CONSERVATOIRE NATIONAL DES ARTS ET METIERS

PARIS

MEMOIRE

Présenté en vue d'obtenir

le DIPLÔME D'INGENIEUR CNAM

SPECIALITE : INFORMATIQUE

OPTION : AISL

LÊ Francis

RECETTE ET DEPLOIEMENT DE L'OUTIL LDD+

Soutenu le 04/01/2011

JURY

PRESIDENT : TREVES Nicolas

MEMBRES :
CNAM : PRINTZ Jacques, BARTHELEMY François
RENAULT : CHANIAT Dominique
ATOS-ORIGIN : GOZE Anne

Remerciement

Les travaux relatifs au présent mémoire ont été effectués à la DSIR de Renault sur le site Novadis au Plessis-Robinson de début février 2007 à fin décembre 2007. Ils ont été dirigés par M. Dominique CHANIAT, Chef de l'UET Performance des Process SI. Mon responsable hiérarchique direct à Atos-Origin Intégration France est Mme Anne GOZE Responsable Qualité et Gestion des Risques du Contrat Renault. Mme Anne GOZE a participé au projet PRIMA côté Atos-Origin. La partie pédagogique du mémoire a été faite sous la direction des professeurs du CNAM Nicolas TREVES et de Jacques PRINTZ.

Je remercie :

- mes professeurs du CNAM, M. Nicolas TREVES et M. Jacques PRINTZ, pour avoir rendu possible ce mémoire.
- mon responsable à Renault, M. Dominique CHANIAT, qui m'a supervisé sur le projet et qui m'a autorisé à utiliser mon travail comme support de mon mémoire.
- Mon supérieur hiérarchique à Atos-Origin, Mme Anne Goze, pour avoir participé à mon jury.
- M. Gérard BESSON avec qui j'ai travaillé sur l'outil LDD+. S'il n'y avait pas eu une bonne entente entre nous, le projet n'aurait peut-être pas abouti. Je remercie aussi son successeur Jean-Philippe DECROIX pour ses explications.
- Mes collègues LDD+ Mme Marie-Ange DE TAFFANEL, M. Gilles MARCHAND, avec qui j'ai déployé LDD+.
- l'équipe VPMQ chargé du processus GCL en particulier M. Guillaume GIRAUD, Mme Aurélie BROUILLARD, Mme Maud BONNEHON, M. André MOREAU, Mme Nathalie CASSILDE.
- l'équipe SOAC chargé des outils qualité en particulier M. Frédéric SAINTE-ROSE, M. Abdelilah MARKOUM.
- mon entourage pour leur soutien et leurs encouragements, en particulier mon épouse Mme Nhung DOAN et mes amis Mme Emmanuelle OREAL et M. Jerôme HOUDAYER.
- mes camarades du CNAM, en particulier M. Gabriel GOIC.
- toutes les autres personnes avec qui j'ai travaillé dans le cadre de ce projet.

Table des Matières

Index 1
Introduction 2
1 PREMIERE PARTIE : PRESENTATION DU PROJET 4
1.1 Le contexte du projet 5
1.1.1 Ma situation professionnelle 5
1.1.2 La société Atos-Origin 5
1.1.3 Le groupe Renault 6
1.1.4 Le contrat Renault et le projet PRIMA 7
1.2 La démarche qualité CMMI 8
1.2.1 La représentation étagée 8
1.2.2 La représentation continue 9
1.3 Le projet PRIMA 10
1.3.1 Les processus PRIMA 10
1.3.2 Le déploiement d'un processus PRIMA 12
1.3.3 Le déploiement PRIMA et PDCA. 13
1.3.4 Les responsabilités dans un processus PRIMA 14
1.4 Le processus GCL 15
1.4.1 Les objectifs 15
1.4.2 Les principes 15
1.4.3 Les activités 16
1.4.4 Les responsabilités 17
1.4.5 Les outils choisis pour la GCL 17
1.4.6 Les participants au déploiement 18
1.5 Le projet « Recette et déploiement de l'outil LDD+ » 18
2 DEUXIEME PARTIE : PRESENTATION DE L'OUTIL LDD+ 19
2.1 L'environnement technique 20
2.1.1 Le travail d'un développeur Mainframe 20
2.1.2 L'environnement Mainframe 20
2.1.3 Les fichiers dans l'environnement Mainframe 21
2.2 Le prédécesseur LDD 23
2.2.1 Les principes de LDD 23
2.2.2 L'interface de LDD 24
2.2.3 L'architecture de LDD 26
2.2.4 Les transferts sous LDD 27
2.2.5 L'inadaptation de LDD pour la GCL 29
2.3 Présentation de l'outil LDD+ 29
2.3.1 L'origine de LDD+ 29
2.3.2 L'architecture de LDD+ 30
2.3.3 Les commandes LDD+ 32
2.3.4 Les fonctions de versionning 32
2.3.5 Les fonctions de traçabilité 36
2.3.6 Un exemple d'utilisation des fonctions Checkin et Checkout 42
2.3.7 Les fonctions de verrouillage de LDD+ 46
2.4 LDD+ et la Gestion de Configuration Logicielle 55
2.4.1 LDD+ et la Gestion de configuration dans CMMI 55
2.4.2 La comparaison de LDD+ avec Subversion 55

3 TROISIEME PARTIE : REALISATION DU PROJET 59
3.1 Les objectifs du projet « Recette et Déploiement de LDD+ » 60
3.2 L'organisation du projet « Recette et Déploiement de LDD+ » 60
3.3 Les étapes du projet « Recette et Déploiement de LDD+ » 60
3.4 La recette de LDD+ 62
3.4.1 Les fonctions testées 62
3.4.2 Le détail des tests de recette 63
3.4.3 Les résultats de la recette 69
3.5 La migration des projets pilotes 70
3.6 Le périmètre de déploiement 71
3.7 Les étapes d'une migration d'un projet 72
3.8 Les sessions de formation 74
3.9 Les opérations menées par l'équipe LDD+ 75
3.9.1 Les outils de migration. 75
3.9.2 Rappel sur l'architecture de LDD+ 77
3.9.3 L'étape de prémigration 78
3.9.4 L'ancienne étape de prémigration 79
3.9.5 L'ancienne étape de préparation 81
3.9.6 Les nouvelles étapes de préparation et de prémigration 82
3.9.7 L'étape de migration 83
3.10 Le suivi du déploiement de LDD+ 86
3.11 Le bilan du déploiement de LDD+ 86
4 QUATRIEME PARTIE : BILAN DU PROJET 88
4.1 Mon point de vue sur LDD+ 89
4.1.1 La qualité de LDD+ 89
4.1.2 Les limites fonctionnelles de LDD+ 89
4.1.3 La très faible maintenabilité de LDD+ 90
4.2 L'avenir de LDD+ 91
4.2.1 Les demandes d'amélioration 91
4.2.2 La pérennité de LDD+ 92
4.3 Les difficultés rencontrées 92
4.3.1 Les délais 92
4.3.2 L'opposition IRN/SIA 93
4.3.3 La phase de préparation 93
4.3.4 L'évolution des procédures 93
4.4 Mon bilan personnel 94
4.4.1 L'enrichissement personnel 94
4.4.2 L'environnement de travail 95
4.5 L'année 2008 95
Conclusion 96
Annexe A : Extrait du support de formation 98
Annexe B : Compte-rendu de prémigration 101
Annexe C : PV de validation pour la migration de LDD vers LDD+ 105
Annexe D : Suivi du déploiement de LDD+. 106
Bibliographie 107
Table des figures 108
Table des tableaux 109

Index

CCC : Comité de Contrôle des Changements	voir 1.4.4
CMMI : Capability Maturity Model Integration	voir 1.2
DAMT : Direction de l'Architecture, des Méthodes et des Technologies	voir 1.1.3
DESI : Direction des engagements des systèmes d'information	voir 1.1.3
DQ : Département Qualité	voir 1.4.6
DSIR : Direction des systèmes d'information Renault	voir 1.1.3
DSPI : Direction du Service et de la Production informatique	voir 1.1.3
GED : Gestion Electronique des Documents	voir 1.4.2
GCL : Gestion de la Configuration Logicielle	voir 1.4
IRN : Identifiant Renault Normalisé	voir 3.6
IPPD : Integrated Product and Process Development	voir 1.2.2
LDD : Lot de développement	voir 2
PCAQ : Process-Champion Assurance Qualité	voir 1.3.4
PGCL : Plan de Gestion Configuration Logicielle	voir 1.4.4
PRIMA : PRocess IMprovement and Assets	voir 1.3
RGCL : Responsable de la GCL	voir 1.4.4
RPL : Renault Programming Langage	voir 1.4.5
RS3 : Renault Systèmes Solutions et Services	voir 1.1.3
SGBD: Système de Gestion de Bases de Données	voir 3.11
SCLM : Software Configuration and Library Manager	voir 2.3.1
SSII : Société de services en ingénierie informatique	voir 1.1.2

Introduction

Le mémoire d'ingénieur est l'étape finale dans le cursus d'ingénieur du CNAM. Ce mémoire et la soutenance qui l'accompagne permet au jury d'apprécier l'aptitude du candidat à exercer des fonctions d'ingénieur. Dans le mémoire, le candidat y décrit les travaux réalisés dans le cadre d'un projet d'ingénieur. En général, ce projet est réalisé dans le cadre du travail du candidat et doit durer au minimum 6 mois. Le sujet du projet est validé par un enseignant habilité du CNAM. Dans mon cas, ce sont M. Trêves et M. Printz qui ont validé mon projet.

En 2005, le constructeur automobile Renault a choisi la SSII Atos-Origin comme unique prestataire pour ses développements informatiques. Pour améliorer la qualité des développements informatiques, Renault et Atos-Origin ont décidé de mettre en place la démarche qualité CMMI. C'est le projet PRIMA (PRocess IMprovement and Assets).

Le projet PRIMA consiste à formaliser les activités de développements sous forme de processus. Un processus est un ensemble d'activités corrélées ou interactives qui transforme des éléments en entrée en éléments de sortie. Le référentiel PRIMA fournit pour chacun des processus tous les éléments nécessaires pour le réaliser : procédures, modèles, guides, notes.

Un projet informatique est un ensemble de traitements et de données qui a pour objectif de remplir une fonction de gestion donnée. Par exemple, le projet TIC correspond à l'alimentation en données clients des outils de gestion de la relation client. Les traitements d'un projet sont réalisés par des composants informatiques comme des programmes, des paramètres ou des scripts d'exécution. Le processus GCL (Gestion de Configuration Logicielle) permet de suivre les évolutions des composants informatiques d'un projet. Pour appliquer le processus GCL il faut des outils qui dépendent de l'environnement informatique du projet. Pour l'environnement Windows/UNIX l'outil GCL choisi est SVN (Subversion). Pour l'environnement Mainframe l'outil GCL choisi est LDD+.

Ingénieur d'étude chez Atos-Origin, j'ai été chargé de mettre en place l'outil LDD+. J'ai commencé par recetter LDD+. Il s'agit de le tester et de vérifier qu'il fonctionne sans anomalies et qu'il applique la GCL. Pendant cette phase de recette, j'ai fait remonter la nécessité d'affiner la gestion des verrous. Puis j'ai déployé LDD+ sur des projets pilotes. Cela a permis d'utiliser LDD+ en conditions réelles et de faire remonter les observations des utilisateurs finaux. Cette phase m'a aussi permis d'élaborer la formation et la procédure de migration.

Ensuite nous avons commencé le déploiement à grande échelle de l'outil LDD+ avec en parallèle la formation des utilisateurs. La procédure de migration nécessitait de fusionner les environnements de développement. Après avoir déployé plusieurs projets et parce que les projets devenaient de plus en plus complexes, il devenait impossible de continuer avec cette fusion. Nous avons alors mis au point une nouvelle procédure de migration qui permettait de contourner la fusion des environnements de développement. Cette nouvelle procédure a permis d'accélérer le déploiement de l'outil LDD+.

A la fin de l'année 2007, tous les projets qui étaient prioritaires pour le déploiement ont migré sous LDD+. Le projet que je présente dans le cadre de mon mémoire d'ingénieur comprend la phase de recette et la phase de déploiement jusqu’à fin 2007.

Le présent mémoire est composé de quatre parties. Dans la première partie, je montrerai que ce projet s’inscrit dans le déploiement du processus GCL qui lui-même fait partie d’un projet de déploiement de la démarche qualité CMMI. Dans la deuxième partie, je présenterai l’outil LDD+ et j’expliquerai comment il permet d’appliquer la GCL. Dans la troisième partie, j’exposerai mes réalisations et j’expliquerai comment mes propositions ont permis d’améliorer l’outil et le processus de déploiement. Enfin dans la quatrième partie, je tirerai le bilan du projet.

Remarque : dans le mémoire, nous avons inversé les couleurs dans les copies d’écran de l’environnement MVS pour faciliter l’impression du mémoire. Les écrans noirs apparaissent en blanc.

1 PREMIERE PARTIE : PRESENTATION DU PROJET

Je commencerai par présenter le contexte du projet : ma situation professionnelle, les sociétés pour qui je travaille, le contrat qui les lie. Puis je montrerai que mon projet fait partie d’un projet plus global le projet PRIMA qui consiste à déployer la démarche qualité CMMI.

1.1 Le contexte du projet

1.1.1 Ma situation professionnelle

Je suis ingénieur d'étude chez Atos-Origin une société de services en ingénierie informatique (SSII prononcé SS2I). Ma spécialité est le système Mainframe.

J'ai été détaché en mission dans la société Renault. Cette mission a commencé fin janvier 2007 et doit se terminer fin 2008. La mission peut se partager en deux parties. La première va de fin janvier 2007 à fin 2007, elle correspond à la recette de LDD+ et au déploiement de LDD+ sur les projets prioritaires. Cette partie est la plus importante car elle a permis au parc applicatif de Renault d'être certifié CMMI niveau 3. La deuxième partie va de janvier 2008 à fin 2008, elle correspond à la fin du déploiement de LDD+ et à d'autres activités.

Le projet sur lequel porte le présent mémoire correspond à la première partie de ma mission chez Renault. Je présente maintenant les sociétés pour qui ce projet s'est fait : le prestataire Atos-Origin et le client Renault.

1.1.2 La société Atos-Origin

Les informations qui suivent sont issues des sites internet de la société Atos-Origin Intégration.
www.atosorigin.com
https://source.atosorigin.com/intranet/

Atos-Origin est une société de services en ingénierie informatique (SSII). C'est la troisième SSII européenne après IBM Global Services et Cap Gémini. L'activité d'Atos-Origin s'organise autour de 3 domaines d'activité : le conseil, l'intégration de systèmes et l'infogérance. Les principales entités qui la composent sont :

- Atos Consulting : conseil en management, en organisation, en processus métiers,
- Atos Intégration : intégration de systèmes,
- Atos Infogérance : gestion déléguée et mise à disposition de ressources informatiques,
- Atos Worldline : solutions de paiement, gestion de la relation client, e-services,
- Atos Euronext Market Solutions (AEMS) : solutions informatiques pour les bourses et intermédiaires financiers.

Atos-Origin emploie 50 000 personnes dans 40 pays, principalement la France, les Pays-Bas, l'Espagne et le Royaume-Uni. Elle est aussi le partenaire informatique mondial des Jeux Olympiques. La société s'est beaucoup développée par rachat de sociétés. La dernière acquisition importante est le rachat de la SSII SEMA. Ses 2 axes de développement sont une activité offshore pour diminuer les coûts et des activités spécialisées plus rentables comme les moyens de paiements (Atos Worldline), les outils financiers (AEMS), les systèmes de santé (BPO Medical).

Je fais partie d'Atos-Origin Intégration France, une entreprise qui compte environ 7000 collaborateurs. Dans la Figure 1, je présente l'organisation de ma société.

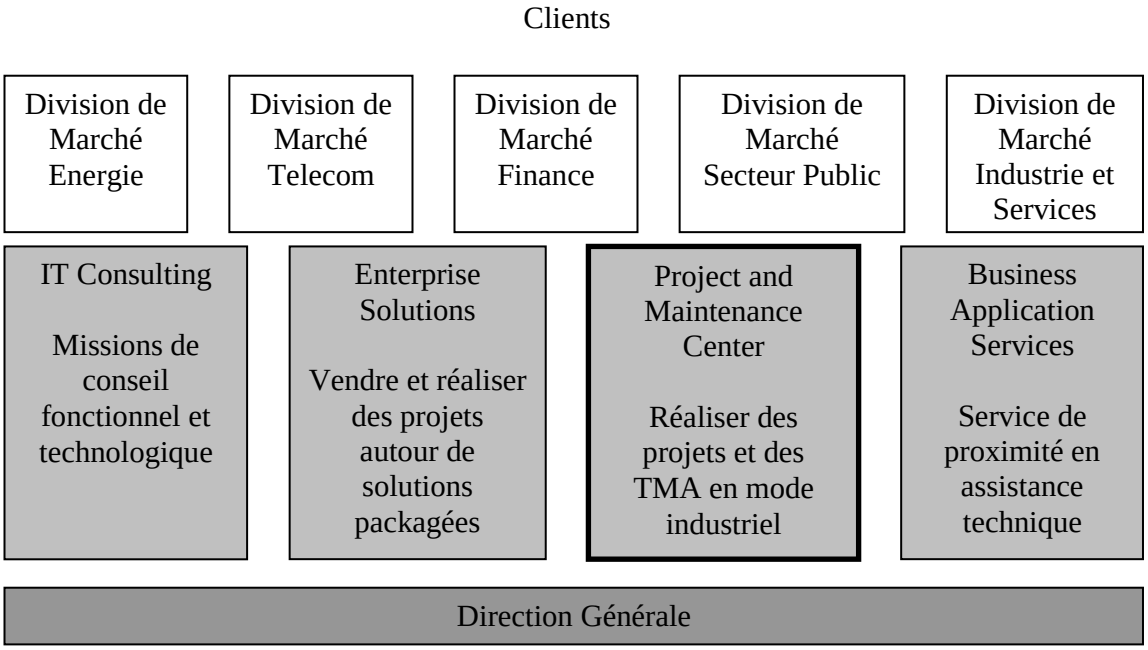


Figure 1 : Organisation de la société Atos-Intégration France

Je travaille pour l'unité de production Project and Maintenance Center.

Les clients sont en relation avec les divisions de marchés qui gère la relation commerciale. Les divisions de marchés sont spécialisées par secteur d'activité du client. L'ensemble des divisions de marchés représentent environ 800 personnes. Pour réaliser les contrats signés, les divisions de marchés s'adressent aux quatre unités de production :

- IT Consulting qui représente environ 200 personnes.
- Enterprise Solutions qui représente environ 1000 personnes.
- Project and Maintenance Center qui représente environ 1900 personnes.
- Business Application Services qui représente environ 2900 personnes qui travaillent chez le client et sous sa responsabilité.

La direction générale rassemble toute les fonctions centrales de l'entreprise (Finance, RH, Juridique) et représente environ 200 personnes. Je travaille dans l'unité de production Project et Maintenance Center pour le client Renault.

1.1.3 Le groupe Renault

Les informations qui suivent sont issues des sites internet de la société Renault.
www.renault.com
http://www.intra.renault.fr/corp/dsir/

Renault est un groupe automobile français de stature mondiale. Il regroupe les marques Renault (France), Samsung (Corée du Sud) et Dacia (Roumanie). En 2006, le groupe Renault a commercialisé 2 433 372 véhicules et réalisé un chiffre d'affaires de 41,5 milliards d'euros. Il compte 128 893 collaborateurs. De plus l'Alliance Renault-Nissan signée en 1999 permet à ces deux sociétés de partager une stratégie et des intérêts communs. Dans le cadre de cette alliance, des structures communes permettent de rapprocher processus industriels et stratégies commerciales tout en respectant culture d'entreprise et identité de marque. L'ensemble Renault-Nissan représente le quatrième groupe automobile mondial.

L'informatique de Renault est gérée par la DSIR Direction des systèmes d'information Renault. Dans la Figure 2, je présente l'organisation de la DSIR.

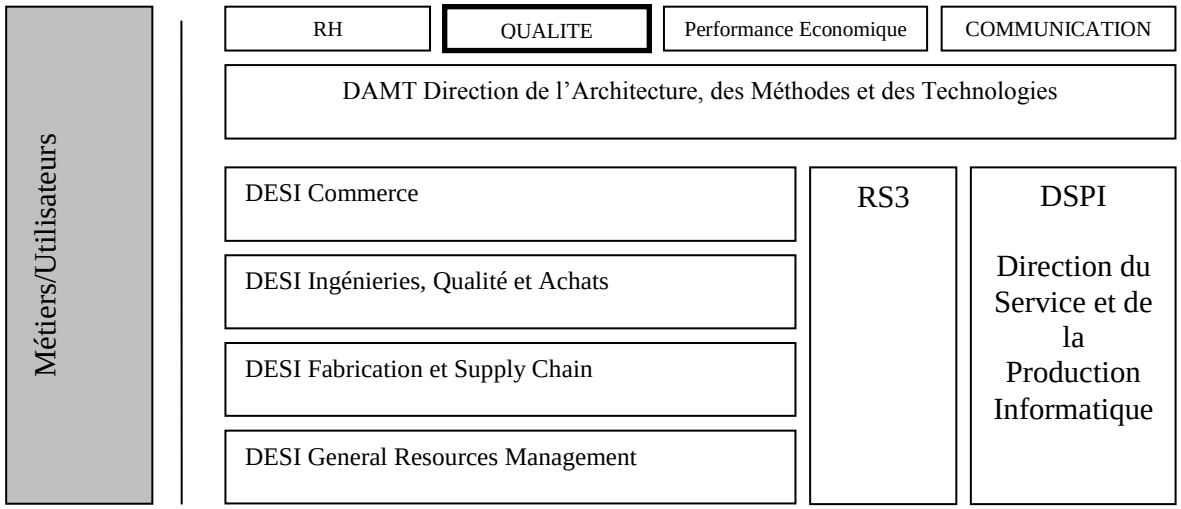


Figure 2 : Organisation de la DSIR

J'étais en mission dans le Département Qualité (DQ).

Pour ses besoins informatiques, les métiers de Renault s'adressent aux DESI (Direction des Engagements des Systèmes d'Information) correspondantes. Après avoir étudié les demandes et les besoins, la DESI réalise le cahier des charges. Le cahier des charges est ensuite soumis à RS3 (Renault System Solutions Services) pour réalisation. Une fois les traitements et les données réalisés, ils sont mis en production pour activité réelle par la DSPI (Direction du Service et de la Production informatique). En support des DESI, RS3 et DSPI, il y a les fonctions centrales que sont la RH, la Qualité, la Performance Economique, la Communication et la DAMT (Direction de l'Architecture, des Méthodes et des Technologies). Le rôle de la DAMT est de définir, promouvoir et s'assurer de l'application des normes et standards techniques et fonctionnels. C'est elle qui fournit support et assistance sur les outils aux DESI, RS3 et DSPI.

La DSIR correspond à environ 3000 collaborateurs Renault et 1300 applications. Pour le projet Recette et déploiement de l'outil LDD+, j'étais en mission dans le Département Qualité de la DSIR. Je présenterai plus en détail la DQ quand je détaillerai les participants du projet.

1.1.4 Le contrat Renault et le projet PRIMA

En 2005, Renault a choisi Atos-Origin comme unique prestataire pour les développements informatiques. Ces deux sociétés ont créé une structure commune Renault Systèmes Solutions et Services (noté RS3 prononcé RS-Cube) responsable des développements informatiques de Renault.

Pour satisfaire au mieux son client Renault, RS3 a décidé de mettre en place la démarche qualité CMMI pour améliorer la qualité des réalisations. C'est le projet PRIMA (PRocess IMprovement and Assets). Ce projet est très vaste et a été découpé en de nombreux sous-projets. Mon projet « Recette et déploiement de l'outil LDD+ » est l'un de ces sous-projets.

Le projet PRIMA consiste à déployer les processus CMMI pour les 800 projets informatiques de RS3. La mission de la Département Qualité dirigée par Alain Hubert a été de mener le projet PRIMA et faire obtenir la certification CMMI3 à RS3. Parmi ces processus CMMI déployés, il y a le processus Gestion de Configuration Logicielle (GCL). Avec le déploiement du processus GCL, il y a le déploiement d'outils GCL comme LDD+ ce qui est l'objet de ma mission et de ce mémoire. Je vais maintenant présenter la démarche qualité CMMI puis le processus GCL.

1.2 La démarche qualité CMMI

Les informations qui suivent sont issues des documents suivants :

Software Engineering Institute, 2006. CMMI for Development Version 1.2, Carnegie Mellon, Pittsburgh

www.sei.cmu.edu/cmmi/models/index.html

BASQUE R., 2009. CMMI 1.2 – Le Modèle. DUNOD, Paris

1.2.1 La représentation étagée

CMMI (Capability Maturity Model Integration) est une démarche qualité développée par la Software Engineering Institute de l'université américaine de Carnegie Mellon. Elle consiste à classifier les processus utilisés par une organisation qui fait du développement informatique, puis à répertorier les meilleures pratiques pour accomplir ces processus et enfin diffuser ces meilleures pratiques. En diffusant et en généralisant les meilleures pratiques, l'organisation gagne en efficacité et en qualité. On dit que l'organisation gagne en maturité.

Dans la démarche CMMI, il existe deux représentations : la représentation étagée et la représentation continue. Dans la représentation étagée présentée dans la Figure 3, une organisation peut être classée selon 5 niveaux de maturité :

- niveau 1 ou niveau initial : les processus sont accomplis mais leur succès est aléatoire et non reproductible.
- niveau 2 ou niveau géré : les processus sont documentés et réutilisables.
- niveau 3 ou niveau défini : les processus sont standardisés et il y a une capitalisation des connaissances.
- niveau 4 ou niveau géré quantitativement : les processus ont des objectifs quantifiables et on applique des actions de correction en cas de dérive.
- niveau 5 ou niveau en optimisation : on met en place des actions de prévention pour anticiper les dérives ainsi qu'un processus d'amélioration continue.

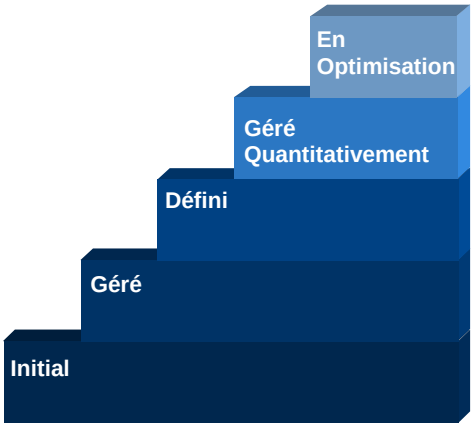


Figure 3 : Représentation étagée du modèle CMMI

On y voit les 5 niveaux de maturité d'une organisation.

1.2.2 La représentation continue

Dans la représentation continue représentée dans le Tableau I, les processus sont regroupés en 22 domaines de processus. Dans une organisation donnée, la maturité de chacun des domaines de processus peut être classée selon 6 niveaux. Ce sont les 5 niveaux de la représentation étagée auxquels on a ajouté le niveau 0 ou niveau incomplet : les processus sont partiellement ou pas appliqués.

Tableau I : Représentation continue du modèle CMMI
On y voit les 22 domaines de processus.
La colonne Maturity Level indique le niveau du modèle étagé auquel ils appartiennent.

