Kendo UI, PrimeFaces, Qooxdoo, RAP, RichFaces, Vaadin et ZK. D'autres toolkits pourront être intégrés avec peu d'effort par exemple: SAP UI5, Siebel Open, UI et Salesforce. Il faudra également tester des applications spécifiques tels que Electron et/ou Webswing...	
	<b>Applications Android</b> : Applications Android natives, applications web mobiles et applications hybrides sur Android à partir d'Android 7 sur des appareils réels et avec l'émulateur d'Android Studio	

## FONCTIONNALITÉS



Technologies GUI de l'application à tester	<b>Applications Windows</b> : Win32 classique, .NET (souvent développé en C#), Windows Presentation Foundation (WPF), Windows Forms, Windows Apps / Universal Windows Platform (UWP) avec éléments de contrôle XAML, applications C++ modernes (p.ex. avec Qt)	
	<b>Systèmes hybrides</b> considérant la combinaison des plusieurs technologies GUI, tels que des composants incorporés dans le navigateur (JavaFX WebView, JXBrowser, SWT-Browser)	
	Les documents PDF peuvent être testés comme n'importe quelle autre application (Vérifications textuelles et graphiques pour des éléments individuels)	
Support GUI selon le système d'exploitation	<b>Applications Java: Swing</b> et <b>JavaFX</b> : Windows, Linux, Unix, macOS <b>SWT</b> : Windows, Linux-GTK; Solaris-GTK sur demande.	
	<b>Applications Web</b> : Windows, Linux, macOS	
	<b>Applications Windows</b> : Windows	
Principe de test	Capture/Relecture pour la génération directe et efficace de séquences de test pour un traitement ultérieur dans des cas de test plus complexes avec contrôle de flux, paramétrage, modularisation et capacités de script avancées. Tous les tests et leurs environnements associés devront pouvoir être personnalisés.	
Structuration de test	Clair et concis grâce à la représentation graphique des cas de test et des nœuds d'action dans une structure arborescente. Les scénarios de tests pourront être structurés de manière modulaire en utilisant des suites de tests et des bibliothèques	
Reconnaissance des composants, robustesse des tests	Détection stable des composants, indépendamment des propriétés géométriques, même pour des éléments complexes comme les arbres et les tableaux dynamiques. Les tests sont robustes et tolérants aux modifications de l'interface graphique.	

## FONCTIONNALITÉS



Réutilisabilité, effort de maintenance	Haute réutilisabilité des modules de test grâce à une structure modulaire, par exemple par des procédures, l'encapsulation des accès aux composants, etc	
Tests basés sur des données	Importation directe de fichiers CSV ou Excel, utilisation de requêtes de base de données SQL, fichiers XML. Toute autre source peut être intégrée via une extension de script.	
Tests basés sur les mots-clés/ Tests basés sur le comportement	Utilisation de mots-clés (Keywords) pour mettre en œuvre et contrôler les cas de test, également au moyen de documents ou d'outils externes de spécification des tests (par exemple, Excel ou des outils de gestion des tests)	
Conteneurs Docker	Tous les composants devront supporter des tests conteneurisés via Docker ou tout autre système de conteneur	
Tests de charge et de performance	Tests de charge et de performance grâce à une exécution synchronisée et parallèle, même sur plusieurs machines. Pour le web en combinaison avec des outils comme JMeter ou NeoLoad.	
Protocoles, documentation des tests, Rapports	Des protocoles clairs et détaillés comprenant des captures d'écran de la situation d'erreur sont toujours générés. Des rapports configurables dans différents formats (HTML, XML, JUnit), la documentation des tests et des procédures peuvent être générés par simple pression d'un bouton ou automatiquement.	
JIRA/REST	JIRA et JIRA PlugIns comme TestRail, Zephyr, X-Ray, TM4J peuvent être intégrés via REST, souvent aussi via des outils CI comme Jenkins.	
Extensibilité par des scripts	Extensions de fonctions libres et contrôles/actions personnalisés par des scripts intégrés (Jython, Groovy et JavaScript). Via l'API de script, accès complet à tous les objets de l'application (SUT) et exécution de son propre code dans l'application ou dans le navigateur.	
Gestion des tests	Homogénéiser les fonctionnalités de base pour les petits scénarios de tests inclus dans chaque composant à tester.	
Intégration continue	Intégration disponible ou possible pour : ALM/QualityCenter de MicroFocus/HP, TestBench d'Imbus, QMettry, Klaros de Verit, TestLink, IBM Rational Quality Manager, Jira et Jira PlugIns tels que TestRail, Zephyr, X-Ray, TM4J, entre autres.	
Bureaux virtuels	Tous les types de bureaux virtuels existants, par exemple: Citrix, VMware, VirtualBox	
Gestion des versions	Bonne versionnabilité, par exemple via Git, SVN/Subversion, CVS, Mercurial, grâce au format XML des fichiers pertinents.	
Suivi des bogues	Possibilité de se connecter via des interfaces ouvertes et REST, par exemple Jira, MantisBT, Bugzilla	

## EXÉCUTION DES TESTS



Préparation des tests	Un assistant de démarrage rapide permet de générer une séquence de démarrage adaptée à l'application testée, en fonction de la technologie GUI sous-jacente.	
Préparation des scénarios de test	Gestion des dépendances pour la préparation et le post-traitement des tests pour des cas de test exécutables indépendamment, y compris la gestion automatique des erreurs.	
Points de vérification	L'enregistrement direct des contrôles standard, les vérifications spécifiques au client peuvent être mises en œuvre de manière variable par du scripting.	
Comparaison d'images	Enregistrement direct des contrôles d'image possible. De nombreux algorithmes, y compris pour les comparaisons d'images floues, et une vue différentielle pratique pour la vérification en cas d'écarts.	
Cartographie des objets	Les informations sur les composants sont stockées de manière centralisée pendant l'enregistrement dans une zone dédiée de la suite de tests, modifiable à tout moment. Mécanismes de recherche de références et de mise à jour automatique.	
Localisation intelligente des objets	Avec les SmartIDs (hashcode d'identifiant unique), les composants peuvent être adressés directement sur la base de propriétés caractéristiques, c'est-à-dire le label associé. La définition d'une portée permet de restreindre la recherche de l'objet, par exemple en cas d'éléments multiples.	
Composants génériques	Le mappage des composants spécifiques de l'interface graphique en composants génériques (boutons, champs de texte...) permet la réutilisation des tests entre les technologies ainsi que l'utilisation d'actions généralement valides sans avoir à capturer chaque composant individuel.	
Exécution du test via la ligne de commande	Exécution en mode batch possible avec des options de configuration étendues via des paramètres de ligne de commande, également pour l'intégration dans des environnements de construction.	
Exécution à distance	Exécution du test également sur des ordinateurs distants en mode daemon/service.	
Traitement des erreurs	Le traitement automatique des erreurs garantit la poursuite de l'ensemble du test sans interruption. Les erreurs sont enregistrées pour une analyse ultérieure.	
Débogueur de test	Fonctionnalité complète de débogage, y compris les points d'arrêt et l'analyse des variables.	

CONVIVIALITÉ		✓
Confort d'utilisation	Utilisation facile et intuitive avec une vue arborescente clairement structurée pour une édition pratique des cas de test, par exemple par copier/coller et glisser/déposer. Capture/relecture pour un démarrage rapide	
Connaissances préalables requises	Aucune connaissance en programmation n'est requise pour une utilisation standard.	
Travail en équipe	Pour les scripts, la connaissance des langages de script standard Jython, Groovy et JavaScript est utile.	

LICENCES ET PRIX		✓
Variante de produits	La multiplicité des environnements utilisés par les utilisateurs doivent apparaître dans les cas de tests et donc les cas d'usage. Ainsi, chaque environnement possible devra idéalement être disponible en différentes variantes de produits variables configurables pour les technologies d'interface graphique prises en charge, par exemple Swing, JavaFX, SWT, Web et Windows.	
Types de licences	Licence développeur - pour créer (et exécuter) des cas de test. Licence runtime - pour exécuter des tests (nocturnes).	
Mécanisme de licences	Toutes les licences devront être flottantes (c'est-à-dire qu'elles ne devront pas être liées à une personne spécifique). Les licences standard doivent fonctionner dans un réseau (local), pour une utilisation inter-réseau (internet), un serveur de licences de licences spécifiques devra être à disposition.	
Acheter ou louer	<i>Gibberish</i> proposera l'option d'achat et la location des vidéos sur une base annuelle, pour les licences de test de charge également des termes plus courts.	

