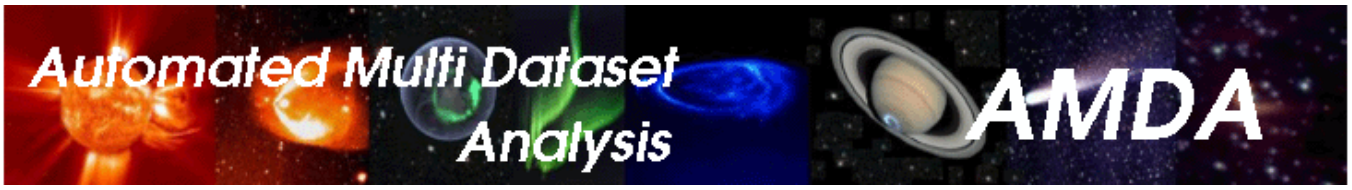
Agence ou Service : NTIC

Projet : AMDA-Kernel



NOYAU AMDA-NG - PLAN DE TEST

Rédigé par : *Chef de projet AKKA* : *Freddy CASIMIR*	Diffusé à : CNES/ IRAP
Approuvé par : Chef de projet CNES : DUFOURG Nicolas	

LISTE DES MODIFICATIONS DU DOCUMENT

Vers.	Date	Paragraphes modifiés	Description des modifications
01.00	23/11/2012	tous	Création du document
01.01	05/02/2013		Modification suite au Sprint 1 de la Release 3
01.02	11/02/2013	P11	Ajout schémas infrastructure AKKA

SOMMAIRE

1 INTRODUCTION 5

1.1 Objet du document 5

1.2 Domaine d'application 5

2 STRATEGIE DE TESTS 6

2.1 Objectifs/Limites 6

2.2 Principes généraux 6

2.3 Eléments concernés 7

2.4 Responsabilites 7

2.5 Documents utilisés 7

2.6 Règles d'identification 8

2.6.1 Identification des tests 8

2.6.2 Identification des résultats de tests 8

2.7 Environnement de tests 11

2.7.1 Configuration matérielle 11

2.7.2 Outils 12

2.8 Critères d'arrêt des tests 13

2.9 Planning et charges 13

2.10 Déroulement de l'activité de tests 14

2.11 Gestion des anomalies 14

3 TEST UNITAIRE 15

3.1 Objectifs/Limites 15

3.2 Documents de référence 15

3.3 Eléments concernés 15

3.4 Jeux de données 15

3.5 Méthodologie 15

4 TEST D'INTEGRATION 16

4.1 Objectifs/Limites 16

4.2 Documents de référence 16

4.3 Eléments concernés 16

4.4 Jeux de données 16

4.5 Méthodologie 16

4.6 Scénarios de tests 16

4.7 Résultats de tests 16

5 TESTS DE VALIDATION 17

5.1 Objectifs/Limites 17

5.2 Documents de référence 17

5.3 Eléments concernés 17

5.4 Jeux de données 17

5.5 Méthodologie 17

5.6 Scénarios de tests 17

5.7 Résultats de tests 18

6 TESTS DE NON-REGRESSION 19

6.1 Methodologie de tests de non-régression 19

6.1 Scénarios de tests 19

6.2 Résultats de tests 19

7 MATRICE DE COUVERTURE DES TESTS 20

8 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R) 21

9 GLOSSAIRE ET ABREVIATIONS 22

9.1 Glossaire 22

9.2 Abréviations 23

1 INTRODUCTION

1.1 OBJET DU DOCUMENT

L'objectif de l'activité de tests est de vérifier et démontrer la conformité du logiciel aux spécifications validées.

Le présent document décrit l'activité de tests mise en œuvre par l'équipe projet AKKA Technologies dans le cadre du projet AMDA-Kernel.

Il est constitué par le plan de tests, qui décrit la stratégie de tests, le déroulement global de l'activité de tests, ainsi que son suivi.

Le document est complété au fur et à mesure de l'avancement du projet.

Dans le cadre de ce projet développé en mode agile (SCRUM), nous entendons par spécifications validées, les Users Stories (USs) prises en comptes et leurs Tests d'Acceptations (TAs). Les USs et les TAs sont de la responsabilité du Product Owner.

1.2 DOMAINE D'APPLICATION

Ce document couvre la stratégie de test appliquée aux programmes en ligne de commande :

- EXE_AMDA_Kernel,
- amdaXMLRequestTool,
- amdaParameterGenerator.

Ces programmes sont tous trois décrits en détail dans les documents [R1] et [R3].