Table 3.2 Process Areas and Their Associated Categories and Maturity Levels

Process Area	*Category*	*Maturity Level*
Causal Analysis and Resolution	Support	5
Configuration Management	Support	2
Decision Analysis and Resolution	Support	3
Integrated Project Management +IPPD	Project Management	3
Measurement and Analysis	Support	2
Organizational Innovation and Deployment	Process Management	5
Organizational Process Definition +IPPD	Process Management	3
Organizational Process Focus	Process Management	3
Organizational Process Performance	Process Management	4
Organizational Training	Process Management	3
Product Integration	Engineering	3
Project Monitoring and Control	Project Management	2
Project Planning	Project Management	2
Process and Product Quality Assurance	Support	2
Quantitative Project Management	Project Management	4
Requirements Development	Engineering	3
Requirements Management	Engineering	2
Risk Management	Project Management	3
Supplier Agreement Management	Project Management	2
Technical Solution	Engineering	3
Validation	Engineering	3
Verification	Engineering	3

Dans la représentation étagée, une organisation est d'abord au niveau 1. Pour passer au niveau 2, il faut que tous les 7 domaines de processus notés 2 dans le tableau précédent soient de maturité niveau 2. Pour passer au niveau 3, il faut que tous les 18 domaines de processus soient de niveau 3. Il s'agit des 7 domaines du niveau 2 et des 11 domaines notés 3 dans le tableau précédent. Pour passer au niveau 4, il faut que tous les 20 domaines soient de niveau 4. Il s'agit des 18 domaines précédents et des 2 domaines notés 4. Pour passer au niveau 5, il faut que tous les 22 domaines soient au niveau 5.

Pour qu'un domaine de processus soit correctement accompli, il faut que les objectifs spécifiques (Specific Goal) qui lui sont rattachés soient correctement accomplis. Le domaine de processus qui nous intéresse est « Configuration Management » ou « Gestion de la configuration ». Il repose sur les trois objectifs spécifiques :

- SG1 : établir des référentiels.
- SG2 : suivre et contrôler les modifications.
- SG3 : établir l'intégrité.

1.3 Le projet PRIMA

Les informations qui suivent sont issues du site intranet du projet PRIMA.
http://www.intra.renault.fr/projet/primab/

Au début, l'objectif de RS3 est que la maturité de son organisation soit de niveau 3 à fin 2006. Cette date a ensuite dérivé pour atteindre 2008. Une évaluation par un organisme officiel certifiera cet acquis. Cette certification a été effectuée par la société QLABS.

1.3.1 Les processus PRIMA

L'ensemble des tâches que peut accomplir RS3 pour Renault a été classifié en processus. Dans le jargon RS3 on utilise aussi l'anglicisme process pour processus. Le résultat de cette classification est le référentiel PRIMA. Les processus ont été regroupés en 8 domaines de processus selon la nature du service rendu. Ces 8 domaines sont représentés dans le logo du projet PRIMA présenté dans la Figure 4.



Figure 4 : Logo du projet PRIMA avec les 8 domaines de processus

Les huit domaines sont :

- Gérer un service
- Manager l'organisation
- S'engager à livrer un service
- Piloter la réalisation de services
- Réaliser un projet, une maintenance, les autres services
- Installer et recetter un service
- Fournir un support à un service
- Fournir le support

Le détail des 8 domaines est présenté dans la Figure 5 :

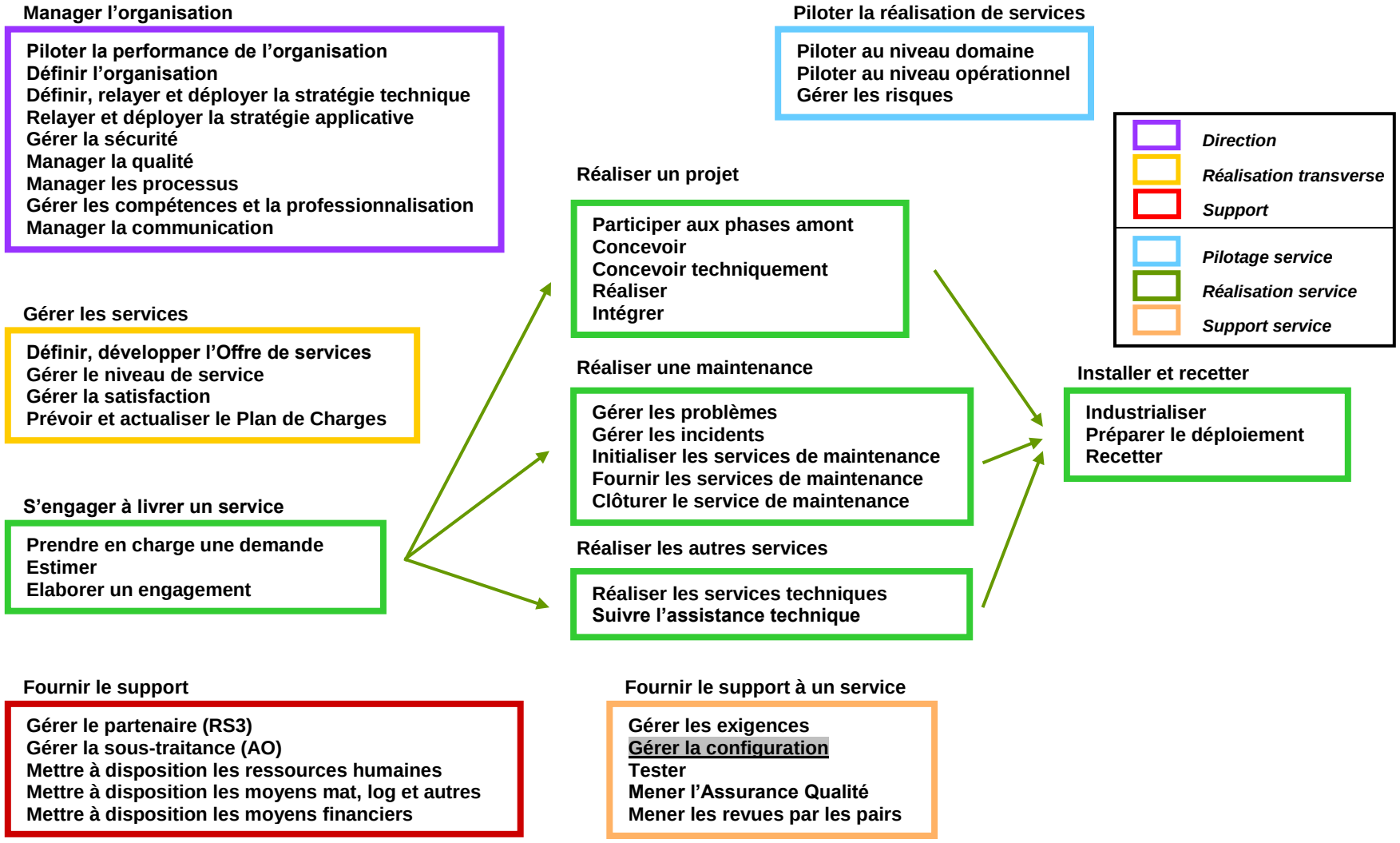


Figure 5 : Les processus du projet PRIMA

Chaque boîte correspond à un des 8 domaines de processus à l'exception du domaine« Réaliser un projet, une maintenance, les autres services » qui a été partagé en 3 boîtes. Le processus Gérer la configuration logicielle fait partie du processus Gérer la configuration (en surligné gris) qui lui même fait partie du domaine Fournir le support à un service (en bas au centre).

Le processus qui nous intéresse est le processus Gestion de Configuration Logicielle. C'est un sous-processus du processus « Gérer la configuration ». Dans la Figure 5, ce processus surligné en gris se trouve dans le domaine de processus « Fournir le support à un service ». Parmi les 44 processus, trois sont considérés comme prioritaires car ils nécessitent la mise en place d'outils et donc de formation lourde. Ce sont des processus du domaine « Fournir un support à un service » :

- Gérer les exigences (GEX)
- Gérer la configuration qui lui-même se partage en deux sous-processus : gestion électronique des documents (GED) et la gestion de configuration logicielle (GCL). Le projet recette et déploiement de l'outil LDD+ dans lequel j'interviens fait partie du sous-processus GCL.
- Tester

On constate que pour les 22 domaines de processus de représentation continue CMMI (voir Tableau I) diffèrent des 44 process PRIMA (voir Figure 5). Les processus PRIMA ont été choisi pour leur aspect opérationnel et on a vérifié qu'ils couvrent les 22 domaines de processus du modèle CMMI. Par exemple le domaine de processus CMMI « Configuration Management » est couvert par le processus PRIMA « Gérer la configuration ». Cette couverture a été vérifiée et validée par l'organisme certificateur QLABS.

1.3.2 Le déploiement d'un processus PRIMA

Selon la norme ISO9000, un processus est un ensemble d'activités corrélées ou interactives qui transforme des éléments d'entrée en éléments de sortie. Le référentiel PRIMA fournit pour chacun des processus tous les éléments nécessaires pour le réaliser : procédures, modèles, guides, notes. Il se matérialise sous la forme d'un site Intranet qui centralise tous les documents. Chaque processus est formalisé par sa carte d'identité process. Dans ce document, on trouve entre autre l'objet du processus, ses finalités, les entrées, les sorties, les activités ou tâches, les critères d'efficacité du processus. Le déploiement d'un processus PRIMA suit un cycle qui est présenté dans la Figure 6 suivante :

- De nouvelles procédures ont été définies et/ou de nouveaux outils ont été choisis.
- Ces nouvelles procédures et/ou nouveaux outils sont déployés sur des entités opérationnelles pilotes.
- A l'issue de ces déploiements et selon les retours, une décision de GO/ NOGO est prise.
- En cas de GO, les nouvelles procédures et/ou nouveaux outils sont déployées sur toutes les entités opérationnelles.
- Les utilisateurs sont formés et accompagnés pour les appliquer.
- Une fois le déploiement terminé, on mesure et on surveille l'application.
- Les retours d'expérience et les propositions d'amélioration sont remontés pour éventuellement élaborer de nouvelles procédures et/ou de nouveaux outils. Ce qui redéclenche un nouveau cycle de déploiement.

Ce cycle correspond au niveau 3 de la représentation étagée, le niveau défini. Dans le niveau CMMI 3, les processus sont institutionnalisés c'est-à-dire qu'ils sont définis au niveau de l'organisation.

Tous les projets doivent appliquer les mêmes processus et s'il y a des exceptions, ils sont répertoriés et il doit y en avoir le minimum. Pour s'assurer que les processus soient communs, il faut insister sur la formation, l'accompagnement, la mesure et la surveillance. De plus dans le niveau 3, tous les projets doivent participer à l'enrichissement des processus. Cela se fait par les retours d'expérience et par les propositions d'amélioration c'est ce qu'on appelle la capitalisation.

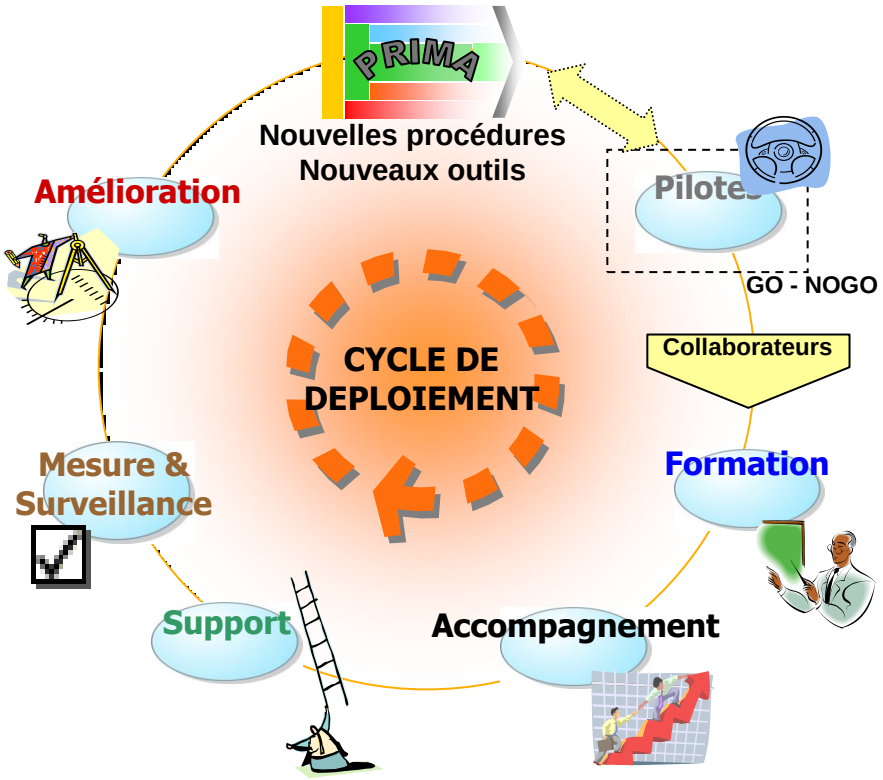


Figure 6 : Cycle de déploiement d'un processus PRIMA

On y voit les 2 exigences du niveau 3 CMMI : l'institutionnalisation et la capitalisation.

1.3.3 Le déploiement PRIMA et PDCA.

Les informations qui suivent sont issues du site Wikipedia sur PDCA.
http://fr.wikipedia.org/wiki/Roue_de_Deming
http://www.hci.com.au/hcisite2/toolkit/pdcacycl.htm

Le cycle de déploiement PRIMA de la Figure 6 peut se comparer à la roue de Deming de la Figure 7.

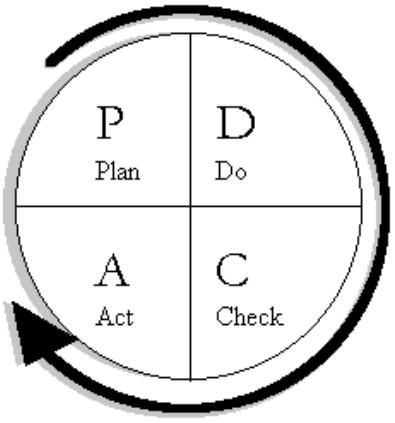


Figure 7 : Roue de Deming de la méthode PDCA

La roue de Deming est une illustration de la méthode de gestion de la qualité PDCA. Cette méthode, qui comme PRIMA vise à établir un cercle vertueux d'amélioration, comporte 4 étapes :

- Plan. D'abord on étudie les problèmes qu'on veut résoudre, les améliorations qu'on veut apporter. Puis on identifie les solutions à apporter.
- Do. On met en place les changements nécessaires d'abord à petite échelle ou sur des projets expérimentaux. Cela permet de tester les changements sans que l'organisation soit bouleversée en cas de problème.
- Check. On vérifie que les changements apportent les résultats attendus, que ceux-ci ne se dégradent pas au cours du temps et que de nouveaux problèmes ne sont pas apparus.
- Act. Si l'expérience a donné satisfaction, on change d'échelle afin que toute l'organisation profite des changements.

Dans les deux cycles, on s'appuie sur l'expérience accumulée pour améliorer la qualité. Par contre, dans le cycle de déploiement PRIMA (voir la Figure 6) la phase d'expérimentation est limitée aux projets pilotes. On insiste surtout sur la formation et l'accompagnement des utilisateurs.

1.3.4 Les responsabilités dans un processus PRIMA

Les responsabilités au niveau de chaque processus sont : le ou les propriétaires du processus, le ou les PCAQ (Process-Champion Assurance Qualité terme inventé par le projet PRIMA), les utilisateurs du processus.

Le propriétaire du processus est le responsable du processus. En premier lieu, il supervise et valide la formalisation du processus. Si un outil est nécessaire, il choisit celui qui est le plus adapté aux besoins. Puis il organise le déploiement et s'assure de l'application. Enfin il suit les indicateurs de performance et décide des actions correctives s'il y a des écarts par rapport aux objectifs. Suite aux remontées des utilisateurs, il s'occupe de l'amélioration du processus. Pour la gestion de configuration, le propriétaire du processus était Guillaume Giraud.

Le PCAQ apporte du support au déploiement du processus dans chaque entité opérationnelle. Il aide au démarrage du processus en l'adaptant aux contraintes de l'entité opérationnelle et vérifie son application dans le temps par des audits. Il fait régulièrement un bilan du processus et fait remonter les améliorations possibles. Pour la gestion de configuration, les PCAQ étaient : Aurélie Brouillard, Maud Bonnehon, Jean-François Renault et André Moreau.

Les utilisateurs du processus reçoivent formation et support pour appliquer le processus. Ils doivent s'assurer que le processus est correctement appliqué. Ils sont régulièrement évalués par le biais d'un entretien. Ces entretiens sont l'occasion de retours d'expérience et de propositions d'amélioration. Ce qui distingue CMMI d'autres démarches qualité est qu'on va au-delà de la vérification de l'application du processus en s'assurant de l'implication personnelle de l'utilisateur.

Nous allons nous intéresser au processus gestion de la configuration logicielle (GCL) sur lequel porte mon projet.

1.4 Le processus GCL

Les informations qui suivent sont issues du support de formation au processus GCL.
PRIMA-SS#01-SF-393-Processus Gestion de configuration logiciel - V2.1.ppt

1.4.1 Les objectifs

La gestion de configuration a plusieurs objectifs :

- identifier de façon univoque les constituants d'un projet. Ces constituants peuvent être des sources, des exécutables, des documents, des informations concernant le matériel, des informations concernant les données de test. Ces constituants sont désignés par le terme d'ARTICLES DE CONFIGURATION.
- identifier les REVISIONS des fichiers sources. Chaque modification d'un fichier source donne une nouvelle version d'un fichier source qu'on appelle REVISION. On utilise le terme de REVISION pour le distinguer du terme VERSION. L'ensemble des articles de configuration sous une révision donnée forme une VERSION du projet informatique.
- justifier l'origine de chaque REVISION. Il faut que chaque REVISION soit justifiée par une demande fonctionnelle ou technique
- tracer l'historique des révisions et des versions. Cela permet de protéger les articles de configuration des dégradations et de pouvoir reconstituer une VERSION à la demande.
- gérer le travail commun de plusieurs personnes en gérant les conflits qui apparaissent quand deux personnes travaillent en même temps sur le même fichier source.

La gestion de configuration logicielle permet donc de répondre à des questions comme :

- qui a modifié un article de configuration et qu'est-ce qui a été changé ?
- pour quelle raison un article de configuration a été modifié ?
- quelles sont les articles de configuration et leurs révisions associées à telle version ?

La gestion de configuration permet de maîtriser les évolutions du projet :

- en assurant la traçabilité des actions
- en assurant la coordination des mises à jour concurrentes
- en liant chaque modification à une demande d'évolution ou de correction
- en permettant de connaître précisément les composants d'un transfert

1.4.2 Les principes

La gestion de la configuration s'appuie sur trois éléments :

- une BASELINE. Une BASELINE stocke les informations des ARTICLES DE CONFIGURATION dans le temps. Elle suit leur évolution. A un moment donné, pour une VERSION donnée, la BASELINE fournira les ARTICLES DE CONFIGURATION et leur REVISION. En suivant dans le temps les VERSIONS, une BASELINE permet de comparer deux VERSIONS. Pour modifier un ARTICLE DE CONFIGURATION, il faut justifier cette modification par une demande d'évolution ou de correction. En stockant ces informations, on peut à l'inverse retrouver la demande d'origine d'une modification.

- une REFERENTIEL. Un REFERENTIEL stocke tous les d'ARTICLES DE CONFIGURATION ainsi que toutes leurs REVISIONS. A un moment donné, l'état d'un REFERENTIEL correspond à une VERSION donnée. Grâce aux informations de la BASELINE, un REFERENTIEL peut reconstituer n'importe quelle VERSION en récupérant les REVISIONS correspondantes des ARTICLES DE CONFIGURATION.

- une gestion des DEVELOPPEMENTS CONCURRENTS. Il 's'agit de gérer le travail commun de plusieurs personnes en gérant les conflits qui apparaissent quand deux personnes travaillent en même temps sur le même fichier source.

La gestion de configuration est constituée de sous-processus selon la nature des objets gérés :

- la GED (Gestion Electronique des Documents) gère la documentation
- la GCL (Gestion de Configuration Logicielle) gère les objets informatiques

Plusieurs critères distinguent la GED et la GCL. Dans notre cas, cette distinction se fait sur le codage des fichiers gérés. La GED gère des documents en général des fichiers Word ou Excel ou PDF. Leur codage rend très compliqué le stockage des différentes versions d'un fichier par delta : on ne peut pas facilement reconstituer une version d'un fichier à partir d'une autre version et de leurs différences. A l'inverse La GCL gère en général des fichiers au format texte. Avec ce format, on peut facilement reconstituer la version d'un fichier par delta.

1.4.3 Les activités

Le processus GCL s'appuie sur 5 activités :

- AC01 Initialiser. Il s'agit mettre en place les procédures et de former les acteurs.
- AC02 Créer le référentiel. Il s'agit mettre en place les outils. On crée le référentiel et on le remplit avec les articles de configuration s'ils existent déjà.
- AC03 Modifier le référentiel. Il s'agit de modifier les articles de configuration selon une demande d'évolution ou de correction. La demande est identifiée et les modifications sont enregistrées.
- AC04 Livrer à une autre entité. Une autre entité est un service qui va recevoir les articles de configurations modifiés à partir du référentiel soit pour les utiliser soit pour les mettre en place pour utilisation. Le bon de livraison recense toutes les modifications et chaque modification est rattachée à une demande d'évolution ou de correction.
- AC05 Auditer. Il s'agit de vérifier que le processus GCL est correctement appliqué et de faire remonter les remarques et les améliorations. Cette vérification se fait soit en interne par le projet, c'est une revue de configuration, soit en externe par les PCAQ c'est un audit.

Comme nous l'avons expliqué dans la présentation de la représentation continue de CMMI (voir 1.2.2), le processus CMMI « Gestion de la configuration » repose sur trois objectifs spécifiques (Specific Goals) :

- SG1 : établir des référentiels.
- SG2 : suivre et contrôler les modifications.
- SG3 : établir l'intégrité.

Pour s'assurer que le processus GCL PRIMA est conforme au processus GCL CMMI, le Département Qualité (DQ) a vérifié que les 5 activités du processus GCL PRIMA permettent de remplir les trois objectifs spécifiques de la gestion de configuration CMMI selon le Tableau II.

Tableau II : Les activités PRIMA et les attentes CMMI pour la GCL

	AC01 Initialiser	AC02 Créer le référentiel	AC03 Modifier le référentiel	AC04 Livrer à une autre entité	AC05 Auditer
SG1 Etablir les référentiels	X	X	X		
SG2 Suivre et contrôler les changements		X	X		X
SG3 Etablir l'intégrité				X	X

1.4.4 Les responsabilités

Le responsable de gestion de configuration logicielle (RGCL) est responsable de l'application de la procédure telle qu'elle est décrite dans le plan de gestion de configuration logicielle (PGCL). Avant chaque livraison, le responsable de GCL réunit le Comité de Contrôles des Changements (CCC). Le CCC acte le contenu de la livraison dans le bon de livraison :

- les demandes livrées et recettées,
- les demandes non livrées ou livrées mais non recettées,
- les anomalies de recettes connues.

Le résultat du CCC est le bon de livraison pour décision d'acceptation de la livraison. Le PGCL est un document important. Il est spécifique au projet. Il liste les différents rôles et leurs activités. C'est un document opérationnel susceptible d'évoluer avec le projet. Il est élaboré et mis à jour par le responsable GCL.

1.4.5 Les outils choisis pour la GCL

Plus que tout autre processus, la GCL a besoin d'outil. Dans le cadre du projet PRIMA, l'outil choisi sera différent selon l'environnement technique du projet. Pour l'environnement Windows/UNIX, l'outil qui a été retenu est Subversion. L'outil Subversion sera présenté dans le paragraphe 2.4.2. Avant le projet PRIMA, les projets qui appliquaient la GCL utilisaient surtout CVS ou PVCS. L'avantage de Subversion est qu'il est Open Source et que son référentiel est accessible par un simple navigateur. Pour l'environnement SAP, la GCL est gérée en interne par le produit. Pour l'environnement Mainframe (voir 2.1.2), c'est l'outil LDD+ qui a été choisi. LDD+ est l'outil sur lequel porte mon mémoire d'ingénieur. Chaque outil a été sélectionné après étude comparative parmi les outils disponibles dans le marché à l'exception de LDD+ qui est un développement interne.

Les informations qui suivent m'ont été transmises par Gérard Besson (voir 1.4.6) quand je suis arrivé sur le projet. Pour choisir un outil de GCL pour l'environnement Mainframe, Renault a étudié les produits disponibles sur le marché. Les deux produits commerciaux les plus connus sont Endevor édité par Computer Associates et Changeman édité par Serena. Ils n'ont pas été choisis pour plusieurs raisons :

- ils étaient chers. Le peu d'informations que j'ai pu trouvées indiquent un prix autour de 100000 dollars.
- ils étaient lourds. Ils auraient demandé une modification importante des environnements de recette et de production. Je suppose qu'il aurait fallu renommer de nombreux fichiers, adapter les compilateurs. Ce qui supposait des tests très poussés pour vérifier que les traitements existants étaient conservés.
- on n'était pas certain que ces produits puissent gérer RPL (Renault Programming Langage) un langage et un environnement de programmation spécifique à Renault (voir 3.9.4). Il y avait un risque que ces produits ne puissent pas utiliser les fichiers et des compilateurs RPL.
- ces outils auraient demandé aux développeurs de modifier leurs méthodes de travail, ce qui n'était pas réalisable dans les délais. Ils perdaient la gestion des lots apportés par LDD et l'interface et les procédures de transferts étaient modifiées. La notion de lot et les procédures de transferts sont présentées dans la présentation de LDD (voir 2.2).

Pour ces raisons de coût, de simplicité et de délai, on a préféré un développement interne à partir d'un outil qui existait déjà. Le résultat de ce développement interne est l'outil LDD+. '+' car son prédécesseur s'appelait LDD.

1.4.6 Les participants au déploiement

Le projet PRIMA a été mené par le Département Qualité (DQ) composée d'une trentaine de membres et dirigée par Alain Hubert. La DQ est constituées de deux équipes.

La première équipe, dirigée par Patrick Lemaître, est l'équipe VPMQ (Validation Produits et Management Qualité). L'équipe VPMQ est constituée de PCAQ (Process Champions Assurance Qualité). Les PCAQ s'occupent de l'accompagnement fonctionnel des projets pour la mise en place des processus. Une fois le processus GCL mis en place, ils auditent le projet pour vérifier qu'il est correctement appliqué. Les PCAQ qui s'occupent de la GCL sont : Aurélie Brouillard, Maud Bonnehon, Jean-François Renault et André Moreau qui s'occupe plus spécifiquement de l'environnement Mainframe. Ils sont encadrés par Guillaume Giraud, propriétaire du processus « Gérer la Configuration » (voir 1.3.4).

La deuxième équipe, dirigée par Dominique Chaniat, est l'équipe SOAC (Support des Outils d'Audit de Code). L'équipe SOAC est responsable de la partie outillage de la qualité. Pour la GCL, Frédéric Sainte-Rose et Abdelilah Markoum s'occupent de Subversion et je m'occupe de LDD+. Ponctuellement j'ai été aidé par Marie-Laure de Taffanel et par Gilles Marchand.

Une autre personne venant de la DAMT (voir 1.1.3) est intervenue dans le déploiement de LDD+. C'est Gérard Besson. C'est l'intervenant le plus important car c'est lui qui a créé et réalisé LDD+. J'ai travaillé en étroite collaboration avec lui pour mettre au point, tester et valider LDD+.

