Université de Mons
Faculté des Sciences
Institut d'Informatique

Qualification de modules pour framework Open Source
Rapport de Stage

Directeur du stage : Mr Tom MENS

Maître du stage : Mr Christophe PONSARD

Rapport de stage réalisé par
Dimitri DURIEUX

Stage réalisé au CETIC

Année académique 2011-2012

Remerciements

Je tiens à remercier le *Centre d'Excellence en Technologies de l'Information* et de la Communication (CETIC) de m'avoir accueilli et accompagné tout au long de ce stage.

Je tiens à remercier *M. Christophe Ponsard*, responsable du département Software and System Engineering du CETIC, de m'avoir donné l'opportunité d'effectuer ce stage au sein de son département, ainsi que d'avoir accepté d'être mon maître de stage.

Je tiens à remercier *M. Tom Mens*, professeur de l'université de Mons *UMons*, d'avoir accepté d'être mon directeur de stage.

Je tiens à remercier *M. Jean-Christophe Deprez*, coordinateur scientifique au sein du CETIC, pour ses conseils, son encadrement et son aide lors de la compréhension du projet QualOSS.

Je tiens à remercier *M. Frédéric Fleurial Monfils*, responsable qualité et coordinateur d'équipe au sein du CETIC, pour ses conseils, son encadrement et son aide lors de la compréhension des outils issus du projet QualOSS.

Je tiens à remercier *M. Nicolas Devos*, chef de projet au sein du CETIC, pour son aide et ses conseils dans la compréhension des outils de développement.

Je tiens à remercier *M. Jacques Flamand*, ingénieur de recherche senior au sein du CETIC, pour son aide dans la compréhension du projet QualOSS ainsi que de ses outils.

Je tiens à remercier également toute autre personne ayant contribuée de près ou de loin à la réalisation de ce stage.

Table des matières

Introduction

Lors de cette seconde année de Master en Sciences Informatiques, l'université de Mons (**UMons**) offrait aux étudiants la possibilité d'effectuer un stage de 10 semaines dans une entreprise extérieure. L'objectif principal consistait à confronter les connaissances acquises par l'étudiant tout au long de son cursus, face à un ou plusieurs projets pratiques d'une entreprise.

Ce stage permettait également à l'étudiant d'acquérir et de comprendre les connaissances propres au monde du travail. Pour renforcer cette expérience, l'étudiant se devait de trouver lui-même son stage, comme il le ferait lors de la recherche d'un emploi (ex. : rédiger une ou plusieurs candidatures spontanées et/ou participer à des entretiens).

Pour ma part, le stage s'est déroulé dans le département Software and System Engineering (SSE) du centre de recherche nommé Centre d'Excellence en Technologies de l'Information et de la Communication (**CETIC**). Le **CETIC** correspondait à mes attentes et proposait une suite de sujets de stages dans des domaines variés, tels que l'ingénierie logicielle, le web sémantique, les réseaux, les systèmes embarqués, etc.

Dans le cadre de ce stage, le sujet choisi fût la *Qualification de module pour framework Open Source*. Ce sujet consistait au développement d'une plate-forme d'évaluation multi-critères de logiciels open-sources et de génération de rapports.

Le cas étudié est celui de la génération du rapport d'évaluation de la maturité du consortium **OW2** pour les projets hébergés au sein de sa forge.

Ce document consiste en un court rapport sur le travail réalisé.

La section 1 présente le **CETIC**, ses objectifs et ses principes de fonctionnement. La section 2 introduit la méthodologie **QualOSS** et le consortium **OW2**, formant le contexte dans lequel a évolué le stage. La section 3 détaille le sujet de stage proposé ainsi que ses objectifs et ses contraintes. La section 4 présente les outils utilisés par la plate-forme d'évaluation. La section 5 décrit de manière détaillée la plate-forme d'évaluation développée au travers de ses fonctionnalités. La section 6 présente et interprète les résultats obtenus par l'application sur un ensemble de projets d'**OW2** et vérifie si l'ensemble des contraintes définies dans la section 3 ont été respectées. Enfin, une conclusion reprend ce que l'étudiant a apporté à l'entreprise et ce que le stage a apporté à l'étudiant.

1 Présentation de l'entreprise

Le *Centre d'Excellence en Technologies de l'Information et de la Communication* ou **CETIC** est un centre de recherches appliquées accrédité par la région wallonne, dédié au support des industries. Sa mission est d'aider à la compétitivité des industries wallonnes en mettant à leur disposition une expertise de qualité dans le domaine des Technologies de l'Information et des Communications (TIC).

L'expertise du CETIC vient principalement de son implication active dans des projets de recherche et des développements technologiques européens et wallons.

Dans le but d'offrir un large panel de services aux entreprises, le CETIC possède 3 départements d'expertise, chacun spécialisé dans un des aspects importants des systèmes logiciels actuels.

Premièrement, le département Software and System Engineering (SSE), où s'est organisé le stage, a pour mission d'aider les entreprises à améliorer leurs activités de développement logiciel et leurs processus IT (technologie de l'information). Le département couvre l'ensemble des activités techniques impliquées dans le cycle de vie d'un développement logiciel : l'ingénierie des exigences, le design architectural, le développement orienté modèle, la qualité de code, ainsi que le design et la couverture des tests.

Deuxièmement, le département des Software and Services Technologies (SST) couvre les zones clés dans le contexte des TIC : architecture orientée services, Cloud computing, le contenu intelligent et la sémantique, ainsi que les technologies open-sources. Il fournit aux entreprises les compétences et les approches indispensables au développement de nouvelles capacités, nécessaires afin de prendre l'avantage sur les dernières tendances et une avance technologique significative dans les industries TIC. Il aide également à la sélection et à l'implémentation des technologies appropriées.

Troisièmement, le département Embedded and Communication Systems (ECS) développe des solutions innovantes dans l'intégration des systèmes embarqués basés sur les technologies de communication sans fil et leur logique de programmation. ECS étudie les nouvelles technologies et processus dans le monde des systèmes embarqués et développe des "proof-of-concept" visant à l'application de solutions efficaces sur des problèmes industriels.

2 Contexte de départ

Avant d'expliquer en détails de travail réalisé tout au long de ce stage, il est nécessaire d'introduire les éléments les plus importants du contexte dans lequel il a évolué.

Premièrement, cette section présentera le consortium **OW2**, notre cas d'étude, qui permettra de décrire le problème faisant l'objet du stage.

Deuxièmement, cette section introduira le projet **QualOSS**. Il a été utilisé dans le cadre du stage comme modèle de qualité.

2.1 OW2

Le consortium *OW2* [6] est une **communauté open-source indépendante**, s'engageant à rendre disponible à tous les meilleurs et les plus fiables logiciels d'infrastructure informatiques d'entreprise, y compris les middlewares, les plates-formes d'applications et technologies de cloud computing.

La **mission** du *consortium OW2* est de développer une base de code de logiciels open-sources et de favoriser une **communauté vivante** et un **business ecosystem**.

OW2 a été lancé le 1^{er} Janvier 2007 à la fusion d'ObjectWeb et d'OrientWare, deux grandes communautés du middleware open-source constituées de sociétés IT industrielles, de start-ups, d'organisations universitaires et de personnes en provenance du monde entier.

Pour le développement de logiciels, les activités de la communauté sont organisées en **projets**. Un projet regroupe les actions correspondant à un ou plusieurs composants logiciels techniques ou à l'intégration des différents composants techniques en vue de construire une plate-forme.

La maturité logicielle

La *maturité logicielle* est utilisée dans le cas du consortium OW2 pour définir le cycle de vie d'un logiciel. Ce cycle de vie est l'ensemble des étapes définissant l'évolution du logiciel.

La définition de ce cycle de vie a été réalisée par le **comité technologique** d'OW2. Ce comité est constitué principalement des chefs les plus motivés des projets de l'écosystème.

Ce comité a défini un cycle de vie divisé en 3 phases :

- *incubation* : état d'un **nouveau** projet ;
- *mature* : état d'un projet **évalué comme mature** par OW2 (voir ci-dessous) ;
- *archive* : état d'un projet ayant montré une **inactivité** de sa communauté [1].

Avant de faire partie du consortium, un nouveau projet doit envoyer une demande au comité technologique qui l'évalue ensuite. Le projet doit développer une application du type **middleware**. Il doit posséder une l**icence compatible** avec les règles d'IPR (Intellectual Property Rights) d'OW2. L'équipe de développement doit avoir la volonté de se mettre en **synergie avec les autres projets** de l'écosystème. Et enfin, le projet doit pouvoir **propager des releases** et **favoriser une communauté**.

Lorsque le projet souhaite passer mature, sa communauté doit remplir la **Mature Transition Checklist** (**MTC**) et la faire parvenir au comité technologique pour évaluation. Le comité évalue le projet et en fonction des résultats, accepte ou non qu'il devienne mature.

La Mature Transition Checklist

La *Mature Transition Checklist* (MTC) est un document rédigé par le comité technologique pour définir la maturité logicielle selon OW2.

La MTC reprend un **ensemble de 9 critères** regroupés en deux catégories : les critères techniques et les critères communautaires. Chaque critère reprend un aspect important des projets logiciels open-sources actuels :

1. les sources, la documentation et les binaires ;
2. la compilation ;
3. l'usage du projet ;
4. la gestion du contrôleur de versions ;
5. les conventions de codage ;
6. Software Quality Assurance Trustworthiness (SQuAT) ;

1. OW2 n'a pas encore clairement défini les conditions amenant à l'archivage d'un projet

7. committers ;
8. dashboard ;
9. l'activité.

Chacun de ces critères demande une **suite d'informations** et décrit ce que le projet doit posséder. Par exemple, pour la compilation, les informations demandées sont l'URL de la documentation décrivant la compilation du projet et la description du le mécanisme d'intégration continue utilisé. De plus, le projet doit être compilable sans erreurs et ses tests doivent être réalisés avec succès.

Cependant, il y a une exception : le **critère SQuAT**. Il s'agit du programme de qualité logicielle d'OW2. Il demande un ensemble de rapports mettant en évidence les aspects suivants :

- le contrôle de la propriété intellectuelle et des licences utilisées dans le projet ;
- des mesures de la qualité ;
- des indicateurs de maturité et de confiance.

Comme il n'existe pas d'outil spécifique pour ce critère, il est non-obligatoire.

2.2 QualOSS

QualOSS [4] est un projet européen de trois années, auquel le CETIC a activement participé. Il avait pour but de définir une méthode d'évaluation de logiciels open-sources multi-critères.

L'objectif de la méthode est de donner la possibilité aux entreprises d'évaluer le risque lié à une initiative open-source qu'elles souhaiteraient inclure dans une de leurs solutions logicielles. L'évaluation du risque se subdivise en celles d'un ensemble de critères.

La figure 1 (page suivante) montre l'organisation des critères définis par la méthode. Les critères blancs sont évalués tandis que les critères grisés ne le sont pas. La méthode mesure le risque sur une initiative open-source par l'analyse de deux critères principaux : sa capacité d'évoluer (*evolvability*) et sa robustesse (*robustness*).

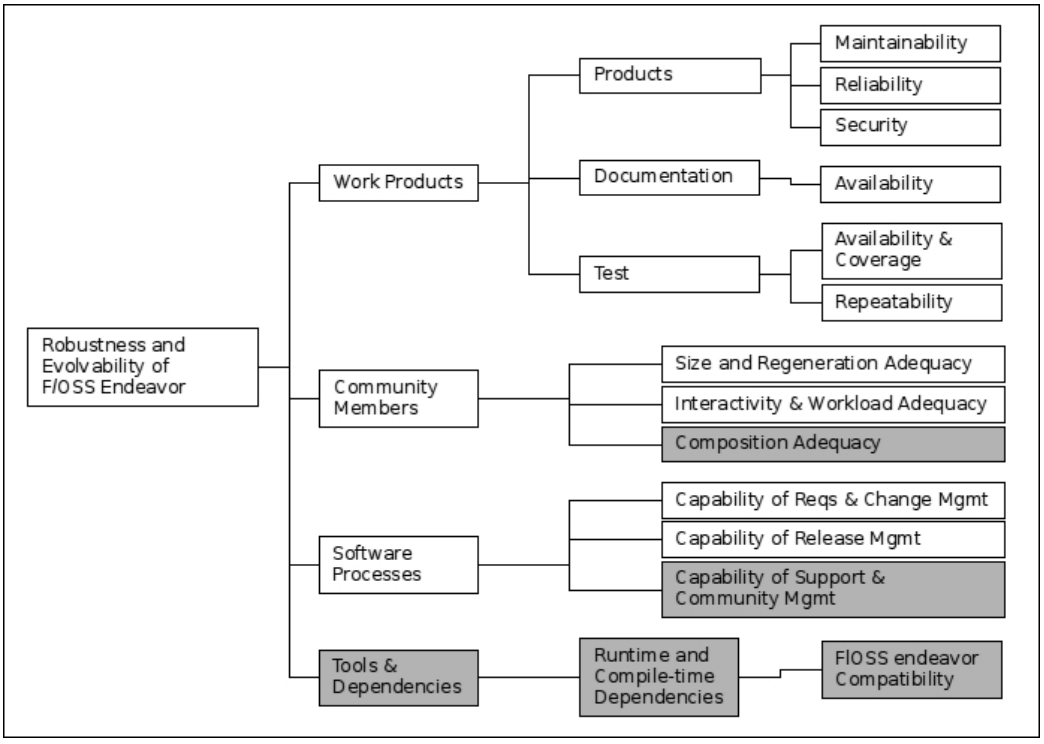

FIGURE 1 – Modèle de qualité hiérarchique de la méthode d'évaluation QualOSS

Ensuite, ces critères sont qualifiés selon les entités constituant l'initiative :

– les produits de travail (*workproduct*) ;
– les membres de la communauté (*community members*) ;
– les processus logiciels (*software processes*).

