Université du Québec
École de technologie supérieure

COURS MGL 804

SUJET : ÉVALUATION DE LA MAINTENABILITÉ DES PRODUITS LOGICIELS DU CCI

RAPPORT FINAL

Franklin Kamsong

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MONTRÉAL
HIVER 2012

TABLE DES MATIÈRES

LISTE DE FIGURES

INTRODUCTION

La norme ISO/IEC 25040 fait partie de la série standard SQuaRE et contient les exigences générales pour l'évaluation de la qualité des produits logiciels. La norme ISO/IEC 25040 a pour but de définir des exigences et des recommandations pour l'évaluation de la qualité des produits logiciels, de fournir la description du processus pour l'évaluation de la qualité ainsi que de préciser les concepts généraux. Le processus décrit dans la norme peut être utilisé pour l'évaluation des différents types de logiciel et peut être appliqué pendant ou après le processus de développement.

CHAPITRE I PRÉSENTATION DE LA NORME ISO/IEC 25040

1. Introduction

La série de normes SQuaRE (Systems and software Quality Requirements and Evaluation) est développée afin de remplacer progressivement la série de normes ISO/IEC 9126 et la série ISO/IEC 14598. L'objectif général de la création de la série de standards SQuaRE est de passer à une organisation logique, enrichie et une série unifier couvrant les exigences de spécifications de la qualité du logiciel et l'évaluation de la qualité des logiciels. Le but de cette série est d'aider avec des spécifications et l'évaluation de la qualité des produits informatiques, les acquéreurs et les développeurs. La série SQuaRE établit les critères pour la spécification des exigences de qualité du produit, leurs mesures et leurs évaluations. Il comprend un modèle de qualité qui permet d'aligner les définitions de qualité du client avec les attributs du processus d'évaluation.

La norme ISO/IEC 25040 contient les exigences et les recommandations pour l'évaluation de la qualité des logiciels. Elle décrit le processus d'évaluation de la qualité et les exigences nécessaires à l'application du dit processus. Le processus d'évaluation de la qualité s'applique à une large gamme de logiciels (commercial, propriétaire, etc.), et peut être utilisé pendant et après le processus de développement et/ou le processus d'acquisition des logiciels. Cette norme est destinée aux responsables de l'évaluation des logiciels qui peuvent être de l'organisation développeur, de l'organisation acquéreur et des évaluateurs indépendants. La norme ne s'applique pas aux aspects suivant des logiciels : exigence fonctionnelle, exigence d'affaires, etc.

2. Modèle de référence

Le processus d'évaluation de la qualité des logiciels comporte des contraintes, des ressources, des entrées et des sorties du processus comme le montre la figure 1.

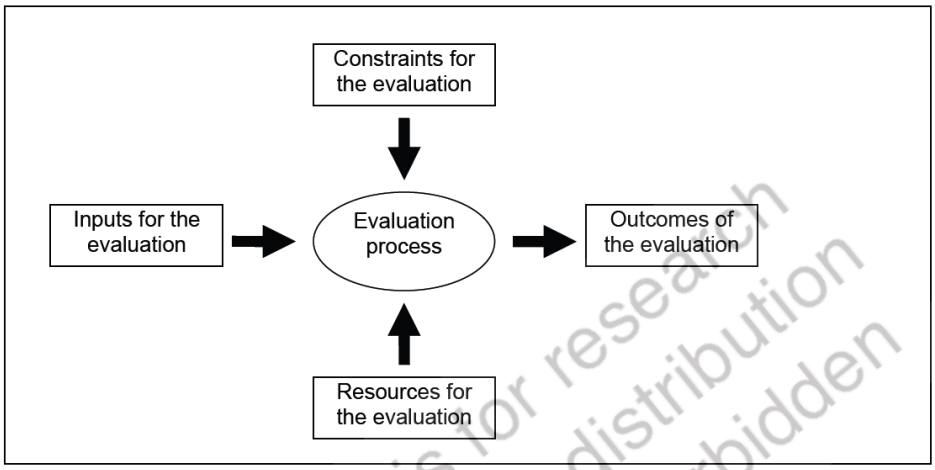

Figure 1. Modèle de référence général pour le processus d'évaluation de la qualité des logiciels

Ce modèle s'applique aux responsables de l'évaluation de la qualité des logiciels et propose de baser l'évaluation sur la spécification des exigences de qualité des logiciels en utilisant au préalable ISO/IEC 25030 et en énonçant clairement les objectifs et les critères d'évaluation.