1.5 Le projet « Recette et déploiement de l'outil LDD+ »

Pour pouvoir déployer le processus GCL sur tous les projets qui utilisent l'environnement Mainframe, il fallait que tous ces projets utilisent l'outil LDD+. Quand je suis arrivé fin janvier sur le projet, une première version de l'outil LDD+ existait. Cette version reposait sur SCLM (Software Configuration and Library Manager). SCLM est un outil des versionning de fichiers présent en standard dans l'environnement Mainframe. Après des tests de performance menés par Gérard Besson (voir 3.3), on a constaté qu'on ne pouvait pas garder SCLM. Il a été décidé de redévelopper LDD+ pour se passer de SCLM. Une fois ce développement fait, j'ai été chargé de tester et recetter l'outil. Une fois l'outil recetté, on m'a confié la tâche de déployer LDD+. C'est donc la recette et le déploiement de l'outil qui constitue le projet de ce mémoire.

Avant de présenter les réalisations accomplies pour ce projet, je vais maintenant présenter l'outil LDD+ sur lequel repose ce projet. Cela permettra de mieux comprendre mes réalisations.

2 DEUXIEME PARTIE : PRESENTATION DE L’OUTIL LDD+

Je commencerai par exposer l’environnement technique pour mieux comprendre la suite. Puis je présenterai le prédécesseur LDD (Lot De Développement) et je montrerai qu’il n’est pas adapté pour la GCL. Puis j’exposerai LDD+ et en particulier ses fonctions de GCL. Les informations qui suivent sont issues du support de formation à LDD+que j’ai moi-même rédigé : Formation-LDD+-080314.ppt

2.1 L'environnement technique

2.1.1 Le travail d'un développeur Mainframe

Les développeurs informatiques qui travaillent dans l'environnement Mainframe sont les principaux utilisateurs de l'outil LDD+. J'explique ici leur travail. Un développeur informatique travaille avec des fichiers. Cela peut être des fichiers de données ou des fichiers sources. Un fichier source contient du code informatique pour contrôler les traitements. Certains fichiers sources sont compilés, c'est à dire traités pour fournir un exécutable. Un traitement informatique, par exemple l'affichage d'une liste de véhicules, est le résultat de l'exécution d'un ou de plusieurs exécutables. Certains fichiers sources comme les scripts ou les paramètres n'ont pas besoin d'être compilés pour être utilisés. Le travail d'un développeur informatique consiste à :

- concevoir un ou des sources selon une demande des commanditaires. On parle d'analyse.
- réaliser le ou les sources à partir du résultat de l'analyse.
- compiler les sources qui en ont besoin.
- tester les exécutables. Le développeur va exécuter les traitements correspondant aux sources développés et vérifier qu'ils produisent les résultats attendus.

Une fois les tests validés, le développeur procède à une livraison en recette. Cela consiste à transférer les sources et leurs exécutables dans un autre environnement, l'environnement de recette, où les commanditaires pourront tester et vérifier que les développements correspondent à ce qu'ils ont commandés. On parle de phase de recette. Une fois la recette validée, les sources et leurs exécutables sont livrés en environnement de production afin que les traitements s'exécutent sur les données réelles nécessaires au bon fonctionnement de l'entreprise. Les environnements de test, recette et production sont en général disjoints pour qu'il n'y ait pas d'interférences entre les traitements.

LDD+ est l'outil de GCL pour l'environnement Mainframe. C'est un environnement très différent de l'environnement Windows/UNIX. Pour mieux comprendre la suite du mémoire, je vais vous présenter les particularités de l'environnement Mainframe.

2.1.2 L'environnement Mainframe

Le terme « Mainframe » désigne un environnement informatique particulier. Cet environnement existait avant UNIX et Windows. Maintenant il est surtout utilisé pour les traitements de masse dans l'informatique de gestion. Un système « Mainframe » est un système centralisé contrairement aux systèmes Windows/UNIX.

Pour ces derniers, chaque machine, poste de travail ou serveur, est indépendant : il faut qu'il y ait un système d'exploitation qui s'exécute sur chacune des machines même s'ils peuvent communiquer entre eux via le réseau. Pour un système « Mainframe », il n'y a qu'une seule machine qui exécute le système d'exploitation et les développeurs y accèdent via un terminal. De nos jours, les terminaux n'existent plus. On utilise un logiciel, appelé émulateur, qui s'exécute sur un machine Windows et qui reproduit le fonctionnement d'un terminal.

L'interface dans l'environnement Mainframe est rustique. Elle est basée sur des écrans constitués de 24 lignes et 80 colonnes. Dans chacune des positions correspond un caractère sans possibilité d'animation graphique. On saisit les données et les commandes à travers des champs. Dans la Figure 8, je présente l'écran d'accueil lorsqu'on vient de connecter.

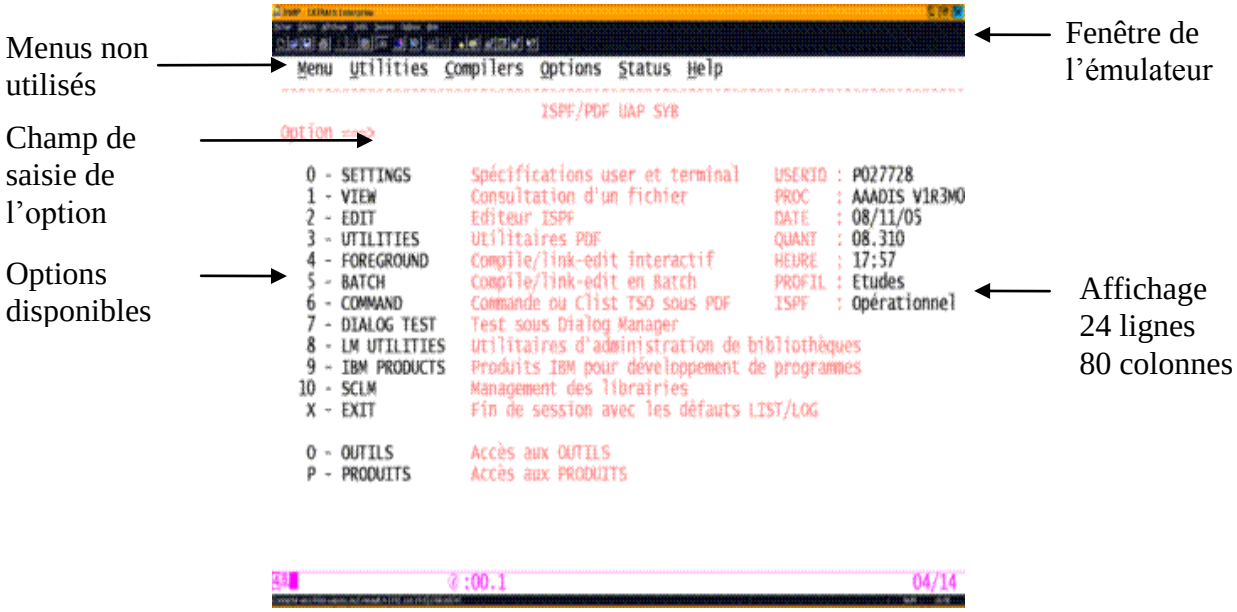


Figure 8 : Ecran d'accueil de l'environnement Mainframe

2.1.3 Les fichiers dans l'environnement Mainframe

Le stockage des données est aussi particulier. Il n'y a pas de notion de répertoires. Il existe différents types de fichiers. Deux types nous intéressent ici. Le premier type correspond aux fichiers simples. Ils sont semblables à ce qu'on peut trouver dans l'environnement Windows/UNIX. En général, on y trouve les données de traitement. Le deuxième type correspond aux bibliothèques qui sont sans équivalent dans l'environnement Windows/UNIX. Les bibliothèques sont des fichiers partitionnés c'est-à-dire découpés en blocs appelés membres. Les membres se manipulent comme des fichiers simples : ils ont un nom et on peut les créer, éditer, supprimer. Tous les membres d'une même bibliothèque possèdent le même format qui est défini au moment de la création de la bibliothèque. Dans les bibliothèques sont stockés les objets de programmation : fichiers sources, exécutables, paramètres. LDD+ repose sur des bibliothèques.

Dans les Figure 9 et Figure 10, je présente un exemple d'affichage de bibliothèques et de membres pour utilisation. Dans la Figure 9, je demande l'affichage de toutes les bibliothèques dont le nom commence par DEV.SCRM.SOURCE.

Dans cette liste de bibliothèques, je choisis la bibliothèque DEV.SCRM.SOURCE.COB pour visualiser son contenu. La liste des membres de la bibliothèque DEV.SCRM.SOURCE.COB s'affiche dans la Figure 10. On y voit des informations sur les membres.

La colonne Name donne le nom du membre qui est sur 8 caractères maximum. La colonne Size donne le nombre de lignes dans le membre. La colonne Created donne la date de création du membre. La colonne Changed donne la date et l'heure de la dernière modification. La colonne ID donne l'identifiant de la personne à l'origine de la dernière modification.

La codification de nommage des bibliothèques est expliquée plus loin dans le paragraphe 2.2.1. Chacun des membres qu'on voit correspond au fichier source d'un programme COBOL. On les manipule comme des fichiers en les créant, en les éditant et en les supprimant. C'est pour cela que dans la suite je parlerai de membre comme de fichier.

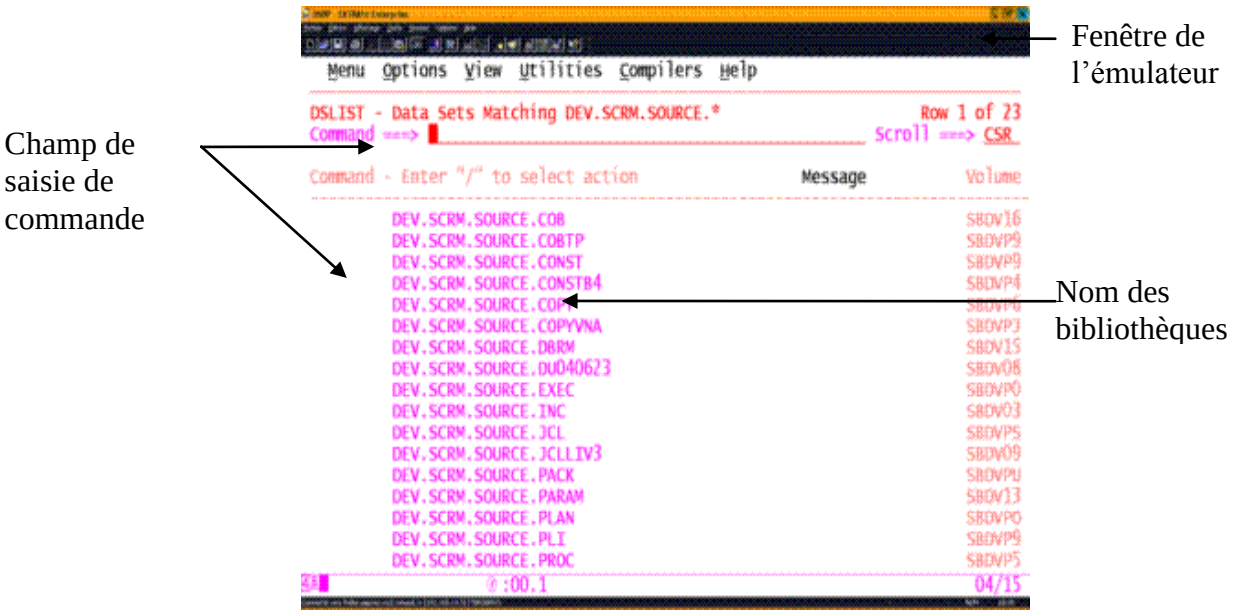


Figure 9 : Liste des bibliothèques

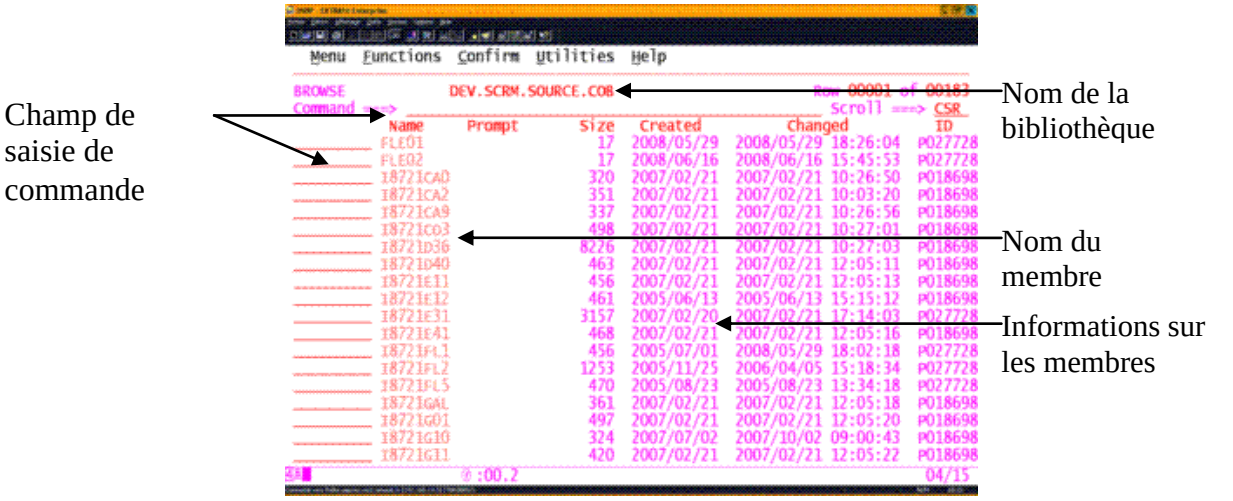


Figure 10 : Liste de membres dans la bibliothèque DEV.SCRM.SOURCE.COB.

2.2 Le prédécesseur LDD

2.2.1 Les principes de LDD

Chez Renault l'outil de travail du développeur dans l'environnement Mainframe était LDD avant le déploiement de LDD+. LDD (Lot De Développement) est un outil développé en interne par Renault. Il a été conçu et réalisé par Gérard Besson en CLIST et REXX. CLIST et REXX sont des langages spécifiques à l'environnement Mainframe pour développer des outils. LDD permet d'éditer, de compiler et de transférer les fichiers sources. L'intérêt de LDD est la capacité à pouvoir regrouper les fichiers sources spécifiques à une modification dans des lots quelque soit le type des fichiers sources.

Dans l'environnement Mainframe, les fichiers sources gérés par LDD sont stockés dans des bibliothèques dont le nom est codifié. Exemple : la bibliothèque DEV.SCRM.SOURCE.COB contient les fichiers sources de type COBOL. La bibliothèque DEV.SCRM.SOURCE.SQL contient les fichiers sources de type SQL. La codification de nommage des bibliothèques se présente ainsi :

- l'identifiant DEV indique que la bibliothèque est utilisée pour le développement. L'identifiant RE7 indique que la bibliothèque fait partie des bibliothèques miroirs de recette (voir 2.2.3). L'identifiant OPE indique que la bibliothèque fait partie des bibliothèques miroirs de production (voir 2.2.3).
- l'identifiant CRM indique le SIA (Système d'Information Autonome). Le SIA est un identifiant de 3 caractères. En général, le SIA identifie un projet informatique qui utilise l'environnement Mainframe (voir 3.6). Le S présent avant le SIA est ajouté par convention.
- L'identifiant SOURCE indique que la bibliothèque contient des fichiers sources. L'identifiant LOAD aurait indiqué que la bibliothèque contient des fichiers exécutables.
- l'identifiant COB et SQL indique le type du fichier source contenu dans la bibliothèque.

Il existe différents types de fichiers sources. Les principaux types sont :

- COB pour les programmes en langage cobol
- COPY pour les fichiers sources réutilisables dans plusieurs programmes en langage cobol
- PLI pour les programmes en langage PL/1
- INC pour les fichiers sources réutilisables dans plusieurs programmes en langage PL/1
- SQL pour les fichiers sources qui utilisent le SGBD DB2 (voir 3.11)
- PACK pour les fichiers sources qui contiennent les paramètres d'accès au SGBD DB2
- CONST pour les paramètres de traitement

Le type est repris en suffixe dans le nom des bibliothèques. Ainsi la bibliothèque DEV.SCRM.SOURCE.COB contient des modules de types COB.

L'environnement Mainframe est mono tâche : un développeur ne peut travailler que sur un membre à la fois. De plus, si le développeur travaille par exemple sur un fichier source COBOL puis veut travailler sur un fichier source SQL, il doit passer de la bibliothèque des sources COBOL vers la bibliothèque des sources SQL.

On voit donc que sans LDD, le développeur serait obligé de passer de bibliothèque en bibliothèque. De plus dans ce système de stockage par bibliothèque de type, on est incapable de regrouper tous les modules modifiés pour demande donnée.

Grâce à LDD, on peut créer un lot, c'est à dire un regroupement logique, qui ne va contenir que les fichiers sources qui correspondent à la modification. Exemple : si une modification demande la modification d'un fichier source CONST const1 et d'un fichier source COB cob2, on trouvera dans le lot les fichiers sources const1 et cob2 sans être gênés par les autres sources.

Pour cela LDD utilise des pointeurs vers les bibliothèques de sources. C'est pour cela qu'on parle de regroupement logique car les fichiers sources de nature différentes d'un même lot ne sont pas stockés physiquement dans la même bibliothèque.

La Figure 11 présente le principe de LDD. Il y a le lot FLE01 qui contient deux fichiers sources const1 de type CONST et cob2 de type COB. Les fichiers sources const1 et cob2 sont stockés physiquement dans leur bibliothèques respectives DEV.SCRM.SOURCE.CONST et DEV.SCRM.SOURCE.COB.

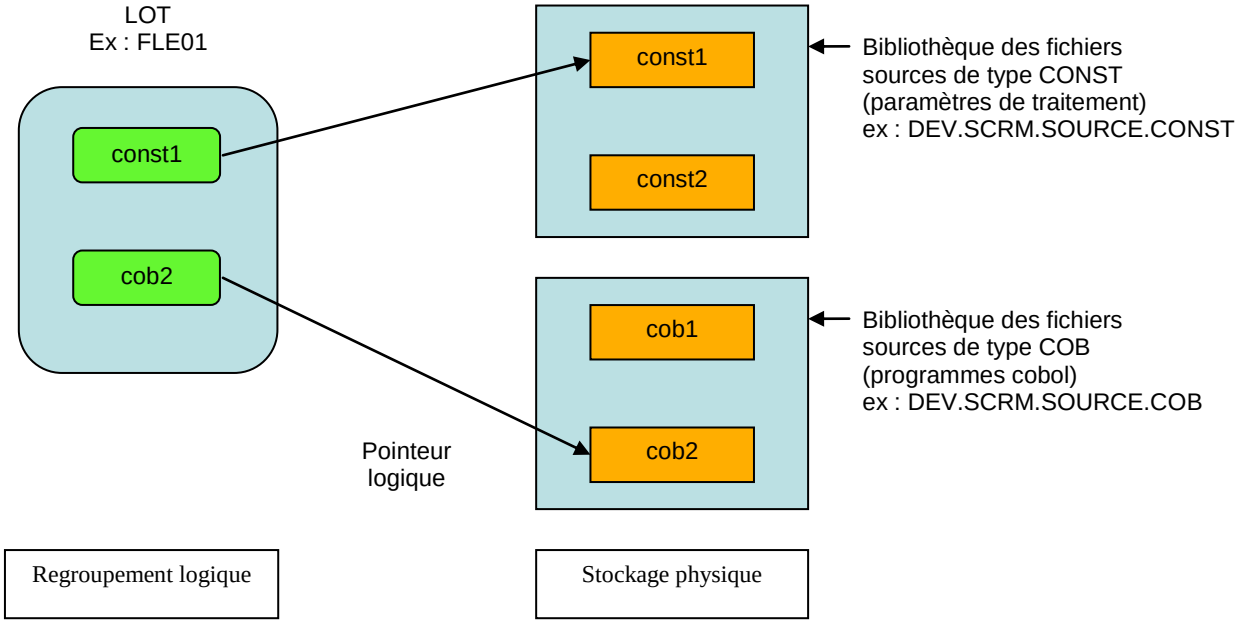


Figure 11 : Principe d'un lot

Dans un lot on peut travailler sur des objets de types différents indépendamment de leur stockage physique.

2.2.2 L'interface de LDD

La Figure 12 est une capture d'écran qui montre le contenu d'un lot affiché par l'interface de LDD. Dans cet exemple : nous avons le lot ANO716B qui contient trois fichiers sources de type PLI, un fichier source de type JCL, un fichier source de type SQL. Pour faciliter la lecture, j'ai retranscrit son contenu.

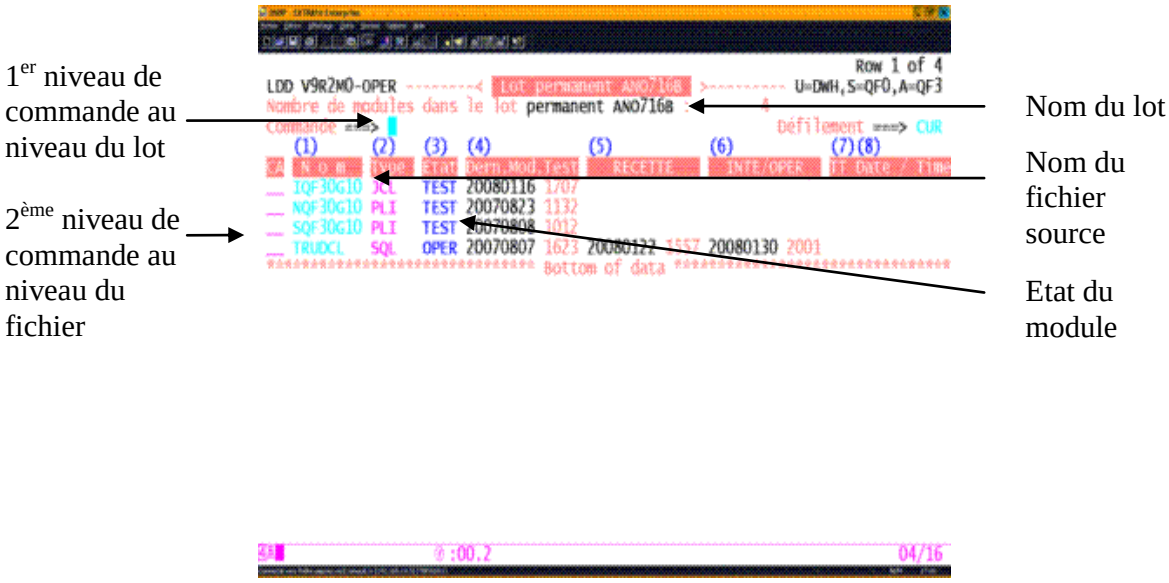
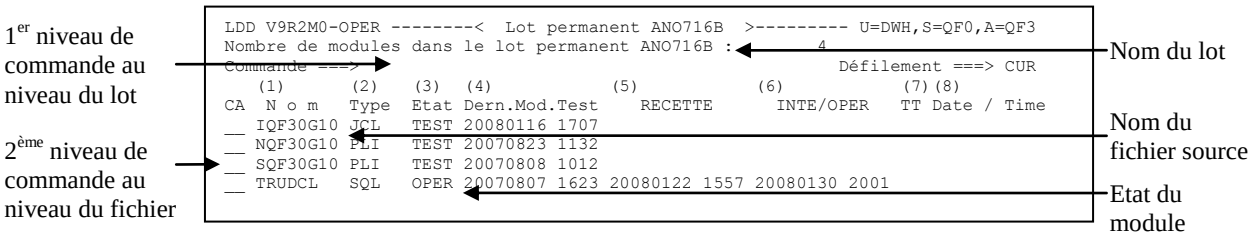


Figure 12 : Interface de LDD



Retranscription de la Figure 12.

Il existe deux niveaux de commande. Dans le premier niveau de commande, dans le champ Commande, on saisit les commandes qui concernent le lot en entier. Par exemple pour sauvegarder le lot, on saisira la commande SAVE. Dans le deuxième niveau de commande, dans le champ CA devant le nom du fichier source, on saisit les commandes qui concernent le fichier source seul. Par exemple pour sélectionner un fichier source pour l'éditer, on saisira la commande S.

En dessous du champ Commande on a la liste des fichiers sources du lot. Comme l'affichage est limité à 24 lignes et que les 6 premières lignes sont déjà prises, on ne peut afficher les fichiers sources que par page de 18 fichiers sources. Pour accéder aux autres fichiers, il faut passer de page en page. Dans l'interface de LDD, les informations sont présentées en colonne :

- la colonne CA permet de saisir les commandes de second niveau qui agissent sur le fichier source seul. Par exemple pour sélectionner un fichier source pour l'éditer, on saisira la commande S.
- la colonne NOM donne le nom du fichier source
- la colonne TYPE donne le type du fichier source. PLI correspond à un programme en PL/1. JCL correspond à un script de traitement. SQL correspond à un fichier source qui utilise la base de données DB2. Ce type est repris comme suffixe dans le nom de la bibliothèque de stockage (voir 2.2.1)
- la colonne ETAT indique si le fichier source a été transféré vers un autre environnement. L'état TEST indique que le fichier est dans l'environnement de développement et que le développeur peut le modifier. L'état RECE indique que le fichier a été transféré dans l'environnement de recette pour être validé. L'état OPER indique que le fichier a été transféré dans l'environnement de production pour les traitements de production.

- la colonne Dernière modification indique la date et l'heure de la dernière modification en TEST.
- la colonne RECETTE indique la date du dernier transfert en environnement de recette.
- la colonne INTE/OPER indique la date du dernier transfert en environnement de production.
- la colonne « TT Date / Time » indique si elle est renseignée qu'un transfert est en cours.

2.2.3 L'architecture de LDD

Je présenterai ici l'architecture puis le fonctionnement de LDD pour montrer qu'il est inadapté à la GCL. Elle sera aussi utile lorsque j'expliquerai le fonctionnement de LDD+. La Figure 13 présente l'architecture de LDD.

