Configuration Interface for MEssage ROuting

Plan d'Assurance Qualité Logicielle

Date : 05/04/07
Version : 1.1
Statut : diffusable

Auteurs : **BAGNARD Natacha**
FOROT Julien

Tables des révisions

Version	Date	Modifications
0.1	17/01/07	Création du document
0.2	01/02/07	Modification du document suite à une lecture de Jérôme Camilleri
1.0	20/03/07	Validation du document par Jérôme Camilleri
1.1	05/04/07	Modification suite au premier audit avec Martine Tasset (fiche audit1_040407.odt)

Table des matières

1.But, portée, responsabilité

1.1.Objectif du document

Le plan d'assurance qualité (PAQL) a pour but de définir les méthodes et outils utilisés par le projet, ainsi que les mesures à prendre et les étapes pour contrôler et s'assurer de la qualité du projet.

Tout document produit sera soumis au contrôle de qualité. Il devra être conforme aux règles définies dans ce document. Tout document non conforme devra être corrigé.

1.2.Portée du document

Ce document est destiné :

- au responsable du stage : Jérôme Camilleri
- à la consultante : Martine Tasset
- au jury du Master Pro GI pour l'évaluation du stage
- à l'équipe du projet : Natacha Bagnard et Julien Forot

1.3.Responsabilités associées

Le responsable qualité est chargé de la rédaction du présent PAQL ainsi que de veiller à son application et son évolution, en collaboration avec le chef de projet. C'est à lui de décider des actions à entreprendre si le PAQL n'est pas appliqué.

1.4.Procédure d'évolution du PAQL

Le PAQL est élaboré au début du projet. A chaque étape, il est susceptible d'être modifié, auquel cas la modification sera indiquée dans la table de l'historique.

2.Documentation utilisée

2.1.Documents de référence

Le tableau suivant récapitule les principales sources documentaires qui seront utilisées dans le cadre de ce projet.

But	*Source*
Rédaction des documents	Cours de Y. Ledru
Développement Eclipse	Documentation Eclipse
ServiceMix	Documentation ServiceMix
Petals	Documentation Petals

But	*Source*
JBI	Spécification JBI 1.0 – JSR 208
Développement de la nouvelle version	Documentation produite pour la version précédente

3.Terminologie

Les termes suivant sont les termes spécifiques utilisés dans le PAQL.

- **CdC** : Cahier des Charges
- **CdB** : Cahier de Bord
- **DCG** : Dossier de conception globale
- **DCD** : Document de Conception Détaillée
- **DSE** : Dossier de Spécifications Externes
- **PAQL** : Plan d'Assurance Qualité Logiciel
- **PDL** : Plan de Développement Logiciel
- **MOA** : Maîtrise d'ouvrage
- **MOAd** : Maîtrise d'ouvrage déléguée
- **MOE** : Maîtrise d'oeuvre
- **MOEd** : Maîtrise d'oeuvre déléguée
- **CV** : Cycle de vie

4.Organisation

4.1.Maîtrise d'ouvrage

4.1.1.MOA

La maîtrise d'ouvrage est l'équipe BSOA de la section Recherche et Développement de Bull.

4.1.2.MOAd

La maîtrise d'ouvrage déléguée est représentée par Jérôme Camilleri, de l'équipe BSOA. Son rôle est de donner les objectifs de travail et de valider les résultats obtenus.

4.2.Maîtrise d'oeuvre

4.2.1.MOE

La maîtrise d'oeuvre est le Master2 Pro Génie Informatique de l'UJF et sont représentant : Philippe Lalanda.

4.2.2.MOEd

La maîtrise d'oeuvre déléguée est l'équipe de développement : Natacha Bagnard et Julien Forot. Leur rôle est de proposer une solution logicielle permettant au logiciel Cimero d'évoluer pour satisfaire les nouveaux besoins du MOA et palier aux problèmes de la version précédente, ainsi que de s'assurer de la bonne conduite du projet.