Dans notre contexte les programmes listés ci-dessus:

- sont basés sur une même librairie
- sont diffèrents dans la manière dont on leurs passe les arguments :
  - pour le 1er, les arguments sont passés « à plat » sur la ligne de commande
  - pour les deux autres, les arguments sont passés au travers d'un fichier XML
- doivent être connectés au serveur DDServer(machine : manunja.cesr.fr, port : 5000) pour fonctionner. Celui-ci fournit les données que nos programmes doivent traiter.

Le comportement de ces exécutables est entièrement spécifié sous forme de USs et TAsdans l'outil WEB IceScrum (Outil de gestion de projet en mode Scrumhttp://yahscrum.akka.eu:8080/icescrum/).

Seul les USs et TAs déjà réalisés ou en cours de réalisation peuvent être considérés comme valides et à tester.

Exemple, si des tests de performance sont à faire, ils doivent être écris sous forme de US et de TA,de même pour les tests de robustesse. De manière générale, tout test répond à au moins une US.

Ce document décrit :

- Les tests unitaires
- Les tests d'intégration
- Les tests de validation
- Les tests de non-régression

2 STRATEGIE DE TESTS

2.1 OBJECTIFS/LIMITES

La stratégie des tests a pour objectif de définir le cadre de l'activité des tests en termes de principes généraux, de définition du contenu des tests, d'enchaînement des étapes et de responsabilités.

2.2 PRINCIPES GENERAUX

Quatre types de tests sont effectués sur le projet :

- Les tests unitaires: ce sont les tests effectuéspar le développeur en cours de sprint sur les US qu'il développe. Ces tests sont effectués dans l'espace de développement.
- Les tests d'intégration : ce sont les tests effectués en cours de sprint sur les US courant du sprint dans l'espace d'intégration. Ils comprennent de plus des tests de qualimétrie (vérification de la bonne application des règlede codage, comme le taux de commentaire, la non duplication de code, la bonne gestion de la mémoire, …)
- Les tests de validation : ce sont les tests effectués en fin de sprint sur les US définis dans le sprint.
- Les tests de non-régression : ce sont les tests effectués tous au long sur les US validées dans les sprintsprécédant.

Dans le cadre de ce projet, nous nous appuyons totalement sur les tests d'acceptation définis par le « ProductOwner».Ces tests d'acceptation peuvent être fonctionnels, structurels, de performance, de robustesse etc.

Les mêmes scénarii de tests sont utilisés pour les tests unitaires, d'intégration, de validation et de non régression.

Nous suivons quelques principes de la méthode «BehaviourDevelopmentDriven»BDD:

- La « User Story » et ses tests d'acceptation forment une spécification qui est travaillée par l'équipe SCRUM afin d'obtenir un langage naturel commun entre les développeurs etle métier.
- Un développeur, travaillant sur sa plateforme de développement, choisit une US et commence par automatiser les tests d'acceptation qui y sont associés.
- Le développeur les joue pour confirmer leurs échecs.
- Le développeur effectue un premier développement jusqu'à faire passer avec succès ces tests et ceux mis en place pour les précédentes « User Stories ».
- Cette première version est mise en gestion de configuration.
- Le développeur re-factorise enfin le code pour en obtenir une version plus homogène, sans doublon de code et avec un niveau de qualité de code acceptable (Taux de commentaire, pas de fuite et/ou d'écrasement mémoire).
- Cette nouvelle version est mise en gestion de configuration.
- Le Product Ownervalide que la « User Story » répond bien aux tests d'acception définis lors de la démonstration de fin de Sprint et qu'elle n'a pas d'impact sur les autres US (ou alors de manière acceptable, voir normale).
- Le développeurdoit rejouer avec succès tous les tests automatisés sur la plateforme d'exploitation à chaque livraison de release.

2.3 ELEMENTS CONCERNES

Chaque fonctionnalité décrite par une User Story est testée.

La liste des fonctionnalités se trouve dans l'outil de gestion de projet Scrum:

- IceScrum (http://yahscrum.akka.eu:8080/icescrum/)

Les groupes de fonctionnalités identifiés par le Product Owner se retrouve sous forme de « features»Scrum:

- Couverture fonctionnelle de l'existant
- Fonctionnalités nouvelles
- Evolutivité & Extensibilité
- Performances
- Maintenabilité

2.4 RESPONSABILITES

Le **Product Owner** du projet est responsable de la définition et de la validité des tests d'acceptation des US, il les saisit sous IceScrum.

Comme les exécutables sont testés en ligne de commande, la majorité des tests sont de la forme suivante,inspirée du principe « Given Then When»:

- **Arguments Z en entrée du programme X donnera les résultats Y.**

Le Product Owner fournit les entrées (arguments, fichiers d'entrée) et les sorties (fichiers) au travers de IceScrum et FTP.