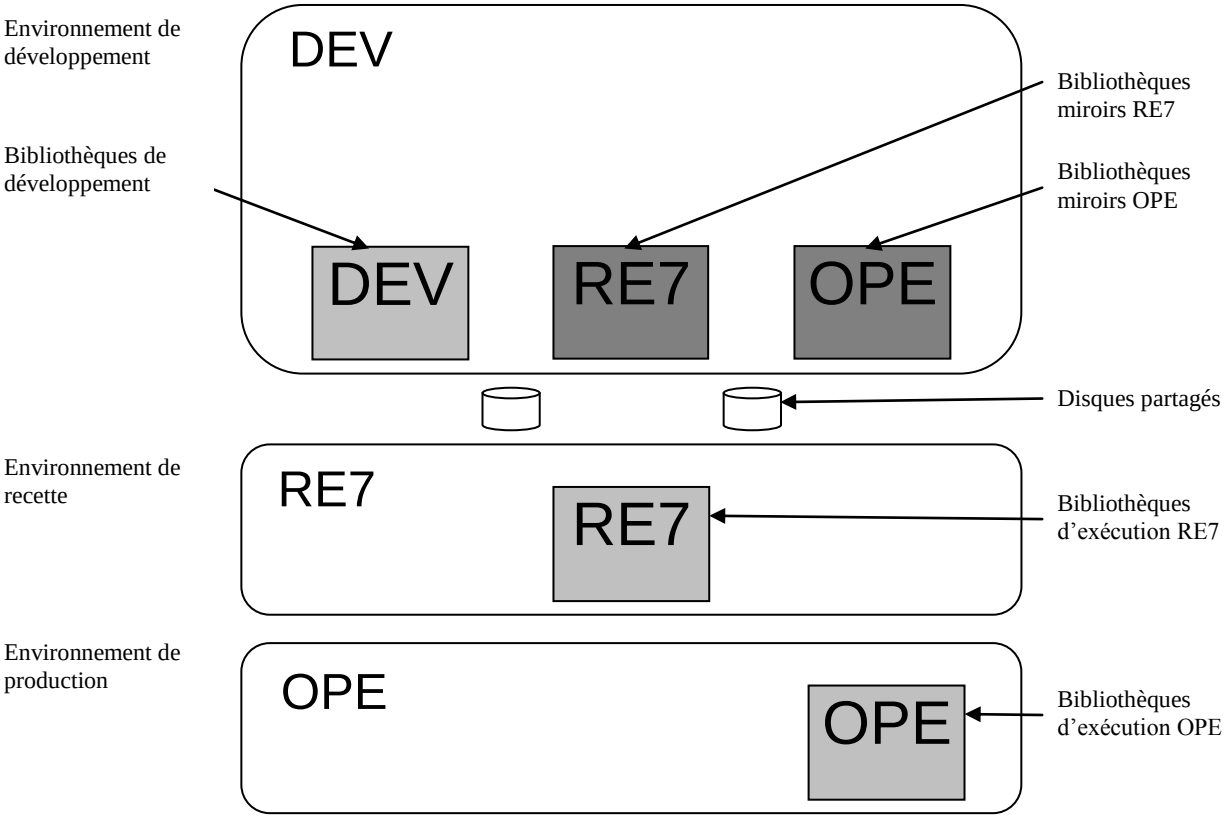


Figure 13 : Architecture de LDD

DEV correspond à l'environnement de développement,
RE7 à l'environnement de recette, OPE à l'environnement de production

Les 3 grands rectangles aux coins arrondis représentent les trois environnements : DEV l'environnement de développement, RE7 l'environnement de recette, OPE l'environnement de production.

Dans l'environnement DEV, le rectangle DEV en gris clair représente les bibliothèques de développement dans lesquelles travaillent les développeurs informatique. Les rectangles RE7 et OPE en gris foncé représentent les bibliothèques miroirs RE7 et OPE. Elles contiennent tous les fichiers sources correspondant aux exécutables présents dans les bibliothèques d'exécution RE7 et OPE. On parle de bibliothèques miroirs car elles permettent de connaître ce qu'il y a dans les environnements RE7 et OPE à partir de l'environnement DEV.

Dans l'environnement RE7, le rectangle RE7 en gris clair représente les bibliothèques d'exécution de l'environnement RE7. Elles contiennent les fichiers nécessaires aux traitements de recette.

Dans l'environnement OPE, le rectangle OPE en gris clair représente les bibliothèques d'exécution de l'environnement OPE. Elles contiennent les fichiers nécessaires aux traitements de production

Les cylindres hors rectangles aux coins arrondis représentent des disques partagés qui sont utilisés pour les transferts de fichiers entre les différents environnements. Ils sont représentés hors des rectangles aux coins arrondis car ils sont sur des machines en relation avec tous les environnements.

2.2.4 Les transferts sous LDD

Les transferts des fichiers sources sont faits par l'équipe en charge de leur modification. Une fois que le développeur Mainframe a terminé son travail de développement, il va procéder à un transfert DEV-RE7. Il déclenche ce transfert directement à partir de l'interface de LDD avec la commande de second niveau R.

Une fois les traitements de transferts DEV-RE7 déclenchés, ceux-ci procèdent à la recopie les fichiers sources dans les bibliothèques miroirs RE7 et au transfert des exécutables dans l'environnement RE7. Les traitements peuvent alors s'exécuter en environnement de recette pour être contrôlés par les commanditaires. C'est la phase de recette. Le détail des traitements de transfert sont présentés dans la Figure 14.

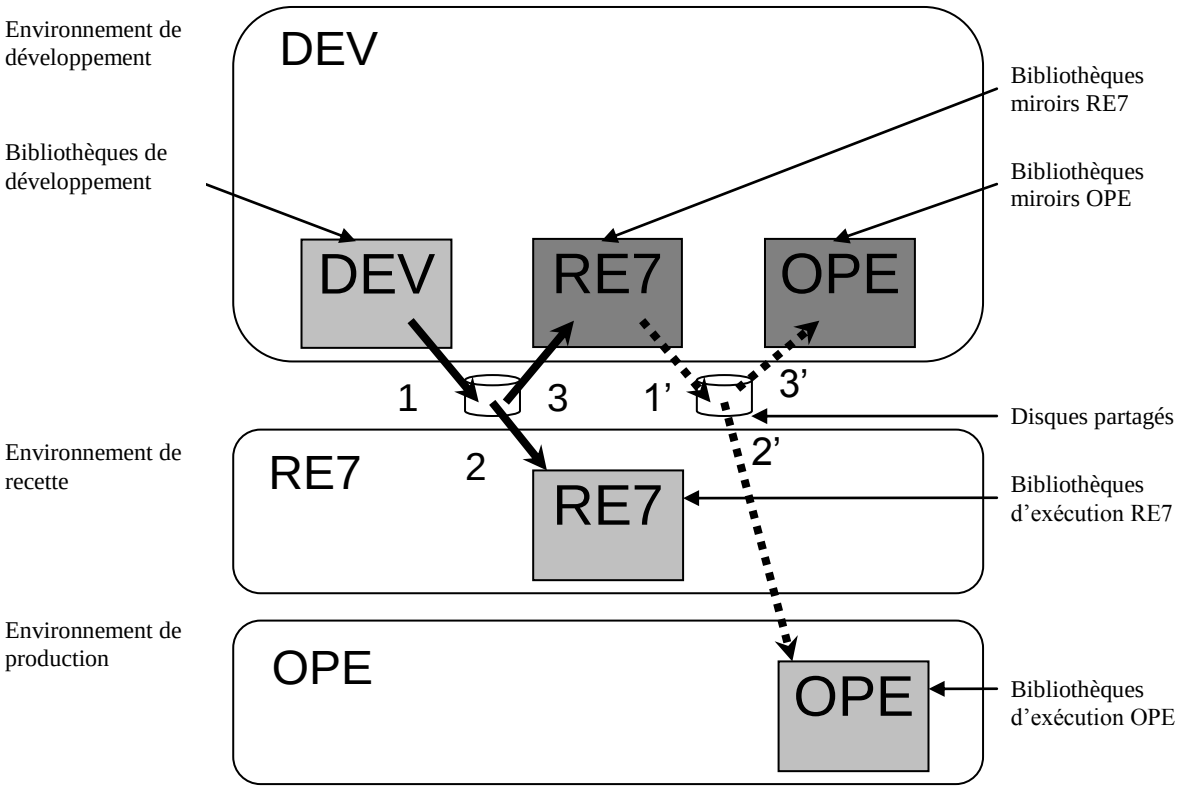


Figure 14 : Les étapes des transferts dans LDD

1-2-3 sont les 3 étapes du transfert DEV-RE7. 1'-2'-3' sont les 3 étapes du transfert RE7-OPE.

Le transfert DEV-RE7 se fait en trois étapes :

- l'étape 1 consiste à recompiler et à recopier les fichiers sources et leurs exécutables dans le disque partagé.
- l'étape 2 consiste à recopier les exécutables dans les bibliothèques d'exécution RE7.
- l'étape 3 consiste à recopier les sources et les exécutables dans les bibliothèques miroirs. Si l'étape 3 s'est déroulée sans problème alors l'état du fichier source transféré passe de TEST à RECE dans LDD.

Quand la phase de recette est terminée, les fichiers sources et les exécutables sont transférés dans l'environnement OPE pour les traitements en production. C'est la phase de mise en production. Une fois que l'utilisateur final donne son accord, l'équipe de développement déclenche les traitements de transfert RE7-OPE avec la commande de second niveau O. Comme pour le transfert DEV-RE7, le transfert RE7-OPE se passe aussi en 3 étapes semblables à celles du transfert DEV-RE7. L'état du fichier source transféré passe alors de RECE à OPER dans LDD.

A l'état RECE ou OPER, le fichier source ne peut pas être modifié. Pour qu'il soit modifiable à nouveau, il faut le faire repasser à l'état TEST soit par le retour RE7-DEV s'il est à l'état RECE soit par le retour OPE-DEV s'il est à l'état OPER. Le retour consiste à changer l'état du fichier source, ce qui ne correspond pas à un véritable transfert. La Figure 15 montre les différents états que peut prendre un fichier source.

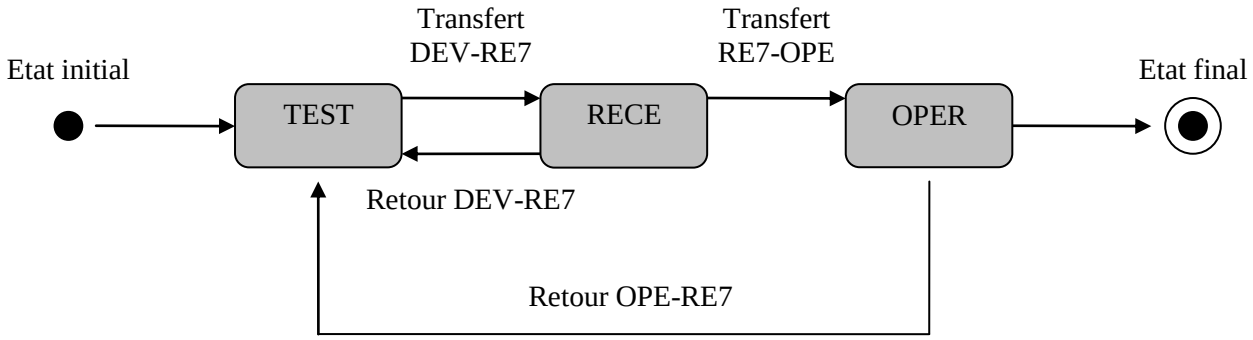


Figure 15 : Diagramme état-transition d'un fichier source

On voit les 3 états LDD : TEST, RECE, OPER.
Les Retours DEV-RE7 et OPE-RE7 permettent de retrouver l'état TEST.
L'état TEST est le seul état qui permet les modifications du fichier source.

2.2.5 L'inadaptation de LDD pour la GCL

LDD est incompatible avec la GCL telle qu'on l'a présentée précédemment dans les objectifs de la GCL (voir 1.4.1) pour deux raisons :

- On voit dans la Figure 14 que LDD procède par recopie de fichier et que ce soit pendant le travail de développement ou après chaque transfert, la nouvelle version d'un fichier remplaçait la précédente. Dans le jargon informatique, on dit que la dernière version « écrase » la précédente. Donc la version précédente d'un fichier était à chaque fois perdue sauf en cas de sauvegarde manuelle. De ce fait, l'outil LDD ne permet pas la GCL car il n'y a pas de référentiel qui stocke les différentes versions d'un fichier.

- L'absence de version entraine l'absence d'historique sur les versions. Pour un jour donné, on était incapable de savoir ce qui avait été modifié ou transféré ni de déterminer ou de reconstituer l'état d'un projet car il n'y a pas de baseline pour stocker l'historique des versions.

En l'absence de référentiel et de baseline, LDD ne peut par gérer la GCL. C'est pourquoi dans le cadre du déploiement de la démarche CMMI, on ne pouvait pas garder LDD.

2.3 Présentation de l'outil LDD+

2.3.1 L'origine de LDD+

Comme je l'ai expliqué dans le choix des outils de GCL (voir 1.4.5), Renault a décidé de modifier LDD afin qu'il incorpore les fonctions nécessaires à la GCL :

- les fonctions de gestion des versions de fichiers appelées aussi fonctions de versionning,
- les fonctions de traçabilité pour suivre les modifications dans le temps,
- les fonctions de verrouillage pour gérer les développements concurrents.

On a ainsi obtenu l'outil LDD+. En 2006 une première version de LDD+ a été élaborée. Cette version V0 (voir 3.3) reposait sur l'outil SCLM (Software Configuration and Library Manager) un gestionnaire de version de fichiers, fourni en standard avec l'environnement Mainframe. Mais des tests de performance effectués en Février 2007 ont montré qu'on ne pouvait utiliser SCLM. C'est pourquoi, il a été décidé de réécrire LDD+. C'est cette version V1 (voir 3.3) de LDD+ que j'ai été chargé de tester et valider, puis de déployer.

2.3.2 L'architecture de LDD+

L'architecture de LDD+ par rapport à celle de LDD est présentée dans la Figure 16. Les fichiers et bibliothèques que LDD+ possède en plus par rapport à LDD sont stockés dans l'environnement DEV. Ce sont les bibliothèques de versionning et les fichiers baseline représentées par des rectangles avec le motif en pointillé. Les bibliothèques de versionning contiennent les fichiers versionning. Comme je l'expliquerai plus loin dans la gestion des versions dans LDD+, les fichiers versionning stockent les différentes versions d'un fichier. Les fichiers baseline enregistrent tous les transferts d'un environnement à un autre. Pour chaque module transféré, on y enregistre son numéro de version. Les fichiers baseline permettent de reconstituer la photo du projet à un moment donné. La notion de baseline est présentée dans les principes du processus GCL (voir 1.4.2).

Il y a trois ensembles de bibliothèques de versionning, une par environnement. Par exemple, les bibliothèques de versionning DEV contiennent les versions des fichiers sources DEV. Il n'y a que deux fichiers baseline, un pour l'environnement RE7, un pour l'environnement OPE. Il n'y a pas de fichier baseline pour l'environnement DEV pour deux raisons :

- il n'y a pas de transfert vers l'environnement DEV,
- en environnement de DEV, un fichier source peut être en cours de modification sans qu'il y ait de version identifiée et sans version identifiée, il ne peut pas y avoir de GCL.

Dans LDD+ les processus de transferts sont constitués de 4 étapes au lieu de 3. Les 3 premières étapes sont identiques à celles qui existent dans LDD. La quatrième étape consiste à recopier les fichiers versionning dans les bibliothèques de versionning et d'enregistrer le transfert dans le fichier baseline. Cette étape sera présentée plus en détail dans l'exemple d'utilisation des fonctions de versionning (voir 2.3.6). La gestion des états TEST-RECE-OPER (voir Figure 15) reste la même.

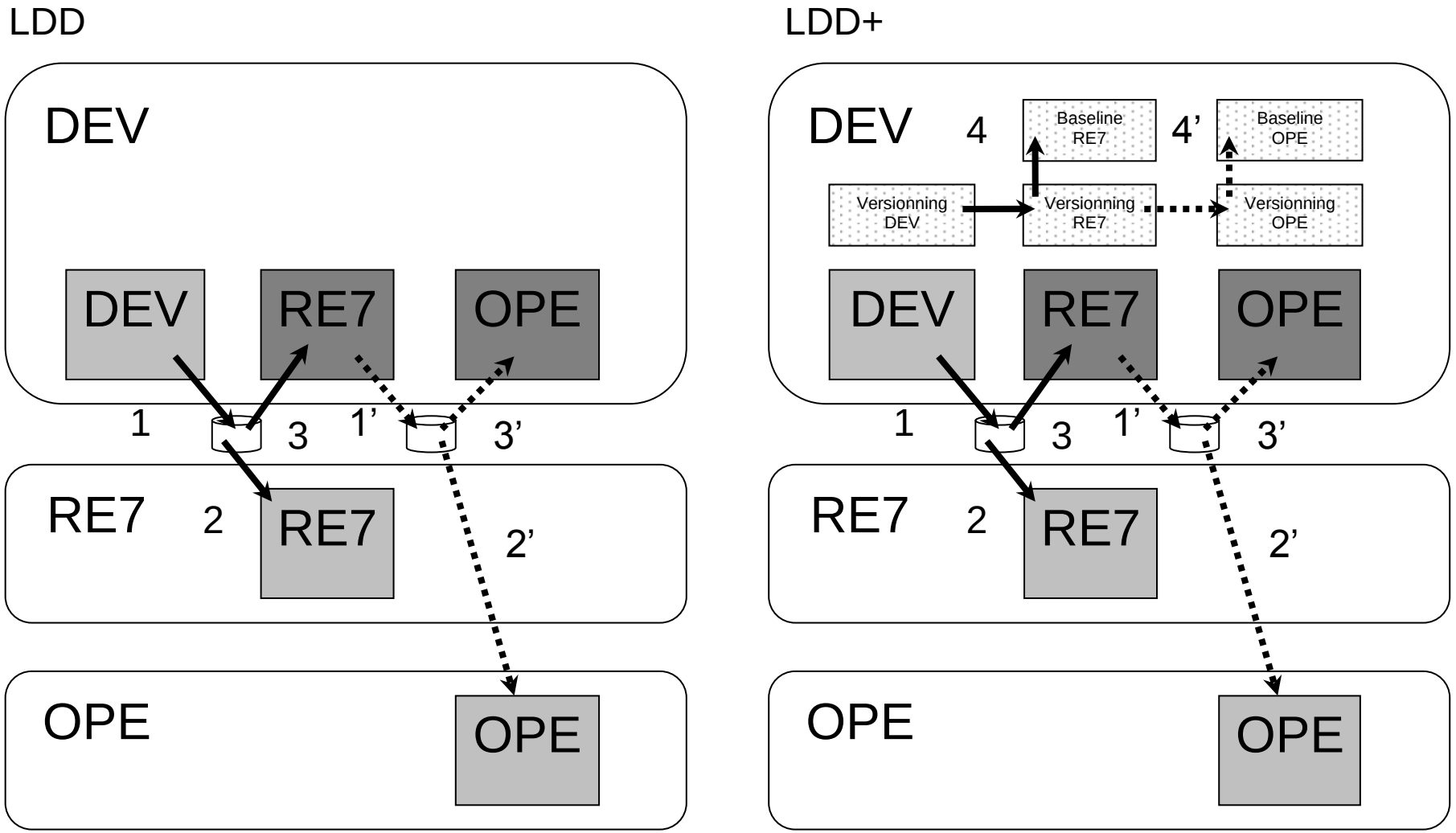


Figure 16 : Comparaison de l'architecture LDD/LDD+

Par rapport à LDD, LDD+ possède en plus des bibliothèques de versionning et des fichiers baseline.
Pour les transferts, il y a une étape supplémentaire pour mettre à jour les fichiers versionning et les fichiers baseline.

2.3.3 Les commandes LDD+

LDD+ est une évolution de LDD qui permet d'appliquer la GCL. On a ajouté à LDD :

- des fonctions de versionning et en particulier :
  - la fonction Checkin pour enregistrer une version
  - la fonction Checkout pour récupérer une version
- des fonctions de traçabilité pour suivre les modifications dans le temps,
- des fonctions de verrouillage pour gérer les développements concurrents.

L'interface de LDD a été conservée, ce qui a rassuré les utilisateurs. La plupart des commandes utilisables sous LDD restent utilisables sous LDD+. LDD+ apporte des commandes supplémentaires liées à la GCL. Elles ne sont pas très nombreuses et on peut se rappeler des principales grâce au mnémotechnique VOLUBICE. Ces commandes sont :

- ZV pour visualiser les versions d'un fichier source
- ZO pour faire un Checkout
- ZL pour verrouiller un fichier source avec un verrou Lock
- ZU pour lever un verrou Lock
- ZB pour visualiser la baseline pour le suivi des transferts
- ZI pour faire un Checkin
- ZC pour savoir si un fichier source est verrouillé
- ZE pour extraire les versions d'un fichier source

En reprenant la deuxième lettre des commandes dans l'ordre ci-dessus, on reconstitue le mot VOLUBICE. A cela s'ajoute deux autres commandes :

- QT pour avoir l'historique des opérations.
- UNLOCK pour déverrouiller un fichier source sans qu'on soit celui qui a posé le verrou

Six commandes présentes dans LDD ont été bloquées dans LDD+. Elles étaient très peu utilisées et surtout elles n'étaient pas conformes avec la gestion de configuration. C'étaient des commandes qui déclenchaient des transferts spéciaux entre DEV et OPE. Leur spécificité est que les fichiers étaient transférés dans des bibliothèques spéciales qui n'étaient pas les bibliothèques officielles de production. De ce fait, ces bibliothèques spéciales ne pouvaient faire partie du système géré en configuration car dans ce cas un fichier source pouvait se retrouver en deux exemplaires alors que la gestion de configuration impose un exemplaire unique.

Dans une utilisation courante, la commande la plus importante à retenir est ZI pour le Checkin. Nous allons maintenant détailler les différentes fonctions. Cela nous permettra d'étudier comment LDD+ réalise la GCL.

2.3.4 Les fonctions de versionning

La gestion des versions est la partie la plus importante de la GCL. Il faut que l'outil de GCL soit capable d'enregistrer les différentes versions d'un fichier source et de les restituer. Il faut aussi que l'outil soit capable de « marquer » chacune de ces versions, c'est-à-dire de lier chaque version à une étiquette où on enregistrera la justification de la modification. Par exemple, il faut qu'on puisse dire que la version 1.04 du fichier source de type COB cob1 correspond à la correction de l'anomalie 133. Les fonctions de versionning sont :

- la fonction Checkin pour enregistrer une version (Commande ZI)
- la fonction Checkout pour récupérer une version (Commande ZO)
- la fonction Visualisation des versions (Commande ZV)
- la fonction Extraction des versions (Commande ZE)

En général, nous avons pu constater que seule la fonction Checkin est vraiment utilisée. L'environnement Mainframe étant centralisé, un fichier est commun à tous les développeurs. Donc ceux-ci ne travaillent en général qu'à partir de la dernière version du fichier.

2.3.4.1 La fonction Checkin

La fonction Checkin (commande ZI) permet d'enregistrer une version d'un fichier source. Au moment de l'enregistrement, l'outil demande à l'utilisateur de fournir l'étiquette c'est-à-dire la justification de cette version, ex : correction de l'anomalie X, mise en place de l'évolution Y. Grâce à cette étiquette, nous pouvons remonter à la demande à l'origine de la modification. Dans LDD+, toutes les versions d'un fichier source sont enregistrées les unes après les autres dans un fichier versionning associé au fichier source.

La Figure 17 présente la fonction Checkin. Pour enregistrer la nouvelle version, LDD+ crée une nouvelle version du fichier versionning. LDD+ écrit d'abord un enregistrement entête correspondant à la nouvelle version. Dans l'enregistrement entête, il y a le numéro de version (incrément de 1 par rapport au numéro de la version précédente) et d'autres informations comme l'identifiant de la personne qui a procédé au Checkin, la date et l'heure de l'enregistrement, et l'étiquette de la version. Puis LDD+ recopie le contenu du fichier source et enfin LDD+ recopie le contenu de l'ancien fichier versionning. Ainsi la version la plus récente se trouve au début du fichier versionning et la partie la plus ancienne en fin.

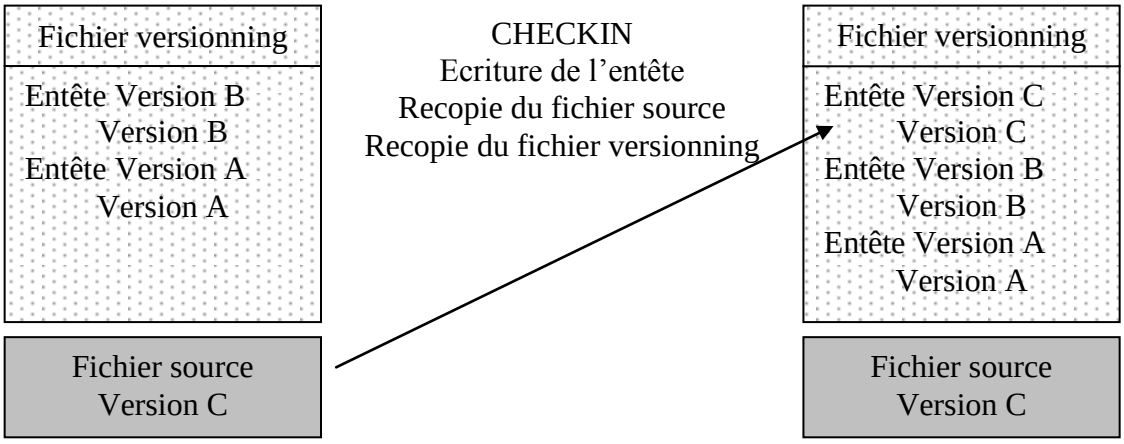


Figure 17 : Principe de la fonction Checkin

La Figure 18 montre le contenu d'un fichier versionning. Il s'agit du fichier versionning du programme P8721Q04 de type COB. La première ligne contient l'enregistrement entête. Cet enregistrement entête est suivi de la copie du programme. L'enregistrement entête est retranscrit dans la Figure 19.

La dernière version enregistrée est la version 01.03. Elle a été enregistrée le 12 juillet 2007 à 17h49 par P027728. Cette version comporte 355 lignes et son format est fixe et de 80 caractères par lignes (« FB,80 »). L'étiquette de la version indique que cette version est issue de l'anomalie 115120 enregistrée dans l'outil CHIPRE. CHIPRE est l'outil qui centralise les anomalies. QC (Quality Center) aurait indiqué l'outil qui centralise les tests. « Ano RCI » est la description de l'anomalie.

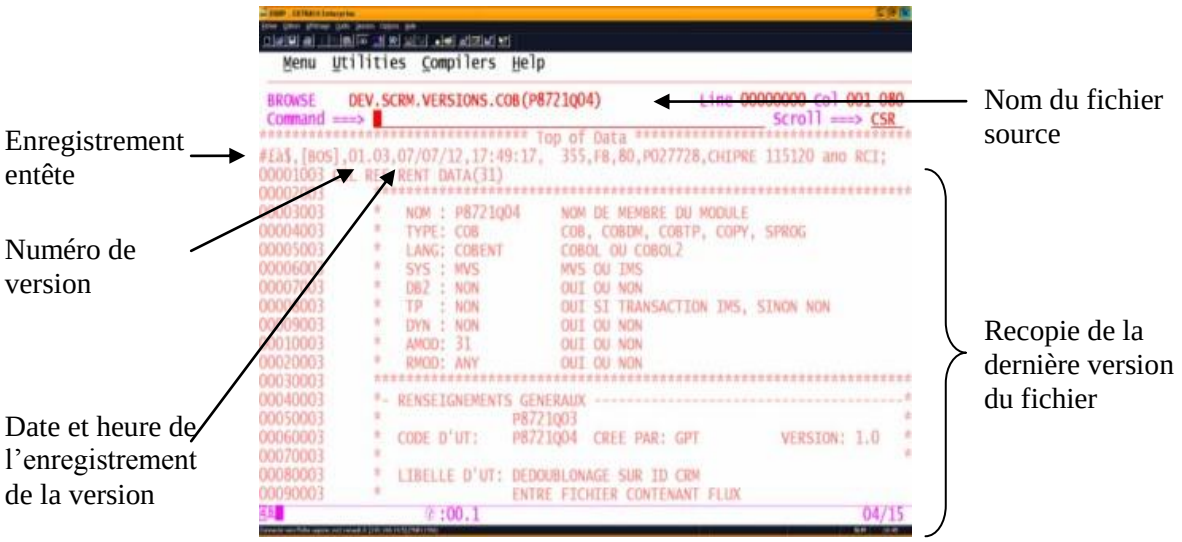


Figure 18 : Contenu d'un fichier versionning

La version la plus récente du programme COB P8721Q04 est la version 1.03.