Chaque entité est ensuite évaluée par rapport à une liste de **caractéristiques**.

Dans le cas des produits de travail, la méthode évalue la fiabilité, la maintenabilité et la sécurité du produit logiciel (*product*). Elle chiffre également la disponibilité de la documentation (*documentation*), ainsi que la couverture, la disponibilité et la répétabilité des tests (*test*).

Dans le cas de la communauté, la méthode analyse nombre de ses membres et sa capacité à se reconstituer. Elle évalue également l'interactivité des membres et l'adéquation de leur charge de travail.

Dans le cas des processus logiciels, la méthode évalue la capacité de gestion des exigences, des changements, ainsi que de la parution des livrables.

QualOSS comprend également un ensemble de scripts automatisant en partie récupération des valeurs des indicateurs définis par son modèle de qualité.

Artefacts, métriques et mesures

Dans le cadre de l'évaluation logicielle, un *artefact* est un ensemble entités composant d'une initiative open-source (les sources, les logs du contrôleur de versions, la liste de mails, etc.), sur lequel l'évaluation logicielle applique des métriques pour en déduire des mesures.

Les *mesures* sont des valeurs définies par une *métrique logicielle* appliquée à un *artefact*.

Une *métrique logicielle* définit une mesure calculée sur un artefact. Il peut s'agir du nombre de lignes de code ou de l'évolution d'un ensemble des membres d'une communauté. Elles servent à mesurer une caractéristique (sens général) d'un produit logiciel. Elles sont définies par rapport au type d'artefacts qu'elles mesurent (mesurer le nombre de lignes de code dans la documentation n'aurait aucun de sens).

Il est possible qu'une métrique soit fonction de mesure d'autres métriques (ex. : des moyennes, des pentes, des rapports du nombre de commentaires par rapport à celui de ligne de codes). Dans ce cas, on parle de *métriques agrégées*.

Indicateurs

L'évaluation d'une caractéristique s'effectue sur un ensemble de valeurs, nommées *indicateurs*, définies selon le paradigme Goal-Question-Metrics[2] appliqué depuis la caractéristique. L'objectif (*Goal*) est défini par le problème, le contexte, le point de vue (ex. : project manager, développeur), l'attention sur la qualité (ex. : "maintenabilité") et l'objet.

Pour chacune de ses caractéristiques, la méthodologie QualOSS définit un ensemble d'indicateurs évaluant l'importance du risque, sur base des valeurs de métriques.

Chaque indicateur possède un critère évalué, une valeur brute et une valeur indicative. Le critère évalué définit clairement ce qu'est la valeur évaluée (ex. : la pente du taux de commits), la valeurs brute définit une valeur directement calculée depuis une ou plusieurs métriques. Enfin, la valeur indicative possède une valeur de -1 à 3,5 décrivant l'importance d'un risque vis à vis de la valeur brute correspondante :

- 0,5 pour un risque fort ;
- 1,5 pour un risque moyen ;
- 2,5 pour un risque faible ;
- 3,5 pour un risque négligeable.

Le rapport entre les valeurs indicatives et les valeurs brutes est établi sur un ensemble de 3 seuils. Ces seuils ont été calibrés sur un nombre de projets open-sources allant de 20 à 300 et d'un terme définissant si le risque est proportionnel à la valeur des seuils (plus la valeur brute est grande, plus le risque est fort) ou inversement (plus la valeur brute est grande, plus le risque est faible).

3 Présentation du sujet

Le CETIC souhaitait le développement d'une **plate-forme d'évaluation automatique du risque** sur des modules open-sources, basée sur la méthode issue du projet européen **QualOSS** (*c.f.* sous-section 2.2).

Cependant, le développement d'une telle plate-forme aurait dépassé la durée allouée par le stage. C'est pourquoi il a été choisi de spécialiser la plate-forme tout en sachant qu'elle pourrait être étendue par la suite. Il a été décidé que cette plate-forme s'appellerait CHOOSE.

Dans le cadre de ce stage, nous nous sommes spécifiquement intéressés à **un cas d'étude** : l'évaluation de projets du consortium **OW2** (*c.f.* sous-section 2.1). Plus précisement, l'objectif était le développement d'une plate-forme d'évaluation un projet d'OW2 et générant sa Mature Transition Checklist. Cette plate-forme devait réaliser ces actions le plus automatiquement possible.

Le choix d'OW2 comme cas d'étude nous a permis de se limiter au :

- langage **JAVA** ;
- contrôleur de versions **Subversion** [9] ;
- bug tracker **JIRA** [1].

À un très haut niveau, le processus utilisé devait correspondre à celui présenté dans la figure 2 (ci-dessous). L'utilisateur paramétrera (URLs du dépôt, de la documentation, nom du projet, etc.) l'analyse en choisissant l'initiative ainsi que les caractéristiques souhaitées (ex. : uniquement la maintenabilité mais pas la communauté) ($ParametersValues$).

Ensuite, à l'aide de ces paramètres et des diverses entités ($Project's\ source\ repository$, $Projet's\ Dashboard$ et $Project's\ Documentation$) constituants l'initiative choisie, un processus ($Analytic\ processus$) gèrera le calcul des métriques propres au modèle de maturité d'OW2 et assurera leur stockage dans une base de données ($Measures$).

Enfin, l'utilisateur pourra directement valoriser ces résultats en envoyant le rapport généré à la communauté du projet concerné ou au comité technologique OW2 ($report$) ou encore, en affichant les résultats sur un site web prévu à cet effet ($UI(web)$).

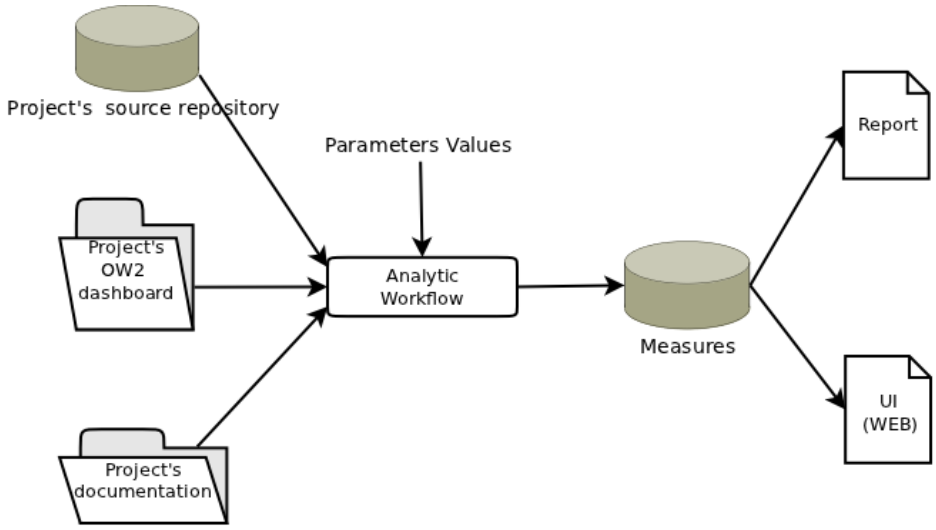

FIGURE 2 – Visualisation du processus.

3.1 Exigences fonctionnelles

Les *exigences fonctionnelles* sont la liste des fonctionnalités devant être fournies par le système à ses utilisateurs.

Dans notre cas, il s'agit de :

- **analyser une initiative** pour en collecter les valeurs des métriques de maturité et celles des indicateurs : sur base du modèle de maturité défini par **OW2** et de la méthodologie **QualOSS**, les métriques des indicateurs seront collectées et les valeurs des indicateurs seront calculées. Cette collecte suit des étapes prédéfinies dans un processus. La mise en oeuvre de ce processus sera réalisée via **Bonita**[3]. L'analyse devra être uniquement paramétrée en début de processus.
- **stocker les valeurs** des métriques, des indicateurs et les paramètres identifiants leurs sources (description de l'initiative, version, date de l'analyse, etc.) dans une base de données : les données seront stockées dans une base de données, de même que les indicateurs définis dans le modèle de qualité **QualOSS**. Ces indicateurs permettront la comparaison avec ceux d'autres initiatives analysées (benchmark).
- **afficher la liste des valeurs des indicateurs dans un rapport web** : une interface utilisateur sous forme d'un site web permettra de rechercher une initiative, de comparer les résultats de différentes analyses d'une même initiative et de comparer différentes initiatives open-sources. De plus, dans le cas où l'analyse pour une initiative donnée n'aurait pas été effectuée, le site web donnera la possibilité d'effectuer une demande d'analyse pour celle-ci.
- **générer un rapport** reprenant la liste des valeurs des indicateurs : un rapport à l'image du formulaire d'évaluation d'**OW2** devra pouvoir être généré sous les formats PDF et ODT.

3.2 Exigences non-fonctionnelles

Les *exigences non-fonctionnelles* sont les contraintes définissant la qualité voulue de l'application, que ce soit au point de vue de sa portabilité, de son évolutivité, de sa fiabilité, de sa performance, etc. Pour ce projet, il a été convenu de se baser sur la norme ISO/IEC 9126 pour définir clairement ces exigences.

Fiabilité

Maturité	Lorsqu'une erreur se produit, un message explicatif devra être affiché à l'utilisateur. Il devra comprendre au moins l'étape où elle s'est produite.
Tolérance aux fautes	Lorsqu'une erreur se produit pendant l'exécution du processus, il faut que la base de données reste consistante.

Convivialité

Compréhensibilité	Les formulaires de paramétrisation d'une initiative ne devra pas contenir de champs ambigus. De plus, chaque étape sera numérotée.
Opérabilité	Le maximum de la paramétrisation devra s'effectuer en début de processus.

Efficacité

Comportement dans le temps	L'analyse automatique d'un critère doit prendre moins de temps que sa version manuelle.
Utilisation des ressources	Le processus **Bonita** stockera l'ensemble des valeurs pour éviter leur recalcul.

Maintenabilité

Analysabilité	Un système d'édition de processus visuel sera utilisé.
Changeabilité	Il devra être possible de modifier facilement le processus, que se soit pour insérer ou supprimer des tâches.
Testabilité	Le processus sera découpé en tâches dont les entrées et les sorties seront clairement identifiables. De plus, les tests seront organisés par tâches.

Portabilité

Adaptabilité	La plate-forme devra être suffisamment générique pour être portable sur d'autres écosystèmes. La structure de la base de données devra permettre l'ajout de nouveaux indicateurs et de nouvelles métriques.
Facilité d'installation	La plate-forme devra pouvoir être installable et utilisable sous les systèmes Linux, Windows et MACos.

4 Présentation des outils

Lors de ce stage, plusieurs outils logiciels ont été utilisés, que ce soit des librairies, des outils de développement ou des outils de design. Ils ont permis de gagner beaucoup de temps de développement, ce qui était important car la durée du stage est limitée.

Cette section présente les outils utilisés pour le développement ou le design de la plate-forme d'évaluation.

4.1 Bonita Open Solution

Bonita Open Solution [3] est une suite de logiciels open-sources composée d'un éditeur WYSIWYG [2] de processus et d'un moteur d'interprétation des processus, tous deux écrits en JAVA et compatible avec MS Windows et Linux.

L'*éditeur de processus* est une application permettant de décrire des processus en **Bussiness Processing Model Notation** ou **BPMN**. Un processus **BPMN** est un ensemble d'entités reliées par des arcs formant un ou plusieurs chemins. Ces entités peuvent être des événements (ex. : début, fin, réception d'un message ou d'une exception), des tâches humaines (entrer des données dans un formulaire), des tâches de services (ex. : rechercher une information en base de données, traitements) ou des appels à un autre processus. Pour les tâches humaines, il est possible de déterminer des utilisateurs et/ou des groupes d'utilisateurs pour permettre le développement de processus multi-utilisateurs.

En plus de l'édition de processus, Bonita permet de **définir des formulaires** depuis un ensemble de widgets de base (liste, boite d'édition, case à cocher, etc.).