Le jeu de donnée DDServer est également de sa responsabilité.

Par exemple, les résultats attendus peuvent être :

- Le code retour du programme,
- Des phrases à trouver dans le fichier de journalisation,
- Les fichiers résultats du traitement,
- …

L'équipe de développement SCRUM AKKA est responsable de l'automatisation de ces tests et du passage avec succès de ces tests.

Une « User Story » ne peut être présentée que lorsque ses tests d'acceptation passent ainsi que tous les tests des Users Stories précédemment déclarées terminées.

Le Product Owner est l'arbitre des tests que l'équipe présente comme obsolètes. En effet, une US peut voir sa fonctionnalité surchargée, contredite ou … par une nouvelle.

2.5 DOCUMENTS UTILISES

Les scenarii des tests sont des pages FitNesse.Ces pages sont gérées en configuration sous svn(http://svn-06.akka.eu/svn/AMDA-KERNEL/) dans le même projet que les sources : dans le répertoire test/FitNesseRoot.

Les rapports des tests sont généréset gérésdans Jenkins, Sonar et FitNesse

Voir le §2.7.2 pour plus d'information sur ces outils.

2.6 REGLES D'IDENTIFICATION

2.6.1 Identification des tests

Les tests d'acceptation sont identifiés via les numéros fournis par IceScrum :

- Numéro de la US
- Numéro du test d'acceptation

Exemple : US 32: Id 31

Sous FitNesse une page de test estécrite par User Story, cette page contientles tests d'acceptation concernant laUS.

Cette page est rangée dans une arborescence ReleaseX/SprintY(voir le répertoire d'une livraison test/FitNesseRoot/ReleaseS/)

Dans la page nous répétons l'objet de la « User Story » au début avec un lien vers IceScrum.

Avant chaque scénario de test l'objet de celui-ci est repris de IceScrum.

ReleaseS. ReLease1. SprinT4.

UserStory46 [add child]

Test
Edit
Properties
Refactor
Where Used
Search
Files
User Guide
Test History

USER STORY 46: TRAITEMENT DE GRANDE INTERVALLE DU TEMPS

http://yahscrum.akka.eu:8080/icescrum/p/AMDAKERNEL-46

DESCRIPTION DES TESTS

Construct and output values of parameter for large time interval when several calls to DDServer are needed. Size of 'time block' is to be defined in the system.

ID-81 OUTPUT PARAMETER VALUES FOR LARGE TIME INTERVAL

Input: paramID : 'imf'
StartTime: '20080020000000000\0'
TimeInt: '00000050000000000\0'
Output: ascii file and log of access to DD Server

TESTS :