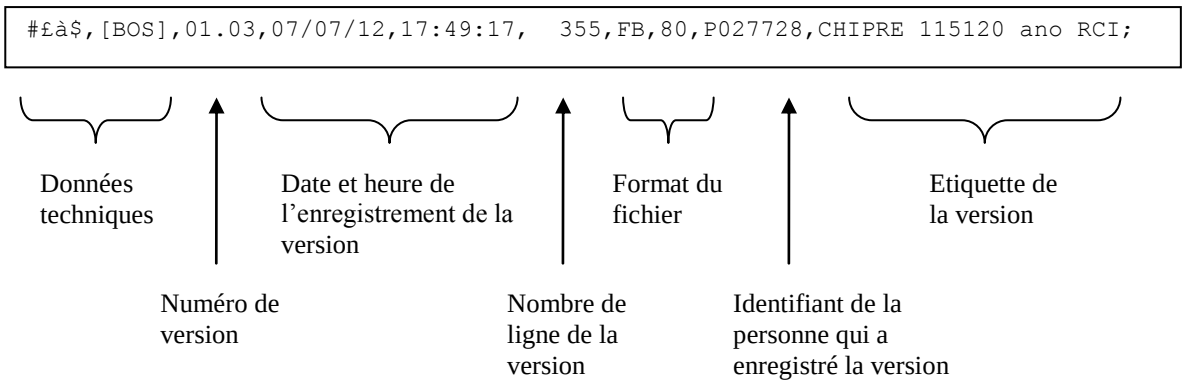


Figure 19 : Transcription d'un enregistrement entête

Contrairement à d'autres outils, avec LDD+ on n'est pas obligé d'enregistrer une version pour pouvoir compiler et exécuter pour tester le fichier source. Il est conseillé de n'enregistrer une version que lorsqu'on est sûr de ses modifications. Cela évite une trop grande prolifération des versions non significatives. En général, on enregistre une version quand on est prêt à transférer le fichier source en recette ou en production.

La fonction Checkin ne met à jour que le fichier versionning DEV. Nous verrons son utilisation dans l'exemple d'utilisation des fonctions de versionning (voir 2.3.6). Pour s'assurer que l'enregistrement des versions se fait, il est obligatoire de faire un Checkin avant tout transfert. Ainsi, on est sûr qu'en recette et en production, on ne trouve que des versions enregistrées.

2.3.4.2 La fonction Checkout

La fonction Checkout (Commande ZO) est présentée dans la Figure 20. Elle permet de récupérer une ancienne version d'un fichier source à partir du fichier des versions. A l'usage, c'est une fonction qui est peu utilisée car étant donnée la nature centralisée de l'environnement Mainframe, les développeurs travaillent en général à partir de la dernière version. Quand le développeur procède à un Checkout, LDD+ lui affiche la liste des versions. Il sélectionne la version qui l'intéresse et LDD+ la recopie dans le fichier source. Il faut faire attention quand on fait un Checkout, car le contenu du fichier source avant Checkout est perdu s'il n'a pas été sauvegardé par un Checkin.

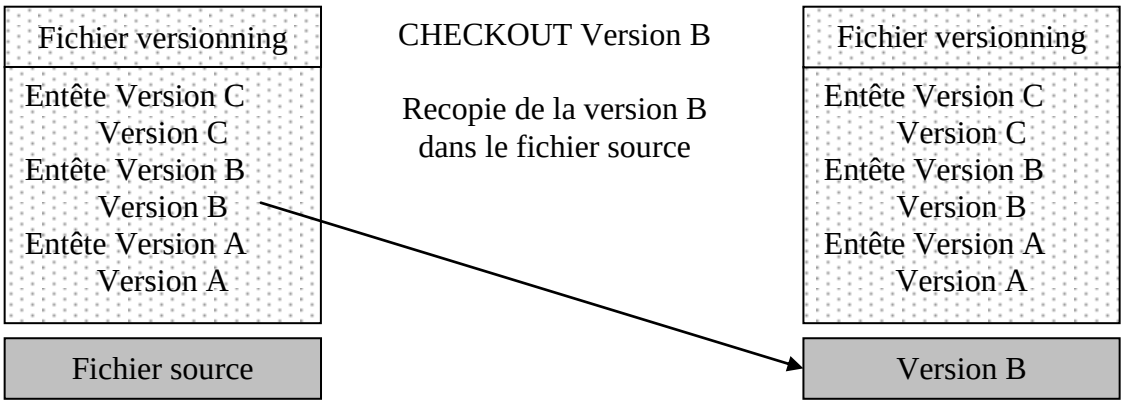


Figure 20 : Principe de la fonction Checkout
Ici le développeur a fait un Checkout de la version B.

2.3.4.3 La fonction Visualisation des versions

La fonction Visualisation des versions (Commande ZV) permet de visualiser toutes les versions qui ont été enregistrées par un Checkin. La visualisation par la commande ZV du fichier versionning est très rudimentaire. On ne peut que paginer pour aller d'une page à l'autre. En plus, cette visualisation ne permet pas de recherche de chaîne de caractères ce qui limite fortement son intérêt. A cause de ces limites et pour aussi pouvoir comparer différentes versions d'un fichier source, on a développé la fonction Extraction des versions.

2.3.4.4 La fonction Extraction des versions

La fonction Extraction des versions (Commande ZE) permet d'extraire toutes les versions qui ont été enregistrées par un Checkin Avec l'avancement du projet, il est apparu que les utilisateurs pouvaient avoir besoin de pouvoir comparer différentes versions d'un fichier source, opération qu'on ne pouvait pas faire avec la commande ZV. C'est pour cela qu'a été mise au point la commande ZE. La commande ZE extrait toutes les versions d'un fichier source à partir de son fichier versionning. La commande ZE produit autant de fichiers que de versions dans une bibliothèque temporaire. On peut alors manipuler normalement ses fichiers. On peut y faire des recherches de chaînes de caractères ou utiliser les outils de comparaison de fichiers pour connaître les différences entre 2 versions. La Figure 21 présente la fonction Extraction des versions.

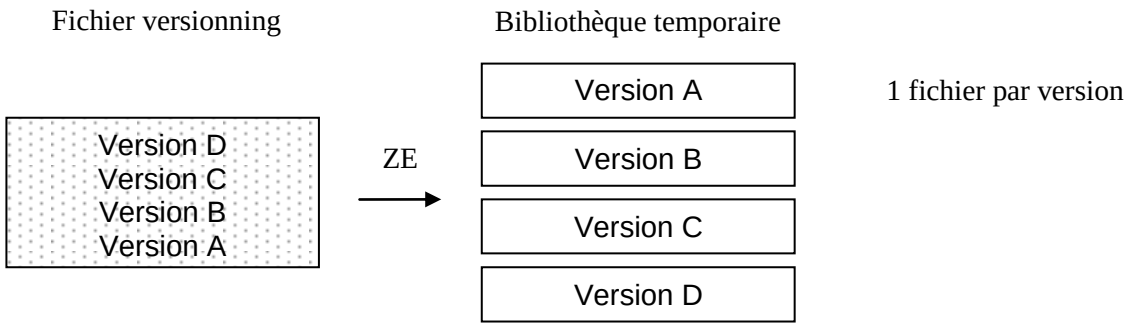


Figure 21 : Principe de la fonction Extraction des versions

2.3.5 Les fonctions de traçabilité

Les fonctions de traçabilité sont indispensables à la GCL car elles gèrent l'historique des versions. Grâce à ces informations, la GCL permet suivre l'évolution du projet et de reconstituer si nécessaire une version du projet. Dans LDD+, il y a deux fonctions de traçabilité :

- le suivi des transferts (Commande ZB)
- le suivi des opérations (Commande QT)

2.3.5.1 Les blocs de transferts de fichiers

Pour mieux expliquer le suivi des transferts, je vais d'abord expliquer les blocs de transferts de fichiers dans LDD+. Avec LDD+, les transferts se font par bloc. Lorsque le développeur décide de transférer des fichiers sources, il sélectionne dans le lot tous les éléments qu'il veut transférer. Ceci constitue un bloc de transferts. La Figure 22 montre la constitution d'un bloc de transfert.

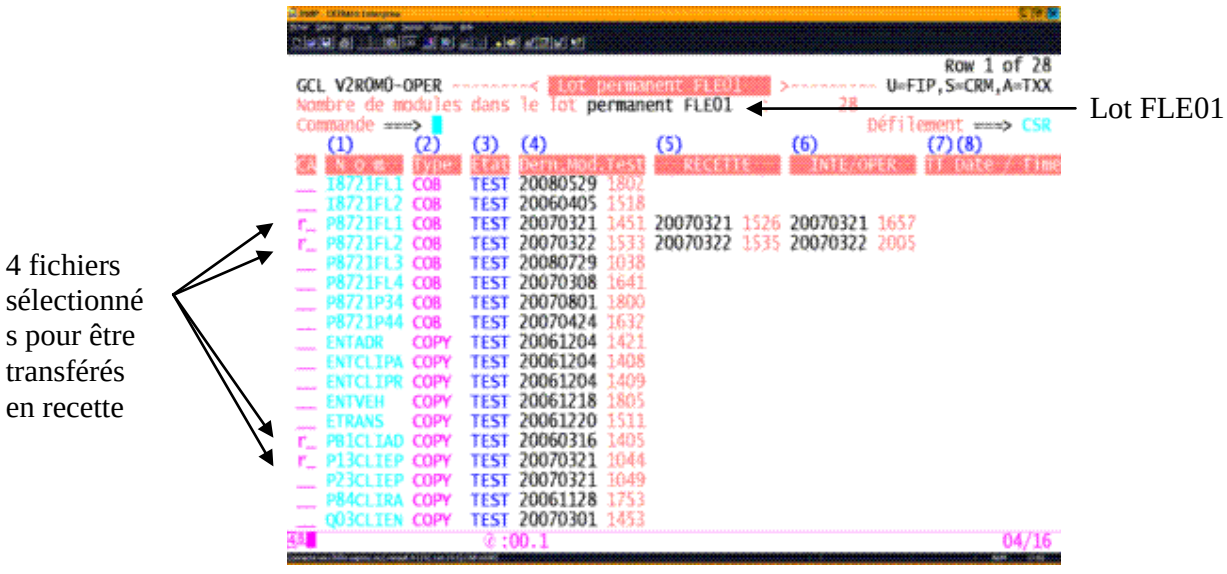


Figure 22 : Bloc de transfert

Le bloc de transfert sera constitué des 4 fichiers sélectionnés par la commande r saisie devant leur nom.

Le bloc de transfert de la Figure 22 est constitué des 4 fichiers sélectionnés. Tous les fichiers d'un bloc de transfert appartiennent à un même lot et sont transférés en même temps. Dans ce cas, leur date et heure de transfert sont identiques.

2.3.5.2 Le suivi des transferts

Une livraison correspond au transfert d'un ou de plusieurs fichiers sources. Le contenu d'une livraison dépend des évolutions et corrections prévues. Avec LDD, il était difficile de savoir quels étaient les fichiers sources transférés et quand ils ont été transférés. Avec LDD+, les informations sur les transferts sont enregistrées dans les fichiers baseline. Il y en a un pour l'environnement RE7 et un pour l'environnement OPE. Il n'y a pas de fichier baseline pour l'environnement de DEV pour 2 raisons :

- il n'y a pas de transfert vers l'environnement de DEV
- en environnement de DEV, un fichier source peut être en cours de modification sans qu'il y ait de version identifiée et sans version identifiée, il ne peut pas y avoir de gestion de configuration.

La Figure 23 suivante montre la structure d'un fichier baseline. Chaque ligne correspond au transfert d'un fichier source. Les lignes sont ajoutées au fur et à mesure à la fin du fichier baseline. Pour une meilleure lisibilité, la Figure 23 a été retranscrite.

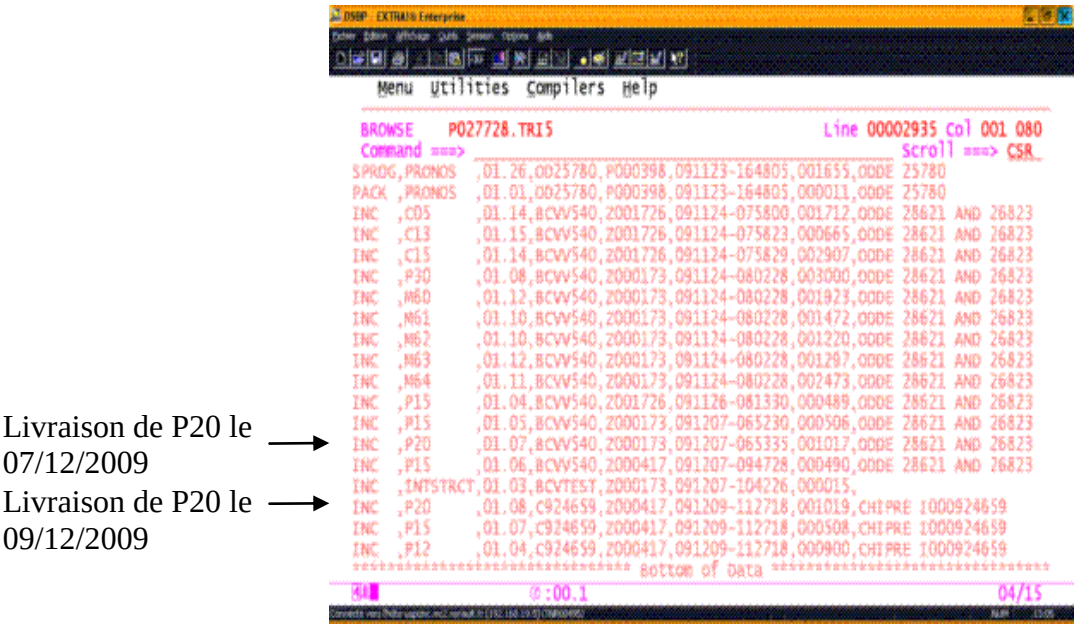
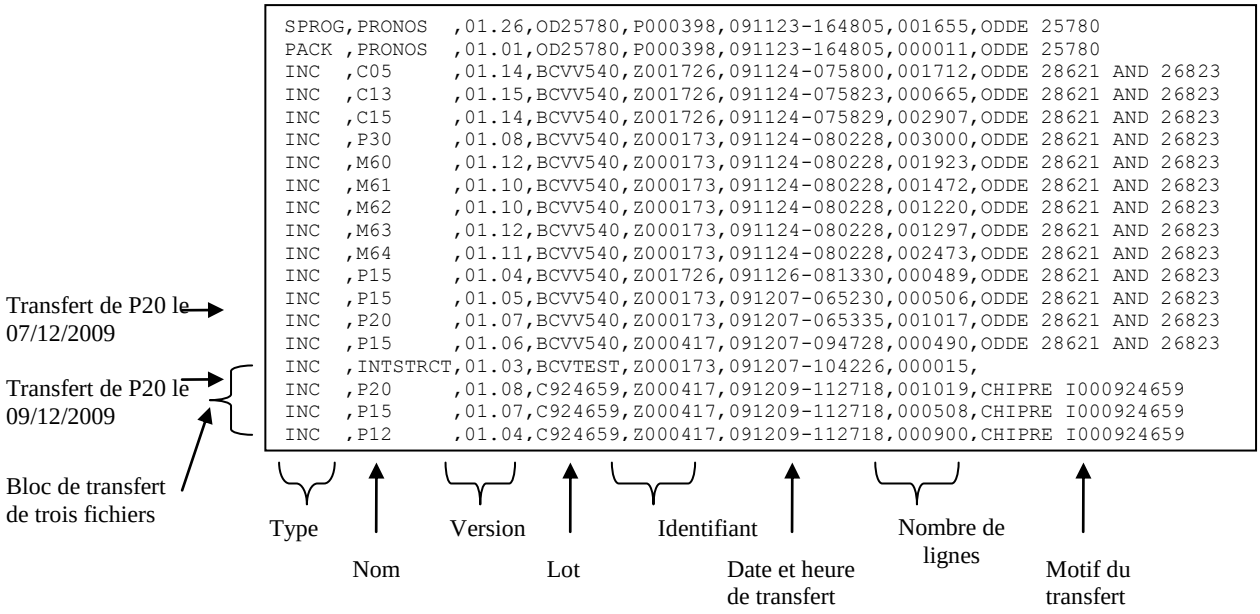


Figure 23 : Contenu d'un fichier Baseline
Ceci est une extraction du fichier baseline
pour restreindre l'affichage à une largeur de 80 caractères.



Transcription de la copie d'écran de la Figure 23.

Parmi les informations enregistrées, les plus importantes sont :

- le type du fichier source
- le nom du fichier source
- la version transférée du fichier source
- le lot dans lequel s'est fait le transfert
- l'identifiant de la personne qui a fait le transfert
- la date et l'heure du transfert du bloc de transfert
- le nombre de lignes du fichier source transféré
- le motif du transfert.

CHIPRE indique qu'il s'agit d'une anomalie. ODDE indique qu'il s'agit d'une évolution. Ce motif n'est pas toujours saisi. C'est pourquoi il est absent pour certaines lignes.

On voit dans la Figure 23 que le fichier source P20 de type INC a été transféré le 07/12/09 puis le 09/12/09. Le transfert du 09/12/09 fait partie d'un bloc de transfert contenant les fichiers sources P20, P15 et P12 de type INC.

Pour visualiser le fichier baseline, les développeurs utilisent la commande ZB. Cette commande permet de visualiser les transferts et leur contenu. Quand le développeur utilise la commande ZB, LDD+ lui demande d'abord l'environnement pour lequel il veut voir les transferts (RE7 ou OPE). Une fois l'environnement choisi, LDD+ affiche l'ensemble des transferts regroupés par bloc de transfert. La Figure 24 montre un exemple de résultat de l'utilisation de la commande ZB.

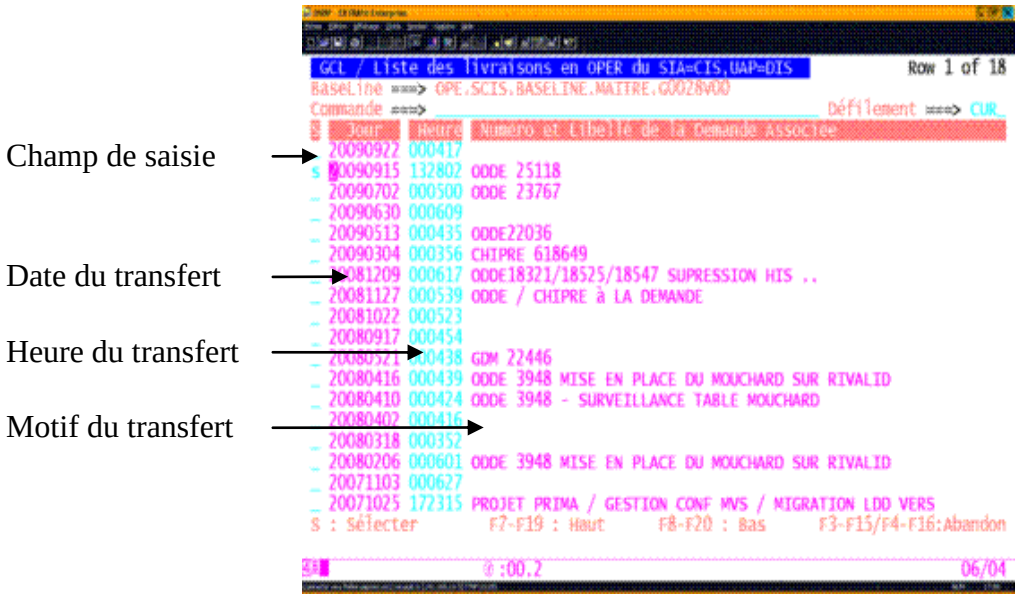
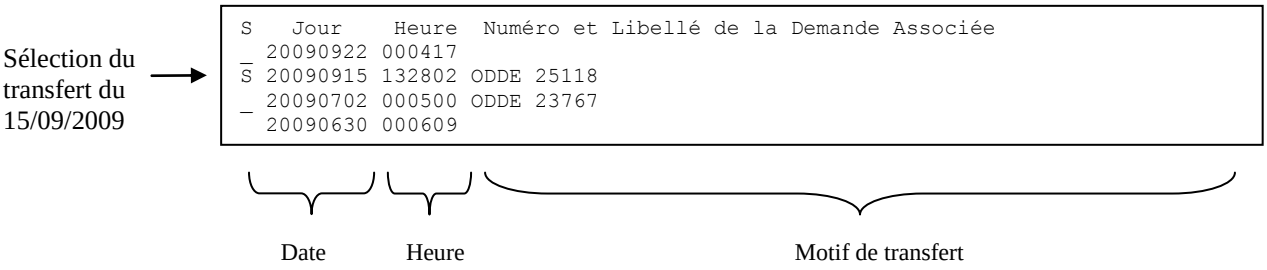


Figure 24 : Liste des transferts

Le bloc de transfert du 15/09/2009 a été sélectionné.



Transcription des 4 premières lignes de la Figure 24.

Le bloc de transfert du 15/09/2009 a été sélectionné avec la commande S.

Cette liste affichée est obtenue à partir du fichier baseline avec un tri sur les dates de transfert. A chaque ligne correspond un bloc de transfert. On voit la date et l'heure du bloc de transfert ainsi que son libellé. Dans ce libellé, on indique l'origine des transferts afin de mieux les reconnaître. La saisie de ce libellé, n'est pas obligatoire, c'est pourquoi certains transferts n'ont pas de libellé. Devant chaque bloc de transfert, il y a un champ de saisie qui permet de le sélectionner. La Figure 24 montre une liste de transferts dans laquelle on a sélectionné le transfert du 15/09/2009.

Une fois le bloc de transfert sélectionné, LDD+ affiche son contenu, c'est-à-dire l'ensemble des fichiers du bloc de transfert. Cette liste est obtenue à partir du fichier baseline avec un tri dans lequel on précise la date de transfert. La Figure 25 montre le contenu du bloc de transfert du 15/09/2009. Dans ce bloc de transfert, il y a trois fichiers. Le premier de ces fichiers est le fichier source PROQUEST de type INC. C'est sa version 01.06 qui a été transféré le 15/09/2009.

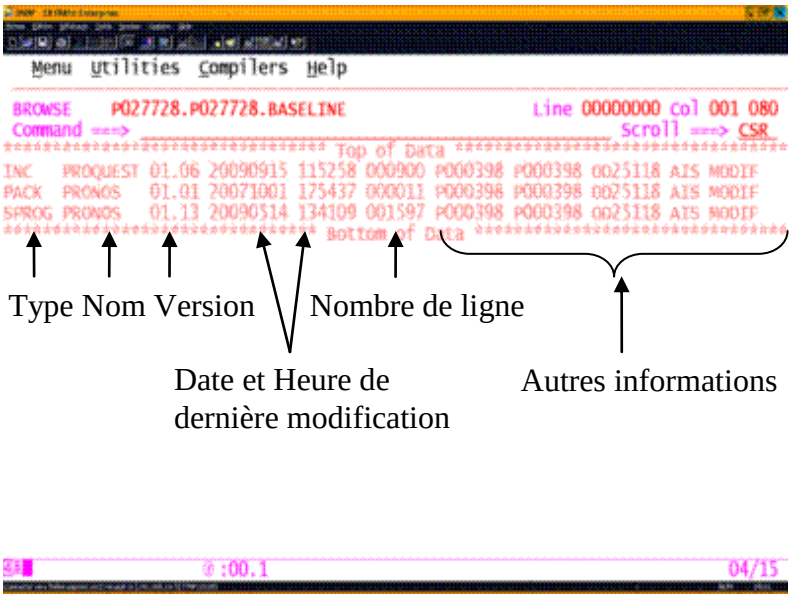


Figure 25 : Contenu d'un transfert

Le transfert du 15/09/2009 correspond au transfert de 3 fichiers sources.

A partir du fichier baseline, on peut aussi obtenir la photo d'un projet à une date donnée c'est-à-dire la liste des fichiers sources et leurs versions à une date donnée. Pour cela, on fait un tri sur le fichier baseline en précisant qu'on veut la version des modules la plus ancienne avant la date voulue.

2.3.5.3 Le suivi des opérations

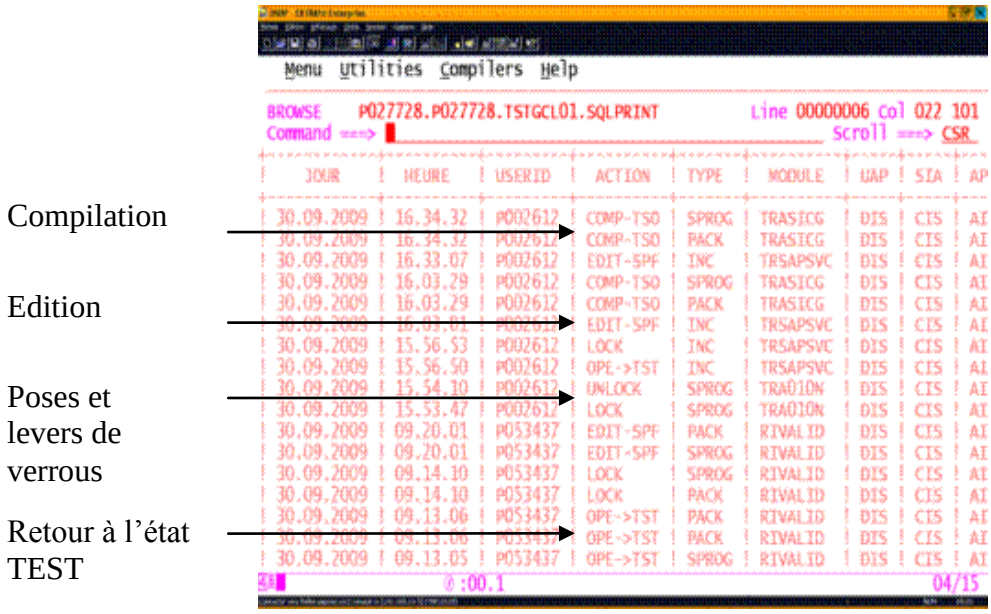
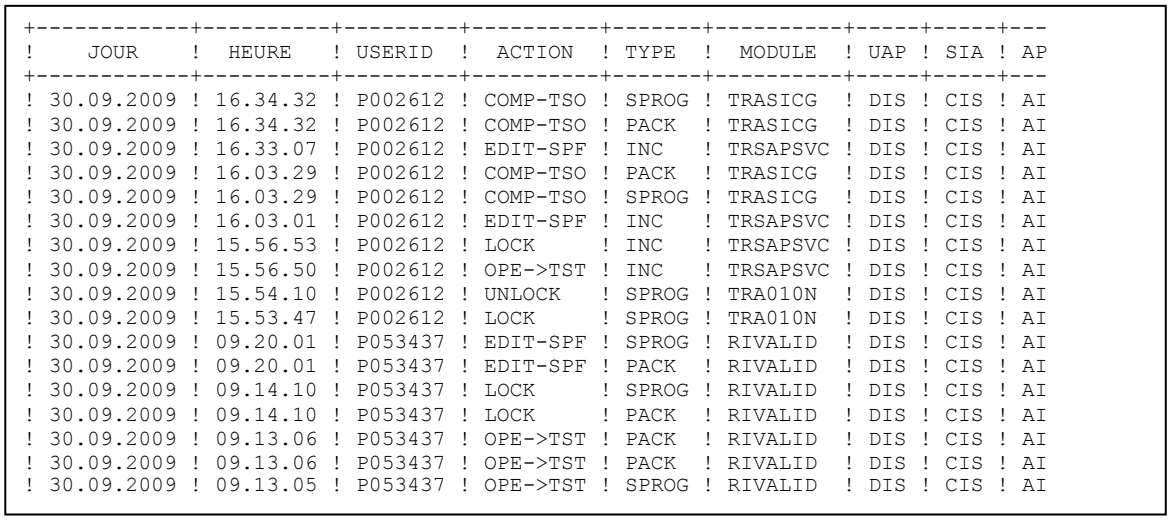


Figure 26 : Suivi des opérations

On voit qu'il y a eu des compilations, des éditions, des poses de verrous et des retours à l'état TEST.