Bonita utilise également un concept de **connecteur**. Un connecteur est un programme JAVA offrant un service qu'il est possible d'ajouter à une tâche. Il peut s'agir d'une requête en base données, de l'utilisation d'un web service, etc. Bonita offre un ensemble de connecteurs par défaut pour l'interfaçage avec un large panel de systèmes de gestion de base de données (ex. : Postgres, Access, MySQL, Oracle, Ms SQLServer, etc.), pour l'execution de scripts Groovy ou système, pour l'interfaçage avec SAP, pour l'envoi d'email, etc.

2. What You See Is What You Get

Mais surtout, toute la puissance de l'éditeur de processus de Bonita est qu'il offre la possibilité au développeur de **créer ses propres connecteurs**, grâce à un éditeur JAVA intégré. Cette fonctionnalité est d'autant plus intéressante qu'il est possible de les exporter pour les inclure dans d'autres projets. De plus, la communauté de Bonita publie des connecteurs téléchargeables gratuitement (après inscription) sur le site de Bonitasoft à l'adresse `http://www.bonitasoft.org/exchange/index.php`.

Il est également possible d'inclure des librairies dans les processus. Elles peuvent être ensuite utilisées dans les scripts Groovy ou dans les connecteurs.

La figure 3 (ci-dessous) présente un processus réalisé par l'éditeur de processus de Bonita. Les tâches *Select your country* et *Select your city* sont des tâches humaines offrant à l'utilisateur de spécifier la localisation analysée. La tâche humaine *get location weather* affiche les prévisions météorologiques de la ville choisie. Les tâches possédant le symbole en forme de prise de courant (ex. :*get location weather*) sont des tâches utilisant au moins un connecteur, dans le cas de l'exemple, il s'agit du connecteur vers des web services Les événements sont représentés à l'aide de symboles circulaires :

- *start* et *end* sont les événements de début et de fin du processus ;
- les deux *server crashes* sont des événements les problème lors de l'utilisation des connecteurs.

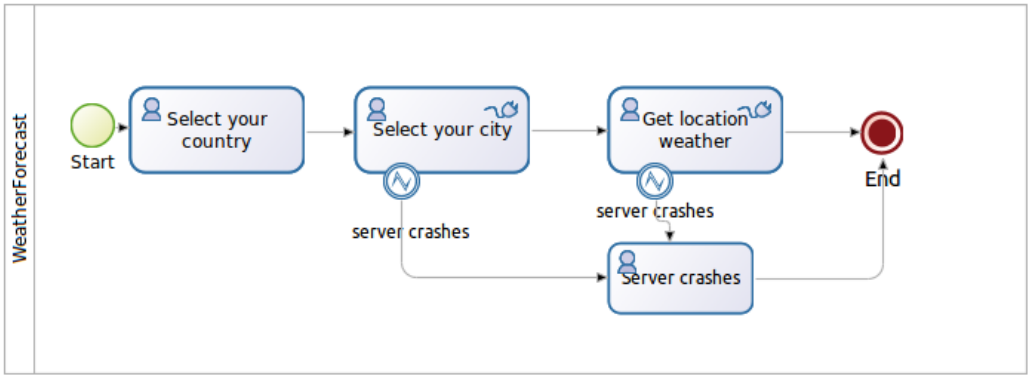

FIGURE 3 – Exemple de processus édité par Bonita Open Solution

Une fois l'édition d'un processus terminée, l'application permet de l'**exporter en une archive BAR** (Bonita ARchive). Cette archive reprend le processus, mais aussi toutes ses dépendances (librairies et connecteurs) ; l'archive se suffit donc à elle-même.

Le *moteur de processus* permet le déploiement d'une ou plusieurs archives BAR et l'interprêtation de leurs processus. Il se présente sous la forme d'une interface web divisée en deux parties :

- un panneau d'administration permettant de déployer des archives et de gérer les processus en cours d'utilisation ;
- un panneau utilisateur permettant de lancer des processus et de générer les formulaires utilisateur.

Dans le cadre du stage, nous avons choisi Bonita Open Solution comme environnement de développement principal, car il permet de :

- **développer rapidement** l'application. Le stage étant limité à 10 semaines, un résultat devait être rapidement atteint ;
- **regrouper** les scripts de QualOSS dans **un processus unique**, car les outils d'évaluation de QualOSS sont un ensemble de scripts dont l'utilisation procédurale est exprimable facilement par l'éditeur de processus ;
- offrir à l'utilisateur la **puissance de calcul** de la machine hébergeant le moteur de processus indépendamment de celle de l'utilisateur ;
- **éviter à l'utilisateur une installation lourde**, QualOSS utilise des scripts ayant des dépendances, dont des systèmes de gestion de base de données.

4.2 HTMLUnit

HTMLUnit est une librairie JAVA permettant d'**émuler un client web** sans GUI [3]. Elle est capable d'**imiter** le comportement de Microsoft Internet Explorer 6 à 8 et Mozilla Firefox 3.6. Elle est sous licence Apache 2 et est donc open-source. Elle offre également la possibilité d'interpréter le Javascript.

À la base, cette librairie avait été développée pour effectuer des tests de sites web, mais sa capacité à émuler un client en fait un outil de choix pour l'analyse de pages web.

De plus, l'interprétation d'une page web par le client crée, d'une *manière robuste*, un objet JAVA Document. Cet objet définit un DOM [4] XML, sur lequel un utilisateur de la librairie peut effectuer des requêtes XPath.

3. Graphic User Interface
4. Document Object Model

Nous définissons *manière robuste* par sa capacité de gérer les pages HTML, même si le format de la page n'est pas du xHTML.

Dans le cadre du projet, HTMLUnit est utilisé pour extraire des informations des pages web relatives au projet analysé (ex. : page d'acceuil, dashboard).

Pour effectuer notre choix, nous avons testé trois autres logiciels capables de créer un DOM XML depuis une page HTML (JTidy, Jericho et le standard Sun javax.swing.text.html.parser.Parser), mais seul HTMLUnit était capable de télécharger la page web, d'interprêter le Javascript et de créer le DOM XML sur toutes les pages nécessaires au projet.

4.3 Jasper Report

Jasper Report est un **générateur de rapport** efficace et de qualité. Il est livré sous forme d'une librairie JAVA ou dans une suite logicielle de Business Intelligence. Il permet depuis une *source de données* et d'un *template* de générer un rapport.

Les *sources de données* peuvent être le résultat d'une requête depuis une base de données (ex. : Postgres, MySQL), mais aussi depuis un document XML. Une seule source de données peut être définie par rapport, mais un rapport peut contenir des sous-rapports dont chacun peut possèder sa propre source de données.

Le *template* est un canvas au format XML définissant comment les données issues de la source de données seront affichées. Elle définit, par exemple, des tableaux, des libellés, des calculs tels que des sommes mais aussi des expressions conditionnelles.

Dans le cadre du projet, Jasper Report est utilisé pour générer la Mature Transition Checklist. Le CETIC avait choisi deux possibilités de générateurs de rapports : Jasper Report ou BIRT. Cependant, l'équipe qualité avait une meilleure expérience de Jasper Report.

4.4 iReport

iReport est un **éditeur WYSIWYG** permettant de conceptualiser des **templates de rapport** pour Jasper Report.

Il permet de dessiner directement le template à l'aide de glisser-déplacer des éléments depuis une toolsbox.

Dans le cadre du projet, il a été utilisé pour définir la génération de la Mature Transition Checklist d'OW2.

5 La plate-forme CHOOSE

La *plate-forme CHOOSE* est la partie principale de l'application développée durant le stage. Elle consiste en un ensemble de processus Bonita, de scripts développés pour le projet QualOSS, d'une base de données et d'une Application Public Interface (API), gérant les interactions entre la base de données et les processus Bonita.

Les fonctionnalités principales de la plate-forme sont au nombre de 3 :

1. la création d'une nouvelle analyse ;
2. l'exécution d'une analyse ;
3. la génération du rapport.

Ces fonctionnalités sont utilisées dans l'ordre cité ci-dessus pour l'analyse d'un projet afin d'en récupérer la Mature Transition Checklist.

Cette section présente en détails chacune de ces 3 fonctionnalités. Pour plus de détails au sujet de la base de données ou de l'API veuillez vous référer à l'annexe B.

5.1 La création d'une nouvelle analyse

La *création d'une nouvelle analyse* consiste en l'étape décrivant le projet à analyser. Cette description est composée de plus de **50 paramètres** définissants le projet (nom, version), des endroits où télécharger les artefacts, mais aussi des informations demandées dans les critères d'**OW2** (*c.f.* sous-section 2.1 pour OW2 et annexe A pour la liste complète des paramètres).

La difficulté de cette étape était son automatisation tout en ayant des paramètres valides. La validité des paramètres est importante car ils sont utilisés par l'analyse. Pour ce faire, l'application CHOOSE permet une correction par l'utilisateur des paramètres récupérés.

De plus, il est possible que certains paramètres soient difficiles à rechercher, voir introuvables parmi les sources de données du projet (pages web). C'est pourquoi l'utilisateur doit pouvoir les compléter.

La *création d'une analyse* est divisée en :

1. La **description** du projet à analyser ;
2. La **récolte** des paramètres ;
3. La **correction** des paramètres par l'utilisateur ;
4. La **sauvegarde** en base de données.

La description du projet à analyser demande des informations de base telles que le nom du projet, sa version, la date de release correspondante à la version analysée et un commentaire.

Une fois cette description réalisée, l'application utilise l'API pour instancier un objet `Analysis`. Cet objet sera ensuite utilisé pour le stockage de tous les paramètres avant sa sauvegarde en base de données.

La récolte des paramètres

La *récolte des paramètres* consiste en un ensemble d'outils extrayant, depuis des pages web, les informations relatives au projet analysé. Ces outils ou extracteurs sont chacun des connecteurs Bonita développés spécifiquement pour récupérer les informations de la ou les pages web correspondantes.

De plus, des heuristiques ont été élaborées pour les informations absentes des pages web, mais pouvant être inférées depuis les données présentes.

Les pages web téléchargées sont :

- la page web OW2 des *projets par maturité* ;
- la page web OW2 des *projets par fonction* ;
- le tableau de bord OW2 (*dashboard*) du projet à analyser ;
- la *page d'accueil du projet* à analyser ;
- la *page d'accueil du projet Bamboo*.

Il y a un extracteur par page web, sauf pour la *dashboard* et la page des *projets par fonction* car les informations issues de ces pages appartiennent au même critère de la MTC [5].

L'extracteur des *projets par maturité* n'a pas pour but la récupération d'informations propres à un projet, mais à tous les projets. Cet extracteur récupère la liste

5. Pour rappel : la Mature Transition Checklist d'OW2

des projets d'OW2 ainsi que l'url vers leurs *dashboards*. Cela permet de simplifier la sélection d'un projet et d'être certain que l'utilisateur tente de décrire un projet d'OW2.

L'extracteur des *projets par fonction* et de la *dashboard* récupère deux pages web et les analyse. L'objectif est de lister l'ensemble des informations propres à la complétude de la dashboard de la MTC. Cette récupération est importante car elle liste un grand nombre d'informations utilisées par les heuristiques pour déterminer les informations manquantes.

L'extracteur de la *page d'accueil* du projet permet de récupérer un ensemble de liens à l'aide d'un système de matching simplifié. Un projet OW2 possède une page web dite d'accueil pouvant être externe, mais devant être renseignée dans la *dashboard* du projet. Cette page est très intéressante car elle contient souvent les liens vers la documentation, les binaires et la convention de code.

Pour chaque type de lien recherché, une liste de termes a été définie. Ces termes sont soit des mots (ex. : "documentation"), soit un ensemble de mots (ex. : "developper corner").

Lors de son utilisation, pour chaque type de lien, l'extracteur recherche dans la page web le premier lien dont le label ou la dernière partie de son URL contient au moins un terme présent la liste du type recherché.

De plus, pour plus de facilité, la correspondance s'effectue sur une version altérée des termes, du label et de la dernière partie de l'URL des liens. Cette version altérée est la version normale à laquelle l'extracteur supprime les espaces, les underscores et remplace les majuscules par des minuscules. Comme exemple, prenons le cas du lien avec comme URL "http ://example.ow2.org/thisIsASample/Developper_Corner" et comme label "How to help". L'url du lien sera transformée en "developpercorner" et son label en "howtohelp".

L'extracteur de la *page d'accueil du projet Bamboo* est utilisée pour rechercher le Continuous Integration Mechanism utilisé par le projet. Bamboo est un projet centralisant l'intégration continue de plusieurs projets d'OW2. Si un projet l'utilise, OW2 demande qu'il le renseigne dans la MTC. La page d'accueil de Bamboo liste l'ensemble des projets inscrits et leurs liens vers leurs pages personnelles sur Bamboo.

C'est pourquoi cet extracteur a été créé. Il récupère et analyse la page d'accueil de Bamboo. Si le projet y est renseigné, l'extracteur retourne l'URL vers sa page personnelle sur Bamboo.

La correction des paramètres

La *correction des paramètres* consiste en un ensemble de formulaires affichant les informations récupérées par les extracteurs.

Chaque formulaire identifie soit un critère d'OW2, soit un groupement de paramètres indispensables à l'analyse.