3. Processus d'évaluation

Le modèle de référence du processus d'évaluation de la qualité des logiciels met en exergue les processus, les tâches, et les activités détaillés qui seront utilisés pour l'évaluation de la qualité. Il est essentiel que l'implémentation du processus d'évaluation soit assez flexible pour prendre en compte le caractère unique de chaque application, éviter de faire un travail qui n'apporte pas une plus value au logiciel, et fournir un moyen pour établir la confiance au logiciel.

Processus d'évaluation de la qualité des logiciels

La documentation est très importante dans le processus d'évaluation de la qualité. Elle doit décrire toutes les actions posées par l'évaluateur lors de son évaluation et contenir toutes les informations pertinentes requises pour l'interprétation et la gestion du processus d'évaluation de la qualité. Il est aussi important de noter dans le rapport d'évaluation les références détaillées de tous les outils utilisés pour l'évaluation.

Établir les exigences de l'évaluation

Déterminer les objectifs de l'évaluation

L'évaluation de la qualité des logiciels supporte les processus de développement et d'acquisition du produit tout en satisfaisant les besoins des clients et des utilisateurs.

Obtenir les exigences de qualité

Les parties prenantes doivent être identifiées. Ce sont eux qui vont définir les exigences de qualité du produit informatique. Ces exigences seront consignées en utilisant le modèle de qualité décrit dans ISO/IEC 25010.

Identifier les parties du logiciel qui doivent être inclure dans l'évaluation

Toutes les parties à évaluer doivent être identifiées et documentées. L'évaluation des parties est fonction d'où l'on est rendu dans le cycle de vie du logiciel ou de l'objectif de l'évaluation. Au début, de l'évaluation, il est difficile de définir avec précision la liste des produits à évaluer. Mais tout au long du processus, la liste initiale est mise à jour.

Définir la rigueur de l'évaluation

La rigueur de l'évaluation doit être liée à un ensemble de caractéristiques et de sous caractéristiques qui établissent des niveaux d'évaluation attendus, définissant les techniques d'évaluation à appliquer et les résultats de l'évaluation à réaliser.

Spécification de l'évaluation

Sélectionner les mesures de qualité

L'évaluateur doit décrire tous les modules d'évaluations qui vont couvrir toutes les exigences de l'évaluation de la qualité des logiciels. La méthode d'évaluation doit être documentée en prenant en compte les actions nécessaires pour aboutir aux résultats. Dans le cas échéant, tout outil utilisé dans la méthode d'évaluation doit être identifié.

Définir les critères de décision pour les mesures de qualité

Pour toutes les mesures, les critères de décision doivent être définis.

Définir les critères de décision pour l'évaluation

L'évaluateur doit préparer une procédure avec des critères distincts pour différentes caractéristiques de qualité, dont chacun en termes de sous caractéristiques individuelles et mesures de qualités, une combinaison pondérée de sous-caractéristique et de mesures. Les résultats synthétisés devraient être utilisés comme base pour l'évaluation de la qualité du produit.

Conception de l'évaluation

Planification des activités d'évaluation

La planification des activités d'évaluation est faite en prenant en compte la disponibilité des ressources. Et dois définir les points de décision dans le processus d'évaluation qui vont déterminer quand et pourquoi l'évaluation est considérée complétée. Elle ne doit pas avoir des taches dupliquées. Elle doit être faite de façon a diminué les risques d'erreurs et de réduire l'effort d'évaluation prévu, en envisageant au moins les éléments suivants : le budget de l'évaluation, les méthodes, les outils d'évaluations et les activités de l'évaluation.

Exécution de l'évaluation

Faire les mesures

En accord avec le plan d'évaluation, appliquer au logiciel les mesures de qualité sélectionnées au préalable.