JOUR	HEURE	USERID	ACTION	TYPE	MODULE	UAP	SIA	AP
30.09.2009	16.34.32	P002612	COMP-TSO	SPROG	TRASICG	DIS	CIS	AI
30.09.2009	16.34.32	P002612	COMP-TSO	PACK	TRASICG	DIS	CIS	AI
30.09.2009	16.33.07	P002612	EDIT-SPF	INC	TRSAPSVC	DIS	CIS	AI
30.09.2009	16.03.29	P002612	COMP-TSO	PACK	TRASICG	DIS	CIS	AI
30.09.2009	16.03.29	P002612	COMP-TSO	SPROG	TRASICG	DIS	CIS	AI
30.09.2009	16.03.01	P002612	EDIT-SPF	INC	TRSAPSVC	DIS	CIS	AI
30.09.2009	15.56.53	P002612	LOCK	INC	TRSAPSVC	DIS	CIS	AI
30.09.2009	15.56.50	P002612	OPE->TST	INC	TRSAPSVC	DIS	CIS	AI
30.09.2009	15.54.10	P002612	UNLOCK	SPROG	TRA010N	DIS	CIS	AI
30.09.2009	15.53.47	P002612	LOCK	SPROG	TRA010N	DIS	CIS	AI
30.09.2009	09.20.01	P053437	EDIT-SPF	SPROG	RIVALID	DIS	CIS	AI
30.09.2009	09.20.01	P053437	EDIT-SPF	PACK	RIVALID	DIS	CIS	AI
30.09.2009	09.14.10	P053437	LOCK	SPROG	RIVALID	DIS	CIS	AI
30.09.2009	09.14.10	P053437	LOCK	PACK	RIVALID	DIS	CIS	AI
30.09.2009	09.13.06	P053437	OPE->TST	PACK	RIVALID	DIS	CIS	AI
30.09.2009	09.13.06	P053437	OPE->TST	PACK	RIVALID	DIS	CIS	AI
30.09.2009	09.13.05	P053437	OPE->TST	SPROG	RIVALID	DIS	CIS	AI

Transcription de la Figure 26.

Avec LDD+, on est capable de suivre toutes les opérations effectuées par les développeurs. Les opérations suivies sont les poses et les levers de verrous, les modifications, les compilations, les Checkin, les Checkout et les transferts. Elles sont enregistrées dans une base de données. Dans la pratique, ce relevé des opérations n'est pas utilisé. La Figure 26 montre un exemple de relevé d'opérations récupéré avec la commande QT. Pour une meilleure lisibilité la Figure 26 a été retranscrite.

Les colonnes JOUR et HEURE donnent la date et l'heure de l'opération. La colonne USERID donne l'identifiant de la personne qui a accompli l'opération. La colonne ACTION donne le type de l'opération. Les colonnes TYPE et MODULE donnent le nom et le type du fichier sur lequel a agit l'opération. Les colonnes UAP, SIA et APPLI (dont l'affichage est tronqué) donnent des informations techniques en particulier le SIA qui identifie le projet (voir 2.2.1).

L'action COMP-TSO indique que le fichier a été compilé. L'action EDIT-SPF indique que le fichier a été édité. Les actions LOCK et UNLOCK indiquent que le fichier a été verrouillé et déverrouillé. L'action OPE->TST indique une opération de retour à l'état TEST.

Nous allons maintenant étudier un cas d'utilisation des fonctions Checkin et Checkout dans LDD+. Il préfigure les tests que j'ai menés pour la validation de LDD+.

2.3.6 Un exemple d'utilisation des fonctions Checkin et Checkout

2.3.6.1 Situation de départ

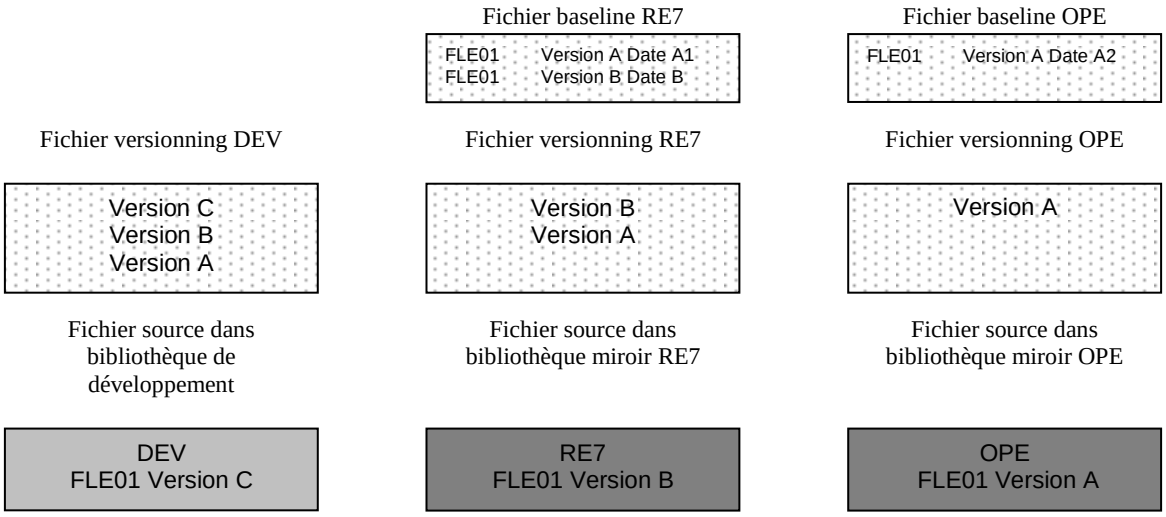


Figure 27 : Utilisation du Checkin et du Checkout
Figure 27A : Situation de départ.

Dans la Figure 27A, nous voyons un fichier source FLE01 qui possède 3 versions. La version A est la plus ancienne et a été transférée d'abord en recette à la date A1 puis en production à la date A2. C'est pour cela qu'on la trouve dans la bibliothèque miroir OPE. Le transfert en production a été enregistré dans le fichier baseline de production (date A2). Le transfert en recette a été enregistré dans le fichier baseline de recette (date A1). La versions B vient après et a été transférée en recette. C'est pour cela qu'on la trouve dans la bibliothèque miroir RE7. Ce transfert en recette a été enregistré dans le fichier baseline de recette (date B) après la version A (date A1). La version C est la plus récente et on la trouve dans la bibliothèque DEV. Ces trois versions sont enregistrées dans le fichier de versionning.

Chacune des versions a été enregistrée par un Checkin. Pour comprendre ce qui suit, il est important de savoir que le développeur ne peut travailler que sur le fichier source dans la bibliothèque DEV. Pour modifier la version en RE7, il est obligé de faire un transfert de DEV vers RE7. De même, pour modifier la version en OPE, il est obligé de faire un transfert de RE7 vers OPE.

Supposons qu'une demande de correction arrive et qu'elle nécessite une correction de la version en OPE sans que les versions en DEV et en RE7 soient modifiées. C'est ce qu'on appelle une correction à chaud. Plus précisément, il s'agit de remplacer la version A qui est en production (OPE) tout en retrouvant la version C en développement (DEV) et la version B en recette (RE7). Dans les figures suivantes nous allons expliquer comment le développeur va procéder.

2.3.6.2 Première étape : Checkout de la version A

La Figure 27B suivante présente la première étape. Elle consiste à récupérer la version A dans la bibliothèque de développement afin d'y apporter des corrections. Pour cela, le développeur va faire un Checkout de la version A. Cette opération va provoquer le remplacement de la version C par la version A.

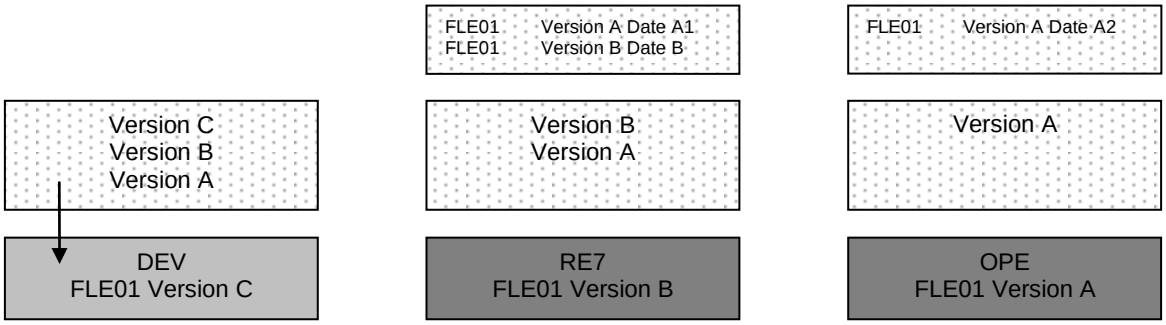


Figure 27B : Checkout de la version de production.

2.3.6.3 Deuxième étape : Correction de la version A

La Figure 27C suivante présente la deuxième étape. Le développeur va travailler à partir de la version A pour apporter ses modifications. Il obtient alors une version D du fichier source.

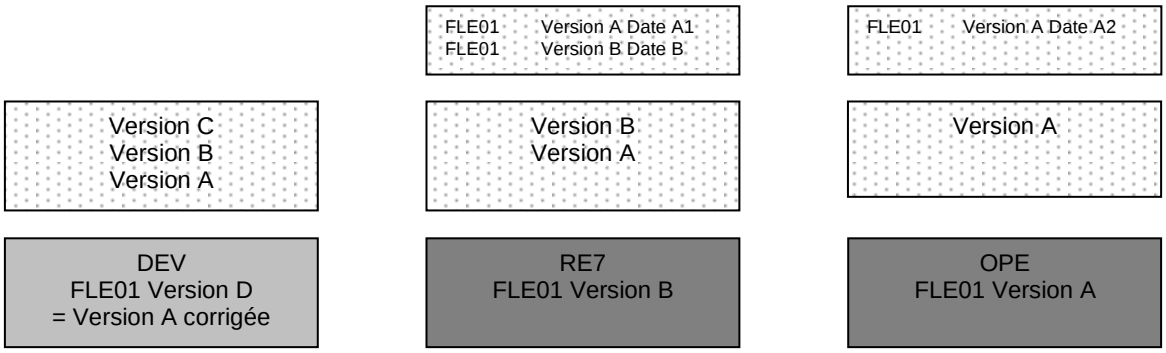


Figure 27C : Correction de la version A.

2.3.6.4 Troisième étape : Enregistrement de la version D par Checkin.

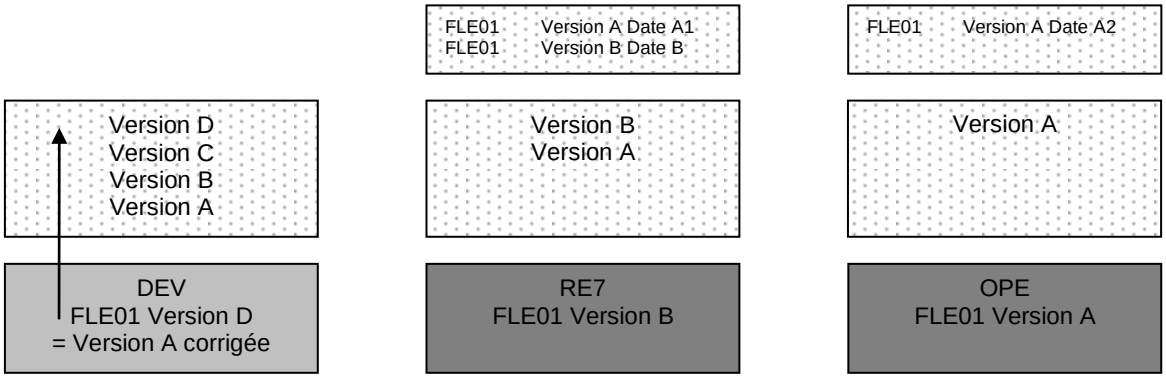


Figure 27D : Enregistrement de la version D par Checkin.

La Figure 27D précédente présente la troisième étape. Le développeur procède au Checkin de la version D. Ce Checkin est indispensable pour procéder au transfert en recette.

2.3.6.5 Quatrième étape : Transfert de la version D en RE7 puis en OPE

La Figure 27E suivante présente la quatrième étape. Pour que la version D se retrouve en production, le développeur va maintenant transférer la version D d'abord en RE7 puis en OPE car il n'est pas possible de faire un transfert directement de DEV vers OPE. Nous constatons que les transferts en RE7 font que la version D a écrasé la version B qui était en RE7 et la version A qui était en OPE.

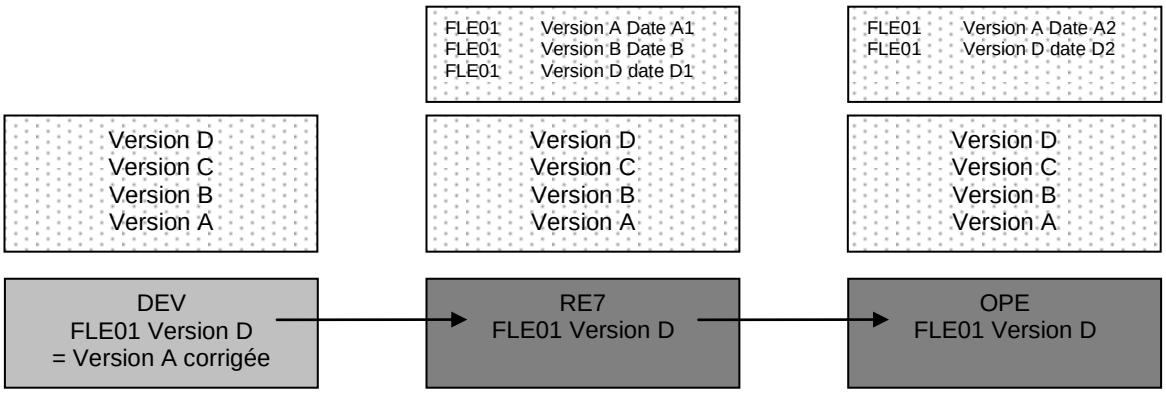


Figure 27E : Transfert de la version D en RE7 puis OPE.

Avec les transferts, les fichiers baseline de RE7 et d'OPE ont été mis à jour. Dans les fichiers versionning RE7 et OPE, on retrouve toutes les versions du fichier versionning DEV car la mise à jour de ces fichiers versionning se fait par copie intégrale. Cette copie intégrale pose clairement problème et sa suppression est une évolution prioritaire. Le problème est qu'on ne retrouve pas dans les fichiers versionning RE7 et OPE les versions qui ont été réellement transférées en RE7 et en OPE. Pour obtenir ces informations, il faut les extraire de la baseline, ce qui n'est pas pratique.

2.3.6.6 Cinquième étape : Checkout de la version B

La Figure 27F suivante présente la cinquième étape. Une fois la version D en production, le développeur veut remettre la version B ou plutôt une version identique à la version B en RE7. Pour cela, il procède à un Checkout de la version B.

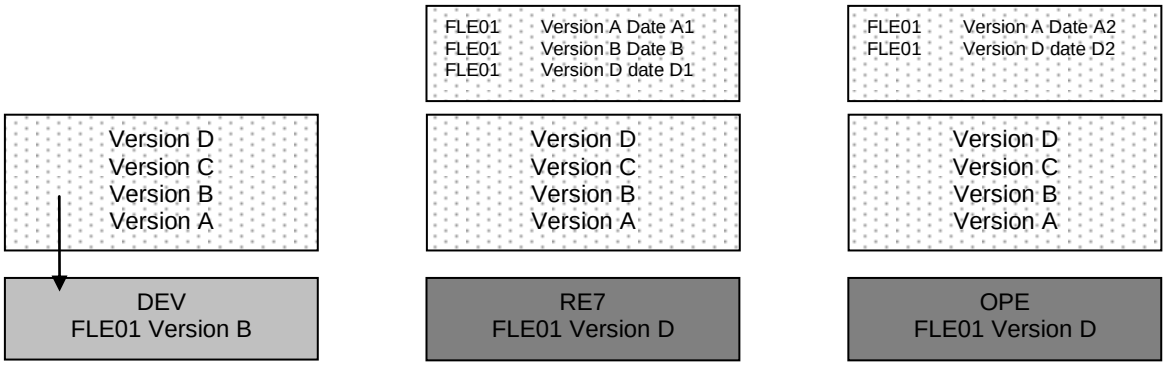


Figure 27F : Checkout de la version B.

2.3.6.7 Sixième étape : Checkin et transfert en RE7 de la version E

La Figure 27G suivante présente la sixième étape. Comme avec LDD+, on ne peut transférer que la version la plus récente, le développeur doit enregistrer une version E identique à la version B par un Checkin. Puis il procède au transfert de la version E en RE7 qui met à jour le fichier versionning RE7 et le fichier baseline RE7.

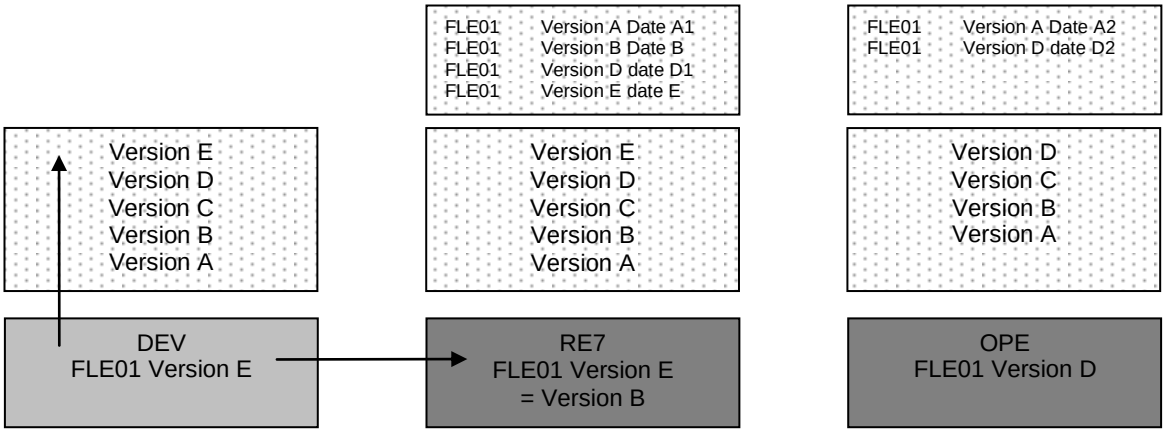


Figure 27G : Checkin et transfert de la version E identique à la version B.

2.3.6.8 Septième étape : Checkout de la version C

La Figure 27H suivante présente la septième étape. Pour retrouver la version C dans la bibliothèque de développement, le développeur procède à un Checkout de la version C. Au final, on obtient une version OPE modifiée (version D) alors que les versions DEV et RE7 n'ont pas changé. En réalité, ce cas n'est pas totalement réaliste. En général, il faut aussi reporter les corrections de la version D sur les versions suivantes : la version E devrait être différente de la version B. Ce report n'est pas assuré par LDD+. Pour cela, il faudrait que LDD+ possède des fonctions de fusion entre versions. Ces fonctions de fusion font partie de la gestion des développements concurrents présentée dans les fonctions de verrouillage (voir 2.3.7.1).

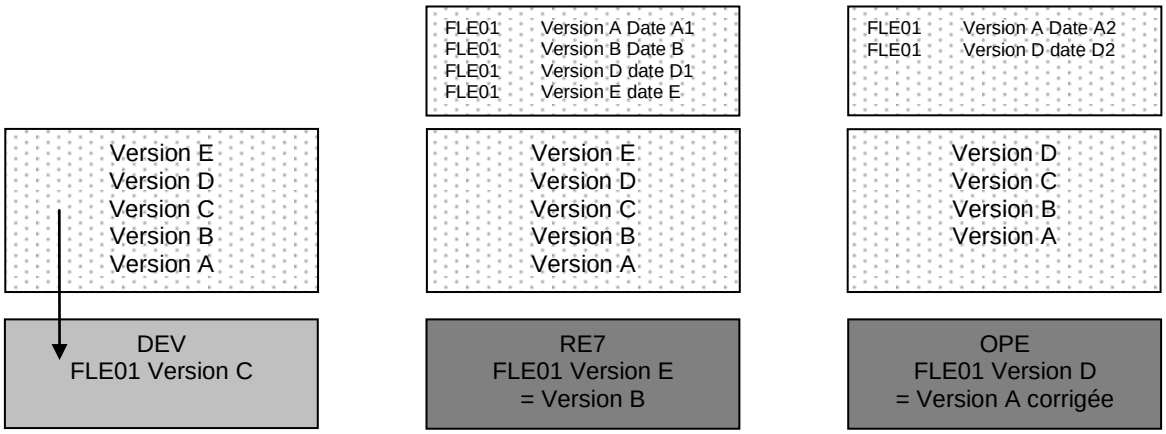


Figure 27H : Checkout de la version C.

2.3.6.9 Remarques

En étudiant les opérations décrites précédemment, on peut faire les remarques suivantes. La mise à jour des fichiers versionning de RE7 et d'OPE doit être améliorée. Elle ne devrait pas se faire par copie intégrale mais par recopie uniquement de la version transférée. Ce n'est pas logique que la version C (voir Figure 27) apparaisse dans les fichiers versionning RE7 et OPE alors qu'elle n'a jamais été transféré ni en RE7 ni en OPE.

Nous avons déjà expliqué (voir 2.3.2) qu'il ne peut pas y avoir de GCL en environnement DEV parce qu'il n'y a pas de fichier baseline DEV pour deux raisons :

- il n'y a pas de transfert vers l'environnement DEV
- un fichier source peut être en cours de modification sans qu'une version soit enregistrée.

Les fichiers versionning RE7 et OPE ne sont utilisés que pour du stockage. On n'est pas obligé de les utiliser pour faire un retour arrière car toutes les versions sont dans le fichier versionning DEV. Comme ces fichiers ne sont stockés que dans l'environnement DEV, ils ne peuvent pas être utilisés pour faire un retour arrière en environnement RE7 ou en environnement OPE.

2.3.7Les fonctions de verrouillage de LDD+

Les informations qui suivent sont issues des documents suivants :
PRINTZ J., 2004. Introduction à la gestion de configuration, CNAM.
Support de cours : Intro-GCONF-10.pdf
COLLINS-SUSSMAN B., FITZPATRICK B., PILATO C., 2008. Version control with Subversion.
http://svnbook.red-bean.com/svn-book.pdf

2.3.7.1 Les modèles collaboratifs

Le processus GCL demande à ce que les développements concurrents soient gérés. Un développement concurrent se produit lors deux développeurs doivent travailler sur le même fichier source. Gérer un développement concurrent consiste à éviter les conflits entre les modifications des différents développeurs. La Figure 28 montre le problème posé par le développement concurrent.

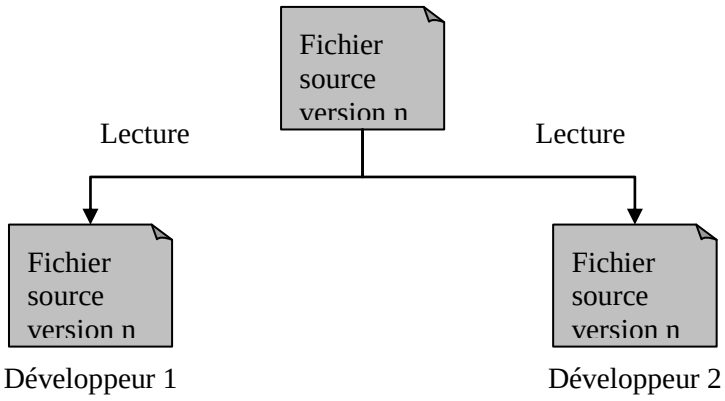


Figure 28 : Problème du développement concurrent
Figure 28A :.Les deux développeurs récupèrent chacun de leur côté la version n du fichier source sur lequel ils doivent travailler.

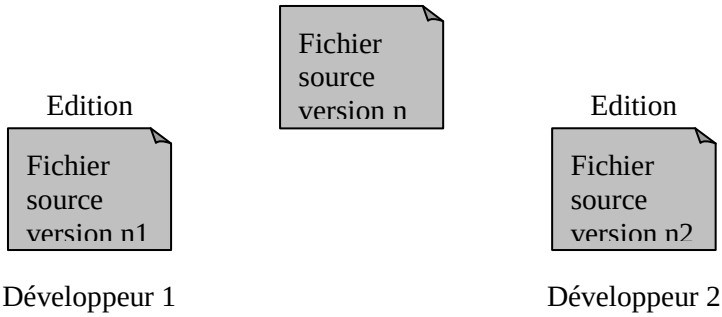


Figure 28B : Chaque développeur travaille sur sa copie de fichier et crée sa propre version.

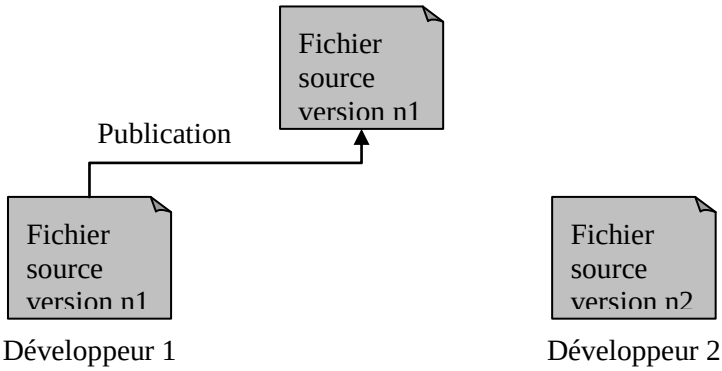


Figure 28C : Le développeur 1 publie en premier sa version.

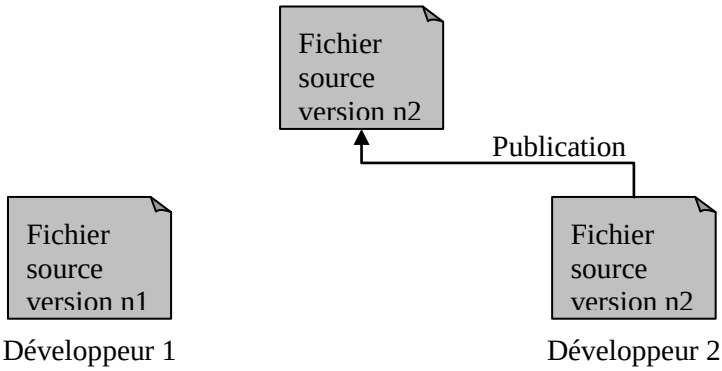


Figure 28D : Le développeur 2 publie sa version et écrase la version du développeur 1. Les travaux du développeur 1 sont perdus.