Après l'extraction des données de la dashboard, mais avant l'utilisation des autres extracteurs, l'application CHOOSE va présenter la liste des informations de la dashboard dans un formulaire. Ces informations ne sont pas modifiables ; CHOOSE considère que les informations issues de la dashboard sont correctes. Il est possible de recharger ce formulaire, ce qui relance une extraction depuis la dashboard du projet. Cette boucle permet à l'utilisateur de compléter et/ou de corriger la dashboard avant de continuer.

Ensuite, après l'application des autres extracteurs, l'utilisateur arrive sur une suite de formulaires correspondants aux critères de la MTC. La liste des formulaires est, dans l'ordre :

1. sources, documentation et binaires
2. compilation
3. usage du projet
4. gestion du contrôleur de versions
5. définition de la liste des versions
6. définition du bug tracker
7. conventions de code
8. activité

Les deux critères manquants (committers et SQuAT) ne nécessitent aucun paramètre. Les informations requises par la MTC seront calculées pendant l'analyse.

La définition de la liste des versions et du bug tracker sont des formulaires regroupant des paramètres indispensables à leurs analyses. La définition des versions sera détaillée plus bas et celle du bug tracker reprend l'URL vers le bug tracker ainsi que son type.

Chaque formulaire reprend les informations sous forme d'une suite de widgets permettant de modifier la valeur du paramètre associé.

A chaque soumission d'un formulaire, l'application affiche le suivant, mais il est également possible de revenir à un formulaire antérieur. Une fois que l'utilisateur soumet le dernier formulaire (activité), l'application sauvegarde l'analyse contenue dans l'objet `Analysis` en base de données à travers l'API.

Le formulaire le plus important est celui de la définition de la liste des versions. Certaines caractéristiques de la méthodologie QualOSS (*c.f.* sous-section 2.2) nécessitent plusieurs versions, car certains des indicateurs qu'elle calcule se basent sur l'évolution entre plusieurs versions.

La méthodologie QualOSS recommande de prendre en considération au moins les deux dernières versions majeures et les deux dernières mineures, en plus de la version à analyser, considérée comme une version mineure. La dernière version majeure correspond également à la moins récente des mineures.

La figure 4 (ci-dessous) reprend les versions prises en compte dans CHOOSE. Il y a les deux versions majeures, la dernière mineure et enfin, la version à analyser. S'il n'y a pas assez de versions, l'utilisateur doit prendre des snapshoots des sources tels qu'il y ait au moins 6 mois entre les mineures et au moins 2 ans entre les majeures, comme le stipule la méthodologie QualOSS.

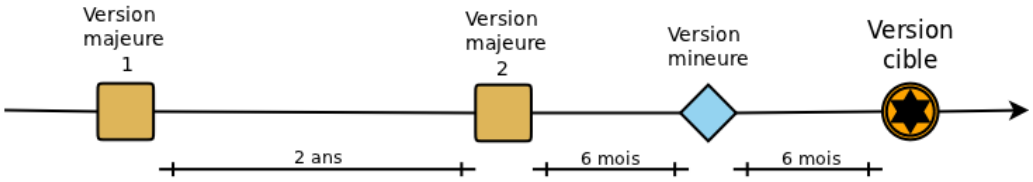

FIGURE 4 – Liste des versions nécessaires à l'analyse

Dans le cas de CHOOSE, le nombre de versions majeures et mineures est une limite maximale : il n'est pas possible de spécifier des versions supplémentaires. Cette limite est suffisante (surtout pour les versions majeures) pour une grande partie des projets d'OW2. En fait, bon nombre de ses projets ne possèdent pas les deux versions majeures requises ou la durée requise (3 ans d'ancienneté). C'est pourquoi, il est possible de ne spécifier qu'une majeure au lieu de deux. Cela permet de couvrir beaucoup plus de projets sans pour autant limiter les possibilités d'analyse.

Pour chaque version, l'utilisateur doit définir la date de release, le numéro de la version correspondante et sa localisation. Il y a deux choix possibles de localisation : l'URL vers une archive ou vers un répertoire d'un dépôt de versions

Subversion. Par défaut, la localisation est le dépôt des sources renseigné dans le formulaire "sources, documentation et binaires".

5.2 L'analyse d'un projet

L'*analyse d'un projet* est la phase durant laquelle l'application calcule les valeurs des métriques sur les différents artefacts du projet.

Le but de l'analyse est de fournir une solution alternative **automatisée** au critère SQuAT de la MTC d'OW2. À la base, ce critère nécessite un rapport d'évaluation de la qualité de Sonar [8], un rapport de licence de FOSSology [5] et une analyse de la maturité selon le modèle de Qualipso [7].

Alors que Sonar et FOSSology sont des logiciels automatisés, le modèle de Qualipso est un modèle composé d'une suite de niveaux ("basic", "medium" et "advanced"), chacun composé d'un ensemble de critères à satisfaire pour y accéder.

Actuellement, Qualipso ne propose pas d'outil pour l'évaluation d'un projet, mais uniquement une fiche d'évaluation. Cependant, cette fiche n'est pas objective : il s'agit de donner manuellement des notes de 1 à 4 pour chaque critère.

C'est pourquoi CHOOSE utilise QualOSS pour l'évaluation complète du critère SQuAT. Il comprend un ensemble de scripts automatisant plusieurs de ses caractéristiques.

Au niveau du processus d'analyse, une fois que l'utilisateur a choisi le projet qu'il souhaite analyser, CHOOSE lui propose de choisir parmi une liste de trois caractéristiques : maintenabilité, fiabilité et communauté.

Lorsque les caractéristiques voulues ont été choisies, CHOOSE demande à l'utilisateur une adresse email. Elle permettra à l'application de le prévenir une fois l'analyse terminée.

Puis, CHOOSE *récupère les artefacts* uniquement nécessaires aux caractéristiques choisies, s'ils ne l'ont pas déjà été lors d'une analyse précédente.

Enfin, CHOOSE *calcule les métriques* sur les artefacts récupérés à l'étape précédente, pour obtenir la liste des mesures nécessaires au calculs des indicateurs de QualOSS.

La récupération des artefacts

La *récupération des artefacts* consiste au téléchargement des artefacts du projet correspondants à l'analyse choisie.

Des connecteurs Bonita ont été implémentés pour permettre que cette étape soit totalement automatisée. Ils ont permis de télécharger les artefacts en local sur la machine hébergeant CHOOSE, sans que la machine de l'utilisateur n'ait à s'en soucier. Cela permet également de centraliser toutes les analyses sur la machine contenant CHOOSE.

Les artefacts récupérés sont :

- Les différentes versions des *sources* ;
- Les *logs du système de contrôle de version* ;
- La *liste du nombre de mails par mois* pour les 6 derniers mois ;
- La *liste des bugs* pour l'ensemble du projet.

Premièrement, la récupération des *sources* consiste en celle de la version analysée et des anciennes, ainsi que de l'extraction de l'ensemble des fichiers sources.

Pour les récupérer, il y a deux solutions dépendantes du type d'URL fournis : soit l'URL pointe vers un répertoire du dépôt (cas 1), soit vers une archive (cas 2). La différence se fait par la présence dans l'URL du préfixe "svn ://" (1) ou par celle du préfixe "http ://" (2).

Dans le cas (1), l'application réalise un téléchargement (*checkout*) depuis le dépôt spécifié vers un dossier d'analyse. Cette étape est limitée par l'utilisation du gestionnaire de contrôle de versions Subversion [9]. De plus, comme le système de gestion le permet, CHOOSE récupère l'état du dépôt à la date fournie lors de la définition des versions.

Il y a cependant un problème : les dépôts comportent souvent des répertoires contenants les anciennes versions du projet (`tags`) et des branches (`branches`). Or, l'application n'a besoin que du point de développement principal, souvent un dossier nommé `trunk`.

On pourrait se dire qu'il suffirait de détecter ce répertoire et d'effectuer un *checkout* depuis ce dossier, mais certains dépôts contiennent une suite de modules, chacun possédant un dossier qui contient, à son tour, un dossier `trunk`. De plus, tous ces dossiers ne sont pas toujours au même niveau dans l'arborescence du dépôt.

Pour résoudre ce problème, la solution proposée consiste en une exploration en largeur (BFS) depuis la racine du dépôt. Cette exploration aura pour but de découvrir l'ensemble des dossiers `trunk` contenus dans le dépôt. Puis, pour chaque répertoire `trunk` trouvé :

- l'application crée le dossier de sortie, comme étant le chemin allant de la racine au `trunk`. Par exemple, pour le chemin svn ://svn.forge.objectweb.org/svnroot/chameleon/trunk, l'application ne créera rien, mais pour svn ://svn.forge.objectweb.org/svnroot/accord/forby/applet/trunk, l'application créera le dossier forby/applet ;
- l'application effectue un *checkout* dans le dossier de sortie.

Dans le cas de l'URL de type (2), il s'agit d'une URL vers une archive. CHOOSE la télécharge dans un répertoire dédié aux archives à l'aide de la commande unix "wget".

Deuxièmement, la *récupération des logs* du système de contrôle de versions consiste à télécharger la liste de ses logs sous les formats texte et XML.

Troisièmement, la *récupération du nombre de mails par mois* consiste en l'analyse des pages de la forge OW2 du projet, pour en récupérer la somme du nombre de mails reçus et envoyés par mois, pour les 6 derniers mois, par rapport à la date de lancement de l'analyse.

Pour ce faire, tout d'abord, CHOOSE utilise HTMLUnit pour télécharger et analyser la page d'accueil de la forge du projet. Il récupère ainsi l'URL vers la page listant des boites email. Avec cette seconde page, CHOOSE procède de la même manière afin de lister les URLs vers les archives des boites email.

À ce stade, CHOOSE filtre les URLs référençant la boite mail des commits et celle des bugs sur base du contenu de leurs noms :

- si elle contient le mot "commit", elle correspond à la boite mails des commits ;
- si elle contient le mot "issue", elle correspond à la boite mails des bugs.

Ensuite, les pages des archives restantes se présentent sous la forme d'un tableau dont chaque ligne reprend le mois, l'année et le nombre d'emails reçus ou envoyés durant cette période, depuis le début du projet. CHOOSE passe en revue chacune des lignes de chaque boite email et récupère uniquement le nombre de celles ne dépassants 6 mois d'ancienneté. Comme les dates sont classées de la

plus récente à la plus ancienne, CHOOSE s'arrête dès qu'il dépasse les 6 mois.

Quatrièmement, la *récupération de la liste de bugs* consiste en une requête HTTP GET retournant l'ensemble des entrées du bug tracker, sous format XML. Cette étape n'accepte que les bug trackers JIRA [1].

Soient les variables `<PROJECT_NAME>` le nom du projet analysé en majuscules et `<JIRA_HOME>` l'URL vers la racine du bug tracker JIRA pour ce projet (pour OW2, il s'agit de "jira.ow2.org"), la requête HTTP permettant de récupérer l'ensemble des entrées du bug tracker est :

```
<JIRA_HOME>/sr/jira.issueviews:searchrequest-xml/temp/
```

suivie par :

```
SearchRequest.xml?jqlQuery=project+\%3D+<PROJECT_NAME>
&tempMax=1000
```

Une fois la liste de bugs récupérée, elle est transformée sous format CSV car les scripts de QualOSS nécessitent ce format. Il existe également une requête HTTP GET retournant la liste des bugs sous format CSV, mais certaines informations devaient être filtrées et réordonnées : l'ordre des informations a de l'importance pour les scripts utilisés. C'est pourquoi, CHOOSE utilise le format XML.

Pour réaliser cette transformation, CHOOSE passe en revue chaque entrée du bug tracker dans le fichier au format XML et pour chacun, il reprend les informations nécessaires et les imprime dans une chaîne de caractères de sortie, dans l'ordre demandé. La séparation des informations suit les règles d'un format CSV avec comme caractère séparateur ' ;'.

Le calcul des métriques

Le *calcul des métriques* est la dernière étape de l'analyse avant la sauvegarde des résultats en base de données. Elle reprend le calcul d'environ 60 métriques réparties dans les 3 caractéristiques évaluées :

- la *communauté* ;
- la *fiabilité* ;
- la *maintenabilité*.

L'ensemble de ces calculs utilise les scripts de QualOSS pour en obtenir les mesures, excepté pour l'analyse de la liste du nombre de mails par mois et pour une partie de l'analyse des logs du dépôt de versions, qui ont tout deux nécessité le développement de connecteurs Bonita spécifiques.

La difficulté de l'analyse était de **comprendre** et **regrouper** ces scripts sous forme d'un processus Bonita. Ce processus devait être identique au processus d'analyse de la méthodologie QualOSS. Ce processus ne devait pas nécessiter d'interaction avec l'utilisateur. Pour un soucis de performance, il devait être également parallélisé.

Pour le calcul de ces métriques, des connecteurs ont été définis pour chaque caractéristique et pour chaque type d'artefacts considéré. Ce qui fait un total de 15 connecteurs.