Appliquer les critères de décision pour les mesures de la qualité

Appliquer les critères de décision, aux valeurs mesurées

Appliquer les critères de décision pour l'évaluation

L'ensemble des critères de décision doit être résumé en caractéristiques et sous caractéristiques produisant des résultats comme une déclaration de la mesure dans laquelle le produit informatique répond aux exigences de qualité. Les résultats de l'évaluation doivent :

- Établir un degré de confiance appropriée que le produit est capable de se conformer aux exigences de l'évaluation.
- Identifier toutes les conditions et limitations spéciales imposées sur l'utilisation du logiciel.
- Identifier les faiblesses ou les omissions dans l'évaluation.
- Identifier les options pour l'utilisation du logiciel

Conclure l'évaluation

Examiner les résultats de l'évaluation

Le requérant de l'évaluation et l'évaluateur doivent réviser les résultats

Créer le rapport d'évaluation

Examiner l'évaluation de la qualité et fournir une rétroaction à l'organisation

L'évaluateur doit examiner les résultats de l'évaluation et du processus d'évaluation, des indicateurs et mesures appliquées. La rétroaction devrait être utilisée afin d'améliorer le processus et les techniques d'évaluation. Quand il est nécessaire d'améliorer les modules d'évaluation, les indicateurs supplémentaires devraient être inclus à la collecte de données, afin de les valider pour une utilisation ultérieure.

Archivage des données

Lorsque l'évaluation est terminée, les données et les éléments d'évaluation doivent être disposés conformément aux exigences de la demande. Lorsque la durée d'archivage spécifiée expire, les données doivent être archivées de nouveau ou détruites de façon sécuritaire.

CHAPITRE II QUALITÉ DE LA MAINTENABILITÉ

1. Facilité d'analyse

La facilité d'analyse d'un produit informatique est caractérisée par l'aisance qu'on a pour identifier les déficiences, diagnostiquer les causes des échecs, et identifier les modifications à implémenter et leurs impacts. Une façon de mesurer cet élément est l'utilisation du MTTR (Mean Time To Repair).

La facilité d'analyse couvre donc la bonne compréhension du produit logiciel par les mainteneurs. Ceci passe par une bonne documentation et une bonne conception du code source rendant la solution développée claire dans l'esprit des membres de l'équipe de maintenance. Alors pour pouvoir cerner toutes les facettes et les réalisations d'un programme, toutes les documentations ainsi que le code source doivent être disponibles.

À la lumière de ce qui précède, une lacune peut déjà être identifiée dans le processus de documentation du CCI (Centre de Compétence en Intégration). En effet, la documentation n'est pas disponible. En effet, selon les politiques de l'entreprise (RONA), lors de la réalisation d'un projet, tous les documents relatifs à ce projet sont déposés dans un répertoire dédié au projet. À la fin du projet, c'est la responsabilité de chaque développeur de faire une copie, des documents pertinents pour son équipe dans un répertoire d'archivage pour l'équipe en question. Mais au sein du CCI, cette pratique n'est pas encore monnaie courante. Ce qui a pour conséquence de rendre la tâche des mainteneurs un peu plus ardue et long. En effet, les documents nécessaires à la compréhension du logiciel ne sont pas disponibles et quand ils le sont, ils ne sont pas à jour.

Un moyen d'avoir accès à ces documents est de faire une demande pour pouvoir consulter les répertoires de projet. Mais la lourdeur administrative qui entoure cette procédure fait qu'il est plus souvent recommandé de faire une inspection du code source avant.

En effet, l'analyse relativement simple du code source fait que c'est cette méthode qui est la plus utilisée pour comprendre le fonctionnement du produit. Ceci est possible par le concept de programmation utilisé par l'outil TIBCO qui permet de faire une programmation dite graphique, la nomenclature des ressources est évocatrice de l'action exercée par ceux-ci (figure 2).