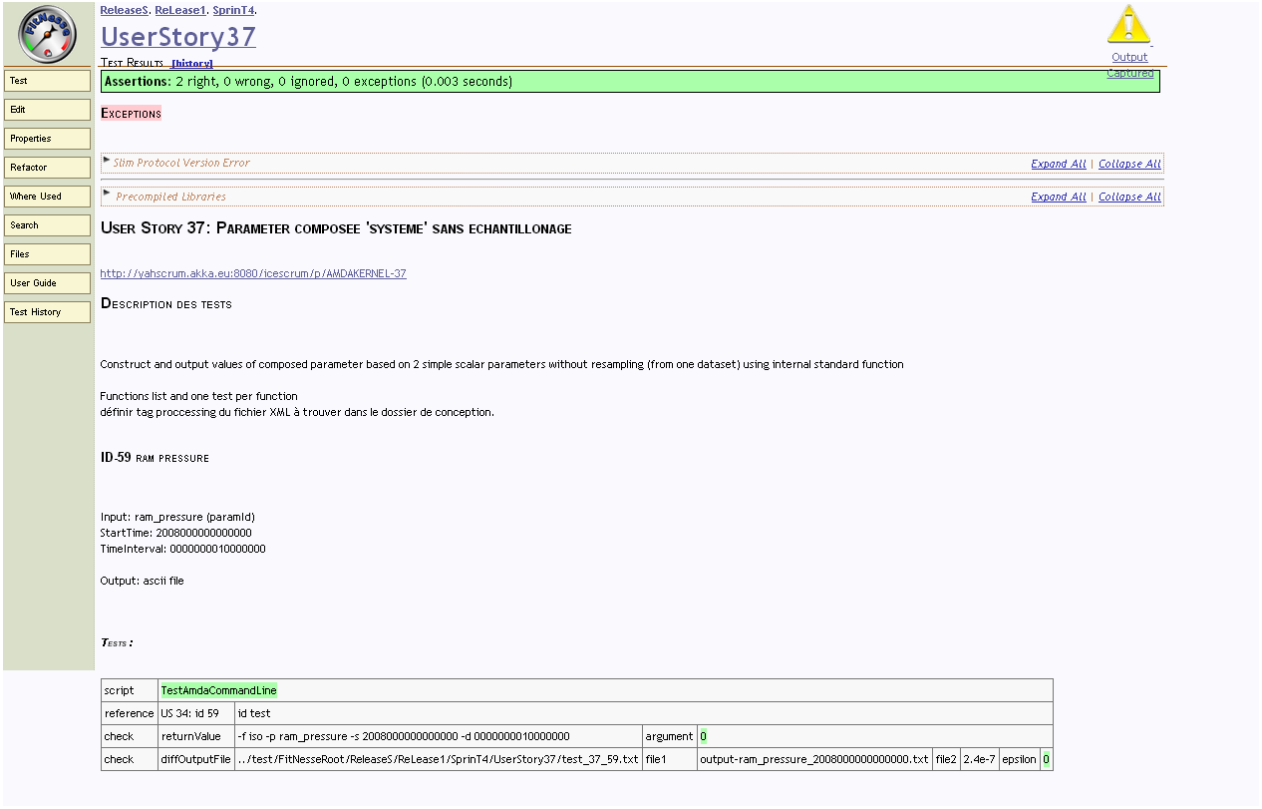
script	TestAmdaCommandLine						
reference	US 46: id 81	id test					
check	returnValue	-f iso -p imf -s 2008002000000000 -d 0000005000000000	argument	0			
check	diffOutputFile	../test/FitNesseRoot/ReleaseS/ReLease1/SprinT4/UserStory46/test_46_81.txt	file1	output-imf_2008002000000000.txt	file2	0	
check	logFile	AMDA-Kernel.ParamOutput	befor	INFO 0x[0-9a-f]* \([0-9]+ ms\)[\t]+AMDA-Kernel.ParamGet - ParamGetDDBase::getOneDDDataBloc DD_GetData\([T:, 0-9<>=:.-]*\) returns = \(1\)	pattern	0	

2.6.2 Identification des résultats de tests

Nous utilisons le langage de FitNesse pour la lecture des résultats de tests.

Les résultats des tests sont classésdans les catégories de conformité suivantes :

- **« right » :** résultat conforme aux exigences
- **« wrong »** : résultat non conforme aux exigences
- **« exceptions » :** test non desservie par le serveur de test



23/11/12 Suite Results: ReleaseS

ReleaseS

Suite Results (history)

Output (preserved)

Test Pages: 11 right, 6 wrong, 0 ignored, 0 exceptions Assertions: 64 right, 22 wrong, 0 ignored, 4 exceptions (50.783 seconds)

Test Summaries

slim:build/bin/CSlimTestServer

2 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT1.UerStory31 (0.004 seconds)
15 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT2.UerStory32 (4.932 seconds)
4 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT2.UerStory34 (3.354 seconds)
13 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT2.UerStory36 (0.005 seconds)
4 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT3.UserStory04 (1.712 seconds)
2 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT3.UserStory12 (0.003 seconds)
13 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT3.UserStory35 (8.106 seconds)
3 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT3.UserStory49 (2.801 seconds)
2 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT3.UserStory54 (0.005 seconds)
2 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT4.UserStory37 (0.005 seconds)
3 right, 0 wrong, 0 ignored, 0 exceptions ReLease1.SprinT4.UserStory46 (0.005 seconds)
0 right, 4 wrong, 0 ignored, 0 exceptions ReLease2.SprinT1.UserStory16 (0.056 seconds)
1 right, 1 wrong, 0 ignored, 0 exceptions ReLease2.SprinT1.UserStory17 (0.002 seconds)
0 right, 5 wrong, 0 ignored, 2 exceptions ReLease2.SprinT1.UserStory29 (0.094 seconds)
0 right, 6 wrong, 0 ignored, 0 exceptions ReLease2.SprinT1.UserStory40 (0.053 seconds)
0 right, 3 wrong, 0 ignored, 0 exceptions ReLease2.SprinT1.UserStory47 (0.002 seconds)
0 right, 3 wrong, 0 ignored, 2 exceptions ReLease2.SprinT1.UserStory59 (0.000 seconds)