Pour cela, ils doivent faire en sorte que :

- Tous les documents demandés soient fournis
- Le planning soit observé et réajusté d'un commun accord si besoin
- Les normes de qualité soient définies et respectées
- La validation des documents soit conforme au PAQL
- La validation du code produit soit conforme au PAQL

4.3.Chef de projet

Le chef de projet sera Julien Forot pour la période de janvier à mi-juin, puis Natacha Bagnard pour la période de mi-juin à mi-septembre.

Le chef de projet est responsable :

- ✔ de la planification du projet
- ✔ du contrôle de l'avancement du projet
- ✔ de la mobilisation des moyens nécessaires de la coordination des travaux de chacun

4.4.Responsable qualité

Le responsable qualité sera Natacha Bagnard pour la période de janvier à mi-juin, puis Julien Forot pour la période de mi-juin à mi-septembre.

Le responsable qualité a pour mission de :

- ✔ définir les règles de retour arrière et les procédures de modification
- ✔ veiller à la diffusion et au respect des règles particulières au projet
- ✔ gérer l'évolution du PAQL

- ✔ superviser les actions de vérification et de validation

5.Cycle de vie

5.1.Motivation du choix

Chaque attente du client peut être atteinte indépendamment des autres. L'utilisation d'un cycle de vie permettant de développer chacun des modules de bout en bout séparément est donc appropriée. Le produit final sera donc livré par lots successifs. Le cycle de vie choisi est un cycle incrémental.

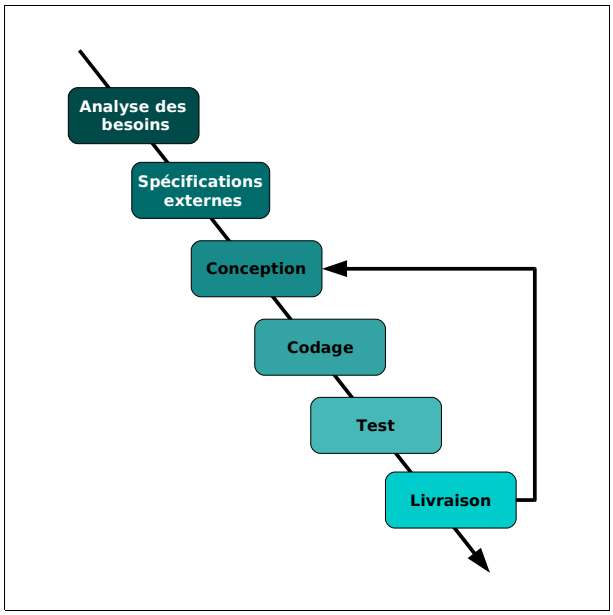

Illustration 1: Cycle de vie incrémental

5.2.Description des étapes du cycle de vie

- **Analyse des besoins** : Cette étape permet de prendre en compte les besoins du client, choisir le cycle de vie à adopter, définir le planning et préparer l'acquisition des outils de programmation.
- **Spécifications externes** : Cette étape permet de définir précisément l'ensemble des fonctionnalités, d'étudier l'interface homme-machine du système, de se former aux outils utilisés et de préparer les tests système.
- **Conception** : Cette étape permet de définir l'architecture interne du logiciel, d'acquérir et d'installer les outils de programmation et de préparer les tests d'intégration.

- **Codage** : Cette étape regroupe la phase de conception détaillée et de codage. Elle permet de détailler précisément la conception globale par module afin de pouvoir les développer et préparer les tests unitaires. Le codage permet de traduire la conception détaillée en un programme.
- **Tests** : Cette étape permet, à l'aide des plans de tests, de tester le logiciel développé dans l'ordre suivant : tests d'intégration, tests système, tests d'acceptation. L'observation et la comparaison avec les résultats attendus permettront de modifier si nécessaire le logiciel.
- **Livraison** : Livraison d'une version logiciel ou du logiciel final.

5.3.Phase du cycle de vie

Voici pour chaque étape du cycle de vie « en V » les documents requis et produits, ainsi que les conditions de passage d'une étape à une autre.