La figure 5 (ci-dessous) présente le processus simplifié utilisé pour calculer et enregistrer les résultats en base de données. En premier lieu, les traitements de chaque caractéristique ont été parallélisés :

- le chemin du **haut** calcule les métriques propres à la caractéristique *communauté* ;
- le chemin du **milieu** calcule les métriques propres à la caractéristique *fiabilité* ;
- le chemin du **bas** calcule les métriques propres à la caractéristique *maintenabilité*.

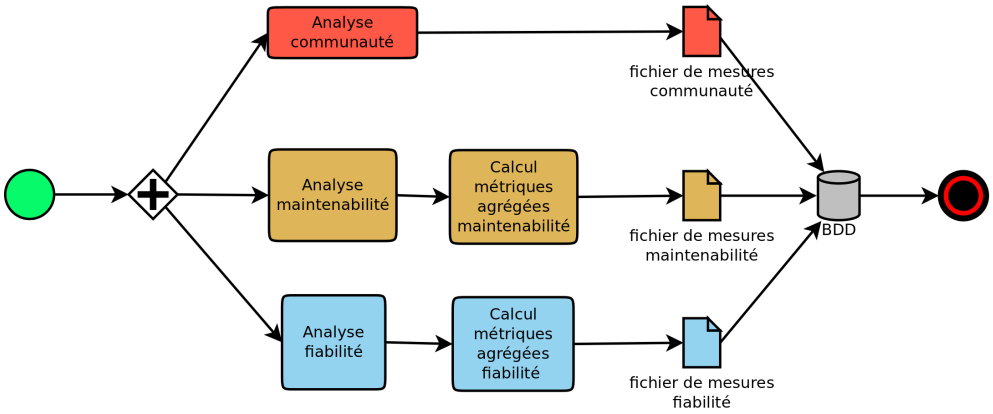

FIGURE 5 – Processus de calcul des métriques.

L'analyse de la *communauté* applique ses métriques sur les logs du dépôt de versions et le nombre de mails par mois. Cette analyse est décomposée en :

- l'analyse des logs du dépôt, calculant les métriques nécessaires à l'évaluation QualOSS, la date du dernier commit et le nombre de commits effectués par la communauté lors des 6 derniers mois ;
- l'analyse de la liste de mails, évaluant la moyenne du nombre de mails par mois.

L'analyse de la *fiabilité* et celle de la *maintenabilité* appliquent leurs métriques sur les sources des différentes versions ainsi que sur la liste des bugs.

L'analyse de la *fiabilité* est décomposée en :

- l'analyse de l'évolution entre les différentes versions des sources ;
- l'analyse des violations de la convention de code sur la version courante ;
- l'analyse de la liste de bugs en fonction des dates de release des versions.

L'analyse de la *maintenabilité* est décomposée en :

- l'analyse de l'évolution entre les différentes versions des sources ;
- l'analyse de la version courante ;
- l'analyse de l'ensemble des bugs.

Ensuite, chaque analyse retourne un fichier de mesures. Ces fichiers sont au format XML et listent pour chaque mesure, le chemin vers l'artefact, le nom de la métrique, le type de l'artefact et enfin, la valeur de la mesure. Ce format de fichier a été défini pour uniformiser les sorties des traitements, qu'elles viennent de scripts ou de connecteurs Bonita.

Enfin, L'API CHOOSE récupère l'ensemble des fichiers de mesures et les insère en base de données.

5.3 La génération du rapport

La *génération du rapport* est l'étape durant laquelle CHOOSE affiche les résultats, sur base desquels il génère un rapport portable, sous forme d'un fichier.

Cette étape est divisée en 3 parties :
- le calcul des indicateurs ;
- l'affichage des résultats ;
- la génération du rapport portable.

Le calcul des indicateurs

Le *calcul des indicateurs* évalue la liste des indicateurs pour chaque caractéristique supportée par l'application, comme défini par la méthodologie QualOSS (*c.f.* sous-section 2.2), ainsi qu'un indicateur de risque global à la caractéristique.

Il était également possible de placer ce calcul après celui des métriques. Cependant, le placer avant l'affichage du rapport permet d'afficher un rapport d'un point de vue différent sans pour autant recalculer les métriques [6].

Pour effectuer ce calcul, une classe a été définie par caractéristique (`CommunityIndicator`, `ReliabilityIndicator`, `MaintainabilityIndicator`), leurs méthodes de calculs étant légèrement différentes :

- Dans le cas de *communauté* et *fiabilité*, la valeur brute équivaut à une métrique.
- Dans le cas de *maintenabilité*, la valeur brute est calculée par :

$$v = r * \frac{n}{d}$$

  Où r est le signe d'une métrique (1 ou -1), n est une valeur de métrique et d est une valeur de métrique ou 1.

Comme il est possible qu'il manque des artefacts pour certains projets (ex. : pas de bug tracker JIRA, le projet possède moins de 2 versions majeures), l'application CHOOSE utilise deux valeurs indicatives supplémentaires agissantes comme un code. Elles définissent le manque d'artefact (valeur de 0) et le calcul non effectué (valeur de -1). Ce système permet de faire en sorte que le visionneur du rapport sache si relancer l'analyse sur un critère apportera de nouvelle(s) information(s) ou pas.

6. La gestion des différents points de vues est une évolution possible de CHOOSE

Pour chaque indicateur, CHOOSE définit une instance de la classe correspondante à la caractéristique associée, reprenant les seuils définis par la méthodologie QualOSS.

Enfin, les indicateurs de risque globaux sont les moyennes des indicateurs de la caractéristique correspondante. Ces moyennes ne prennent pas en compte les indicateurs ne pouvant pas ou n'étant pas calculés.

L'affichage des résultats

L'*affichage des résultats* est constitué d'une suite de rapports.

Le premier rapport affiche les informations du projet analysé comme son nom, sa version, sa date de release de la version ainsi que son commentaire défini lors de la création de l'analyse.

Ensuite, le deuxième rapport reprend la Mature Transition Checklist d'OW2 (*c.f.*, sous-section 2.1) sous format HTML. Grâce à ce format, CHOOSE fait en sorte que chaque hyperlien fasse référence à la page web correspondante ; l'utilisateur peut ainsi naviguer sur les différentes pages du projet facilement. De plus, les informations du critère d*ashboard* sont affichées en clair et non pas au travers d'un "Yes" ou "No" définissant la présence ou non de l'information sur la page.

Puis, un rapport pour chaque caractéristique est affiché. Il reprend une description de la caractéristique, la valeur de son indicateur global de risque, un tableau affichant la liste de ses indicateurs et un commentaire éditable. Pour chaque indicateur, l'application affiche le nom de l'indicateur, une description détaillée de l'indicateur, l'ensemble de ses seuils, sa valeur indicative sous forme d'une couleur et sa valeur brute. Les couleurs des valeurs indicatives sont les suivantes :

- -1 en gris avec le terme "Not yet computed" ;
- 0 en gris avec le terme "can't be computed" ;
- $0,5$ en noir ;
- $1,5$ en rouge ;
- $2,5$ en jaune ;
- $3,5$ en vert.

La figure 5.3 (page suivante) présente un exemple de rapport complet pour la caractéristique communauté, dans le cas du projet OW2 *Chameleon*. En haut, il y a une courte description de la caractèristique, les artefacts sur lesquels l'analyse s'applique et le lien vers le site du projet QualOSS. Puis, se trouve la légende

des couleurs et la valeur de l'indicateur de risque global pour la communauté. Au centre se trouve le tableau des indicateurs suivit du champ du commentaire. Enfin, le rapport se termine par une note concernant le commentaire et les boutons de navigation.

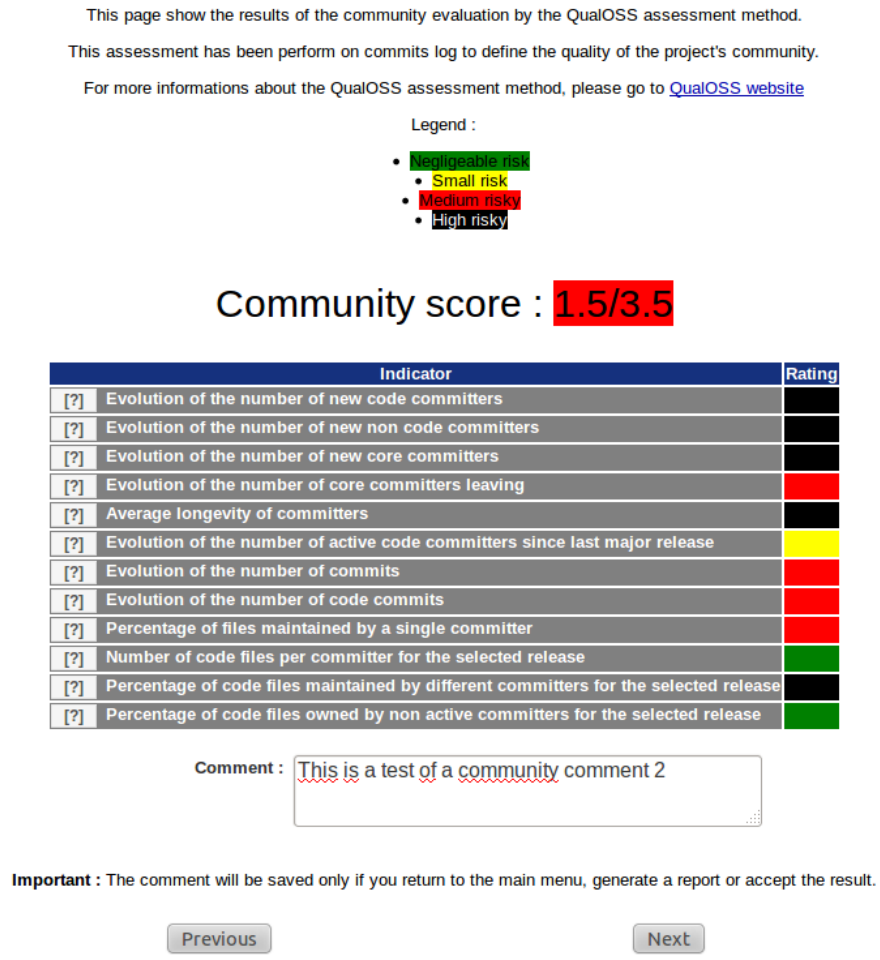

FIGURE 6 – Exemple de rapport d'évaluation de la communauté pour *Chameleon*

Enfin, CHOOSE affiche un formulaire d'action permettant à l'utilisateur de choisir de générer un rapport portable.

La génération du rapport portable

La *génération du rapport portable* permet à l'utilisateur de transformer les résultats d'une analyse sous forme d'un fichier. Le format du fichier est choisi par l'utilisateur dans la liste suivante : RTF, PDF, ODT et DOCX.

Tout d'abord, cette étape utilise l'API pour récupérer sous forme d'un fichier au format XML tous les paramètres d'une analyse, ainsi que les valeurs des métriques et des trois indicateurs de risque globaux.

Ensuite, ce fichier est utilisé comme source de données d'un template de rapport Jasper Report (*c.f.* section 4) pour générer le rapport final.

Enfin, le rapport final correspond à la Mature Transition Checklist d'OW2, avec une modification appliquée au niveau du critère SQuAT, lequel prend en compte la méthodologie QualOSS et la valeur des indicateurs globaux. L'annexe D présente un exemple de la Mature Transition Checklist générée pour le projet *Chameleon*.

6 Résultats

Cette section traite de la validation des résultats obtenus par l'application, au niveau des contraintes fonctionnelles ou non-fonctionnelles.

Premièrement, la *validation fonctionnelle* consiste en la validation de l'ensemble des fonctionnalités définies dans la sous-section 3.1.

Deuxièmement, la *validation non-fonctionnelle* consiste en la validation des exigences non-fonctionnelles définies dans la sous-section 3.2.

6.1 Validation fonctionnelle

Pour chacune des fonctionnalités demandées, la *validation fonctionnelle* a consisté en la vérification de sa présence ou non dans l'application ainsi qu'en la manière dont elle a été validée.

L'analyse d'une initiative

Cette validation a été divisée en deux : celle de la *création d'une analyse* et celle de *l'analyse*.

Tout d'abord, la *création d'une analyse* a été validée sur 21 projets différents. Les informations (si elles sont présentes) sont correctes. Cependant, il y a des problèmes lors de l'utilisation de HTMLUnit sur les page d'accueil de 2 d'entre eux (Contrail et Spago4Q).

Ensuite, dû à nos restrictions pour le bug tracker, pour le gestionnaire de dépôt et surtout pour le nombre de versions, l'*analyse* a été validée sur un ensemble restreint de 4 projets :

- *Chameleon*
- *Spago*
- *Spago4Q*
- *Orchestra*

La table 1 (ci-dessous) présente les résultats des indicateurs de risque globaux obtenus par l'analyse des 4 projets cités ci-dessus (*c.f.* annexe C pour les résultats de tous les indicateurs). Ces résultats ont été validés par la comparaison avec ceux obtenus par une application manuelle de la méthodologie QualOSS (*c.f.* sous-section 2.2). On remarque un risque *faible* pour la fiabilité en moyenne $(2, 56)$ et un risque *faible* à *moyen* pour la maintenabilité en moyenne $(2, 0625)$. Par contre, la communauté possède un plus large panel de valeurs (de $1, 5$ à $2, 42$), la communauté de *Chameleon* étant la plus risquée avec un risque *moyen*.