SUPPORT ET SERVICE		✓
Télécharger et tester gratuitement	Téléchargement gratuit et anonyme de la version démo. La version de démonstration peut être exécutée sans enregistrement de la part d'un utilisateur.	
Installation et application portable	Installation facile sous Windows, macOS et Linux en quelques clics. Possibilité d'utiliser les différents outils via un navigateur web par webapps.	
Support	Assistance directe des développeurs et des testeurs de <i>Gibberish</i> en français, anglais et allemand.	

SUPPORT ET SERVICE		✓
Contrat de maintenance	Le contrat de maintenance (support + mises à jour) sur une base annuelle comprend du support par e-mail et par téléphone.	
Formation, Consultation	Formation ou consultation individuelle sur votre site. Formation ouverte régulièrement chez <i>Gibberish</i> . Toutes les offres sont également disponibles en ligne.	
Documentation	Manuel complet, tutoriel de démarrage, vidéos, blog, fonction de recherche en ligne, aide en ligne par clic droit dans les différents outils de Gibberish, FAQ générale et technique.	





## V. Fonctionnalités et composants testés

Dans ce paragraphe, les fonctionnalités décrites au sein du cahier des charges seront spécifiées et détaillées. De plus, pour chaque test d'application et chaque test de composant, les composants impliqués seront spécifiquement identifiés selon les libellés utilisés au sein de l'architecture cible présentée dans le document de définition d'architecture, telles que :

F1	Télécharger différents types de contenu, en utilisant des liens disponibles au sein même de la vidéo - ces liens pourront renvoyer sur des scènes spécifiques de la vidéo elle-même, ou pointer vers des ressources externes.	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"><li>• Processus de production de médias</li><li>• Vidéo interactive avec médias composite</li><li>• Histoire personnalisable</li><li>• Choix utilisateur</li><li>• Navigation dans l'Histoire</li><li>• Processus d'enregistrement</li><li>• Processus de téléchargement</li><li>• Espace de stockage</li></ul>	
	F1.1	Les vidéos à contenu interactif contiendront des liens vers d'autres ressources à la fois interne, telle qu'une vidéo contenue dans la médiathèque de <i>Gibberish</i> , ou externe vers des ressources accessibles sur d'autres plateformes.
		<b>Composants testés :</b> <ul style="list-style-type: none"><li>• Processus de production de médias</li><li>• Vidéo interactive avec médias composite</li><li>• HISTOIRE PERSONNALISABLE</li><li>• Choix utilisateur</li><li>• Navigation dans l'Histoire</li></ul>
F1.2	L'utilisateur pourra télécharger toutes les ressources proposées par les outils de <i>Gibberish</i> .	
	<b>Composants testés :</b> <ul style="list-style-type: none"><li>• Processus d'enregistrement</li><li>• Processus de téléchargements</li><li>• Espace de stockage</li></ul>	

F2	Prendre des décisions arbitraires à des moments clés de l’histoire - modifier l’histoire elle-même en fonction des choix de l'utilisateur.	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Choix utilisateur</li> <li>• Navigation dans l’Histoire</li> <li>• Balances de l’Histoire</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> <li>• Système de gestion des droits utilisateur</li> </ul>	
	F2.1	La vidéo doit contenir des points d’évènements permettant à l'utilisateur de faire un choix quant à la suite de l’histoire.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Choix utilisateur</li> <li>• Navigation dans l’Histoire</li> <li>• Balances de l’Histoire</li> </ul>
F2.2	Les choix de l'utilisateur doivent être enregistrés dans l’index personnel de l'utilisateur associé à la vidéo.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Choix utilisateur</li> <li>• Index personnel</li> <li>• Système de gestion des droits utilisateur</li> </ul>	

F3	Modifier les angles de visionnage des scènes, tout au long de la vidéo, à l'aide d'une rotation interactive tridimensionnelle (vidéo à 360°)	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Angle de visionnage personnalisable</li> <li>• Balances de l'Histoire</li> </ul>	
	F3.1	Les vidéos doivent contenir les informations nécessaires et suffisantes pour pouvoir visionner les scènes sous différents angles de vue.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Balances de l'Histoire</li> </ul>
F3.2	Des angles de vue prédéfinis sont mis à disposition de l'utilisateur, à des instants précis du déroulement de l'histoire de la vidéo.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Angle de visionnage personnalisable</li> </ul>	

F4	Permettre la navigation interactive de l'utilisateur pour se déplacer chronologiquement à n'importe quel moment de l'histoire - les éléments du menu ou de la table des matières devront permettre, lors de leur sélection, de passer directement au segment spécifique de la vidéo choisi par l'utilisateur	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Navigation dans l'Histoire</li> <li>• Choix utilisateur</li> <li>• Balances de l'Histoire</li> <li>• Index personnel</li> </ul>	
	F4.1	La vidéo dispose de points temporels identifiés, selon les scènes thématiques disponible au sein de l'Histoire.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• Vidéo interactive avec médias composite</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Balances de l'Histoire</li> </ul>
F4.2	L'utilisateur peut choisir et sélectionner un point temporel de l'Histoire, à n'importe quel moment, afin de s'y déplacer et de continuer le visionnage à ce moment.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Navigation dans l'Histoire</li> <li>• Choix utilisateur</li> <li>• Balances de l'Histoire</li> <li>• Index personnel</li> </ul>	

F5	Prendre en charge les retours d'expérience utilisateurs à l'aide d'une fonctionnalité de collecte et de publication d'avis – permettre à l'utilisateur de saisir des données et des commentaires pour chaque scène de la vidéo	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Avis et commentaire personnalisés</li> <li>• Espace de stockage</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> </ul>	
	F5.1	L'utilisateur peut rédiger des commentaires et/ou des avis associés à une vidéo.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Avis et commentaire personnalisés</li> <li>• Espace de stockage</li> </ul>
F5.2	L'utilisateur peut choisir de partager publiquement ses commentaires et/ou avis relatifs à une vidéo, ou décider de les conserver privés.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Choix utilisateur</li> <li>• Index personnel</li> </ul>	

F6	Enregistrer des séquences personnalisées, relatives aux choix décisionnels effectués par l'utilisateur - générer dynamiquement des vidéos personnalisées que l'utilisateur pourra partager, via des liens externes	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> <li>• Espace de stockage</li> <li>• Processus d'enregistrement</li> </ul>	
	F6.1	L'utilisateur peut choisir d'enregistrer tout ou partie de la vidéo qu'il visionne dans son espace personnel.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Processus de production de médias</li> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> <li>• Espace de stockage</li> <li>• Processus d'enregistrement</li> </ul>
F6.2	L'espace personnel de l'utilisateur doit être suffisamment dimensionné pour contenir un enregistrement personnalisé d'une vidéo à disposition.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Espace de stockage</li> <li>• Processus d'enregistrement</li> </ul>	

F7	Créer un index personnel, associé aux contenu privé de l'utilisateur, recensant tous les ajouts que celui-ci aura pu apporter au sein de l'histoire - cet index sera accompagné de filtres permettant de sélectionner spécifiquement chaque type d'ajout personnel, de les retrouver et les isoler du contenu de l'histoire d'origine	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Index personnel</li> <li>• Avis et commentaires</li> </ul>	
	F7.1	Pour chaque vidéo de chaque utilisateur, un index personnel doit être disponible pour y stocker des enregistrements et des commentaires et/ou avis associés à la vidéo en question.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Index personnel</li> <li>• Avis et commentaires</li> </ul>
F7.2	Le nombre d'index personnels à disposition d'un utilisateur est fonction du nombre de vidéos auxquelles il a accès.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Index personnel</li> </ul>	

F8	Gérer des droits utilisateur pour partager ou conserver chaque contenu personnalisé de façon privée et sécurisée – utilisation d’un système de mot de passe personnalisable	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> </ul>	
	F8.1	Un utilisateur doit pouvoir choisir de rendre ses contenus d’index personnels privé ou publiques.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Choix utilisateur</li> <li>• Index personnel</li> </ul>
F8.2	Un contenu interactif paramétré <i>privé</i> ne doit pas être accessible par un autre utilisateur. Un contenu interactif paramétré <i>public</i> doit pouvoir être visionné par un autre utilisateur.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• HISTOIRE PERSONNALISABLE</li> <li>• Index personnel</li> </ul>	