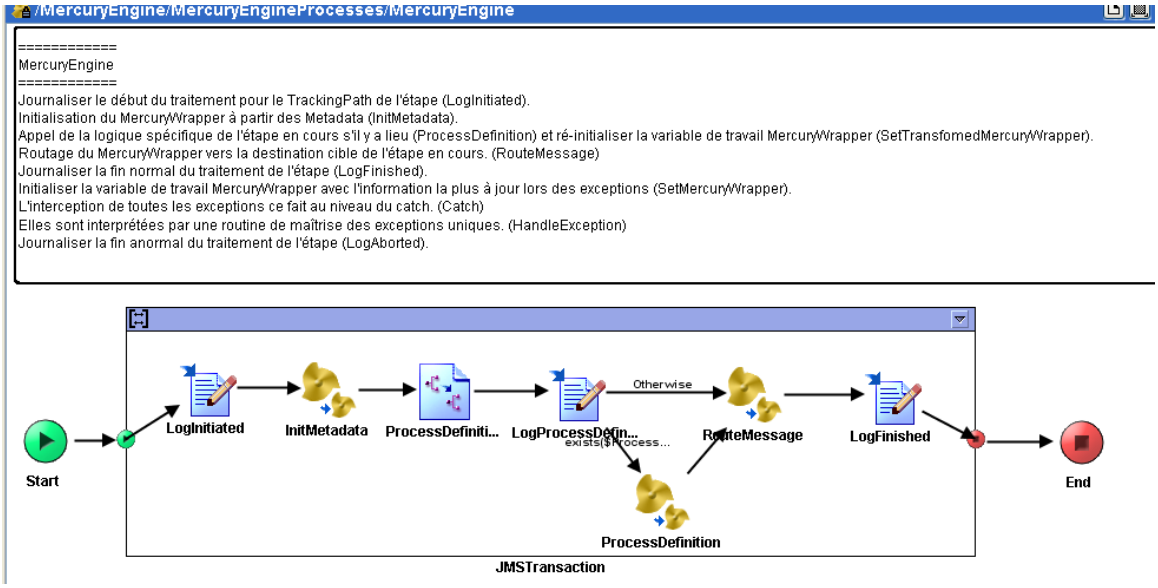

Figure 2 : description d'un « process »

2. Facilité de modification

La facilité de modification est la capacité d'implémenter les modifications spécifiées aux logiciels. Elle est identifiée par la modularité des composantes des systèmes. C'est un concept qui s'intègre bien au paradigme SOA. En effet, au CCI c'est cette architecture qui est utilisé et pour des fins de réutilisabilité, lors de la conception, la solution est pensée sous forme de module. C'est ainsi que l'on retrouvera dans un programme des appels à d'autres programmes. La Figure 2 est un excellent exemple. On peut voir que le process « JMSTransaction » fait appel au process « RouteMessage ». On dit que « RouteMessage » est un sous-process dans ce cas. Les modifications des sous-process n'ont généralement aucun impact sur le process principal, à moins d'une refonte complète de la fonctionnalité réalisée. Il faut aussi noter que les sous-process peuvent aussi faire appel à d'autres process (figure 3), augmentant ainsi la granularité et la complexité du produit final.