Nom du projet	Communauté	Maintenabilité	Fiabilité
Chameleon	1,5	2,21	2,28
Spago	1,92	1,92	2,79
Spago4Q	2,42	2,12	2,89
Orchestra	-1	2	2,28

TABLE 1 – Résultats des analyses

Le score de la communauté de -1 de *Orchestra* provient d'un problème de calcul ne permettant pas l'évaluation de la communauté, c'est pourquoi nous ne la considérons pas.

De plus, *Spago* ne possédait pas deux versions majeures. Les caractéristiques ont donc été évaluées sur une seule version majeure. Cependant, seuls 4 indicateurs (2 pour maintenabilité et 2 pour fiabilité, sur 35 au total) n'ont pas pu être calculés. Ce nombre montre le faible impact du manque de version majeure, ce qui est important pour OW2 qui possède peu de projets avec plus d'une version majeure.

Enfin, *Spago4Q* ne possédait pas de bug tracker. Dans ce cas, l'impact est bien plus important que pour le nombre de versions majeures. 12 indicateurs (6 pour maintenabilité et 6 pour fiabilité) n'ont pas pu être calculés.

Le stockage des valeurs

L'application devait sauvegarder les valeurs des métriques, des indicateurs et des paramètres identifiants leurs sources (description de l'initiative, version, date de l'analyse, etc.) dans une base de données.

La base de données développée spécifiquement pour ce stage (*c.f.* annexe B) stocke l'ensemble des informations relatives à un projet sous forme de paramètres.

De plus, une fois calculées, les métriques sont sauvegardées dans cette même base de données.

Enfin, les indicateurs ne sont pas sauvegardés en base de données, à l'exception des indicateurs de risque globaux. Seuls les indicateurs de risque globaux étaient indispensables pour la génération de la Mature Transition Checklist.

L'affichage des valeurs des indicateurs dans un rapport web

L'application devait pouvoir afficher la liste des indicateurs dans un rapport web. C'est chose faite, car l'ensemble des indicateurs des caractèristiques évaluées sont présentés sous la forme d'un tableau, contenu dans une page web par caractéristique.

La génération du rapport

L'application devait générer la Mature Transition Checklist d'OW2 pour un projet donné ; c'est le cas, car la plate-forme génère un rapport identique à la Mature Transition Checklist. La seule modification est le remplacement du critère SQuAT, par un rapport d'évaluation de QualOSS.

6.2 Validation non-fonctionnelle

Pour chacune des exigences non-fonctionnelles, la *validation non-fonctionnelle* définit si l'exigence a été respectée ou non, ainsi qu'un justificatif.

Fiabilité

Lors de la création d'une analyse, l'application traite les erreurs. Seules les pages web peuvent causer des problèmes lors de leurs analyses. De ce fait, dès qu'une page web n'est pas traitée correctement, elle est ignorée.

Lors de chaque étape de l'analyse, l'application affiche un message d'erreur identifiant l'étape où ce situe le soucis. Dans le cas de la récupération des artefacts, l'application identifie également l'artefact concerné.

L'utilisation de l'API (*c.f.* annexe B) se charge de faire en sorte que les sauvegardes soient atomiques et, donc, que la base de données reste consistante.

Convivialité

L'application décrit le contenu de chacun de ses formulaires de manière claire, détaillant chaque champ pour lever toute ambiguïté.

La paramètrisation s'effectue principalement lors de la création d'une analyse, se plaçant toujours au début de l'ajout d'une nouvelle analyse.

Efficacité

L'analyse prend effectivement moins de temps que sa version manuelle. Lors des lectures des rapports de réunion d'OW2, le conseil technologique avait passé 3 heures sur l'évaluation, selon le modèle de Qualipso. L'application passe au maximum 20 minutes, toutes caractèristiques comprises (cas de Orchestra).

Les processus Bonita développés sauvegardent l'ensemble des valeurs utilisées, évitant ainsi leur recalcul.

Maintenabilité

L'outil principal de développement, Bonita Open Solution, est un système d'édition de processus visuel.

De plus, il permet d'insérer/modifier/supprimer simplement les tâches composant tous les processus de l'application.

Enfin, il permet les tests de chaque connecteur indépendamment des autres, que ce soit dans l'éditeur ou hors de l'éditeur [7].

7. Il est possible d'exporter les sources d'un connecteur, de les importer et les réutiliser après suppression de l'utilisation des classes de Bonita.

Portabilité

L'application est capable d'être modifiée pour être utilisée sur d'autres écosystèmes. Les seules modifications se feront au niveau de l'extraction des données depuis les pages web et de la récupération du nombre d'emails par mois.

Comme l'ensemble des scripts de QualOSS utilise des outils et des commandes développés sous UNIX, l'application est donc limitée à cette architecture. Cependant, comme il s'agit d'une interface web, l'utilisateur n'aura pas besoin de l'installer sur sa machine. Donc, quelle que soit la plate-forme de l'utilisateur, la seule contrainte de portabilité sera l'existence d'un explorateur web moderne interprêtant le Javascript.

7 Travaux complémentaires

Cette section liste les différentes possiblités d'extensions de l'application ou les exigences non-implémentées durant le stage, qu'elles soient fonctionnelles ou non-fonctionnelles.

Tout d'abord, CHOOSE est capable d'analyser 3 caractèristiques de la méthodologie QualOSS sur 7. Il est possible d'ajouter la caractèristique de *sécurité* sans pour autant nuire à l'automatisation de l'analyse. L'ajout des caractèristiques *processus*, *documentation* et *tests* sera plus compliqué, car QualOSS ne propose pas de méthode automatique pour les évaluer.

Comme cité lors de la section 3, le CETIC souhaiterait que l'application englobe d'autres écosystèmes. CHOOSE ne traite que le cas d'OW2, mais il est possible de l'étendre vers d'autres. Étendre à d'autres écosystèmes impliquerait l'ajout de la gestion d'autres langages, bug trackers et gestionnaires de contrôle de versions.

De plus, CHOOSE possède une limite sur le nombre de versions qu'il est capable d'analyser ; il serait intéressant de supprimer cette limite pour obtenir une analyse plus complète.

Lors de la génération du rapport, Jasper Report tronque les contenus dépassant la taille de leurs contenants. Cela pose des soucis lorsque les descriptions sont trop longues. Il faudrait faire en sorte que les champs se redimensionnent afin que ce problème soit résolu.

Enfin, l'utilisateur pourrait choisir une suite de projets pour les comparer. Cette fonctionnalité serait d'autant plus utile si l'utilisateur était capable de définir un ensemble de coefficients aux indicateurs, pour modifier leurs poids dans le calcul des indicateurs de risque globaux.

Conclusion

Du point de vue du **CETIC**, l'application développée correspond à ses attentes, tant au niveau de l'automatisation qu'au niveau de l'évolutivité. Il reste à intégrer une grande partie des caractéristiques de QualOSS, mais leur automatisation aurait pris beaucoup trop de temps par rapport au temps alloué au stage. La création d'une analyse est presque totalement automatisée. La récupération des artefacts et le calcul des métriques le sont complètement.

Au niveau de l'évolutivité, il est facile d'ajouter à l'application de nouveaux paramètres, artefacts et métriques, grâce à la puissance de Bonita Open Solution et à l'architecture de la plate-forme développée. Le calcul des indicateurs est simple et évolutif, il suffit de créer une nouvelle instance de la classe correspondante pour l'ajouter.

L'affichage des résultats montre clairement tous les résultats des analyses, la description de chaque indicateur utilisé et la Mature Transition Checklist correspondante au projet. La génération du rapport portable est identique à la Mature Transition Checklist d'OW2, avec comme seule modification le critère SQuAT.

De plus, les résultats de l'analyse ont été validés sur plusieurs projets d'OW2 et sont identiques à ceux calculés manuellement par la méthodologie QualOSS.

Pour ma part, ce stage au CETIC a été une expérience très enrichissante. Elle m'a permis de confronter mes connaissances au monde du travail, en vue de mieux comprendre la place de l'informatique dans le monde d'aujourd'hui.

Durant ce stage, tout d'abord, il a fallu bien cerner les objectifs par une analyse et une compréhension des exigences. Cette étape s'est répétée lors de l'analyse de la seconde itération du développement spécifique à l'analyse du critère SQuAT.

De plus, j'ai découvert un grand nombre d'outils et amélioré ma connaissance du langage JAVA et de son fonctionnement.

Le fait que le projet était amené à être repris par d'autres personnes m'a permis de rédiger, pour la première fois en anglais, une documentation précise sous la forme d'un wiki. Elle explique chaque partie de l'application : son installation, l'API, sa base de données et ses différents processus Bonita.

Enfin, ma compréhension de l'évaluation logicielle a été renforcée par l'analyse de plusieurs modèles évaluants la qualité, la maturité ou le taux de risque. Tous ces points de vue différents montrent la difficulté et la diversité que possède l'évaluation logicielle.

Références

[1] Atlassian. Jira - track bugs, projects, issues with software development tools. URL : http://www.atlassian.com/fr/software/jira/overview, consulté le 3/01/2012.

[2] V. R. Basili. Software modeling and measurement. the goal/question/metric paradigm. *Computer Science Technical Report Series*, 1992.

[3] BonitaSoft. Bonitasoft | open source business process management and workflow software. URL : http://www.bonitasoft.com/, consulté le 4/10/2011.

[4] J.-C. Deprez. Standard QualOSS Assessment Method - version 1.1. Décembre 2009.

[5] Fossology. FOSSology. URL : http://www.fossology.org/, consulté le 3/01/2012.

[6] OW2. OW2 Consortium. URL : http://www.ow2.org/, consulté le 22/09/2011.

[7] Qualipso. Qualipso, Trust and Quality in Open Source Systems. URL : http://www.qualipso.org/, consulté le 22/09/2011.

[8] Sonar. Sonar. URL : http://www.sonarsource.org/, consulté le 3/01/2012.

[9] The Apache Software Foundation. Apache subversion. URL : http://subversion.apache.org/, consulté le 4/10/2011.

A Liste des paramètres supportés par CHOOSE

Nom du paramètre	Description
analyzedVersionUrl	Url vers le dépôt de versions ou de l'archive contenant les sources de la version analysée
communityComment	Contient le commentaire inséré dans le rapport d'évaluation de la communauté
communityScore	Contient le score évaluant la communauté
maintainabilityComment	Contient le commentaire inséré dans le rapport d'évaluation de la maintenabilité
maintainabilityScore	Contient le score évaluant la maintenabilité
onlyOneVersion	Est vrai si le projet ne possède pas deux version majeures
projectActiveCommitters	Liste des committers actifs
projectAnalysisRoot	Chemin du répertoire où seront placé les fichiers de l'analyse
projectBinRepository	Url vers la liste des binaires
projectbug tracker	Url vers la page d'acceuil du bug tracker
projectbug trackerType	Type du bug tracker (JIRA or BugZilla)
projectBuildDocumentation	Url vers la documentation de compilation
projectCaseStudy	Liste des études cas renseignés sur la dashboard du projet
projectCheckStyleConventionUrl	Url vers la convention de code checkstyle
projectCIMUrl	Url vers le Continuous Integration Mechanism (e.g., la page web Bamboo)
projectCodeConvention	Url vers la convention de code renseignée sur la dashboard du projet
projectDataSheetLink	Url vers la datasheet renseignée sur la dashboard du projet
projectDebugRepository	Url vers le chemin du dépôt utilisé pour le débogage
projectDescription	La description du project renseignée sur la dashboard du projet
projectDocRepository	Url vers la documentation renseignée sur la dashboard du projet
projectForgeLink	Url vers la GForge (forge d'OW2) renseignée sur la dashboard du projet

Nom du paramètre	Description
projectFunction	Fonctionnalité renseignée sur la page web d'OW2 "project by function"
projectHomePageLink	Url vers la page d'acceuil du project renseignée sur la dashboard du projet
projectLastCommitDate	Date du dernier commit
projectLastMajorVersion1Date	Date de release de la moins récente des dernières versions majeures
projectLastMajorVersion1Version	Version de la moins récente des dernières versions majeures
projectLastMajorVersion1Url	Url de la moins récente des dernières versions majeures (archive ou chemin vers le dépôt)
projectLastMajorVersion2Date	Date de release de la plus récente des dernières versions majeures
projectLastMajorVersion2Version	Version de la plus récente des dernières versions majeures
projectLastMajorVersion2Url	Url de la plus récente des dernières versions majeures (archive ou chemin du dépôt)
projectLastMinorVersionDate	Date de release de la plus dernière version mineure
projectLastMinorVersionVersion	Version de la plus dernière version mineure
projectLastMinorVersionUrl	Url de la plus dernière version mineure (archive ou chemin du dépôt)
projectLicense	Liste des licenses utilisées, renseignées sur la dashboard du projet
projectLicenseLink	Liste des urls des descriptions des licenses renseignées sur la dashboard du projet
projectLogoImage	Url du logo affichée sur la dashboard du projet
projectMailingListLink	Url de la mailing list renseignée sur la dashboard du projet
projectMailSupport	Adresse mail utilisée pour le support
projectMainDevRepository	Url vers le dépôt utilisé comme point de développement principal (habituellement, `trunk`)
projectMavenName	Nom du projet sur Maven