	Activités	Documents		Condition
		Requis	Produits	
Analyse des besoins	Étude de l'existant, définition des besoins, analyse des risques et de la faisabilité, préparation des tests d'acceptation, établissement des procédures et plans qualité, choix du CV, planification, estimation des ressources		• CdC • PDL • PAQL • Plan de tests d'acceptation	Validation du Cdc par le MOAd
Spécifications externe	Étude détaillée du logiciel à réaliser, préparation des tests système, mise à jour des documents établis dans l'étape précédente	• CdC	• DSE • Manuel utilisateur • Plan de tests système	Validation des documents par le MOAd

	Activités	Documents		Condition
		Requis	Produits	
Conception	Définition de l'architecture interne du logiciel, définition des composants, préparation des tests d'intégration et des tests unitaires, mise à jour des documents établis dans l'étape précédente	• DSE	• DCG • DCD • Plan de tests d'intégration et de tests système	Validation des documents par le MOAd
Codage	Codage, documentation des composants, mise à jour des documents établis dans l'étape précédente	• DCG • DCD		
Tests	Réalisation des jeux de tests, correction du code si besoin, mise à jour des documents établis dans l'étape précédente	• Ensemble des plans de tests	• Rapports de tests	• Jeux de tests concluant
Livraison	Réalisation des tests d'acceptation par le client	• Jeux de tests d'acceptation		

6.Documentation produite

A l'issue de ce projet, plusieurs documents auront été produits. Certains devront être livrés avec le produit fini, certains seront disponibles si le client les souhaite et certains ne serviront qu'à l'équipe de développement. Le tableau suivant précise pour chaque document sont statut.

Document	Statut
Cahier des charges	Livrable
Plan d'assurance qualité logicielle	Livrable
Plan de développement logiciel	Privé
Dossier de spécifications externes	Livrable
Dossier de conception globale	Consultable
Dossier de conception détaillée	Consultable
Plans de tests	Livrable
Jeu de tests	Livrable
Manuel utilisateur	Livrable

- **Description des différents statuts :**
  - **Livrable** : Doit être fournit au client
  - **Consultable** : Peut être consulté par le client
  - **Privé** : Destiné à l'équipe du projet uniquement

7.Gestion de la configuration

La configuration est l'ensemble des éléments suivants : code source exécutable, outils de développement et de test utilisés, documents et données.

7.1.Outils de travail collaboratif

Voici les outils que nous utiliserons pour gérer le travail collaboratif :

- **CVS** (Concurrent Versions System)

Cet outil permet de partager le code source d'un logiciel et d'intégrer facilement les modifications de fichiers d'un développeur avant de les mettre à disposition des autres utilisateurs.

Il permet également de disposer d'un historique de toutes les modifications et des différentes versions du projet. Le retour à une version précédente en cas de problème est facilité.

- **BullForge**

Bull possède un site qui permet aux équipes de développement de collaborer efficacement grâce a des outils comme les forums et les mailings list. Bullforge fournit aussi des outils de travail collaboratif comme CVS ou Subversion.

- **Mailing list**

L'inscription aux mailing list des projets associés est essentiel pour prendre en main les outils et être au courant des nouvelles versions et des corrections d'erreurs.

7.2.Environnement technique

Le développement sera effectué sous environnement Linux (Debian). Le plugin pour Eclipse sera développé sur la version 3.3 de l'IDE. La JDK 1.5 sera utilisée pour le développement JAVA.

8.Gestion des modifications

8.1.Procédure de modification

Les modifications peuvent avoir deux origines :

- Détection d'une anomalie
- Demande d'évolution

Dans le cas d'une détection d'anomalie, il faut en trouver la source puis la corriger dans les plus bref délais.

Dans le cas d'une demande d'évolution du logiciel par le client, il faut dans un premier temps réaliser une étude de faisabilité, pour ensuite modifier le logiciel en conséquence si cela s'avère réalisable dans les délais impartis.

Les membres du MOEd sont responsables de la mise en oeuvre de ces modifications.

8.2.Règles d'évolution des numéros de version

Documents et applications sont identifiés par un numéro de version, un numéro de révision et un numéro de correction comme suit : « NuméroVersion.NuméroRévision.NuméroCorrection ».

Trois cas peuvent nécessiter un changement de version.