F9	Permettre d’acheter de l’espace de stockage supplémentaire, associé au compte de l’utilisateur, pour pouvoir stocker davantage de contenu personnel associé aux licences de visionnage détenues	
	<b>Ensemble des interactions entre composants testées :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Espace de stockage</li> </ul>	
	F9.1	Un utilisateur peut acheter et/ou louer de l’espace de stockage supplémentaire pour y héberger davantage d’enregistrements personnalisés.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Espace de stockage</li> </ul>
F9.2	La suppression d’une licence associée à une vidéo doit être accompagnée d’une suppression de l’espace de stockage associé à la vidéo.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> <li>• Espace de stockage</li> </ul>	

F10	Permettre d'acheter des licences de visionnage, spécifiques et temporaires, pour des périphériques autres que ceux déclarés par l'utilisateur.	
	<b>Ensemble des interactions entre composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> </ul>	
	F10.1	Les licences de visionnage d'un utilisateur ne sont associées qu'aux périphériques qu'il a lui-même déclarés au sein de son profil utilisateur.
		<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> </ul>
F10.2	Un utilisateur aura le choix d'acheter des licences de visionnage proportionnelles au nombres de périphériques sur lesquels il visionnera ses vidéos.	
	<b>Composants testés :</b> <ul style="list-style-type: none"> <li>• Système de gestion des droits utilisateurs</li> <li>• Système de gestion des achats vidéo</li> </ul>	





## VI. Tests de composants

Les tests d'acceptation de chaque fonctionnalité décrite précédemment suivront le modèle défini au sein du cahier des charges. En outre, les tests d'acceptation décrits au sein du cahier des charges étaient relatifs au scénario, alors que ceux décrits ci-dessous seront relatifs à cas de tests :

Campagne de tests 1 – Gérald ATTARD	
<b>Scénario 1</b>	<b><i>F1 : Télécharger différents types de contenu</i></b>
<b>Cas de tests 1.1</b>	<b><i>F 1.1 : la vidéo contient des liens vers d'autres ressources</i></b>
<b>Étant donné</b>	<i>un ensemble de ressources associé à une vidéo</i>
<b>Quand</b>	<i>la vidéo propose des liens vers un ou plusieurs ressources</i>
<b>Ensuite</b>	<i>l'utilisateur sélectionne un de ces liens</i>
<b>Résultat attendu</b>	<i>l'utilisateur est redirigé vers le nouveau contenu identifié par le lien</i>

Campagne de tests 1 – Gérald ATTARD	
<b>Scénario 1</b>	<b><i>F1 : Télécharger différents types de contenu</i></b>
<b>Cas de tests 1.2</b>	<b><i>F 1.2 : les outils proposés à l'utilisateur permettent de télécharger les ressources liées</i></b>
<b>Étant donné</b>	<i>un ensemble d'outils à la disposition de l'utilisateur</i>
<b>Quand</b>	<i>une vidéo propose de télécharger des ressources</i>
<b>Ensuite</b>	<i>l'utilisateur sélectionne une de ces ressources</i>
<b>Résultat attendu</b>	<i>la ressource sélectionnée est téléchargée avec un outil au choix de l'utilisateur</i>

Campagne de tests 2 – Gérald ATTARD	
<b>Scénario 2</b>	<i><b>F2 : Prendre des décisions arbitraires</b></i>
<b>Cas de tests 2.1</b>	<i><b>F 2.1 : la vidéo contient des points d'évènements</b></i>
<b>Étant donné</b>	<i>un ensemble de points d'évènements associé à une vidéo</i>
<b>Quand</b>	<i>la vidéo arrive à un point d'évènements</i>
<b>Ensuite</b>	<i>l'utilisateur choisit une orientation pour Son Histoire</i>
<b>Résultat attendu</b>	<i>l'Histoire visionnée par l'utilisateur correspond à ses choix</i>

Campagne de tests 2 – Gérald ATTARD	
<b>Scénario 2</b>	<i><b>F2 : Prendre des décisions arbitraires</b></i>
<b>Cas de tests 2.2</b>	<i><b>F 2.2 : l'enregistrement complète l'index personnel</b></i>
<b>Étant donné</b>	<i>une séquence vidéo spécifique choisie par l'utilisateur (temps début – temps fin)</i>
<b>Quand</b>	<i>l'utilisateur décide d'enregistrer cette séquence vidéo</i>
<b>Ensuite</b>	<i>l'enregistrement de la séquence est stockée dans l'espace de stockage personnel de l'utilisateur</i>
<b>Résultat attendu</b>	<i>l'index personnel de l'utilisateur associé à cette vidéo est mis à jour pour indiquer toutes les informations relatives à cet enregistrement</i>

Campagne de tests 3 – Gérald ATTARD	
<b>Scénario 3</b>	<b><i>F3 : Modifier les angles de visionnage</i></b>
<b>Cas de tests 3.1</b>	<b><i>F 3.1 : la vidéo contient les informations de visionnage sous différents angles</i></b>
<b>Étant donné</b>	<i>un ensemble de scènes pouvant être visionnées sous différents angles de vue</i>
<b>Quand</b>	<i>la vidéo propose à l'utilisateur de changer d'angle de visionnage</i>
<b>Ensuite</b>	<i>l'utilisateur choisit l'angle de visionnage qui lui convient le plus</i>
<b>Résultat attendu</b>	<i>la séquence vidéo est visionnée selon l'angle de vue choisi par l'utilisateur</i>

Campagne de tests 3 – Gérald ATTARD	
<b>Scénario 3</b>	<b><i>F3 : Modifier les angles de visionnage</i></b>
<b>Cas de tests 3.2</b>	<b><i>F 3.2 : la vidéo contient les informations de visionnage sous différents angles</i></b>
<b>Étant donné</b>	<i>un ensemble de scènes pouvant être visionnées sous différents angles de vue</i>
<b>Quand</b>	<i>la vidéo propose à l'utilisateur de changer d'angle de visionnage</i>
<b>Ensuite</b>	<i>l'utilisateur choisit l'angle de visionnage qui lui convient le plus parmi une sélection d'angles de visionnage prédéfinis.</i>
<b>Résultat attendu</b>	<i>la séquence vidéo est visionnée selon l'angle de vue choisi par l'utilisateur</i>

Campagne de tests 4 – Gérald ATTARD	
<b>Scénario 4</b>	<b><i>F4: Permettre la navigation interactive</i></b>
<b>Cas de tests 4.1</b>	<b><i>F 4.1 : la vidéo possède des points temporels</i></b>
<b>Étant donné</b>	<i>une vidéo composée de multiples séquences/scènes vidéo</i>
<b>Quand</b>	<i>la vidéo est sélectionnée par l'utilisateur pour être visionnée</i>
<b>Ensuite</b>	<i>l'utilisateur peut choisir un point temporel spécifique</i>
<b>Résultat attendu</b>	<i>la liste des points temporels proposés par la vidéo sont mis à la disposition e l'utilisateur dès le début</i>

Campagne de tests 4 – Gérald ATTARD	
<b>Scénario 4</b>	<b><i>F4: Permettre la navigation interactive</i></b>
<b>Cas de tests 4.2</b>	<b><i>F 4.2 : l'utilisateur choisit un point temporel</i></b>
<b>Étant donné</b>	<i>une vidéo composée de multiples séquences/scènes vidéo</i>
<b>Quand</b>	<i>un utilisateur visionne la vidéo</i>
<b>Ensuite</b>	<i>l'utilisateur peut choisir un point temporel spécifique</i>
<b>Résultat attendu</b>	<i>l'utilisateur visionne la séquence vidéo correspondant à ce point temporel</i>

Campagne de tests 5 – Gérald ATTARD	
<b>Scénario 5</b>	<b><i>F5 : Prendre en charge les retours d'expérience utilisateurs</i></b>
<b>Cas de tests 5.1</b>	<b><i>F 5.1 : l'utilisateur peut rédiger commentaires et avis</i></b>
<b>Étant donné</b>	<i>un utilisateur qui souhaite rédiger une appréciation relative à une vidéo</i>
<b>Quand</b>	<i>l'utilisateur rédige l'avis ou le commentaire relatif à la vidéo</i>
<b>Ensuite</b>	<i>l'avis ou le commentaire est stocké dans l'espace personnel de l'utilisateur</i>
<b>Résultat attendu</b>	<i>l'espace personnel associé à une vidéo contient tous les avis et/ou commentaires rédigés par l'utilisateur</i>