Nom du paramètre	Description
projectOhlohUrl	Url de la page d'accueil du project sur la page d'Ohloh
projectProfessionalSupport	Url vers le site web fournissant le support
projectRepository	Url vers le dépôt utilisé actuellement
projectRepositoryLink	Url vers le dépôt renseignée sur la dashboard du projet
projectRepositoryPolicy	Url vers la police de gestion du dépôt du projet renseignée sur la page d'accueil du projet
projectRepositoryType	Type du dépôt (SVN, GIT or CVS)
projectRevisionIdentifier	Identificateur de la révision (un nombre pour SVN ou un hash pour GIT)
projectRoadMapLink	Url vers la roadmap du projet
projectShortDescription	La description courte renseignée sur la page "project by function" d'OW2
projectStandard	Liste des standards utilisés par le projet
projectStatus	Status du projet (incubator, mature ou archive)
projectTagsRepository	Url vers le dépôt utilisé pour le versionning (habituellement, `tags`)
projectUsageDescription	La description de l'usage du projet
projectUsageLink	Url vers l'usage du projet
releaseDate	La date de release de la version analysée
reliabilityComment	Contient le commentaire inséré dans le rapport d'évaluation de la fiabilité
reliabilityScore	Contient le score évaluant la fiabilité

B Schéma de la base de données

Un *schéma de base de données relationnelle* a été développé spécifiquement pour la plate-forme CHOOSE. Il est issu du schéma définit pour la méthodologie QualOSS afin de stocker les métriques et les artefacts.

Notre schéma permet le stockage des informations propres aux analyses (nom, version, date de parution), les artefacts, les métriques, les mesures et enfin, des informations décrivant le projet (url de la page d'accueil, email de support, lien vers la forge, image du projet, etc).

La principale difficulté dans la définition de ce schéma est qu'il devait être capable de stocker des données d'autres écosystèmes.

C'est pourquoi les informations décrivant un projet ont été nommées des "paramètres". De plus, le nombre de paramètres doit aussi pouvoir évoluer : certains écosystèmes décrivent plus en détails certains aspects de leurs projets que d'autres.

Il y a également une flexibilité accrue au niveau des types d'artefacts et des métriques. Donc, aussi au point de vue des artefacts et des mesures qui sont considérées ou qui pourraient l'être dans les évolutions futures du logiciel.

La figure 7 (page suivante) présente le schéma utilisé.

Dans cette figure, vous pouvez remarquer la présence d'un éclatement nommé `PARAMETERVALUE` entre la table des analyses (`ANALYSIS`) et celles des paramètres (`PARAMETER`). Il permet d'ajouter des valeurs à des paramètres et d'ajouter des paramètres facilement sans modifier le schéma de la base de données.

Les tables des types d'artefacts (`ARTIFACT_TYPE`) et des métriques (`METRIC`) sont assez libres : il est possible d'ajouter des types d'artefacts et des métriques sans modifier le schéma, sous la seule contrainte que le type d'artefacts qu'elles traitent existe.

Le schéma considère qu'une métrique ne peut s'appliquer qu'à un seul type d'artefact. La table contenant les artefacts (`ARTIFACT`) les définit comme étant une "instance" d'un type d'artefact pour une analyse. Le schéma accepte plusieurs artefacts du même type.

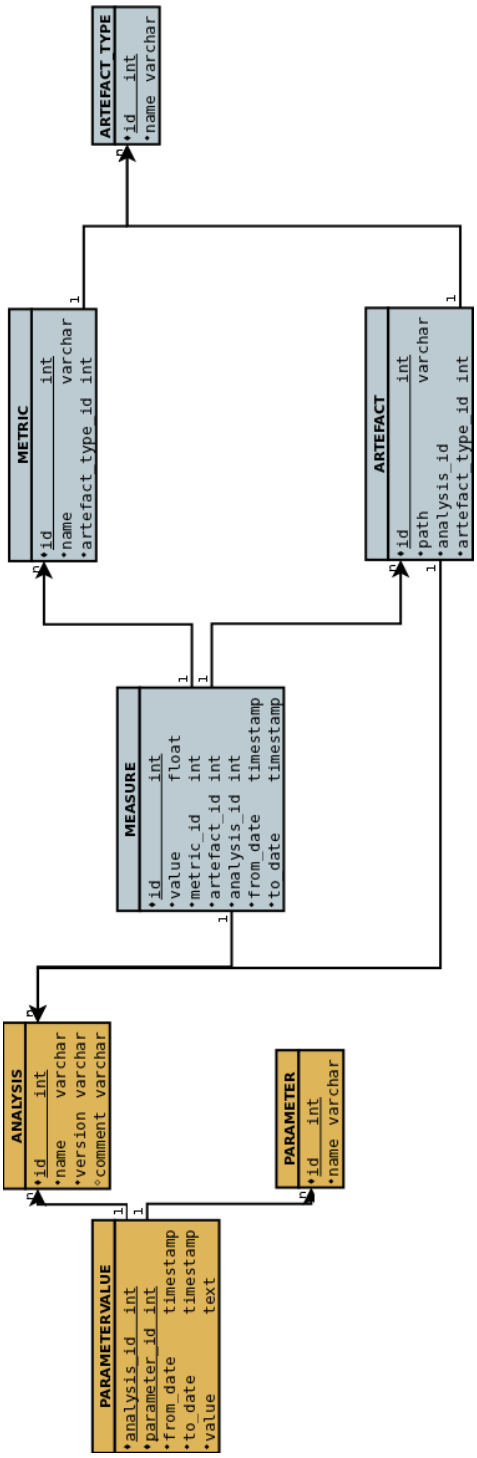

FIGURE 7 – Schéma de la base de données

La table des mesures (`MEASURE`) définit qu'une mesure ne peut appartenir qu'à un artefact et qu'une métrique. Ce qui est logique car une mesure résulte de l'application d'une métrique sur un unique artefact [8].

Enfin, les attributs `from_date` et `to_date` présents sont très utiles. Ils sont utilisés par les mesures et les paramètres. Ce sont des attributs temporels décrivant un historique des valeurs qu'à pris la mesure ou le paramètre correspondant.

Ils ne sont jamais spécifiés dans les requêtes d'insertion, mais uniquement utilisés lors de la sélection de données. Pour modifier ces attributs, des triggers (un par table contenant ce couple d'attributs) permettent de gérer leur valeur. Ces triggers sont appliqués lors de l'insertion d'un nouvel enregistrement dans l'une des tables. Soit now la date et l'heure de l'insertion de l'enregistrement new, ils vont :

1. rechercher l'enregistrement old dont le `from_date` $\leq now$ et dont le `to_date` $> now$;
2. modifier le `to_date` de old en now ;
3. initialiser le `from_date` de new en now ;
4. initialiser le `to_date` de new en une date lointaine [9] ;
5. insérer new.

L'interfaçage entre la base de données et l'application

L'*interfaçage entre la base de données et l'application* CHOOSE consiste en une API (Application Public Interface) développée spécifiquement pour le projet. Elle permet de gérer les comportements des données avant leur insertion dans la base de données, ainsi que de certaines automatisations.

Cette section présente les fonctionnalités de cette API ainsi que les motivations ayants conduites à son développement.

Motivations

Il y deux motivations ayants conduites au développement de cette API.

8. Il existe cependant des métriques dites agrégées fonctions de plusieurs métriques pouvant venir d'artefacts différents (ex. : le nombre de bugs par fichiers)
9. Dernière date possible pour un timestamp

Premièrement, la division entre le système de gestion utilisé et l'application permet la modification du système de gestion de base de données en n'ayant à modifier que l'API.

Deuxièmement, il était nécessaire de sauvegarder atomiquement les données lors de la création d'une nouvelle analyse, de la récolte des artefacts et du calcul des métriques. Cela évitait d'enregistrer partiellement les résultats d'une étape. Pour ce faire, si au moins une erreur se passait lors de l'une de ces étapes, toutes les informations propres à cette étape devaient être supprimées.

Une autre solution consistait à effectuer un "RollBack [10]" en base de données en cas de problème lors d'une insertion, mais dans Bonita l'utilisation des connecteurs permettant d'effectuer des requêtes SQL sur des base de données crée des transactions différentes pour chaque utilisation du connecteur. Le plus simple aurait été de créer un connecteur permettant de faire la sauvegarde atomique d'une suite de requêtes SQL.

Cependant, nous avons préféré le développement d'une API spécifique. Elle permet de diminuer le nombre de connecteurs Bonita et d'uniformiser l'utilisation de ses fonctionnalités dans toute l'application.

Fonctionnalités

Les fonctionnalités de cette API sont :

- la sauvegarde atomique de nouvelles analyses, des résultats de la récolte des artefacts et de ceux du calculs des métriques ;
- l'accès facile aux paramètres, artefacts et mesures ;
- l'insertion automatique des métriques et des types d'artefact.

10. le rollback est une fonction permettant d'annuler la transaction en cours

C Résultats des évaluations

Cette annexe montre l'affichage dans CHOOSE des résultats obtenus par l'évaluation des quatre projets suivants :

- *Chameleon*
- *Spago*
- *Spago4Q*
- *Orchestra*

Chameleon

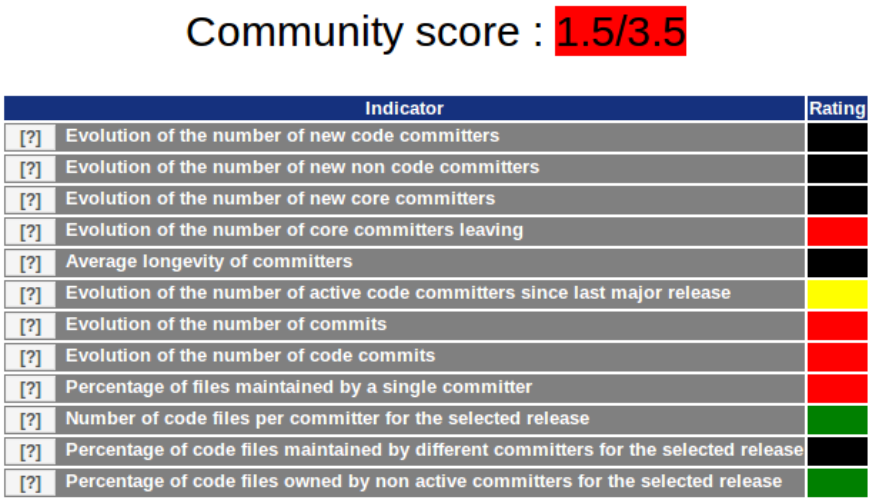

FIGURE 8 – Évaluation de la communauté de Chameleon

Maintainability score : 2.21/3.5

Indicator		Rating
[?]	Percentage of accepted enhancement proposals	
[?]	Rapidity of implementation of enhancement proposals	
[?]	Evolution of change in code between major releases	
[?]	Evolution of change to public interfaces between major releases	
[?]	Evolution of number of lines of code between successive releases	
[?]	Percentage of unassigned issues	
[?]	Rapidity of issue resolution	
[?]	Evolution of change in code between minor releases	
[?]	Evolution of change to public interfaces between minor releases	
[?]	Average efferent coupling of high level modules	
[?]	Average efferent coupling of low level modules	
[?]	Average cyclomatic complexity per defined routine	
[?]	Percentage of commented algorithm	
[?]	Evolution of cyclomatic complexity of defined routines between successive releases	

FIGURE 9 – Évaluation de la maintenabilité de Chameleon

Reliability score : 2.28/3.5

Indicator		Rating
[?]	Average number of bugs in stable releases	
[?]	Slope number of bugs in stable releases	
[?]	Number of bugs by number of stable releases	
[?]	Slope number of bugs by stable releases lifetime	
[?]	Average number of bugs in stable releases specific	
[?]	Slope number of bugs in stable releases specific	
[?]	Average number of new and modified files per day between minor releases	
[?]	Average percentage of modified files between minor releases	
[?]	Average number of coding convention violation	

FIGURE 10 – Évaluation de la fiabilité de Chameleon

Spago

Community score : 1.92/3.5

	Indicator	Rating
[?]	Evolution of the number of new code committers	
[?]	Evolution of the number of new non code committers	
[?]	Evolution of the number of new core committers	
[?]	Evolution of the number of core committers leaving	
[?]	Average longevity of committers	
[?]	Evolution of the number of active code committers since last major release	
[?]	Evolution of the number of commits	
[?]	Evolution of the number of code commits	
[?]	Percentage of files maintained by a single committer	
[?]	Number of code files per committer for the selected release	
[?]	Percentage of code files maintained by different committers for the selected release	
[?]	Percentage of code files owned by non active committers for the selected release	