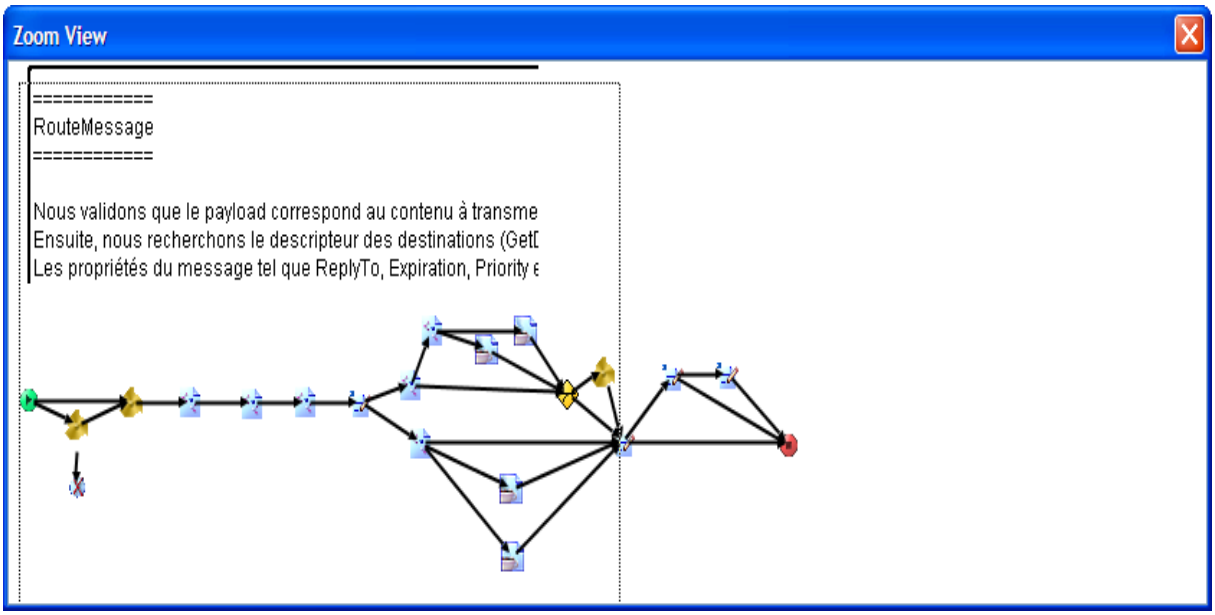

Figure 3 : Process RouteMessage

3. Stabilité

La stabilité d'un système est sa capacité d'éviter les effets indésirés et non envisagés à la suite des modifications du logiciel. Elle couvre la cohésion des données, et le taux d'échec. Un indicateur de la stabilité est la diminution d'erreur dans le système. Ce résultat peut mettre en évidence la maturité du processus de traitement des incidents. Dans notre cas, on a un processus presque entièrement automatisé qui permet de résoudre les incidents et enrichir une base de connaissance. On dispose aussi des statistiques sur l'occurrence d'un incident en particulier, le temps mis pour la résolution et la criticité de ces incidents. C'est donnés sont regroupées et traitée afin de déterminé les priorités à associer à chaque incident pour la mise en place d'un correctif. Ce processus gère aussi la conception de rapports automatiques qui sont mises à la disposition des dirigeants via un portail