Test Output

Test System: slim:build/bin/CSlimTestServer

ReLease1.SprinT1.UerStory31 Top

Exceptions

Slim Protocol Version Error Expand All | Collapse All

Precompiled Libraries Expand All | Collapse All

User Story 31: Connexion à DD_Server

Description du test

Le but du test est de vérfier que le code SimpleTest de DD_client fonctionne correctement.
Dans un premeir temps, nous exeoutons DDLogin qui ouvre une connection aveo DD_serveur
Puis nous testons l'existance de la variable xxx aveo SimpleTest
Et enfin nous vérifions le retour (stdout) de la commande aveo un diff par rapport a un doounent de ref

Test

script	ConnectToDDServer	
check	login	1
check	simpleTest	0

Contents:

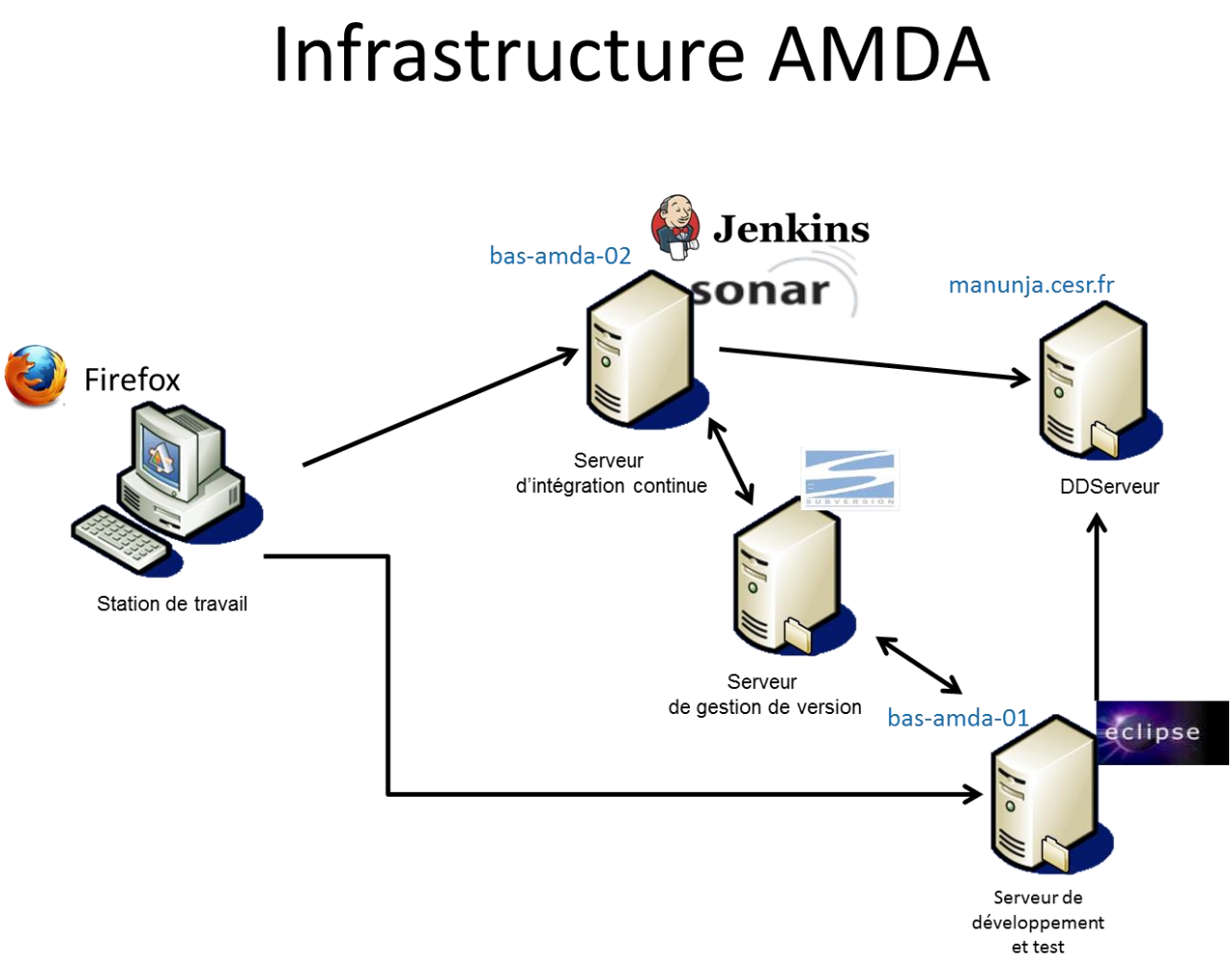
ReLease1.SprinT2.UerStory32 Top

Exceptions

bas-amda-01:8081/ReleaseS?suite 1/19

2.7 ENVIRONNEMENT DE TESTS

2.7.1 Configuration matérielle



Matériel	Type de machine	Système d'exploitation	Configuration
bas-amda-01 bas-amda-02	PC processeurs pentium ou équivalents	Centos 6.3	

2.7.2 Outils

2.7.2.1 FitNesse

FitNesse est un outil de développement collaboratif sous forme de WIKI axé sur les tests d'acceptation.