FIGURE 11 – Évaluation de la communauté de Spago

Maintainability score : 1.92/3.5

	Indicator	Rating
[?]	Percentage of accepted enhancement proposals	
[?]	Rapidity of implementation of enhancement proposals	
[?]	Evolution of change in code between major releases	Can't be computed
[?]	Evolution of change to public interfaces between major releases	Can't be computed
[?]	Evolution of number of lines of code between successive releases	
[?]	Percentage of unassigned issues	
[?]	Rapidity of issue resolution	
[?]	Evolution of change in code between minor releases	
[?]	Evolution of change to public interfaces between minor releases	
[?]	Average efferent coupling of high level modules	
[?]	Average efferent coupling of low level modules	
[?]	Average cyclomatic complexity per defined routine	
[?]	Percentage of commented algorithm	
[?]	Evolution of cyclomatic complexity of defined routines between successive releases	

FIGURE 12 – Évaluation de la maintenabilité de Spago

Reliability score : 2.79/3.5

	Indicator	Rating
[?]	Average number of bugs in stable releases	
[?]	Slope number of bugs in stable releases	Can't be computed
[?]	Number of bugs by number of stable releases	
[?]	Slope number of bugs by stable releases lifetime	Can't be computed
[?]	Average number of bugs in stable releases specific	
[?]	Slope number of bugs in stable releases specific	
[?]	Average number of new and modified files per day between minor releases	
[?]	Average percentage of modified files between minor releases	
[?]	Average number of coding convention violation	

FIGURE 13 – Évaluation de la fiabilité de Spago

Spago4Q

Community score : 2.42/3.5

Indicator		Rating
[?]	Evolution of the number of new code committers	
[?]	Evolution of the number of new non code committers	
[?]	Evolution of the number of new core committers	
[?]	Evolution of the number of core committers leaving	
[?]	Average longevity of committers	
[?]	Evolution of the number of active code committers since last major release	
[?]	Evolution of the number of commits	
[?]	Evolution of the number of code commits	
[?]	Percentage of files maintained by a single committer	
[?]	Number of code files per committer for the selected release	
[?]	Percentage of code files maintained by different committers for the selected release	
[?]	Percentage of code files owned by non active committers for the selected release	

FIGURE 14 – Évaluation de la communauté de Spago4Q

Maintainability score : 2.12/3.5

Indicator		Rating
[?]	Percentage of accepted enhancement proposals	Can't be computed
[?]	Rapidity of implementation of enhancement proposals	Can't be computed
[?]	Evolution of change in code between major releases	Can't be computed
[?]	Evolution of change to public interfaces between major releases	Can't be computed
[?]	Evolution of number of lines of code between successive releases	
[?]	Percentage of unassigned issues	Can't be computed
[?]	Rapidity of issue resolution	Can't be computed
[?]	Evolution of change in code between minor releases	
[?]	Evolution of change to public interfaces between minor releases	
[?]	Average efferent coupling of high level modules	
[?]	Average efferent coupling of low level modules	
[?]	Average cyclomatic complexity per defined routine	
[?]	Percentage of commented algorithm	
[?]	Evolution of cyclomatic complexity of defined routines between successive releases	

FIGURE 15 – Évaluation de la maintenabilité de Spago4Q

Reliability score : 2.83/3.5

Indicator		Rating
[?]	Average number of bugs in stable releases	Can't be computed
[?]	Slope number of bugs in stable releases	Can't be computed
[?]	Number of bugs by number of stable releases	Can't be computed
[?]	Slope number of bugs by stable releases lifetime	Can't be computed
[?]	Average number of bugs in stable releases specific	Can't be computed
[?]	Slope number of bugs in stable releases specific	Can't be computed
[?]	Average number of new and modified files per day between minor releases	
[?]	Average percentage of modified files between minor releases	
[?]	Average number of coding convention violation	

FIGURE 16 – Évaluation de la fiabilité de Spago4Q

Orchestra

Les résultats de l'évaluation de la communauté d'Orchestra ne sont pas affichés, car son analyse n'a pu être effectuée.

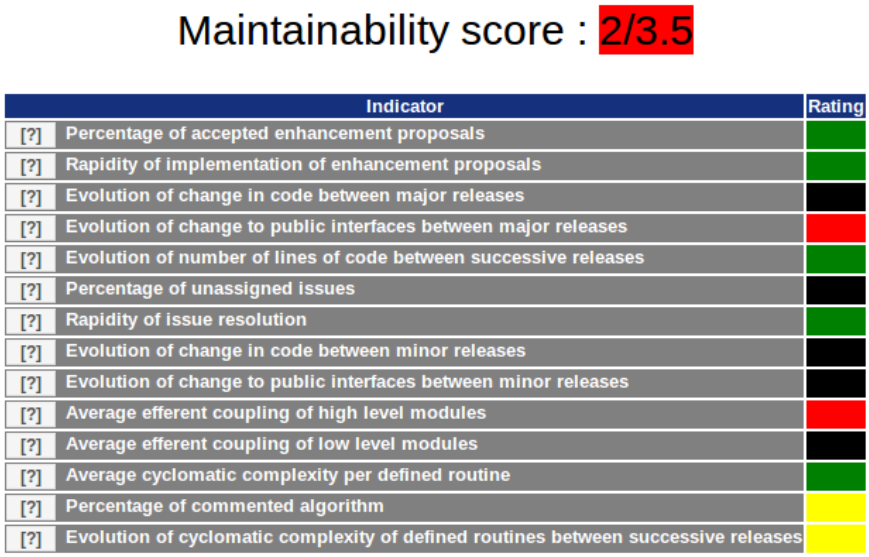

FIGURE 17 – Évaluation de la maintenabilité d'Orchestra

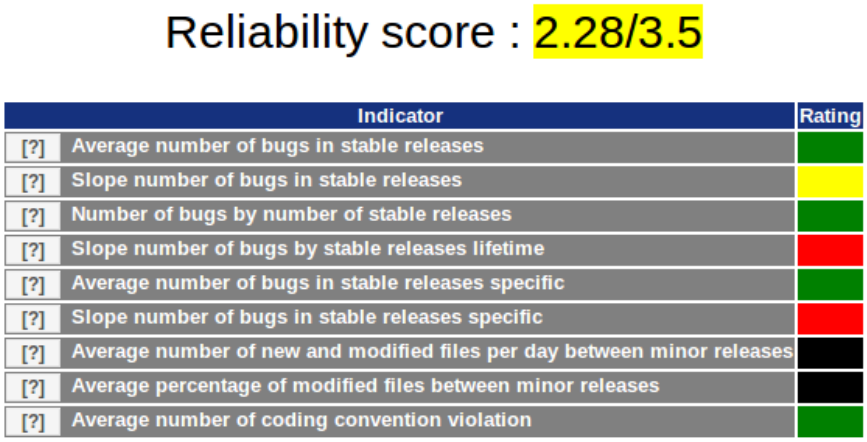

FIGURE 18 – Évaluation de la fiabilité d'Orchestra

D Mature Transition Checklist générée pour le projet Chameleon

Les pages suivantes présentent la Mature Transition Checklist générée par la plate-forme CHOOSE pour le projet d'OW2 *Chameleon*.

Certaines informations n'ont pas pu être trouvées par manque de connaissance du projet. Elles ont été remplacées par des informations arbitraires (ex. : "This is a description of the project CIM" pour la description demandée par le critère 2, verb|Build|). Mais, leurs découvertes ne poseront aucun problème à un membre actif de la communauté, demandant une analyse de son projet.

Dans le cas de l'usage du projet, pour l'information "Proof of usage of the project", nous avons testé le comportement des champs Jasper Report face à une description énorme. Cependant, Jasper Report tronque le contenu pour le faire correspondre à la taille du contenant. Ce problème est connu et est listé dans les travaux complémentaires (*c.f.* section 7).

Mature Transition Checklist

An OW2 project wishing to migrate to the Mature status should fill in this checklist and send it to: tc AT ow2 DOT org.

Technical Criteria

1. Source Code + Documentation + Binaries

The source code of the project must be available on the OW2 infrastructure.
 * Either the OW2 forge is used as the main development point
 * Or the project maintainers must publish on a regular basis their code on OW2
        o This have to be an automatic script
        o Regular basis meaning "Once per release"

The same rule applies for documentation and binaries

URL for source code on the OW2 infrastructure	svn://svn.forge.objectweb.org/svnroot/chameleon
URL for documentation on the OW2 infrastructure	http://wiki.chameleon.ow2.org/xwiki/bin/view/Doc/
URL for binaries on the OW2 infrastructure	http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Downloads

2. Build

The project code-base must build.
 * Build means here compile + tests OK

A continuous integration mechanism must be used to ensure the build is a success.
The way to build the system has to be documented on the web site
 * Like a cookbook explaining required environment and software + step by step build process

Projects could use the OW2 in-house Continuous Integration System (http://bamboo.ow2.org) or provides a link on their own CI system elsewhere.

URL to the documentation about building the project	http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Building
Description of the continuous integration mechanism used by the project	This is a description of the project CIM Url :http://bamboo.ow2.org/browse/CHAMELEON

3. Project's Usage

The projects must show that it is used within the OW2 universe or outside.
Possible Control Points:
* Code sharing with maven repository
* Gives references of client system (using the project)
* Gives references of business users

Proof of usage of the project	This is a description of project usage and a test to long description Experience has shown that as software is fixed, emergence of new and/or Maven name : org.ow2.Chameleon

4. Source Repository Management

The project must have means of separating active development from versioning and bug-fixing in the source repository.
It must be documented on the web site.
Example
* Active development goes into a trunk and branches are used for stabilization.

URL to the repository management policy	http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/ReleaseProcess

5. Code Conventions

A mature project must follow a code convention guideline.
It must be documented on the web site.
This guideline should be enforced by using tools such as checkstyle.
It is not the aim of OW2 to provides a strict code guideline, but OW2 may define in the future a set of conventions that could be used by default by projects.

URL to the code convention	http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/Conventions

6. SQuAT

SQuAT (Software Quality Assurance Trustworthiness) is OW2's software quality program, it provides a set of tools dedicated to software quality.
Using the SQuAT tools, it will be easier to ensure of the quality of a project (IP/Licence management, code style, ...).
A mature project has to publish reports concerning the following aspects:
 * IP/Licence control (Proposed tool: FOSSology)
 * Quality measurements (Proposed tool: Sonar)
 * Trustworthiness and maturity indicators : we propose the QualOSS assessement method.

The following evaluation were performed by the QualOSS assessment method. (http://qualoss.org/)
Each criterion evaluation has a score between 0 and 3.5. The mean is as following :
 * 0 : not yet computed
 * 0.5 : high risk
 * 1.5 : medium risk
 * 2.5 : small risk
 * 3.5 : negligeable risk
The community evaluation is compute on version repository logs, the maintainabilty and reliability ones have been perfomed on the bugs list and sources.

Community evaluation	1.5/3.5
Maintainability evaluation	2.21/3.5
Reliability evaluation	2.28/3.5

Community Criteria

7. Committers The project must have more than 1 active committer.	
Number of committers / URL to the list of committers	Number of active committers : 9 Committers : tony3107,mehdi-d,sauthieg, bourretp,torito,abendt,debbabi,barjo,

8. Dashboard

The project's dashboard must be up to date.

Minimal dashboard's content

Compulsory	
* Project Name	yes
* Project description	yes
* Short project description	yes
* Status: Mature/Incubator/Archive	yes
* Function: (in the list of predefined functions)	yes
* License	yes
* Forge Link	yes
* Mailing List link	yes
* Repository type (SVN, CVS, etc)	yes
* Repository Link	yes
* Project Home Page Link	yes
* Case study	yes
Recommended (not compulsory)	
* Logo Image	yes
* License Link	yes
* Project DataSheet link (the link must be to a PDF file)	no
* Professional support link	yes
* Standard implemented	no
* Project roadmap (link)	no
Proposed	
* Community information (list size, traffic, #committers, ...)	no

Url to the project dashboard	http://www.ow2.org/xwiki/bin/view/ActivitiesDashboard/Chameleon

9. Active

The project must show that it is in activity:

* Exchanges (question + answer) on the mailing lists
    - o Community support
    - o The number of commits on the last 6 months
* Last commits
* The number of mail / month (on the last 6 months).
* Someone must provide support for the project (maybe not commercial support)

Proof of activity: number of mails per months in the past 6 months on the project mailing lists	7.33 mails/month
Proof of activity: number of commits in the past 6 months and date of last commit	Number of commits : 177 Last commits performed on : Mon Nov 28 08:37:09 CET 2011
Proof of activity: support provided (mailing list / other entities)	Email :chameleon@ow2.org Website :http://wiki.chameleon.ow2.org/xwiki/bin/view/Main/MailingLists