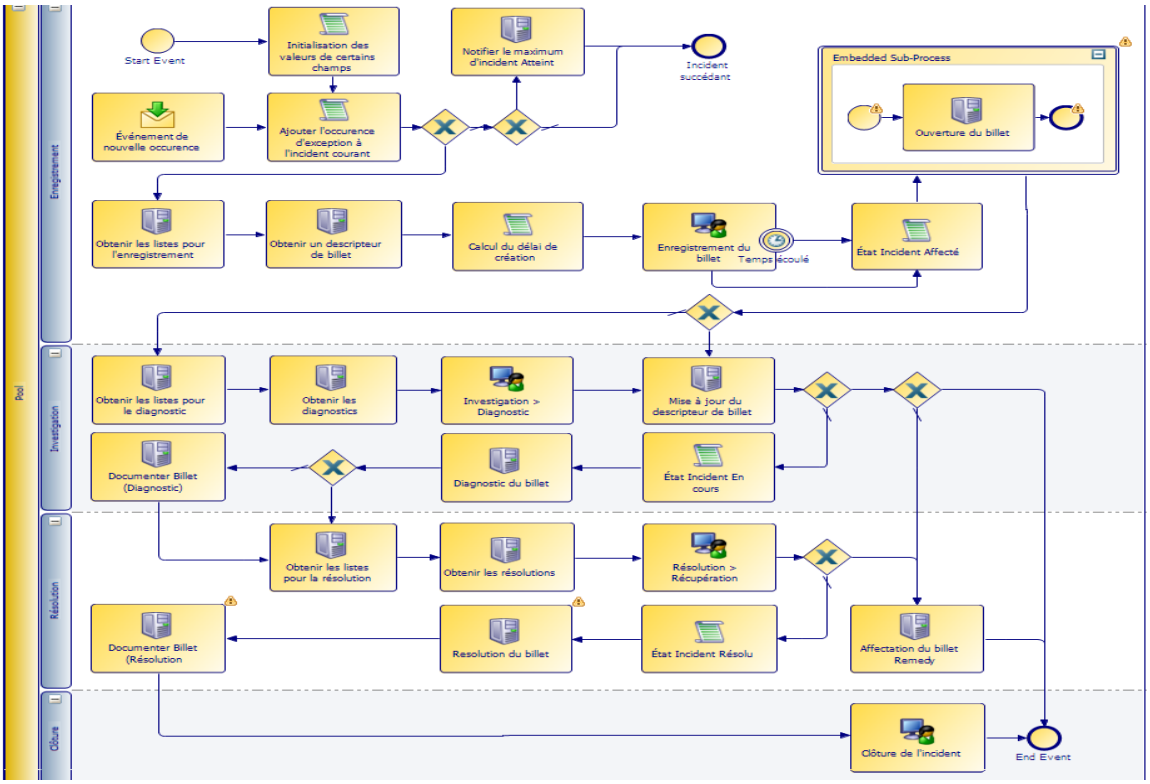

Figure 4 : Processus de traitement des incidents

4. Tests des changements

Le but de cette partie est de décrire la solution technique pour tester les intégrations ESB de façon automatisée.

Les objectifs sont d'effectuer des tests bout en bout d'un coupleur de façon automatisé. La conception et la réalisation des tests devront remplir les objectifs suivants :

- Les tests doivent être faciles à écrire et modifier
- Les tests doivent réutiliser les mêmes données
- Les tests ne doivent nécessiter aucune configuration dédiée pour les tests
- Les tests doivent être déclenchés de façon automatisée
- Les tests doivent être réutilisables pour tester non seulement le coupleur, mais aussi l'application cible

Ces tests doivent être le moins intrusifs possible pour les applications à tester.

Description

La solution proposée est une réutilisation des outils déjà disponibles (Ant) et l'introduction de l'outil de test Java TestNG. Ce dernier (TestNG, http://testng.org) offre une solution complète pour les tests d'intégration. Il permet en autre le chaînage des tests et le regroupement de tests en utilisant les annotations en Java.

Puisque les tests sont écrits en Java, des utilitaires ont été développés pour faciliter l'écriture des tests en utilisant la configuration spécifiée par le coupleur. L'outil utilisera les propriétés des coupleurs pour obtenir la configuration de l'environnement.

Vue d'ensemble de la méthodologie

TestNG est l'outil Java utilisé pour effectuer les tests. L'outil se base sur l'utilisation des méthodes « assertEquals » et « assertTrue » pour valider les valeurs obtenues après l'exécution d'un test.

Un « faux » document source va être soumis au coupleur puis nous allons relire le document en sortie. Pour valider les transformations, nous allons utiliser la méthode « assertEquals » aux champs que nous voulons valider avec TestNG. Des classes utilitaires Java ont été écrites pour envoyer et recevoir des documents sur l'ESB directement à partir de la classe de test Java. De plus, nous avons aussi la possibilité de suivre un pipeline en utilisant l'utilitaire de WireTap de Mercury. De cette façon, il est possible de suivre l'exécution d'un message à l'intérieur du coupleur.

Une fois le document canonique reçu, il est possible de valider les valeurs du document à partir d'une expression XPath. Le retour de cette expression pourra être utilisé avec le « assertEquals » de TestNG pour valider la transformation.

En résumé, pour effectuer un test :

1. On charge un document source
2. On assigne une valeur unique au document
3. On publie le document sur le Bus
4. On attend la « réponse » / publication du document canonique.

5. On valide les valeurs inscrites dans le document canonique.

Gestion des « messages » de test

Une famille d'objet Java a été montée pour permettre les différentes représentations des messages qui transigent sur le BUS. De cette façon, lorsque l'API d'envoie des messages, les propriétés requises pour le bon fonctionnement dans du connecteur Mercury (exemple : JMSFTP et la propriété Filename) seront automatiquement ajoutés.

De plus, des « helpers » ont été ajoutés au PayloadMercury pour obtenir les valeurs des champs souvent utilisés dans le MercuryWrapper.