Dans cet outil, l'équipe Scrum:

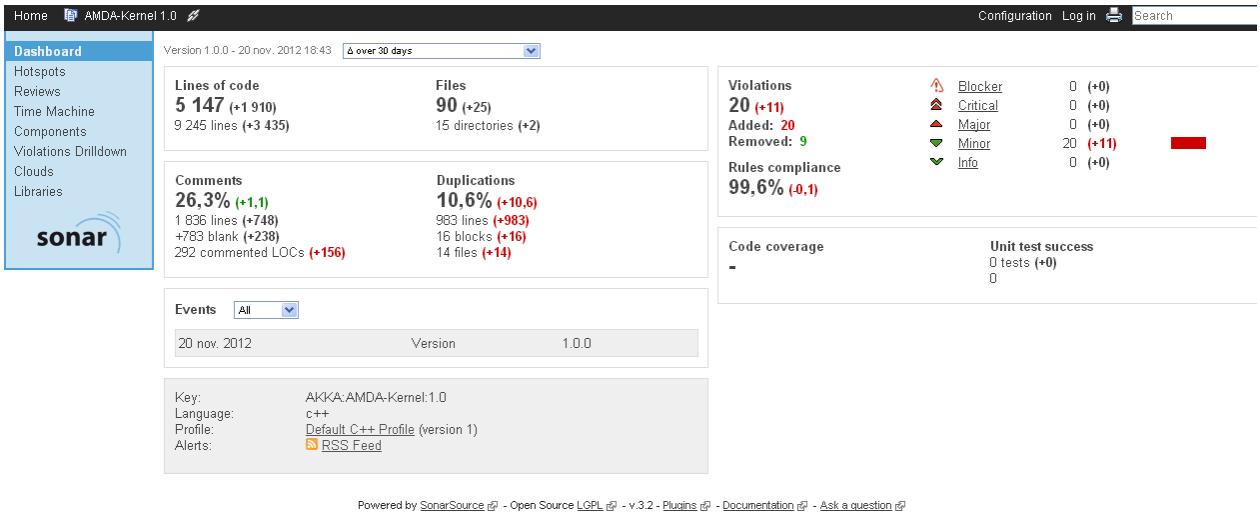
- définitlestests d'acceptation sous forme de scénarii,
- lance l'exécution de tous ou d'une partie de ses tests.

Le principe est que le/les scénario(s) sont passés via le protocole SLIM à un serveur de test qui les exécute et retourne les résultats. FitNesse est donc ordonnanceur et reporteur des tests. Dans notre contexte le serveur de test a été codé en C++.

2.7.2.2 Sonar

Sonar est un outil en ligne qui compile différents rapportsgénérés par d'autres outils de validation de code :

- Validation statique de code :
  - Cppcheck
  - Cxx-Sonar-plugin
- Validation dynamique de code
  - Valgrind



2.7.2.3 Jenkins

Jenkins est un ordonnanceur de tâche, il permet de faire de l'intégration continue et de lancer FitNesse et les analyseurs de code précédents tous les soirs.

2.7.2.4 Liste des outils utilisés

Outil	Plate-forme	Description	Site
Jenkins	bas-amda-02	open-source continuousintegration server	http://jenkins-ci.org/

FitNesse	bas-amda-01 bas-amda-02	scheduler and reporter of tests	
Sonar	bas-amda-02	open source quality management platform	http://www.sonarsource.org/
CXX SONAR Plugin	bas-amda-02	adds basic support of C++ language into Sonar	http://docs.codehaus.org/pages/viewpage.action?pageId=185073817
Cppcheck	bas-amda-01 bas-amda-02	Detects a wide range of problems ranging from performance issues and resource leakage to undefined behaviour. Binary packets are available on/for various platforms. Using the latest release pays off in general; compile from source if in doubt.	http://cppcheck.sourceforge.net/
Valgrind (memcheck)	bas-amda-01 bas-amda-02	Detects various memory management problems at runtime.	http://valgrind.org/

2.8 CRITERES D'ARRET DES TESTS

Test Unitaire

- FitNesse : tous les tests d'acceptation à OK

Test d'intégration :

- FitNesse : tous les tests d'acceptation à OK
- SONAR :
  - Valgrind aucune fuite mémoire.
  - Taux de commentaire > 25%
  - Pas de duplication de code
  - Pas de violation de règle majeure.