Campagne de tests 5 – Gérald ATTARD	
<b>Scénario 5</b>	<b><i>F5 : Prendre en charge les retours d'expérience utilisateurs</i></b>
<b>Cas de tests 5.2</b>	<b><i>F 5.2 : l'utilisateur peut partager ses commentaires relatifs à une vidéo</i></b>
<b>Étant donné</b>	<i>un utilisateur ayant rédigé une appréciation relative à une vidéo</i>
<b>Quand</b>	<i>l'utilisateur enregistre cet avis/commentaire relatif à la vidéo dans son index personnel</i>
<b>Ensuite</b>	<i>l'utilisateur peut choisir de publier cet avis/commentaire</i>
<b>Résultat attendu</b>	<i>un avis/commentaire identifié comme public est mis à la disposition des autres utilisateurs</i>

Campagne de tests 6 – Gérald ATTARD	
<b>Scénario 6</b>	<i><b>F6 : Enregistrer des séquences personnalisées</b></i>
<b>Cas de tests 6.1</b>	<i><b>F 6.1 : l'utilisateur peut enregistrer</b></i>
<b>Étant donné</b>	<i>une séquence vidéo</i>
<b>Quand</b>	<i>l'utilisateur enregistre cette séquence vidéo</i>
<b>Ensuite</b>	<i>la séquence vidéo est stockée dans l'espace personnel de l'utilisateur</i>
<b>Résultat attendu</b>	<i>l'espace personnel disponible de l'utilisateur est réduit de la taille de l'enregistrement réalisé</i>

Campagne de tests 6 – Gérald ATTARD	
<b>Scénario 6</b>	<i><b>F6 : Enregistrer des séquences personnalisées</b></i>
<b>Cas de tests 6.2</b>	<i><b>F 6.2 : l'espace personnel de l'utilisateur doit être suffisamment dimensionné</b></i>
<b>Étant donné</b>	<i>une séquence vidéo</i>
<b>Quand</b>	<i>l'utilisateur enregistre cette séquence vidéo</i>
<b>Ensuite</b>	<i>l'espace personnel de l'utilisateur contient cette séquence</i>
<b>Résultat attendu</b>	<i>l'espace personnel disponible de l'utilisateur doit être suffisamment dimensionné pour contenir cet enregistrement</i>

Campagne de tests 7 – Gérald ATTARD	
<b>Scénario 7</b>	<i>F7 : Créer un index personnel</i>
<b>Cas de tests 7.1</b>	<i>F 7.1 : un index personnel est disponible pour chaque vidéo</i>
<b>Étant donné</b>	<i>une vidéo à disposition d'un utilisateur</i>
<b>Quand</b>	<i>l'utilisateur accède à cette vidéo</i>
<b>Ensuite</b>	<i>un index personnel de l'utilisateur pour cette vidéo est créé</i>
<b>Résultat attendu</b>	<i>l'index personnel associé à cette vidéo est créé</i>

Campagne de tests 7 – Gérald ATTARD	
<b>Scénario 7</b>	<i>F7 : Créer un index personnel</i>
<b>Cas de tests 7.2</b>	<i>F 7.2 : un index personnel est associé à une vidéo</i>
<b>Étant donné</b>	<i>une vidéo disponible pour un utilisateur</i>
<b>Quand</b>	<i>l'utilisateur accède à cette vidéo</i>
<b>Ensuite</b>	<i>un index personnel, associé à cette vidéo, est créé</i>
<b>Résultat attendu</b>	<i>il y aura un index personnel pour chaque vidéo disponible à l'utilisateur</i>

Campagne de tests 8 – Gérald ATTARD	
<b>Scénario 8</b>	<b><i>F8 : Gérer des droits utilisateur</i></b>
<b>Cas de tests 8.1</b>	<b><i>F 8.1 : l'utilisateur choisit la portée de son index personnel</i></b>
<b>Étant donné</b>	<i>un index personnel associé à une vidéo est créé</i>
<b>Quand</b>	<i>l'utilisateur enregistre cet index personnel pour la première fois</i>
<b>Ensuite</b>	<i>Le système demande à l'utilisateur la portée de cet index personnel : privé ou public</i>
<b>Résultat attendu</b>	<i>l'index personnel associé à cette vidéo est créé</i>

Campagne de tests 8 – Gérald ATTARD	
<b>Scénario 8</b>	<b><i>F8 : Gérer des droits utilisateur</i></b>
<b>Cas de tests 8.2</b>	<b><i>F 8.2 : un contenu public est accessible par d'autres utilisateurs</i></b>
<b>Étant donné</b>	<i>un index personnel associé à une vidéo est déclaré public</i>
<b>Quand</b>	<i>Un autre utilisateur veut accéder à cet index personnel</i>
<b>Ensuite</b>	<i>l'utilisateur peut prendre connaissance du contenu de l'index personnel</i>
<b>Résultat attendu</b>	<i>un index personnel public est accessible par d'autres utilisateurs</i>



Campagne de tests 9 – Gérald ATTARD	
<b>Scénario 9</b>	<b><i>F9 : Permettre d’acheter de l’espace de stockage supplémentaire</i></b>
<b>Cas de tests 9.1</b>	<b><i>F 9.1 : l’utilisateur souhaite augmenter la taille de son espace de stockage</i></b>
<b>Étant donné</b>	<i>l’espace de stockage d’un utilisateur</i>
<b>Quand</b>	<i>l’utilisateur souhaite augmenter la taille de son espace de stockage</i>
<b>Ensuite</b>	<i>le système propose différentes tailles supplémentaires à l’utilisateur</i>
<b>Résultat attendu</b>	<i>l’espace de stockage de l’utilisateur est augmenté de la taille choisie</i>

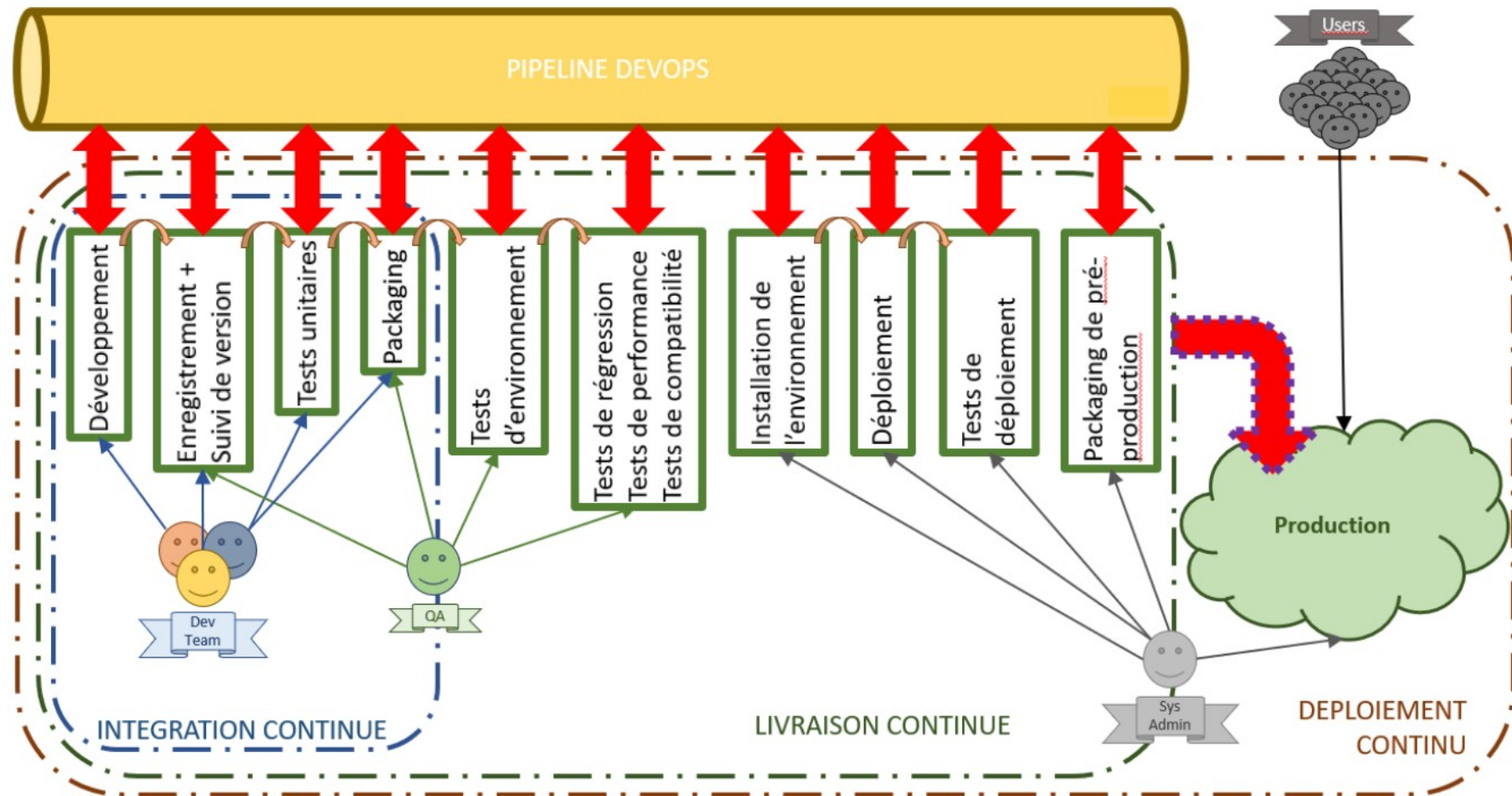
Campagne de tests 9 – Gérald ATTARD	
<b>Scénario 9</b>	<b><i>F9 : Permettre d’acheter de l’espace de stockage supplémentaire</i></b>
<b>Cas de tests 9.2</b>	<b><i>F 9.2 : la suppression d’une licence de visionnage supprime l’espace de stockage associé</i></b>
<b>Étant donné</b>	<i>l’espace de stockage d’un utilisateur</i>
<b>Quand</b>	<i>l’utilisateur supprime une licence associée à une vidéo</i>
<b>Ensuite</b>	<i>l’espace de stockage associé est supprimé</i>
<b>Résultat attendu</b>	<i>la licence supprimé et l’espace de stockage associé ne sont plus disponibles</i>