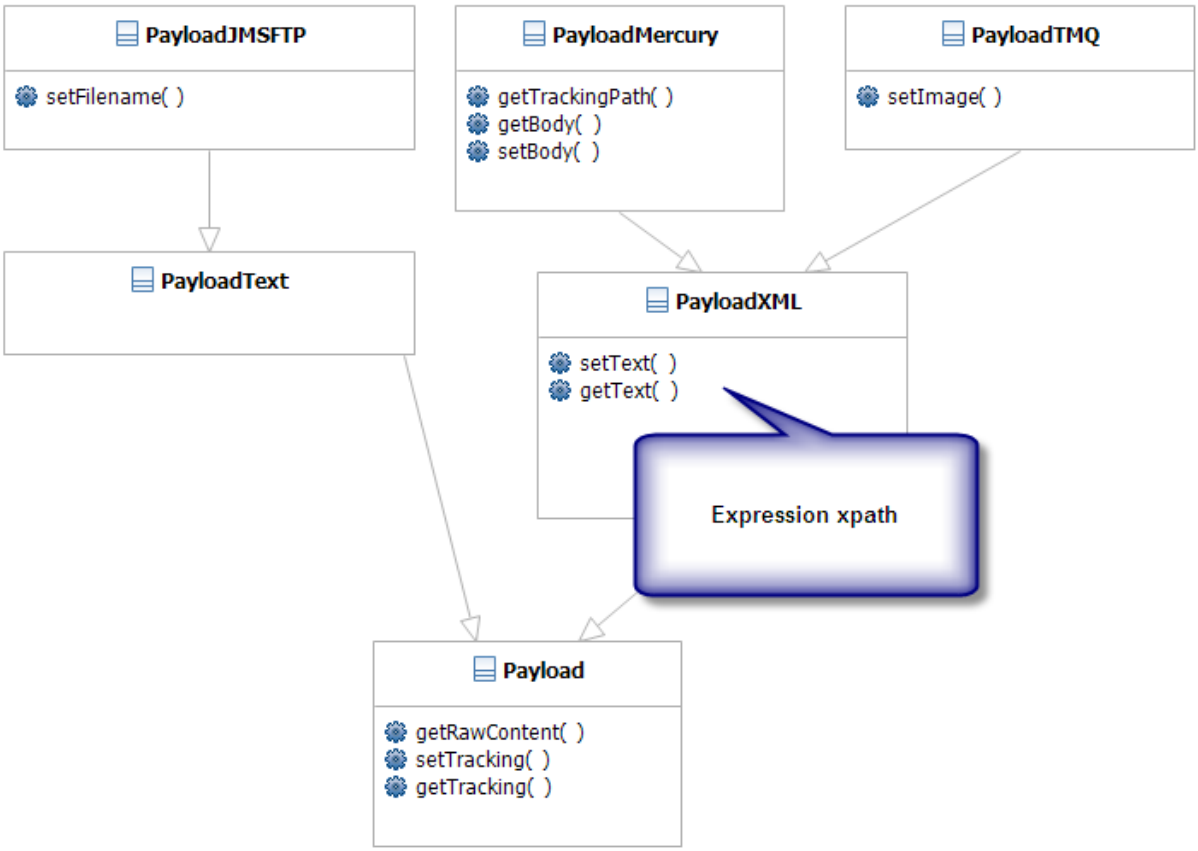

Figure 5 : diagramme de classe PayloadDef

Option de chaînages / groupes

TestNG permet de regrouper et de chaîner les classes de tests. En d'autres mots, ceci nous permet d'exécuter seulement une certaine catégorie de test et permet aussi d'établir les dépendances entre les tests. Dans l'exemple suivant, on indique que le test d'insertion d'un document dans EBS nécessite l'exécution réussie du test de la génération du système source:

```
@Test
public void testMMS() {
    PayloadJMSFTP payload = new PayloadJMSFTP(getRes("batchMMS.txt"));
    payload.setFilename("JRNE" + id + "1209");
    payload.setTracking(id);

    payload.replaceAll("AAAAAAAA", id );

    ems.sendTopic(metadata.getDest("JMSFTPAdapterPublisher"), payload);
    PayloadMercury wrap = ems.wireTap.waitEndPipeline();
    wrap = ems.wireTap.waitEndPipeline();

    // on valide la sortie du canonique
    assertEquals(wrap.getBody("/:JournalEntry/:JournalEntryHeader/:Category"), "MMS APCN");
    assertEquals(wrap.getBody("/:JournalEntry/:JournalEntryDetail[1]/:DebitCredit"), "Credit");
    assertEquals(wrap.getBody("/:JournalEntry/:JournalEntryDetail[5]/:DebitCredit"), "Debit");
}

@Test(dependsOnMethods={"testMMS"})
public void testInsertEBS() {
    PayloadMercury wrap = ems.wireTap.waitAbnd();
    assertTrue(wrap.getExceptionDescription().startsWith("XXRAR_INFA_TERM_INVALID"));
}
```