Au départ, les deux développeurs récupèrent chacun de leur côté la version n du fichier source sur lequel ils doivent travailler (Figure 28A). Ensuite chaque développeur travaille sur sa copie de fichier et crée sa propre version. Le développeur 1 crée la version n1 et le développeur 2 crée la version n2 (Figure 28B). Puis le développeur 1 publie en premier la version. La version n1 remplace donc la version n (Figure 28C). Quand le développeur 2 publie à son tour la version n2 (Figure 28D), celle-ci remplace la version n1. Comme le développeur 2 a travaillé à partir de la version n, la version n2 ne peut pas prendre en compte les travaux du développeur 1 dans la version n1. Ces travaux sont donc « perdus » dans la version n2.

Pour gérer les développements concurrents, il existe deux modèles collaboratifs :

- le modèle « Verrouillage-Edition-Enregistrement »
- le modèle « Extraction-Edition-Fusion »

Le modèle « Verrouillage-Edition-Enregistrement » utilisé par LDD+ repose sur des verrous. Il est présenté dans la Figure 29. Le premier développeur commence par verrouiller le fichier source puis il récupère la version n du fichier source sur lequel il doit travailler (Figure 29A). Plus personne ne peut intervenir sur ce fichier source. Toute tentative du développeur 2 de verrouiller le fichier source aboutira à un échec (Figure 29B). Une fois que le développeur 1 a fini ses travaux, il commence par enregistrer la version n1 puis il déverrouille le fichier source (Figure 29C). Le développeur 2 peut verrouiller à son tour le fichier source. Comme, il est obligé de travailler à partir de la version n1, ses travaux prendront en compte les travaux du développeur 1 (Figure 29D).

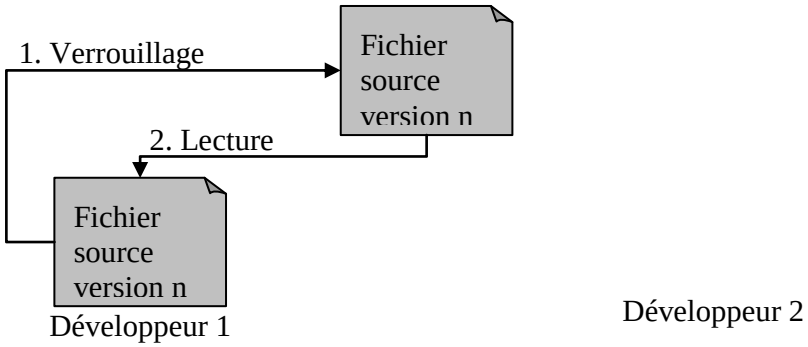


Figure 29 : Modèle « Verrouillage-Edition-Enregistrement »
Figure 29A : Le développeur 1 commence par verrouiller le fichier source puis il récupère la version n du fichier source.

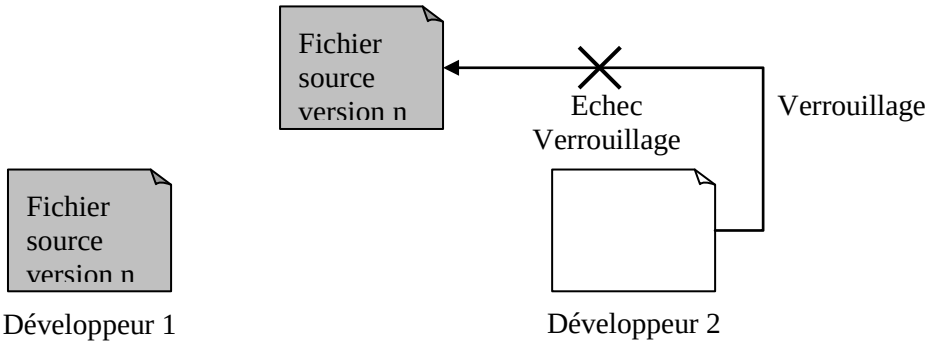


Figure 29B : Si le développeur 2 veut verrouiller le fichier pour travailler, un message d'erreur l'avertit que le fichier source est déjà verrouillé par le développeur 1.

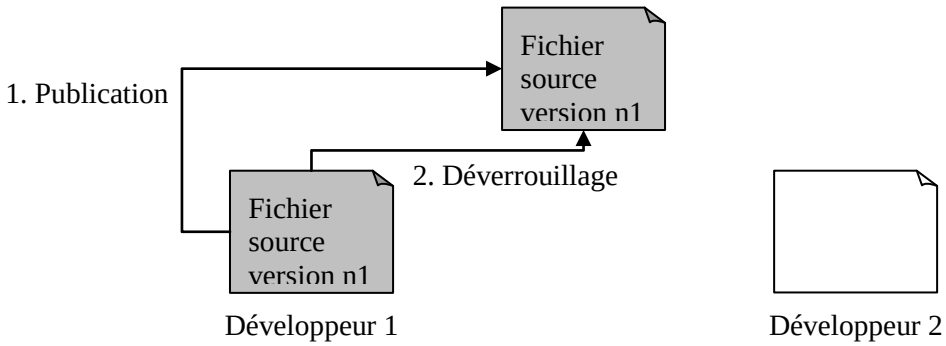


Figure 29C : Une fois, ses travaux terminés, le développeur 1 publie la version n1 du fichier source. Puis il le déverrouille.

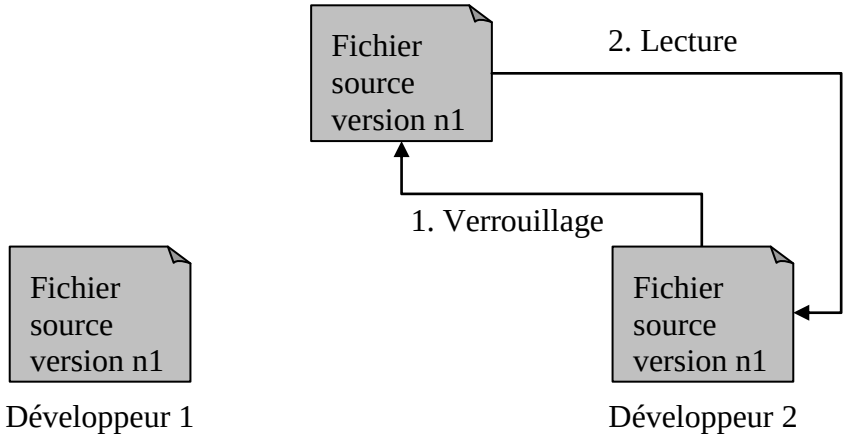


Figure 29D : Le développeur verrouille à son tour le fichier source et travaille sur la version n1. Ses travaux prendront en compte les travaux du développeur 1.

L'avantage de ce modèle est sa simplicité car il évite tous les problèmes de conflit d'édition parallèle. Le premier inconvénient de ce modèle est qu'il ne permet pas de parallélisation des travaux. Le développeur 2 ne peut pas travailler tant que le développeur1 n'a pas déverrouillé le fichier. Le deuxième inconvénient est qu'on risque des situations de blocage en particulier quand un développeur s'absente sans avoir déverrouillé le fichier source. De plus, ce modèle impose de travailler en mode connecté pour que le développeur soit averti en temps réel de la situation des verrous. C'est le modèle utilisé par LDD+ parce que c'est le plus simple à implémenter et que l'environnement Mainframe impose le mode connecté. Dans l'environnement Mainframe, les fichiers sont centralisés et les développeurs ne peuvent pas travailler à plusieurs sur un même fichier.

Le modèle « Extraction-Edition-Fusion » repose sur une stratégie optimiste : on suppose que les cas de conflits seront rares et/ou qu'ils sont facilement résolvables. Chaque développeur va travailler sur une copie du fichier source sans verrouiller le fichier source. S'il y a conflit, le développeur sera averti et procédera à la fusion des versions. La Figure 30 montre son principe.

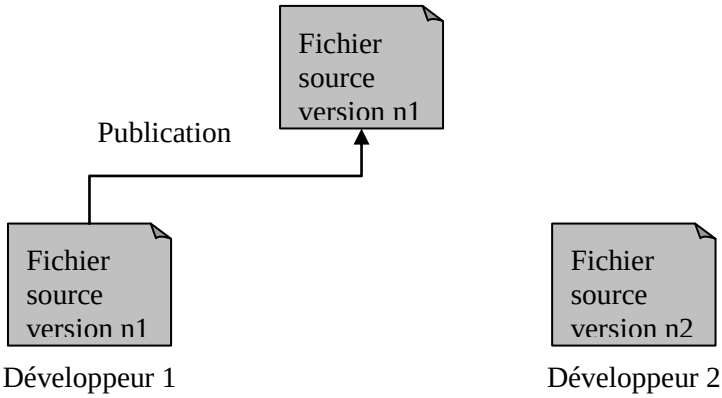


Figure 30 : Modèle « Extraction-Edition-Fusion »
Figure 30A : Comme dans le schéma 28C du problème du développement concurrent, chaque développeur a travaillé sur une copie de la version n du fichier source. Le développeur 1 est le premier à publier sa version.

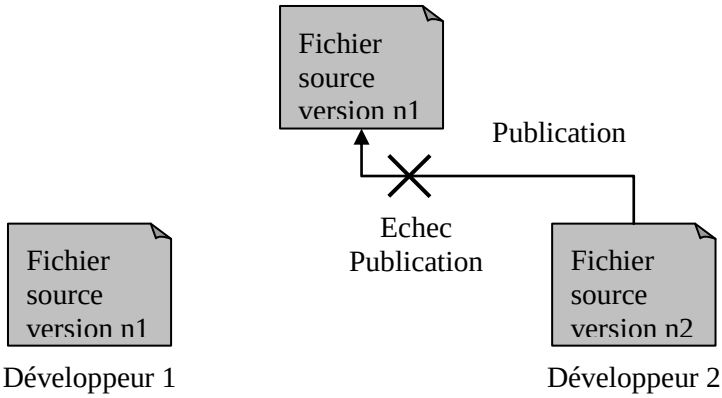


Figure 30B : Lorsque le développeur 2 essaie de publier sa version, il reçoit un message d'erreur qui l'avertit que le fichier source a été modifié.

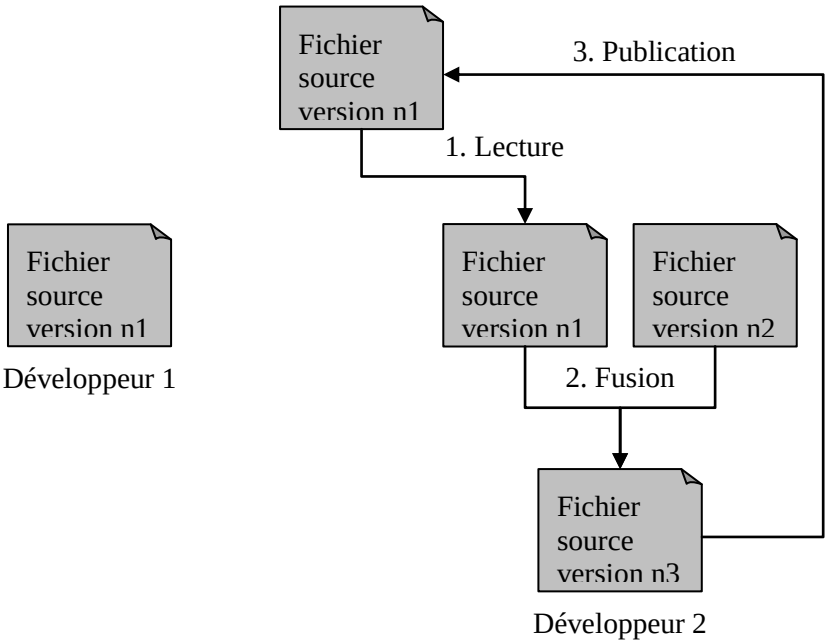


Figure 30C : Le développeur 2 récupère la version n1 du fichier source et la synchronise avec sa version n2 pour obtenir la version n3. C'est cette version n3 que le développeur 2 publiera.

On prend comme point de départ, le moment où le développeur 1 a enregistré le résultat de ses travaux dans la version n1 du fichier source (Figure 30A identique à la Figure 28C). Lorsque le développeur 2 essaie de publier ses travaux dans la version n2, cette tentative échoue car le système a détecté que la version n a été remplacée par la version n1 (Figure 30B). Le développeur 2 récupère la version n1 pour la synchroniser avec la version n2, il obtient alors la version n3. C'est cette version n3 qui sera enregistrée et qui remplacera la version n1 (Figure 30C). Pour synchroniser les versions n1 et n2, le système permettra au développeur 2 de confronter les différences entre les 2 versions. Pour créer la version n3, il choisira les différences à garder (Figure 30D : les différences 1 et 2). Dans certains cas les différences impactent la même partie de code, ce qui obligera le développeur à réécrire ce code pour fusionner les différences (Figure 30D : les différences 3 et 4).

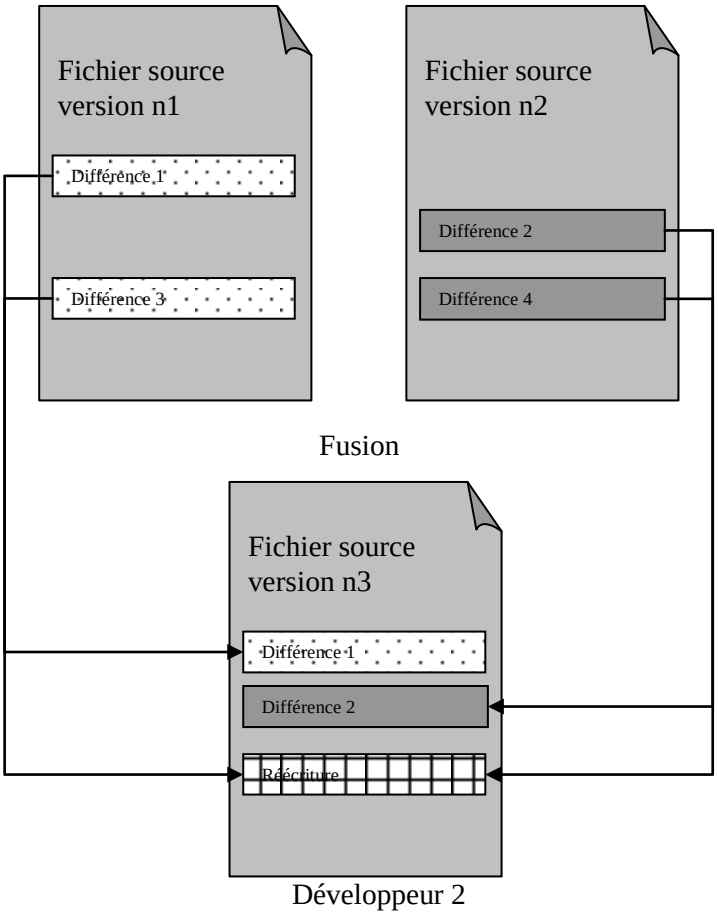


Figure 30D : Détail de la fusion.
Le système de gestion de fusion des versions va afficher les différences entre les versions. Puis le développeur choisira les différences qu'il voudra garder. Dans certains cas, le chevauchement des différences nécessitera une réécriture partielle du fichier source.

L'avantage de ce modèle est qu'il permet la parallélisation des travaux contrairement au modèle « Verrouillage-Edition-Enregistrement ». De plus on n'a pas besoin d'être connecté en permanence au référentiel. Le premier inconvénient est qu'il faut un système de gestion de fusion. Le deuxième inconvénient est que le chevauchement de différence entraîne une réécriture de code qui peut être plus ou moins complexe. En général, malgré ces inconvénients, on préfère utiliser ce modèle car il est plus souple d'utilisation. Il est utilisé par Subversion l'outil de GCL pour l'environnement Windows/UNIX.

2.3.7.2 La nécessité de verrous avec LDD+

A priori de nouvelles fonctions de verrouillage n'étaient pas nécessaires car du fait de sa nature centralisé, l'environnement Mainframe implémente par défaut le modèle « Verrouillage-Edition-Enregistrement ». Il empêche que deux développeurs travaillent en même temps sur un même fichier source. Cette implémentation a deux inconvénients.

Le premier inconvénient est que le verrouillage est limité dans le temps. Il ne dure que pendant le moment où le développeur travaille sur le fichier. Si le développeur suspend son travail, par exemple pour consulter un autre fichier ou parce qu'il a fini sa journée, le fichier est alors déverrouillé.

Le deuxième inconvénient est que le développeur n'est pas averti si un autre développeur a travaillé sur le fichier source.

La Figure 31 illustre le problème. Contrairement à la situation présentée dans la Figure 29A, Le développeur travaille directement sur le fichier source version n. Il n'y a pas de recopie du fichier dans un espace de travail personnel. Le fichier source est déverrouillé dès que le développeur 1 ne travaille plus sur le fichier (Figure 31A). Le développeur 2 peut alors travailler directement travailler sur le fichier source n1 (Figure 31B). Le problème est que le développeur n1 n'a pas été averti que le fichier source est passé à la version n2. Lorsqu'il travaillera à nouveau sur le fichier source, il croira travailler sur la version n1 (Figure 31C).

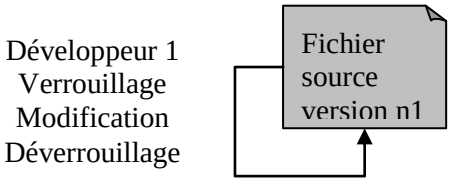


Figure 31 : Limite du modèle « Verrouillage-Edition-Enregistrement » dans l'environnement Mainframe.
Figure 31A : Le développeur 1 travaille directement sur le fichier source version n.

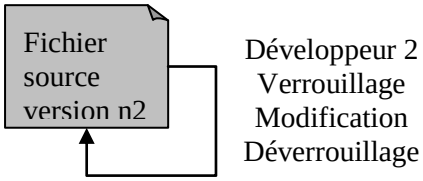


Figure 31B : Le développeur 2 travaille directement sur le fichier source version n1.

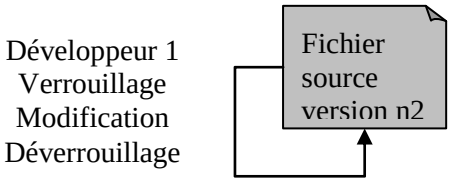


Figure 31C : Le développeur 1 travaille sur la version n2 en croyant travailler sur la version n1.

Ce problème ne se produit pas dans le cas présenté dans la Figure 29 car le déverrouillage est sous contrôle du développeur 1. Celui-ci ne libérera le fichier qu'une fois ces travaux terminés. Le développeur 2 ne peut pas agir sur le fichier source à son insu. Pour résoudre ce problème et avertir les développeurs que des travaux sont en cours sur un fichier, on a mis en place dans LDD+ des fonctions de verrouillage en plus du système « Verrouillage-Edition-Enregistrement » par défaut de l'environnement Mainframe.

2.3.7.3 Les caractéristiques générales d'un verrou LDD+

Chaque fois qu'un développeur utilise un fichier que ce soit pour le modifier ou pour le transférer, LDD+ vérifie l'existence de verrou et s'il n'y en a pas, pose automatiquement un verrou rattaché au développeur. Tant que ce verrou est posé, il n'y a plus que ce développeur qui peut intervenir sur ce fichier source. Si un autre développeur essaie d'accéder à ce fichier source, LDD+ vérifie que le verrou ne lui est pas rattaché et il lui renvoie un message d'erreur. Dans ce cas, le fichier source ne s'affiche qu'en lecture seule.

Pour régler les problèmes qui peuvent se poser en cas d'absence d'un développeur, il existe une commande administrateur de déverrouillage. Cette commande administrateur est réservée aux administrateurs. Il y a deux types d'administrateurs : les administrateurs qui sont rattachés à un projet ou plusieurs projets et les administrateurs tous projets qui peuvent intervenir sur tous les projets. En tant que participant au projet LDD+ je fais partie des administrateurs tous projets.

Il est à noter que les verrous LDD+ ne fonctionnent que sous LDD+. Ils n'empêchent pas de modifier un fichier source en passant par l'éditeur standard Mainframe. C'est une violation d'un principe CMMI mais on n'a pas pu faire autrement. En effet l'environnement Mainframe est fait de telle sorte que si on bloque l'accès à l'éditeur alors on bloque l'utilisation de LDD+. Pour savoir si un fichier source est verrouillé, on utilise la commande ZC. Si le fichier source est verrouillé, LDD+ indique différents renseignements : le type de verrou, l'utilisateur qui a posé le verrou, la date de pose du verrou.

2.3.7.4 Les différents types de verrous LDD+

Dans LDD+, il existe 3 types de verrous :

- le verrou Checkout,
- le verrou Lock,
- le verrou Transfert.

Le verrou Checkout est le plus important car c'est lui qui gère les développements concurrents. Il renforce le système de verrouillage par défaut de l'environnement Mainframe. Il avertit les développeurs que des travaux sont en cours sur les fichiers source. Il est posé par défaut quand l'utilisateur accède au fichier source. Si le fichier source n'est pas modifié alors le verrou Checkout est levé quand l'utilisateur quitte le fichier source car on considère que comme il n'y a pas eu modification, il n'y a pas de travail en cours. A l'inverse, si le fichier source est modifié alors le verrou est maintenu quand l'utilisateur quitte le fichier source car on considère que les travaux ont commencé. Ce verrou empêche toute opération (édition, transfert, Checkin, Checkout). Pour lever le verrou Checkout, il n'y a que deux possibilités : le Checkin et la commande administrateur de déverrouillage. Le Checkin qui est l'enregistrement d'une version d'un fichier source a été présenté dans les fonctions de versionning (voir 2.3.4.1).Il ne peut se faire que si le fichier source a été verrouillé au préalable par un verrou Checkout et que si l'utilisateur qui procède au Checkin est le même que celui qui a posé le verrou Checkout.

Une autre possibilité de poser un verrou Checkout est de faire un Checkout sur le fichier source d'où le nom de verrou Checkout. Le Checkout qui permet de récupérer une version déjà créée a été présenté dans les fonctions de versionning (voir 2.3.4.2). Comme la version choisie écrase le contenu courant du fichier source, le verrou Checkout est posé automatiquement. Il est à noter qu'on ne peut pas faire de Checkout si le fichier source est déjà verrouillé par un verrou Checkout.

Le verrou Lock n'est pas beaucoup utilisé. Lorsqu'un utilisateur souhaite se réserver un fichier source, il peut utiliser la commande ZL pour verrouiller ce fichier source. Le fichier source est alors verrouillé par un verrou Lock. Toute intervention sur le fichier source est alors impossible même par celui qui a posé le verrou. A l'exception de la commande administrateur de déverrouillage, seul l'utilisateur qui a posé le verrou peut le lever en utilisant la commande ZU.

Le verrou Transfert est posé lorsqu'un fichier source est en cours de transfert. Il a été créé pour éviter qu'un fichier source soit modifié pendant qu'il soit transféré.

La Figure 32 présente le diagramme état-transition des verrous LDD+.

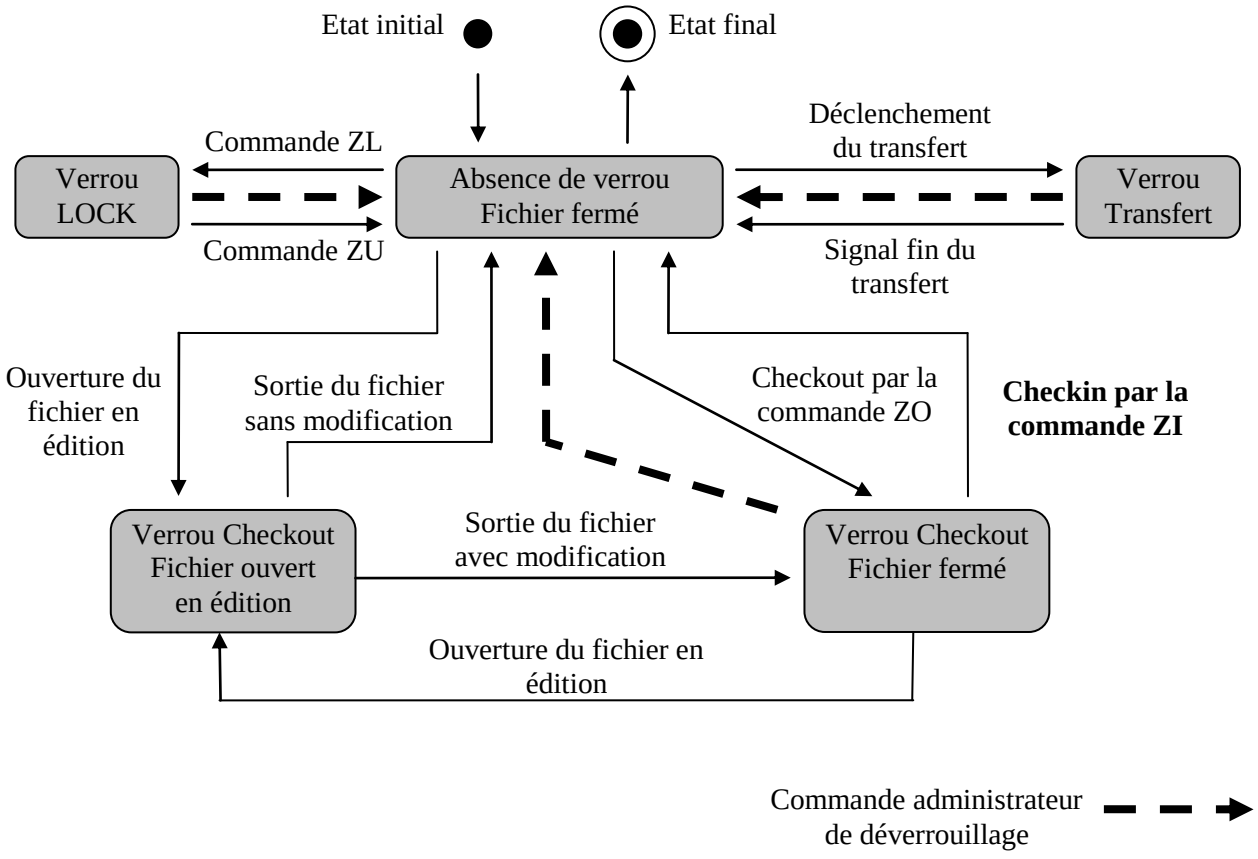


Figure 32 : Diagramme état-transition des verrous LDD+

La commande ZI est la plus importante car le Checkin libère le fichier en levant le verrou Checkout. Cela permet alors à un autre développeur soit de travailler sur le fichier soit de transférer le fichier en recette.
La commande administrateur de déverrouillage est représentée par une flèche en pointillé.

Pour pouvoir lever les verrous en cas d'urgence ou en cas d'absence, il a été créé une commande administrateur de déverrouillage. Pour pouvoir utiliser cette commande, la personne doit être déclarée au préalable. En général, c'est le chef de projet. On a restreint l'accès à cette commande pour que les développeurs respectent les verrous. Grâce à cette commande, la personne peut lever n'importe quel verrou même si elle n'est pas la personne qui l'a posée.

2.4 LDD+ et la Gestion de Configuration Logicielle

Nous allons étudier ici comment LDD+ permet la GCL. Nous commencerons par voir si LDD+ est en adéquation avec la gestion de configuration telle qu'elle est définie dans la démarche qualité CMMI puis nous comparerons LDD+ avec Subversion, l'autre outil de gestion de configuration choisi par Renault pour l'environnement Windows/UNIX.