Campagne de tests 10 – Gérald ATTARD	
<b>Scénario 10</b>	<b><i>F10 : Permettre d'acheter des licences de visionnage</i></b>
<b>Cas de tests 10.1</b>	<b><i>F 10.1 : une licence de visionnage est associée qu'aux périphériques déclarés</i></b>
<b>Étant donné</b>	<i>Les périphériques déclarés par un utilisateur</i>
<b>Quand</b>	<i>l'utilisateur souhaite visionner une vidéo</i>
<b>Ensuite</b>	<i>L'utilisateur doit utiliser un des périphériques déclarés</i>
<b>Résultat attendu</b>	<i>la vidéo n'est que sur un périphérique qui a été déclaré par l'utilisateur</i>

Campagne de tests 10 – Gérald ATTARD	
<b>Scénario 10</b>	<b><i>F10 : Permettre d'acheter des licences de visionnage</i></b>
<b>Cas de tests 10.2</b>	<b><i>F 10.2 : les licences de visionnage achetées seront fonction des périphériques déclarés</i></b>
<b>Étant donné</b>	<i>un ensemble de périphérique déclaré par l'utilisateur</i>
<b>Quand</b>	<i>l'utilisateur souhaite ajouter un nouveau périphérique de visionnage</i>
<b>Ensuite</b>	<i>l'utilisateur doit s'acquitter du prix d'achat pour cette nouvelle licence</i>
<b>Résultat attendu</b>	<i>L'utilisateur peut visionner l'ensemble des vidéos à sa disposition sur ce nouveau périphérique</i>

## VII. Infrastructure préconisée

En se basant sur l'architecture SOA ,définie au sein du DDA, la méthode de développement préconisée sera une méthode AGILE, dont les différentes phases sont présentées ci-dessous :



Au sein du diagramme précédent, les campagnes de tests seront réalisées lors des processus de :

- Intégration Continue, par l'équipe de développement (*Dev Team*) ;
- Livraison Continue par les responsables de l'assurance Qualité (*QA*).

Lors de ces deux précédentes phases, les responsables de tests le réaliseront à des niveaux différents.

En effet, l'équipe de développement réalisera plutôt des tests unitaires, c'est à dire des tests leur permettant de vérifier qu'un extrait de code fonctionne correctement. L'équipe pourra alors utiliser une méthode de type TDD.

Alors que l'équipe d'assurance Qualité se cantonnera davantage vers des tests fonctionnels, c'est à dire des tests servant à s'assurer que toutes les fonctionnalités développées répondent bien au(x) besoin(s). Ces tests visent à démontrer que le logiciel effectue ce pourquoi il a été développé, tout en recherchant d'éventuelles défaillances lorsque le testeur le sollicite sur des valeurs nominales, aux seuils ou hors domaine de définition. Ainsi, peu importe le contexte ou le paramétrage utilisateur demandé, le logiciel doit rester cohérente et stable, tout en renvoyant des données licites autorisées par les développeurs eux-mêmes.

Dans le contexte de tests fonctionnels, ces derniers peuvent être de deux types :

test fonctionnel partiel : dans le contexte de cette étude, en utilisant des méthodologies Agile, c'est ce genre de tests qui seront rencontrés principalement, puisqu'ils seront associées à des livraisons partielles modulaires.

Test fonctionnel total : ce genre de tests n'interviendront alors que lors de la livraison TOTALE du produit, peu importe la méthodologie concernée (Agile ou cycle en V).

## VII.A.Le pipeline DevOps

Le premier élément à décrire dans un pipeline DevOps est le pipeline lui-même.

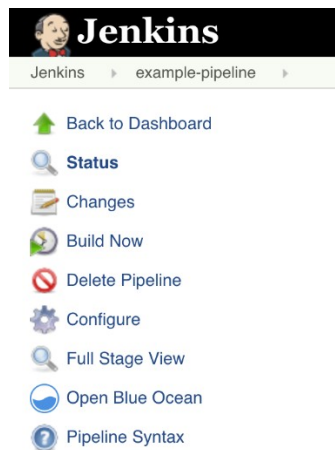
Un pipeline DevOps est un ensemble de pratiques pour lesquelles les équipes de développeurs (Dev) et les équipes d'opérationnels (Ops) concrétisent, testent, et déploient un produit logiciel plus vite et plus facilement que s'ils devaient le faire manuellement, car tout l'intérêt du pipeline réside dans l'automatisation de ces tâches : les développeurs peuvent alors consacrer plus de temps à développer.

Une des premières caractéristiques d'un pipeline est de conserver le process de développement logiciel organisé et focalisé. Ainsi un pipeline DevOps va séquentiellement réaliser les étapes suivantes :

- Planifier ;
- Développer ;
- Compiler ;
- Tester ;
- Déployer ;
- Monitorer (action technique de surveillance d'un sujet).

En termes d'outillage, le choix préconisé s'orientera vers un *Pipeline as Code with Jenkins*, plus communément appelé simplement *Pipeline*.

En effet, *Jenkins Pipeline* est une suite de plugins supportant l'implémentation de l'intégration continue et de la livraison continue au sein d'un pipeline standard déclaré dans une instance standard de *Jenkins*. *Pipeline* fournit ainsi un ensemble extensible d'outils pour la modélisation « *Simple-To-Complex* » d'une pipeline de livraison « *as code* » via le plugin *Pipeline DSL (Domain Specific Language)*.



En termes de prérequis, l'utilisation de *Pipeline* nécessite :

- Une instance de Jenkins 2.x ou supérieur ;
- L'installation du plugin Pipeline (plugin suggéré par défaut à l'installation de Jenkins).

## **VII.B.L'intégration continue**

### **VII.B.1. Développement**

#### ***VII.B.1.a. Phase d'analyse et de conception***

La phase d'analyse est une étape essentielle dans la conception d'un logiciel et un prérequis au développement.

Néanmoins, il est vrai qu'elle n'apparaît pas dans la plupart des diagrammes d'activités techniques car elle est considérée comme une partie réalisée en amont du développement, voire pendant.

#### ***VII.B.1.b. Méthode d'analyse et de conception***

En ce qui concerne la méthode d'analyse, l'*UML* sera à préconisé pour le projet de *Gibberish*.