- Une nouvelle livraison de l'application ou du document :
  - ✔ On incrémente le numéro de version. Les numéros de révision et de correction reviennent à zéro.
- Une évolution :
  - ✔ On incrémente le numéro de révision. Le numéro de correction revient à zéro.
- Une correction d'anomalie(s) :

- ✔ On incrémente le numéro de correction

8.3.Règles d'évolution du statut

Pour chaque document, un statut doit être mentionné. Celui-ci peut-être :

- **Initial** : Le document est en cours de rédaction, il n'est pas diffusable, même en interne.
- **Diffusable** : Le document est rédigé, il peut être diffusé en interne mais il n'a pas encore été validé.
- **Final** : Le document à été validé. Il peut être consulter ou livrer.

9.Méthodes, outils, règles, normes

9.1.Méthodes

La méthodologie UML sera utilisée pour toutes les phases d'analyse de ce projet. Les tests seront préparés pendant les premières étapes du projet, à savoir :

- **Analyse des besoins** : Préparation des tests d'acceptation
- **Spécifications externes** : Préparation des tests systèmes
- **Conception globale** : Préparation des tests d'intégration
- **Conception détaillée** : Préparation des tests unitaires

Ces tests devront être préparés au fur et à mesure de l'avancement du projet. Ainsi, à chaque étape, ils devront être répertoriés dans le document « PlanDeTests*.odt ».

9.2.Outils

Le tableau suivant récapitule les outils qui seront utilisés dans le cadre de ce projet.

Fonctions	*Outils*
Editeur de texte	Open Office Writer
Editeur UML	Dia/OMONDO
Environnement de développement	Eclipse
Développement collaboratif	CVS, BullForge, Mailing list
Gestion de projets	Planner
ESB	ServiceMix, Petals

9.3.Règles et normes à respecter

9.3.1.Règles de présentation

Tous les documents liés à ce projet devront respecter un même modèle :

- **page de garde**
  - ✔ Nom du projet
  - ✔ Nom du document
  - ✔ Date de dernière modification
  - ✔ Numéro de version
  - ✔ Auteur(s)
  - ✔ Cartouche d'entête : Logo Bull, nom du document, logo du projet
- **page suivantes**
  - ✔ Cartouche d'entête : Logo Bull, nom du document, logo du projet
  - ✔ Numéro de page
  - ✔ Nombre de pages

9.3.2.Règles de programmation

Le système sera développé en langage Java. Nous adopterons donc les règles qui s'y appliquent, notamment :

- **En entête de classe**
  - ✔ Date de création de la classe
  - ✔ Auteur
  - ✔ Historique des modifications
- **Pour chaque méthode**
  - ✔ Description succincte de la méthode
  - ✔ Description des paramètres utilisés
  - ✔ Description de la valeur de retour
- **Convention de nommage**
  - ✔ Le nom d'une classe commence par une majuscule. Si ce nom est lui-même

composé de plusieurs noms, chacun d'entre eux commence par une majuscule. Les autres lettres sont en minuscules.

➔ Exemple :

- public class Composant{ }
- public class TypeComposant{ }

✔ Le nom d'une méthode ou d'une variable commence par une minuscule. Si ce nom est lui-même composé de plusieurs noms, chacun d'entre eux commence par une majuscule. Les autres lettres sont en minuscules.

➔ Exemple :

- public void init(){ }
- public void doPost{ }

✔ Le nom des variables doit être significatif pour une meilleure compréhension du code.

Le respect de ces règles de programmation devra être vérifié à l'aide de CheckStyle (plugin Eclipse). Un minimum d'avertissements devra être présent. Aucun seuil n'est fixé car certaines erreurs de CheckStyle n'ont pas un grand impact sur le style de programmation mais nécessite par contre un grand investissement en temps.

Il s'agira donc de réduire au maximum le nombre d'avertissements en perdant un minimum de temps.

9.3.3.Normes sur les comptes-rendus de réunions

Chaque audit donnera lieu à la rédaction d'un compte-rendu. Celui-ci se compose de la manière suivante : « **nom_date** ».