Figure 6 : Exemple de chaine de test

Pour indiquer le chainage, il suffit d'ajouter l'argument « dependsOnMethod » au tag Test de la méthode. TestNG s'occupera de faire l'exécution de cette méthode avant l'exécution de « testInsertEBS » dans l'exemple plus haut.

Pour ce qui est des groupes, il est possible d'identifier une méthode de test en ajoutant le tag « groups ».

Gestion des configurations

Les outils de test utilisent la configuration de déploiement des coupleurs. Avant de démarrer les tests, une tâche Ant s'occupe de traverser le répertoire de tous les coupleurs, puis faire un cumulatif des valeurs de configuration. Ainsi, toute saisie des configurations de l'environnement est évitée.

CONCLUSION

Au vu de tout ce qui précède, on peut faire la remarque que des améliorations sont à apporter au processus actuel pour le rendre plus efficace. En effet, une politique de gestion de documentation devrait être établir au sein du CCI pour s'assurer que l'équipe de support et évolution puisse avoir à leurs dispositions toute la documentation technique nécessaire à une bonne compréhension des logiciels à supporter et à faire évoluer. Le processus devrait aussi être amélioré pour prendre en compte les différentes catégories de la maintenance afin de fournir un rapport plus représentatif des activités de maintenance et d'évolution à la haute direction.

Annexes

A. Exemple de test

Voici un exemple de test avec chaque ligne expliquée pas à pas :

```
public class CanonicalClient extends EsbHelper {
  String id = uniqString();

  @Test
  public void client() {
    PayloadMercury req = new PayloadMercury(getRes("CanonicalClient.xml"));
    req.setTracking(id);
    req.setBody("/:XXR__HZ__PARTIES__V/:T__PARTY__NAME", id);

    EmsReceiver rec = ems.prepareTopic(metadata.getDest("CanonicalClient"));
    ems.sendQueue(metadata.getDest("EBSCouplerInboundNotify"), req);
    PayloadMercury rep = (PayloadMercury) rec.getWaitMsg();
    assertEquals(rep.getBody("/:CustomerPartyMaster/:Name[@sequenceName='LegalName']"), id);
  }
}
```

A. Le nom de la classe a le nom du document à tester. Chaque classe de test doit hériter de la classe EsbHelper pour obtenir les services de « test » à EMS.

B. uniqString() permet d'obtenir un numéro unique pour effectuer le test. Ce numéro sera utilisé dans le Tracking Path et le document pour valider unicité du message.

C. @Test permet d'indiquer à test NG que cette méthode doit être exécutée lors des tests. Si du chainage est nécessaire, nous pouvons ajouter (dependsOnMethods={" NOM DE METHODE "}) à la définition de l'annotation. De cette façon, nous pouvons tester la transformation source et la transformation cible avec les mêmes tests.

D. Définition Java du nom de la méthode.

E. On obtient le document initiateur du test. Ces documents sont disponibles en fichier dans le répertoire « res » et le nom de la classe qui effectue le test. Dans ce cas ça sera « res/CanonicalClient/CanonicalClient.xml »

F. On réinitialise le tracking path pour s'assurer de l'unicité du document

G. On réinitialise une valeur dans le document pour s'assurer de l'unicité lors de la transformation
H. On prépare la lecture du topic où sera publiée le canonique. Nous obtenons le nom du topic à partir des métadonnées qui sont dans le répertoire « work/build »
I. On envoie le message de test au coupleur. Nous utilisons aussi les métadonnées pour obtenir le nom de la file.
J. On attend le message canonique.
K. Nous testons la valeur publiée dans le canonique en utilisant la méthodologie du framework TestNG.