En effet, UML est particulièrement utilisé et adapté en Génie Logiciel et en conception orientée objet. Son principal intérêt est de modéliser GRAPHIQUEMENT les différents éléments constituant un système, selon 3 types de diagramme :

- des diagrammes de structure,
- des diagrammes de comportement,
- des diagrammes d'interaction.

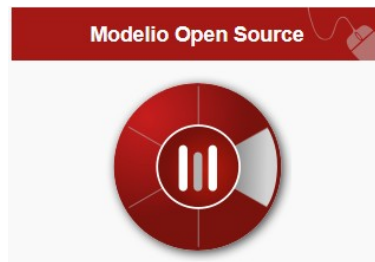
Les principaux diagrammes mentionnés précédemment sont détaillés dans le tableau ci-dessous.

Catégorie	Type de diagramme	Description
Diagramme de structure	Diagramme de classes	Représente les classes
	Diagramme d'objet	Affiche l'état d'un système à un moment donné
	Diagramme des composants	Affiche les dépendances et les composants de structure
	Diagramme de structure composite	Divise les modules ou les classes en leurs composantes et clarifie leurs relations.
	Diagramme de paquetage	Regroupe les classes en paquets, représente la hiérarchie et la structure des paquets
	Diagramme de déploiement	Affiche la distribution des composants aux nœuds de l'ordinateur
	Diagramme de profil	Illustre les relations d'usage au moyen de stéréotypes, de conditions aux limites, etc.
Diagrammes de comportement	Diagramme de cas d'utilisation	Représente divers usages
	Diagramme d'activité	Décrit le comportement de différents processus (parallèles) dans un système.
	Diagramme d'état-transition	Documente la façon dont un objet passe d'un état à un autre par le biais d'un événement.
Diagrammes d'interaction	Diagramme de séquence	Représente le moment des interactions entre les objets.
	Diagramme de communication	Affiche la répartition des rôles des objets au sein d'une interaction.
	Diagramme de temps	Démontre la limitation temporelle pour les événements qui mènent à un changement d'état.
	Diagramme d'aperçu d'interaction	Montre comment les séquences et les activités interagissent.

### VII.B.1.b.i. Outil d'analyse

*Modelio* est un outil de modélisation UML, intégrant également la modélisation du BPMN. Il propose ainsi un ensemble d'outils supportant la démarche MDA et est compatible avec la norme UML 2.0.

Modelio permet ainsi de générer tous les diagrammes UML décrits dans le paragraphe précédent et permet d'étendre ses fonctionnalités « de base » en ajoutant des extensions de modélisation telles que le BPMN, l'architecture entreprise, l'analyse des objectifs, la définition des dictionnaires, le SOA, le SysML...



### VII.B.1.c. Outil de développement

L'environnement de Développement sera à choisir en fonction de la nature du projet et du type de technologie appréhendée.

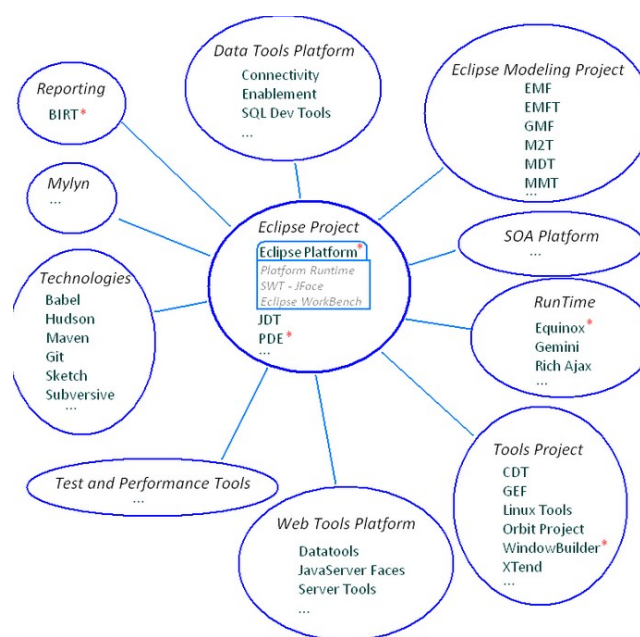
Dans le cas d'un projet JAVA, l'environnement préconisée est Eclipse, développé par la Fondation

Eclipse (<https://www.eclipse.org/>)



Eclipse est à la fois un IDE, un framework et une plateforme de développement, selon que l'on considère le projet, ses composants ou la résultante de leur assemblage.

Le diagramme suivant présente une vue technique de ses possibilités :



\* Eclipse Projects relevant to RCP



## VII.B.2. Suivi de version

Un logiciel de gestion de version est composé d'une arborescence de fichiers permettant de conserver toutes les versions des fichiers, ainsi que leur historique de différence entre chaque mise à jour (*check out*).

Cette notion de suivi de version s'accompagne intrinsèquement par la notion de branche de version ou branche de développement.

Avant d'aborder le type de branchement, préconisé au sein des *Gibberish* parmi tous les types de branchement existant, il est nécessaire de décrire l'objectif de la société : **Avoir le minimum de branches possibles.**

A ce stade, il est normal de se poser la question quant à la légitimité d'un tel axiome. Après tout, avoir beaucoup de branches différentes « *devrait être* » synonyme de flexibilité ou de persistance. Par expérience, il n'en est pas du tout ainsi.

En effet, le fait d'avoir beaucoup de branches engendrent beaucoup de gaspillage de temps et de ressources, et c'est précisément cela que le DevOps s'efforce de minimiser, voire de supprimer.

Aussi, afin de réduire le champ de divergence entre branches et améliorer la Qualité ainsi que la productivité de reprise de code, cette étude se bornera à avoir le minimum de branche de version possible...autant que faire se peut et en fonction des briques fonctionnelles développées.

### VII.B.2.a. Les branches de développement

#### VII.B.2.a.i. Un mouvement en deux temps

Le mouvement général du développement à plusieurs équipiers, ou à plusieurs équipes, est toujours un mouvement en deux temps :

- un temps de développement à l'abri des modifications simultanées des autres ;
- un temps d'intégration.

Du point de vue du gestionnaire de sources, la création d'une « *branche* » dans laquelle on enregistre les versions successives est le procédé qui permet d'isoler les développements de ceux d'autres développeurs, ou de séparer les versions de deux équipes de développement distinctes.

#### VII.B.2.a.ii. Choisir son tempo

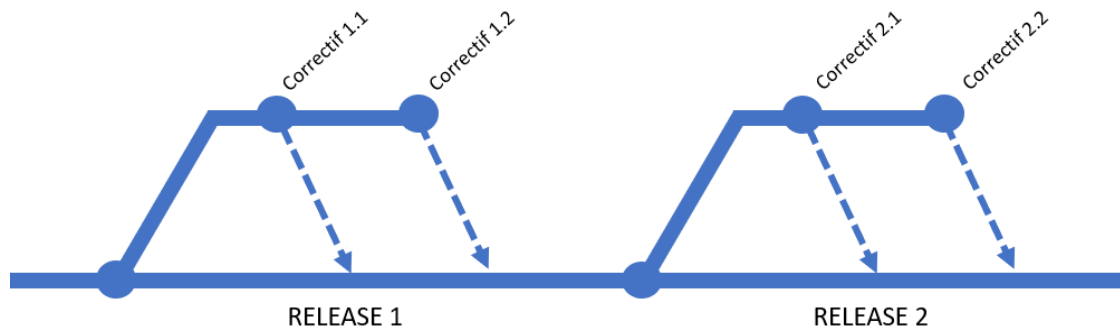
La question restante porte alors sur le rythme, c'est à dire la cadence à donner à ce mouvement de va-et-vient : faut-il produire beaucoup, se protéger longtemps et fusionner au plus tard, ou au contraire consolider souvent ses travaux en tant qu'équipier et ses développements en tant qu'équipe ?

Du point de vue de *Gibberish*, cette étude privilégiera la consolidation périodique et rapprochée afin d'obtenir une itération de codage de l'écart la plus brève possible, la fusion la plus fréquente et la pulsation la plus rapide possible.

De toutes les options de branchement existantes et en suivant l'axiome de Qualité que *Gibberish* s'impose au travers de sa politique d'entreprise, énoncée dans le cahier des charges, cette étude privilégiera ainsi le branchement par *release* adapté au contexte industriel.