✔ nom : nom de la réunion (audit1, audit2 ou audit3)
✔ jj : le numéro du jour sur 2 chiffres
✔ mois : le numéro du mois sur 2 chiffres
✔ aa : le numéro de l'année sur 2 chiffre

Par exemple, le compte-rendu de l'audit du 15 mars 2007 serait nommé : « audit1_150307 ».

9.3.4.Normes sur les documents livrables

Chaque document possède un numéro de version de la forme x.y.
Tout incrément du premier indice (x) implique un changement notoire du document. Tout

incrément du second indice (y) implique une modification minime du document.

9.3.5.Normes sur les fiches de relecture

Chaque diffusion d'un document donne lieu à une relecture et au remplissage d'un fiche de relecture. Celle-ci se compose de la manière suivante : «**nomDocAssocié_Relecteur_Date**».

- ✔ nomDocAssocié : Le nom du document abrégé associé à la fiche de relecture
- ✔ Relecteur : Le nom du relecteur
- ✔ Date : La date de création de la fiche

Par exemple, la fiche de relecture associée au Plan d'Assurance Qualité (PAQL) créer par Jérôme Camilleri le 7 février 2007 sera nommée : « PAQL_Camilleri_06022007 ». Un modèle de fiche est fourni : « **Modele_fiche_relecture** » ainsi qu'une fiche explicative annexe : « **PAQL_Fiche_relecture** ».

10.Exigences et évaluation de la qualité

10.1.Facteurs

Les facteurs de qualité suivant ont été identifié comme important pour ce projet et devront être validés :

- **Maintenabilité** : Aptitude du logiciel à pouvoir être corrigé facilement.
- **Portabilité** : Aptitude du logiciel à être transféré d'un matériel et/ou d'un environnement logiciel à un autre.

10.2.Critères

Chaque facteur est lié à des critères précis. Parmi ceux-ci, un minimum de 4 critères par facteur devra être assuré.

Maintenabilité	*Portabilité*
• Tracabilité • Instrumentation • Consistance • **Simplicité** • **Modularité** • **Auto-description** • Concision • Communicabilité	• **Simplicité** • **Modularité** • **Auto-Description** • **Indépendance logicielle** • **Indépendance machine**

Principaux critères retenus Autres critères

10.2.1.Motivation des choix

Les critères tracabilité, instrumentation, consistance et communicabilité ont été écartés. Seul les critères les plus importants pour ce projet ont été conservés.

10.3.Évaluation

La partie suivante présente les mesures qui seront effectuées pour quantifier le respect de chaque critère sélectionné. Les conditions de validation de chaque critère seront également précisées.

10.3.1.Modularité

La modularité est l'aptitude d'un logiciel à être composé de modules indépendants. Comme le langage de programmation utilisé dans ce projet est JAVA, la modularité sera calculée en fonction du nombre de lignes de code (LOC) composant chaque classe.

La taille maximale recommandée d'une classe JAVA est de 500 lignes de code. Au delà, le code devient trop complexe.

Une classe sera considérée « valide » si elle respecte l'expression suivante :

ClasseValide = NombreDeLignesDeCode < 500

Le critère « modularité » sera donc mesuré par l'expression suivante :

Modularité = 10(NombreDeClassesValides /NombreTotalDeClasses)*

Condition de validation de ce critère :

La note sur 10 obtenue devra être supérieure à 8.5.

10.3.2.Auto-description

L'auto-description est l'aptitude d'un logiciel à fournir la description de chacune de ses fonctions. Comme le langage de programmation utilisé dans ce projet est JAVA, il sera possible de rédiger de la javadoc ainsi que des commentaires avant chaque fonction pour spécifier son rôle, ses attributs, sa valeur de retour, ... De la même façon, le corps des fonctions devra être commenté afin de rester compréhensible.

Une classe sera considérée « valide » si elle respecte l'expression suivante :

ClasseValide = NombreDeLignesDeCommentaires/NombreDeLignesDeCode > 30%

Le critère « auto-description » sera donc mesuré par l'expression suivante :

Auto-Description = 10(NombreDeClassesValides /NombreTotalDeClasses)*

Condition de validation de ce critère :

La note sur 10 obtenue devra être supérieure à 8.5.