Test de validation :

- FitNesse : tous les tests d'acceptation à OK et couvre toutes les USs livrées

Test de Non régression

- FitNesse : tous les tests d'acceptation à OK

2.9 PLANNING ET CHARGES

Phase de développement d'une User Story : le temps d'écriture et d'exécution des tests est inclus dans l'évaluation en points de la User Story.

Phase d'intégration : l'automatisation des tests lors de la phase de développement et le passage de ces tests tous les soirs par un outil d'intégration continue, comme Jenkins,réduit le coup des passages de test aux corrections d'éventuelles régressions.

2.10 DEROULEMENT DE L'ACTIVITE DE TESTS

Le déroulement des tests unitaires est à la discrétion du développeur.

Le déroulement des tests d'intégration, de non-régressiona lieu de façon automatisé par jenkinstoutes les nuits. Le rapport est consultable en ligne sur l'outil tous les matins.

Les tests de validation sont déroulés pendant la phase de démo de fin de sprint via l'outil FitNesse.

2.11 GESTION DES ANOMALIES

L'intégration étant faite en continue, une anomalie constatée, l'a été au plus tôt et son traitement devient la priorité de la journée.

3 TEST UNITAIRE

3.1 OBJECTIFS/LIMITES

Les tests ont pour objectif de vérifier que le code de la US en développement est OK.

Ces tests sont à la discrétion des développeurs, Il se base sur l'outil FitNesse. Ces tests deviennent les tests d'intégration en fin de développement de la US.

3.2 DOCUMENTS DE REFERENCE

IceScrum

3.3 ELEMENTS CONCERNES

Les exécutables :

- EXE_AMDA_Kernel,
- amdaXMLRequestTool,
- amdaParameterGenerator.

3.4 JEUX DE DONNEES

Voir : §2.4

Les fichiers sont gérés en configuration.

La base de données est fournie par l'IRAP et est une instance dédiée aux tests sur manunja.cesr.fr:5000.

3.5 METHODOLOGIE

Tests boîte noire :

- un cas de test pour des données en entrée de l'élément de plus haut niveau,

La méthodologie utilisée est dite directe, tous les éléments sont intégrés en une seule fois : pas de bouchon.

4 TEST D'INTEGRATION

4.1 OBJECTIFS/LIMITES

Les tests ont pour objectif de vérifier que le code de l'ensemble des développeurs fonctionneencore, après intégration en gestion de configuration.

Ces tests sont automatisés par Jenkins. Ils ne s'appliquent qu'aux tests du sprint courant :

Ils consistent en :

- une récupération des sources à partir de la gestion de configuration (branche trunk).
- une compilation et une exécution des scénarii de tests enregistrés dans FitNesse.

4.2 DOCUMENTS DE REFERENCE

IceScrum

4.3 ELEMENTS CONCERNES

Les exécutables :

- EXE_AMDA_Kernel,
- amdaXMLRequestTool,
- amdaParameterGenerator.

4.4 JEUX DE DONNEES

Voir : §2.4

Les fichiers sont gérés en configuration.

La base de données est fournie par l'IRAP et est une instance dédiée aux tests sur manunja.cesr.fr:5000.

4.5 METHODOLOGIE

Tests boîte noire :

- un cas de test pour les données en entrée de l'élément de plus haut niveau,

La méthodologie utilisée est dite directe, tous les éléments sont intégrés en une seule fois : pas de bouchon.

4.6 SCENARIOS DE TESTS

Jobs Jenkins: AMDA-Kernel_Integration et AMDA-Kernel_SONAR.

4.7 RESULTATS DE TESTS

Jobs Jenkins: AMDA-Kernel_Integration et AMDA-Kernel_SONAR. Et la page SONAR

5 TESTS DE VALIDATION

5.1 OBJECTIFS/LIMITES

Les tests ont pour objectif de vérifier que les objectifs du client sont atteints.

Ces tests sont automatisés par Jenkins. Ils sont constitués par l'ensemble des tests de non régression et des tests d'intégration.

Ils consistent en :

- une récupération des sources à partir de la gestion de configuration (branche trunk).
- une compilation et une l'exécution des scénarii de tests enregistrés dans FitNesse.

5.2 DOCUMENTS DE REFERENCE

IceScrum

5.3 ELEMENTS CONCERNES

Les exécutables :

- EXE_AMDA_Kernel,
- amdaXMLRequestTool,
- amdaParameterGenerator.

5.4 JEUX DE DONNEES

Voir : §2.4

Les fichiers sont gérés en configuration.