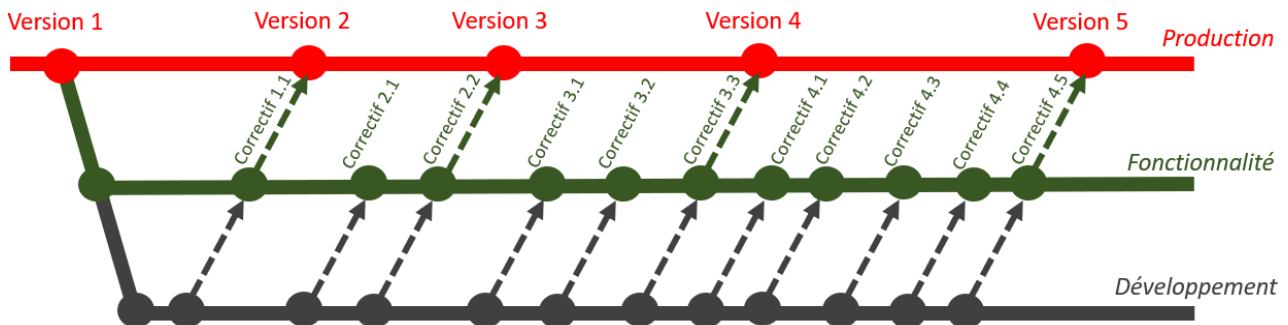
### VII.B.2.b. Le branchement par *release*

Par définition, l'intégration continue se dispense de branches sauf lors des mises en production. Ainsi, une branche de version d'Intégration Continue « *pure* » devrait ressembler au schéma suivant :



Néanmoins, dans un contexte industriel tel que celui de *Gibberish*, cela reste utopique ; il n'en est pas moins vrai, qu'en terme d'Amélioration Continue, cette étude adoptera un modèle y tendant, voire y ressemblant fortement.

Aussi, la proposition ci-dessous se veut être une combinaison de l'axiome décrit précédemment (« avoir le minimum de branche possible ») tout en prenant en compte l'aspect opérationnel sur lequel l'appliquer, c'est un consensus des deux propositions :




Ainsi, le consensus obtenu dans l'exemple ci-dessus peut se décomposer de 3 branches :

- production ou pré-production selon le contexte du projet ;
- fonctionnalité appelée couramment *release* ;
- développement - cette branche est la somme des branches locales de chaque développeur de ou des équipe(s)).

En outre, chaque version est considérée par un numéro Majeur de version et chaque *release* est considérée par un numéro Mineur de version ; en prenant comme base d'exemple le schéma précédant, la version 5 correspond à la *release* 4.5 .

### VII.B.2.c. Outil

Cette étude n'ayant pas encore les spécifications techniques des outils de développement utilisés chez *Gibberish*, elle préconisera l'outil gratuit de gestion de version nommé *Tuleap*  **tuleap** de la société *Enalean SAS*©.

Tuleap est un outil offrant plusieurs fonctionnalités s'intégrant dans le pipeline DevOps, dont notamment une compatibilité avérée avec *Jenkins* présenté dans un paragraphe précédent.

En ce qui concerne le suivi de version, *Tuleap* intègre nativement un module d'intégration de branches *Git*, *GitHuh*, *svn* et *cvs* en son sein.

Ainsi, quand un « *push* » (*check out*) est effectué dans un dépôt *Tuleap-git*, la tâche liée au *Jenkins* sera automatiquement réalisée.

En terme de dépôts, *Tuleap-git* en proposent deux types :

- les projets de référence : branche par défaut, se focalisant sur l'enregistrement et le suivi des versions officielles des dépôts de chaque projet ;
- les projets clone (*fork*) : chaque membre du projet peut cloner la branche référence afin de la publier dans son espace de travail personnel. Ceci permet de publier et d'intégrer le développement de modèle personnel tout en protégeant le branche référence du projet. Le *check-out* sera alors réalisé quand le développement personnel sera mature et aura passé tous les tests unitaires avec succès.

Pour plus de détails, les sources de cette étude sont issues de <https://plugins.jenkins.io/tuleap-git-branch-source/>



### VII.B.3. Package

Le terme *Package* est utilisé ici en termes d'archive et non pas en termes programmatique d'organisation ou de structuration du code lui-même.

Ainsi, lorsque le produit logiciel sera suffisamment mature et possèdera les fonctionnalités minimales pour en effectuer une première version, un *package* sera créé afin d'obtenir une archive de l'incrément réalisé.

Concrètement, en prenant comme base un projet à dominante Java, et suite à l'écriture du code constituant le produit logiciel lui-même, le principal outil sera naturellement le binaire « *jar* » inclus au sein du *JDK* (*Java Development Kit*).

```
jar
Usage: jar [OPTION...] [ [--release VERSION] [-C dir] files] ...
Try `jar --help' for more information.

jar --help
Usage: jar [OPTION...] [ [--release VERSION] [-C dir] files] ...
jar creates an archive for classes and resources, and can manipulate or
restore individual classes or resources from an archive.

Examples:
# Create an archive called classes.jar with two class files:
jar --create --file classes.jar Foo.class Bar.class
# Create an archive using an existing manifest, with all the files in foo/:
jar --create --file classes.jar --manifest mymanifest -C foo/ .
# Create a modular jar archive, where the module descriptor is located in
# classes/module-info.class:
jar --create --file foo.jar --main-class com.foo.Main --module-version 1.0
  -C foo/ classes resources
# Update an existing non-modular jar to a modular jar:
jar --update --file foo.jar --main-class com.foo.Main --module-version 1.0
  -C foo/ module-info.class
# Create a multi-release jar, placing some files in the META-INF/versions/9 directory:
jar --create --file mr.jar -C foo classes --release 9 -C foo9 classes
```



## **VIII. Hypothèses et risques**

### **VIII.A. Hypothèse en Ressources Humaines**

Tel qu'il a été présenté dans le cahier des charges, le document de définition et d'architecture ainsi que dans le présent plan de tests, l'architecture préconisée se basera sur la SOA.

Aussi, pour mettre en place cette architecture techniquement, Gibberish devra constituer une équipe de développement, de préférence AGILE pour la concrétiser et la mettre en œuvre.

En fonction des ressources mises à disposition, l'équipe de QA chargée des tests fonctionnels, aussi bien d'intégration que de livraison, devrait à minima être constituée de trois ETP.

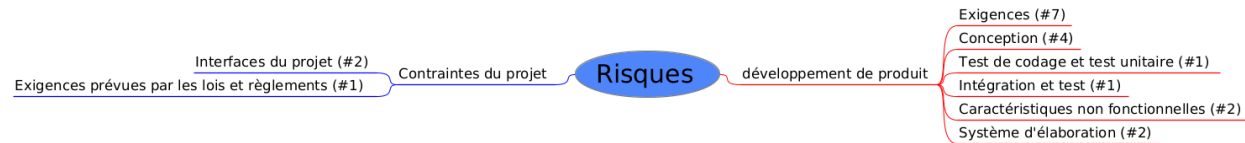
### **VIII.B. Hypothèse temporelle**

En ce qui concerne, la durée des campagnes de tests définis, celles-ci seront à évaluer en fonction du nombre de cas de tests dénombrés au sein de chaque scénario de la campagne.

Pour prévoir une estimation moyenne d'une campagne de tests, il serait souhaitable de partir sur une durée de trois semaines par campagne (et donc par fonctionnalité testée) pour un effectif de trois ETP, tel que préconisé dans le paragraphe précédent.

## VIII.C. Risques

Les principaux domaines et sous-domaines de risque pris en compte sont répertoriés ci-dessous selon 2 domaines principaux ‘*Contraintes du projet*’ et ‘*développement de produit*’.