10.3.3.Indépendance logicielle

L'indépendance logicielle est l'aptitude d'un logiciel à ne pas être lié, de par son fonctionnement, à un environnement logiciel particulier. Pour respecter ce critère, le logiciel issu de ce projet devra fonctionner pareillement sous Windows XP-2000 et sous Linux (Debian).

Le critère « indépendance logicielle » sera donc mesuré par l'expression suivante :

Indépendance logicielle = 10(FonctionneIndependamentDeL'OS)*

Condition de validation de ce critère :

La note sur 10 obtenue devra être égale à 10.

10.3.4.Indépendance matérielle

L'indépendance matérielle est l'aptitude d'un logiciel à ne pas être lié, de par son fonctionnement, à un environnement matériel particulier. Pour respecter ce critère, le logiciel issu de ce projet devra fonctionner pareillement (sans prendre en compte la performance) sur différentes configurations de machines.

Le critère « indépendance matérielle » sera donc mesuré par l'expression suivante :

Indépendance matérielle = 10(FonctionneSurPlusieursConfigurationsDeMachines)*

Condition de validation de ce critère :

La note sur 10 obtenue devra être égale à 10.

10.3.5.

10.3.6.Simplicité

La simplicité est l'aptitude d'un logiciel à avoir un fonctionnement interne compréhensible facilement. Pour cela, des règles de programmation ont été prises (voir chapitre IX.3.B).

Une fonction sera considérée « valide » si elle respecte l'expression suivante :

Fonction*Valide = NormesDeProgrammationRespectées (vrai ou faux)*

Le critère « simplicité » sera donc mesuré par l'expression suivante :

Simplicité = *10*(NombreDeFonctionsValides /NombreTotalDeFonctions)*

Condition de validation de ce critère :

La note sur 10 obtenue devra être supérieure à 8.5.

11.Reproduction, protection, livraison

11.1.Procédure de reproduction

Aucun procédure de reproduction particulière n'est prévu.

11.2.Protection du logiciel

Aucune protection du logiciel n'est prévu.

11.3.Livraison et installation

Le logiciel final sera remis à l'équipe BSOA. Il contiendra le code source de l'application, ainsi que le plugin développés.

Les documents livrables seront fournis dans un répertoire « Cimerov2Doc ». Les règles de nommage de ces fichiers sont explicitées dans le chapitre 9.4.

Ce plugin sera installé dans l'IDE eclipse sur une machine de l'entreprise Bull. Un autre exemplaire sera utilisé pour la soutenance de stage de l'équipe de développement (septembre 2007).

Lors de la livraison du produit à Bull, une procédure complète d'installation sera fournie (dans le manuel utilisateur) afin de permettre à des personnes étrangères à l'équipe de développement d'installer et d'utiliser le produit sans difficultés.

12.Suivi de l'application du PAQL

12.1.Validation des documents

Tous les documents rédigés seront relus par le responsable qualité et modifiés en cas de non-conformité avec le présent Plan d'Assurance Qualité Logicielle.

De la même façon, les documents de type « livrables » seront relus et devront être validés par le MOEd (Jérôme Camilleri).

Les trois audits prévus permettront également de vérifier le bon respect du PAQL. En effet, pendant ces réunions, la situation du produit et l'avancement du projet seront examinés méthodiquement.

12.2.Relecture

La lecture croisée sera effectuée au minimum par les 2 membres de l'équipe de développement (l'un rédige, l'autre relit) et par le MOEd.

L'auteur d'un document assure la réalisation des corrections proposés par les relecteurs et gère le changement des numéros de versions.

L'enchaînement des tâches est le suivant :

- Création du document
- Diffusion aux relecteurs potentiels avec la fiche de relecture[1] associée
- Intégration des modifications par l'auteur et modification éventuelle du numéro de version en fonction des modifications effectuées
- Validation finale, après plusieurs cycles de diffusion-correction, effectuée par le MOEd Jérôme Camilleri

1 Voir document annexe : PAQL_Fiche_relecture.odt, pour plus de précision