La base de données est fournie par l'IRAP et est une instance dédiée aux tests sur manunja.cesr.fr:5000.

5.5 METHODOLOGIE

Tests boîte noire :

- un cas de test pour les données en entrée de l'élément de plus haut niveau,

La méthodologie utilisée est dite directe, tous les éléments sont intégrés en une seule fois pas de bouchon.

5.6 SCENARIOS DE TESTS

Jobs Jenkins : AMDA-Kernel_Integration, AMDA-Kernel SONAR et AMDA-Kernel_NonRegression.

5.7 RESULTATS DE TESTS

Jobs Jenkins : AMDA-Kernel_Integration, AMDA-Kernel SONAR, AMDA-Kernel_NonRegression et de la page SONAR

6 TESTS DE NON-REGRESSION

6.1 METHODOLOGIE DE TESTS DE NON-REGRESSION

Via l'outil Jenkins tous les tests d'acceptation des Users Stories non obsolètes sont systématiquement repassés.

Si un des tests ne passe plus après la modification du produit, la priorité est donnée à la résolution de cette régression.

6.1 SCENARIOS DE TESTS

Jobs Jenkins : AMDA-Kernel_NonRegression.

6.2 RESULTATS DE TESTS

Jobs Jenkins : AMDA-Kernel_NonRegression.

7 MATRICE DE COUVERTURE DES TESTS

Disponible sous IceScrum

8 DOCUMENTS APPLICABLES ET DE REFERENCE (A/R)

A/R	Référence	Titre
R1	CDPP-AR-32500-382-SI	Dossier d'architecture du noyau AMDA-NG
R2	CDPP-MI-32500-440-SI	Manuel d'installation de AMDAKernel
R3	CDPP-CD-32500-436-SI	Dossier de conception du noyau AMDA-NG

9 GLOSSAIRE ET ABREVIATIONS

9.1 GLOSSAIRE

Terme	Définition
Campagne de tests	Période continue d'activités de tests, comprenant tout ou partie des scénarii de tests
Cas de test	Un cas de tests est défini par l'état des données en entrée du test, les actions à mener par le testeur et les résultats attendus.
Jeux de tests	Il s'agit de créer des cas concrets d'utilisation de l'application à partir : • de sa description formalisée dans les spécifications • des règles définies dans le plan et dossier de tests (principalement en matière d'ergonomie) La définition de jeux de tests permet de prévoir et de formaliser la manière et les critères sur lesquels l'application va être recettée. Les jeux de tests ne peuvent être définis qu'une fois les spécifications validées.
Jeux de données	Ensemble de données de tests utilisées dans le même scénario (ou jeux d'essai)
Scénario de tests	Enchaînement chronologique de cas de tests, cohérent fonctionnellement et dont le résultat est un état stable du système.
Test boîte blanche (ou test structurel)	Méthode de test qui consiste à concevoir les données d'entrée et les résultats attendus en examinant la structure interne de l'objet à tester.
Test boîte noire (ou test fonctionnel)	Méthode de test qui consiste à concevoir les données d'entrée et les résultats attendus à partir des fonctions spécifiées de l'objet à tester, sans examiner sa structure interne.
Test de bon fonctionnement	Les tests de bon fonctionnement font partie des tests de validation. Il s'agit de tests effectués sur l'environnement cible (ou un environnement de simulation) et qui consistent à s'assurer que le système fonctionne normalement dans cet environnement. Au minimum, les tests de bon fonctionnement consiste à lancer l'application et à s'assurer qu'il s'exécute normalement sans blocage. Les tests de bon fonctionnement peuvent aussi consister à repasser tout ou partie des tests de validation effectués sur l'environnement de tests, voire des tests complémentaires nécessités par des différences entre les environnements.
Test d'intégration	Activité permettant de vérifier que les interfaces communes à plusieurs composants permettent bien de réaliser le comportement attendu entre ces composants. Les tests d'intégration répondent aux exigences de conception.
Test unitaire	Activité permettant de vérifier qu'un composant, pris isolément, satisfait à ses exigences fonctionnelles et techniques. Les tests unitaires répondent à la conception détaillée si elle a été réalisée.

Terme	Définition
Test de validation	Activité permettant de vérifier que le logiciel, dans son ensemble, satisfait à ses exigences fonctionnelles et techniques y compris dans son environnement cible. Les tests d'intégration répondent aux exigences de spécification.

9.2 ABREVIATIONS

Abréviation	Nom détaillé
DT	Dossier de tests
FT	Fiche de test
MCE	Matrice de couverture des exigences
US	User Story
TA	Test d'acceptation