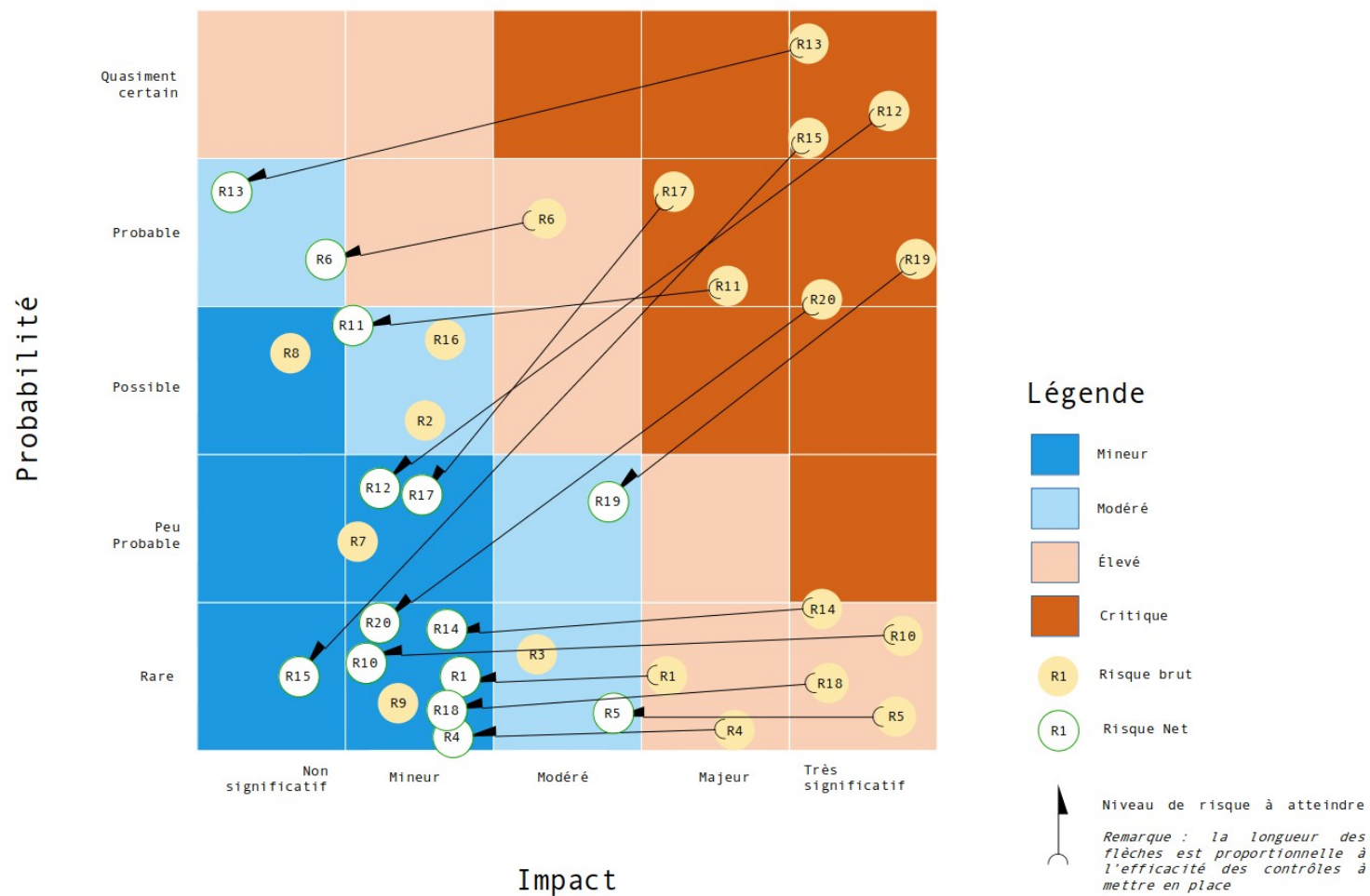


Id.	Risque	Description du risque	Type de risque	Domaine du risque	Impact	Probabilité	Action	Responsabilité	Date de l'examen	Efficacité des contrôles
R1	Stabilité	Stabilité du système précaire	Exigences	Développement de produit	Majeur	Rare (<10%)	S'assure de la qualité et de la stabilité des matériels et logiciels choisis	Marie M		
R2	Exhaustivité	Choix réduit des services proposés par l'extranet	Exigences	Développement de produit	Mineur	Possible (30-50%)	Le système doit proposer les mêmes ressources que le précédent	Alex Z		
R3	Clarté	Manque de clarté des services offerts	Exigences	Développement de produit	Modéré	Rare (<10%)	Le nouveau système doit être tout aussi convivial que l'ancien	Alex Z		
R4	Validité	Informations affichées non valides ou périmées	Exigences	Développement de produit	Majeur	Rare (<10%)	Les informations affichées doivent être actuelles, exactes et valides	Marie M		
R5	Faisabilité	Services proposés dépassent le budget	Exigences	Développement de produit	Très significatif	Rare (<10%)	Estimer l'adéquation de la réalisation des services relativement au budget à disposition	Pierre PARKER		
R6	Unicité	Informations dupliquées et/ou contradictoires	Exigences	Développement de produit	Modéré	Probable (50-90%)	S'assurer que les informations sur l'extranet sont issues d'une source uniques	Pierre PARKER		
R7	Ampleur	Ressources technologiques sous-évaluées	Exigences	Développement de produit	Mineur	Peu probable (10-30%)	Les ressources techniques doivent être au moins dimensionnées pour accueillir les partenaires, les fournisseurs et les clients	Alex Z		
R8	Fonctionnalité	Fonctionnalités inadaptées	Conception	Développement de produit	Non significatif	Possible (30-50%)	Les fonctionnalités devront être systématiquement validées par le client sur site de l'équipe AGILE	Gérald ATTARD		
R9	Difficulté	Difficulté de conception et/ou de réalisation rendant le projet irréalisable	Conception	Développement de produit	Mineur	Rare (<10%)	S'assurer que les besoins exprimés par <i>Gibberish</i> sont fonctionnellement et techniquement réalisables	Marie M		
R10	Interfaces	IHM confuses	Conception	Développement de produit	Très significatif	Rare (<10%)	Les IHM de l'extranet devront fournir les mêmes services que l'ancien SI	Alex Z		

Id.	Risque	Description du risque	Type de risque	Domaine du risque	Impact	Probabilité	Action		Responsabilité	Date de l'examen	Efficacité des contrôles
R11	Contraintes informatiques	Contraintes techniques relatives à l'existant non prises en compte	Conception	Développement de produit	Majeur	Probable (50-90%)	L'extranet devra être compatibles avec les technologies employées par le SI existant		Pierre PARKER		
R12	Mise à l'essai	Les fonctionnalités ne sont pas testées	Test de codage et test unitaire	Développement de produit	Très significatif	Quasiment certain (>90%)	Mettre en place une politique de TDD pour assurer la qualité du code		Pierre PARKER		
R13	Environnement	Non prise en compte de l'environnement technique	Intégration et test	Développement de produit	Très significatif	Quasiment certain (>90%)	Analyser exhaustivement les structures matérielles et logiques lors de l'étude de l'existant		Pierre PARKER		
R14	Fiabilité	Informations affichées non pertinentes ou erronées	Caractéristiques non fonctionnelles	Développement de produit	Très significatif	Peu probable (10-30%)	S'assurer de la pertinence des informations affichées pour fournir des services de qualité		Gérald ATTARD		
R15	Sécurité	Permissivité de connexion	Caractéristiques non fonctionnelles	Développement de produit	Très significatif	Quasiment certain (>90%)	S'assurer de l'activation des comptes et de l'intégrité des utilisateurs accédant à l'extranet		Gérald ATTARD		
R16	Connaissance	Les besoins métiers ne sont pas répondus	Système d'élaboration	Développement de produit	Mineur	Possible (30-50%)	Les services rendus par l'extranet doivent être au moins équivalents à ceux traduits par le SI existant		Marie M		
R17	Convivialité	Services de l'extranet brouillon	Système d'élaboration	Développement de produit	Majeur	Probable (50-90%)	Les services de l'extranet doivent pouvoir être accessibles avec le moins de clic possible		Gérald ATTARD		
R18	Entrepreneurs délégués	Représentant de compte non identifiés	Interfaces du projet	Contraintes du projet	Très significatif	Rare (<10%)	Identifier les représentants de compte pour les fournisseurs, les partenaires et les clients		Alex Z		
R19	Entrepreneur principaux	Fournisseurs, Partenaires et Client non identifiés	Interfaces du projet	Contraintes du projet	Très significatif	Probable (50-90%)	Identifier les représentants de compte pour les fournisseurs, les partenaires et les clients		Alex Z		
R20	Respect de la vie privée	Informations saisies accessibles publiquement	Exigences prévues par les lois et règlements	Contraintes du projet	Très significatif	Probable (50-90%)	Les informations relatives aux entreprises et aux différents intervenants doivent être sécurisées autant pendant les phases de transaction que celle de stockage.		Gérald ATTARD		

## VIII.C.1. Cartographie des risques

La cartographie présentée ci-dessous fait référence au paragraphe §Risques détaillé supra.





GIBBERISH