2.4.1 LDD+ et la Gestion de configuration dans CMMI

Nous avons vu dans les principes de la gestion de configuration (voir 1.4.2) que celle-ci s'appuie sur trois éléments :

- une baseline
- un référentiel
- une gestion des développements concurrents

Dans la présentation de LDD+, nous avons vu que :

- la baseline est constitué par les fichiers baselines. Dans ces fichiers baselines, on retrouve toutes les informations pour suivre les modifications dans les environnements RE7 et OPE. Ces informations permettent de reconstituer l'ensemble d'un projet à une date donnée ;
- le référentiel est constitué par les fichiers versionning. Les fonctions de versionning permettent d'enregistrer et d'extraire ces versions. De plus, l'étiquette renseignée au moment du Checkin permet de remonter aux demandes à l'origine des modifications ;
- la gestion des développements concurrents est assurée par les verrous LDD+ selon le modèle collaboratif « Verrouillage-Edition-Enregistrement ».

Les fichiers baseline n'étant présents qu'en environnement RE7 et OPE, il n'y a de GCL que dans ces environnements (voir 2.3.2). Cela a été officiellement constaté et accepté car les environnements de RE7 et OPE sont les plus importants. C'est au vu de tous ces éléments que LDD+ a été validé comme outil de gestion de configuration dans le cadre de la certification CMMI.

2.4.2La comparaison de LDD+ avec Subversion

Nous allons ici comparer le fonctionnement de LDD+ avec Subversion, l'outil de GCL choisi par Renault pour l'environnement Windows/UNIX. Subversion est un outil représentatif des outils de GCL dans l'environnement Windows/UNIX. Nous allons voir que LDD+ utilise des mécanismes beaucoup plus simples.

Les informations sur Subversion qui suivent sont extraites du document suivant :

COLLINS-SUSSMAN B., FITZPATRICK B., PILATO C., 2008. Version control with Subversion.

http://svnbook.red-bean.com/svn-book.pdf

2.4.2.1 La gestion des développements concurrents

Dans la présentation des fonctions de verrouillage de LDD+ (voir 2.3.7), Nous avons vu que LDD+ utilise le modèle collaboratif « Verrouillage-Edition-Enregistrement » avec deux niveaux de verrouillage :

- le verrouillage spécifique à l'environnement Mainframe qui empêche deux développeurs de travailler en même temps sur le même fichier.
- le verrouillage de LDD+ qui complète le système de verrouillage mainframe en avertissant que des travaux sont en cours sur un fichier.

Subversion utilise le modèle « Extraction-Edition-Fusion » (voir 2.3.7.1) qui est plus souple d'utilisation.

2.4.2.2 Le stockage des versions

Avec LDD+, les différentes versions d'un fichier source sont stockés les unes après les autres dans le fichier versionning associé au fichier source. On parle de sauvegarde totale. Avec Subversion, la version initiale est sauvegardée puis ce sont les différences entre les versions et la version initiale qui sont sauvegardées dans un fichier archive. On parle alors de sauvegarde incrémentale par delta.

Les avantages du premier système (LDD+) sont sa simplicité et la possibilité de consulter facilement les différentes versions. Son inconvénient est de demander beaucoup d'espace de stockage. Le principal avantage du second système est le gain de place puis que seules les modifications sont enregistrées au lieu des versions complètes. Cela facilite aussi les comparaisons entre versions et évite de stocker des informations redondantes. Son principal inconvénient est que cela demande un système de gestion complexe et que ce n'est pas toujours adapté à certains types de fichiers comme les images ou des objets binaires.

2.4.2.3 La gestion des versions du projet

LDD+ ne gère que la version des fichiers source, il ne gère pas les versions d'un projet dans sa globalité. Le retour arrière vers la version antérieure d'un projet devra se faire manuellement et fichier par fichier à partir des informations extraites des fichiers baseline. A l'inverse, Subversion permet de gérer les versions d'un projet. Dans la Figure 33, nous présentons le contenu d'un référentiel Subversion visualisé par l'intermédiaire du navigateur Microsoft Internet Explorer.

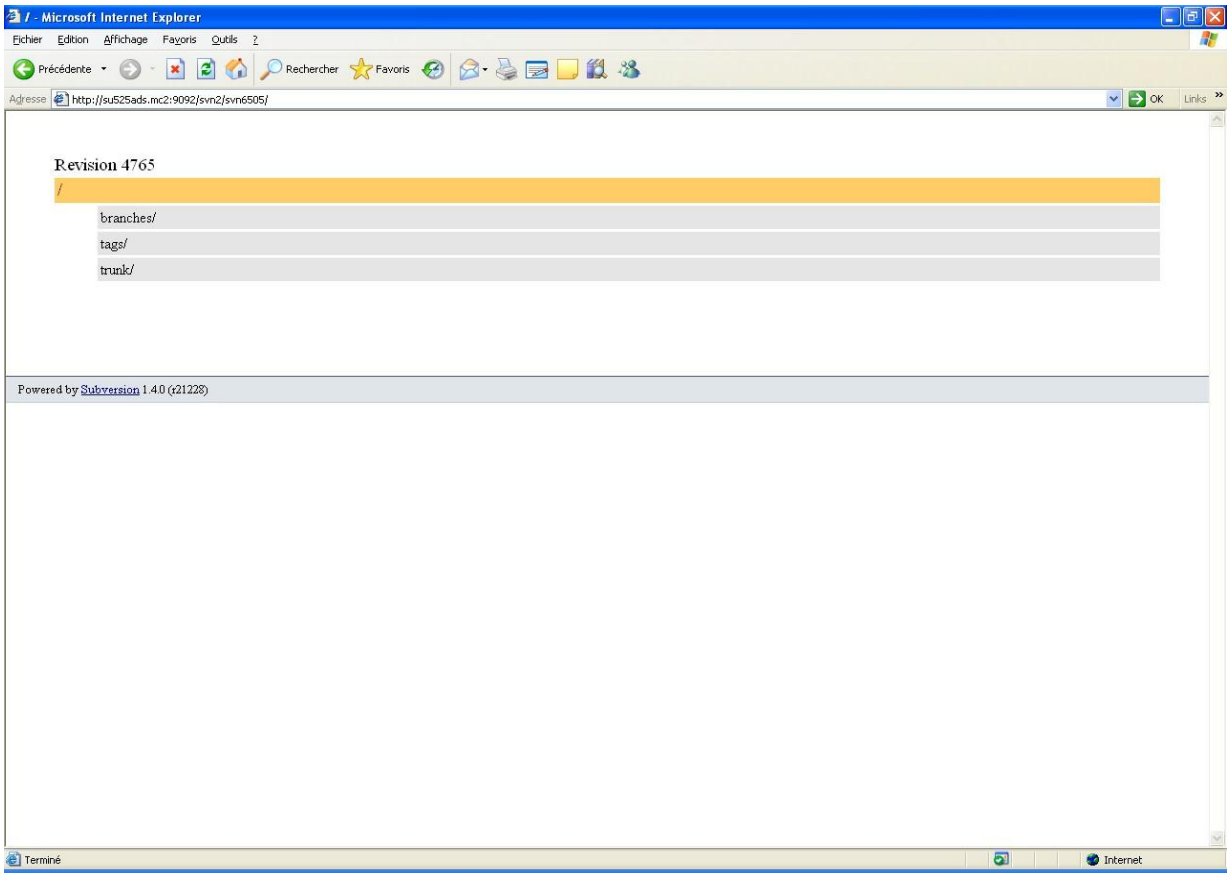


Figure 33 : Contenu d'un référentiel Subversion

On y voit :

- un numéro de révision (4765)
- le répertoire tronc (trunk en anglais)
- le répertoire des branches (branches)
- le répertoire des étiquettes (tags)

Le numéro de révision permet d'identifier l'état d'un référentiel. Chaque fois que le référentiel est modifié, le numéro de révision est incrémenté. Ici la dernière révision du référentiel est la révision 4765. Si on veut accéder à un état antérieur du référentiel, on précise le numéro de révision correspondant.

Le tronc contient la version de référence du projet. Dans ce dossier, on trouvera tous les fichiers sources du projet dans leur dernière version. C'est à partir du tronc que sont extraites les copies des fichiers sources sur lesquelles vont travailler les développeurs selon le modèle collaboratif « Extraction-Edition-Fusion » (voir 2.3.7.1).

Les branches contiennent des versions alternatives du tronc. On utilise des branches lorsqu'on veut publier des modifications sans modifier le tronc. Une fois que les travaux sont finis, on peut fusionner les branches avec le tronc pour avoir une nouvelle version du projet.

Les étiquettes correspondent à des photos du tronc. Lorsque le tronc est prêt à être livré, on crée une nouvelle étiquette auquel on rattache un numéro de version. Grâce à cette étiquette, on peut faire revenir le tronc à la version correspondant à cette étiquette. Au moment de la livraison, l'équipe de développement va préciser l'étiquette ou le numéro de version correspondant. L'équipe en charge de la livraison utilisera le référentiel Subversion et l'étiquette de livraison pour extraire les fichiers à livrer.

2.4.2.4 La gestion des transferts

LDD+ permet de gérer les transferts. Cette gestion des livraisons existait déjà avec LDD et les commandes n'ont pas changé. Si on veut transférer un fichier source vers l'environnement de recette, on utilise la commande R. Si on veut transférer un fichier source vers l'environnement de production, on utilise la commande O. A l'inverse, Subversion ne gère pas les livraisons. C'est au projet d'extraire les fichiers à livrer et de les installer vers l'environnement voulu.

Nous allons maintenant aborder la troisième partie dans laquelle sont exposées les réalisations que j'ai accomplies dans le cadre du projet « Recette et déploiement de l'outil LDD+ ».

3 TROISIEME PARTIE : REALISATION DU PROJET RECETTE ET DEPLOIEMENT DE LDD+

Je commencerai par présenter la chronologie des principales étapes du projet. Puis je détaillerai chacun d’elles. Dans ce projet, j’ai réalisé la recette de LDD+. Puis nous avons procédé à la migration de LDD vers LDD+ pour les applications en environnement Mainframe. Durant ce déploiement, j’ai accompagné les projets et j’ai organisé des séances de formation.

3.1 Les objectifs du projet « Recette et Déploiement de LDD+ »

Le projet « Recette et Déploiement de LDD+ » qui fait l'objet de ce mémoire a duré 11 mois. Son objectif est de recetter et de déployer l'outil LDD+. Recetter consiste à tester LDD+ pour vérifier qu'il réalise les fonctions voulues et qu'il n'y ait pas d'anomalies. Déployer consiste à faire en sorte que les projets informatiques de Renault passent de l'utilisation de LDD à l'utilisation de LDD+.

Dans ce qui suit, la notion de projet que je vais utiliser diffère de la notion de projet objet du mémoire. Dans ce cas, un projet correspond à une équipe de développeurs chargés de s'occuper d'une application correspondant à une fonction de gestion donnée. Par exemple, le projet TIC correspond à l'alimentation en données clients de l'application de gestion de la relation client. Selon la nature de l'application gérée, le projet utilise ou non l'environnement Mainframe. Tous les projets qui utilisent l'environnement Mainframe utilisent LDD. Ils doivent donc dans le cadre du projet PRIMA remplacer LDD par LDD+. Cette opération de remplacement s'appelle la migration LDD-LDD+.

Dans ce qui suit, j'utiliserai aussi les notions de :

- projets pilotes. Ce sont les premiers projets à passer de LDD vers LDD+. Ils sont appelés pilotes car ils servent de « cobaye » pour les projets suivants.
- projets cibles. Les projets utilisant LDD étant trop nombreux, il a fallu sélectionner les projets prioritaires pour le passage de LDD vers LDD+. Les projets sélectionnés sont appelés projets cibles.

3.2 L'organisation du projet « Recette et Déploiement de LDD+ »

J'ai présenté l'équipe SOAC dans la présentation des participants du déploiement du projet PRIMA (voir 1.4.6). Pour le déploiement de LDD+, en plus de moi, il y a eu 2 autres personnes qui sont intervenues :

- Marie-Laure de Taffanel de mars 2007 à octobre 2007
- Gilles Marchand d'octobre 2007 à mars 2008

J'ai évalué que pour ce projet qui a duré 11 mois, à l'exception du premier mois où j'étais seul, il y a eu deux personnes à temps plein sur 10 mois. En prenant en compte un mois de congés, cela nous fait environ 1 + 2* 9 = 19 mois homme.

3.3 Les étapes du projet « Recette et Déploiement de LDD+ »

Le tableau suivant donne les principales étapes du projet avec en parallèle l'évolution des différentes versions de LDD+.

Tableau III : Chronologie du projet

Date	Etape du projet	Version LDD+
Fin janvier 2007	Arrivée dans l'équipe SOAC	V0 Version avec SCLM
Mi février 2007	Décision de reprendre le développement de LDD+	
Mars 2007	Recette de la nouvelle version de LDD+ Recensement des projets à migrer.	V1 Première version sans SCLM
Fin Mars 2007	Migration des projets pilotes	V2 Version corrigée suite à la recette
Juin 2007	Premières formations LDD+	V3 Version corrigée et améliorée
Aout 2007	Début du déploiement de LDD+	
Mi octobre 2007	Nouvelle procédure de migration	
18/12/2007	Migration GPI, dernier des projets cibles	

Comme pour LDD, LDD+ a été conçu et réalisé par Gérard Besson (voir 1.4.6) en CLIST et REXX. CLIST et REXX sont des langages spécifiques à l'environnement Mainframe pour développer des outils. La charge de développement est difficile à évaluer car Gérard Besson a beaucoup travaillé à partir de chez lui et en dehors des horaires de bureau.

La première version V0 de LDD+ a été développée durant l'année 2006 et s'appuyait sur SCLM. SCLM est un outil de gestion de version de fichiers fourni en standard dans l'environnement Mainframe. Cette version V0 n'utilisait pas encore les fichiers versionning tels qu'on les a décrits dans la présentation LDD+. Pour vérifier que cette version V0 était utilisable, Renault a procédé à des tests de migration. Ces tests simulaient la bascule d'un projet de LDD vers LDD+. Ils consistaient à créer les premières versions des fichiers sources. Renault voulait évaluer les durées de migration sur un échantillon de fichiers sources donnés.

Voici les résultats obtenus pour la version V0 : pour migrer 1000 fichiers sources, il fallait 6 heures. Donc pour le projet GPI qui contient 86000 fichiers sources, il aurait fallu geler les développements pendant 216 heures soit 22 jours. Cette durée était inacceptable. On a choisi le projet GPI comme base car c'est le plus gros projet en environnement Mainframe. Après des études d'optimisation, on a constaté que SCLM était l'élément limitant. Gérard Besson m'a expliqué que SCLM n'était pas optimisé et consommait beaucoup d'accès fichiers. Comme SCLM était un outil livré en standard avec l'environnement Mainframe, il était impossible d'intervenir dessus. C'est pourquoi il a été décidé mi février 2007 de redévelopper LDD+ en se passant de SCLM. Les versions d'un fichier source au lieu d'être gérés par SCLM seraient gérées directement par LDD+ dans un fichier versionning associé.

Même si Gérard Besson a pu s'appuyer sur les développements déjà faits pour la version V0, ce redéveloppement lui a demandé beaucoup d'efforts pour être prêt à temps. Le résultat de ce redéveloppement a été la version V1.

Dans la version V1 par rapport à la version V0, Gérard Besson a développé la gestion des versions dans des fichiers versionning en remplacement de SCLM, la gestion des fichiers baseline, la gestion des verrous. Les tests de migration faits avec la version V1, avec comme échantillon la totalité du projet GPI, soit 86000 fichiers ont donné une durée de migration de 3 heures. Cette durée est beaucoup plus faible que celle obtenue avec la version V0. Renault a donc gardé la version V1 de LDD+.

J'ai recetté la version V1. Suite aux anomalies trouvées lors de la recette et à mes remarques, Gérard Besson a corrigé et modifié la version V1 pour obtenir la version V2 (voir 3.4.3). Avec la version V2, nous avons migré trois projets pilotes vers LDD+ en nous ménageant une possibilité de retour arrière. Suite à la migration des projets pilotes et en ajoutant de nouvelles fonctionnalité, une nouvelle version V3 a été développée à partir de la version V2. C'est cette version V3 qui a été utilisée pour le déploiement des projets cibles. Les nouvelles fonctionnalités de la version V3 sont mineures, ce sont :

- L'extraction des versions par la commande ZE (voir 2.3.4.4)
- Le rajout dans le suivi des opérations (voir 2.3.5.3) la version des fichiers sources enregistrée par un Checkin et la version récupérée par un Checkout
- La commande administrateur de déverrouillage (voir 2.3.7.4)

3.4 La recette de LDD+

3.4.1 Les fonctions testées

La recette de l'outil LDD+ consiste à tester et valider son fonctionnement. Pour cela, j'ai élaboré les scénarios de test puis je les ai exécutés. En cas d'anomalies ou de limites constatées, je remontai les informations vers Gérard Besson pour correction et/ou évolution. On peut regretter que le développement de LDD+ ait été fait sans spécifications. Mais les délais extrêmement courts ne nous le permettaient pas. J'ai donc élaboré la recette en même temps que je découvrais LDD+. Les tests de recette ont été consignés dans un document EXCEL. La recette comportait quatre parties.

La première partie des tests concerne les fonctions de versionning. Il s'agissait de tester les fonctions :

- Edition et création d'un nouveau fichier source
- Checkin (commande ZI voir 2.3.4.1)
- Présence au préalable d'un verrou Checkout
- Affichage de l'écran de saisie du commentaire du Checkin
- Saisie du commentaire
- Abandon du Checkin
- Création du fichier de version et enregistrement de la première version
- Enregistrement de la version pour les versions suivantes
- Vérification des informations liées à la version
- Visualisation des versions (commande ZV voir 2.3.4.3)
- Affichage de la liste des versions
- Sélection d'une version
- Affichage de la version

- Checkout (commande ZO voir 2.3.4.2)
- Vérification de l'absence de verrou
- Affichage de la liste des versions
- Visualisation des versions comme pour ZV
- Sélection d'une version

La deuxième partie des tests concerne les fonctions de verrouillage. Contrairement aux fonctions de versionning qui existaient déjà dans la version V0, les fonctions de verrouillages ont été créées avec la version V1. Les tests étaient plus complexes car ils nécessitaient l'intervention de deux utilisateurs, l'un pour verrouiller, l'autre pour tester le verrou. Tester le verrou consiste à vérifier que le deuxième utilisateur ne puisse pas modifier le fichier source verrouillé par le premier utilisateur. Nous avons aussi vérifié le fonctionnement des verrous Lock (commandes ZL et ZU) et la visualisation des verrous (commande ZC).

La troisième partie des tests concerne les fonctions de transferts. Il s'agit de vérifier que les transferts fonctionnent et qu'ils étaient correctement enregistrés dans les baselines. La modification des procédures de transfert entre LDD et LDD+ a été très lourde. Il a fallu tester le transfert de tous les types d'objets un par un. Nous avons aussi testé l'accès aux baselines (commande ZB) pour pouvoir suivre les transferts.

La quatrième partie des tests concerne les fonctions restantes comme l'extraction des versions d'un fichier source dans une bibliothèque de travail (Commande ZE) et la commande de déverrouillage réservée aux administrateurs (commande UNLOCK).

3.4.2 Le détail des tests de recette

Pour faire les tests, j'ai d'abord élaboré des scénarios de tests. Un scénario correspond à un cas d'utilisation qui fait appel à une ou plusieurs fonctions. Dans un scénario, il y a une série de pas de tests. Chaque pas de test correspond à une action qui doit provoquer un résultat attendu. Si le résultat attendu n'est pas obtenue, une anomalie est créée et est remontée au développeur. Pour la recette LDD+, les scénarios de tests ont été consignés dans un document Excel. Dans la page suivante, on trouve deux exemples de scénario, un pour les fonctions de versionning sans anomalie constatée, un pour les fonctions de verrouillage avec une anomalie constatée.

Conditions initiales : Entrer dans un lot LDD+.

Création Checkin						
Libellé du pas de test	Testeur	Résultat attendu	Date d'exécution	Status	Anomalie Réf	Anomalie Résumé
Création d'un nouveau module	FLE	mire gcl + état ????	09/03/07	OK		
ZC	FLE	message d'erreur : état invalide	09/03/07	OK		
ZV	FLE	message d'erreur : état invalide	09/03/07	OK		
ZO	FLE	message d'erreur : état invalide	09/03/07	OK		
ZI	FLE	message d'erreur : état invalide	09/03/07	OK		
ZL	FLE	message d'erreur : état invalide	09/03/07	OK		
ZU	FLE	message d'erreur : état invalide	09/03/07	OK		
R	FLE	message d'erreur : incompatibilité stats	21/03/07	OK		
S	FLE	affichage du module vide	09/03/07	OK		
F3	FLE	mire gcl + état ????	09/03/07	OK		
ZC	FLE	message d'erreur : état invalide	09/03/07	OK		
ZV	FLE	message d'erreur : état invalide	09/03/07	OK		
ZO	FLE	message d'erreur : état invalide	09/03/07	OK		
ZI	FLE	message d'erreur : état invalide	09/03/07	OK		
ZL	FLE	message d'erreur : état invalide	09/03/07	OK		
ZU	FLE	message d'erreur : état invalide	09/03/07	OK		
R	FLE	message d'erreur : incompatibilité stats	21/03/07	OK		
S	FLE	affichage du module vide	09/03/07	OK		
modification	FLE	affichage du module modifié	09/03/07	OK		
F3	FLE	mire gcl + état TEST + verrou checkout	09/03/07	OK		
ZC	FLE	affichage liste des verrous **avec** module	09/03/07	OK		
F3	FLE	mire gcl	09/03/07	OK		
ZV	FLE	message d'erreur : absence de version	09/03/07	OK		
ZO	FLE	message d'erreur : verrou checkout par X	18/03/07	OK		
ZL	FLE	message d'erreur : verrou checkout par X	18/03/07	OK		
ZU	FLE	message d'erreur : verrou checkout par X	18/03/07	OK		
R	FLE	message d'erreur : incompatibilité stats	21/03/07	OK		
ZC	FLE	affichage liste des verrous **avec** module	09/03/07	OK		
F3	FLE	mire gcl	09/03/07	OK		
ZV	FLE	message d'erreur : absence de version	09/03/07	OK		
ZO	FLE	message d'erreur : verrou checkout par X	09/03/07	OK		
ZL	FLE	message d'erreur : verrou checkout par X	09/03/07	OK		

ZU	FLE	message d'erreur : verrou checkout par X	09/03/07	OK		
R	FLE	message d'erreur : incompatibilité stats	21/03/07	OK		
ZI	FLE	masque saisie commentaire	18/03/07	OK		
F3	FLE	mire gcl	18/03/07	OK		
ZI	FLE	masque saisie commentaire	18/03/07	OK		
saisie commentaire vide	FLE	message d'erreur : commentaire vide	09/03/07	OK		
saisie que des blancs	FLE	message d'erreur : commentaire vide	09/03/07	OK		
saisie "??"	FLE	checkin : mire gcl	09/03/07	OK		
ZC	FLE	affichage liste des verrous **sans** module	09/03/07	OK		
F3	FLE	mire gcl	09/03/07	OK		
ZV	FLE	affichage liste version + version 1.00 + "??"	09/03/07	OK		
S	FLE	Affichage de la version 1.00	09/03/07	OK		
F3	FLE	affichage liste version + version 1.00 + "??"	09/03/07	OK		
F3	FLE	mire gcl	09/03/07	OK		
R	FLE	mire gcl + TR	21/03/07	OK		
A	FLE	mire gcl	21/03/07	OK		

Figure 34 : Scénario de test « Création Checkin ».

Les pas de tests marqués en gris sont des pas de tests marqués pour les explications qui suivent.
Certaines pas de test ont été rajoutés après coup car on n'y avait pas pensé avant d'où une date d'exécution plus tardive.

On indique d'abord les conditions initiales pour exécuter le scénario. Puis on trouve le nom du scénario. Dans le premier exemple présenté dans la Figure 34, le scénario Création Checkin permet de tester la création d'un fichier source et l'enregistrement d'une version un Checkin. Dans ce scénario, on commence par créer un fichier source. Puis on teste une série de commandes qui doivent provoquer des messages d'erreurs. On parle dans ce cas de tests aux limites qui permettent de vérifier que les commandes qui ne sont pas en condition d'être utilisées sont correctement bloquées. On vérifie ainsi que LDD+ résiste à des erreurs de saisies.

Une fois la modification faite (pas de test F3 marqué en gris), le fichier source est verrouillé par un verrou Checkout. Après avoir verrouillé l'existence du verrou Checkout avec la commande ZC et refait un test aux limites, on procède à un Checkin (pas de test ZI marqué en gris). Au moment du Checkin, nous avons aussi fait un test aux limites en essayant de ne rien saisir ou de ne saisir que des blancs. Dans ces deux cas, un message d'erreur apparait en nous demandant de refaire une saisie. Une fois que le Checkin est fait, on vérifie que le verrou Checkout a été levé (pas de test ZC marqué en gris) et que la version a été correctement créée (pas de test ZV marqué en gris).

Certaines pas de test ont été rajoutés après coup car on n'y avait pas pensé avant d'où une date d'exécution plus tardive. Ici les pas de tests faisant appel à la commande de transfert R ont été exécutées le 21/03/2007.

Conditions initiales : accès à LDD+, mire gcl, repérer un module non verrouillé, User1 et User 2 travaillent sur 2 lots différents

User1 Checkout-Checkin						
Libellé du pas de test	Testeur	Résultat attendu	Date d'exécution	Status	Anomalie Réf	Anomalie Résumé
User 1 ZO	GBN	affichage liste version	14/03/07	OK		
User 1 S	GBN	checkout : mire gcl	14/03/07	OK		
User 2 ZC	FLE	affichage liste des verrous --avec-- module	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZV	FLE	affichage liste version	14/03/07	OK		
User 2 S	FLE	affichage de la version	14/03/07	OK		
User 2 F3	FLE	affichage liste version	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZO	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZI	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZL	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZU	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 R	FLE	message d'erreur : module verrouillé par X	22/03/07	OK		
User 2 S	FLE	message d'erreur : module verrouillé par X	14/03/07	KO OK	Anomalie 01	Accès au module alors que le module est verrouillé par un autre user Anomalie corrigée le jour même
User 1 S	GBN	affichage de la version	14/03/07	OK		
User 2 ZC	FLE	affichage liste des verrous --avec-- module	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZV	FLE	affichage liste version	14/03/07	OK		
User 2 S	FLE	affichage de la version	14/03/07	OK		
User 2 F3	FLE	affichage liste version	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZO	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZI	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZL	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZU	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 R	FLE	message d'erreur : module verrouillé par X	22/03/07	OK		
User 2 S	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 1 Modification	GBN	affichage de la version modifiée	14/03/07	OK		
User 1 F3	GBN	mire gcl	14/03/07	OK		
User 2 ZC	FLE	affichage liste des verrous --avec-- module	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		

User 2 ZV	FLE	affichage liste version	14/03/07	OK		
User 2 S	FLE	affichage de la version	14/03/07	OK		
User 2 F3	FLE	affichage liste version	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZO	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZI	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZL	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 ZU	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 2 R	FLE	message d'erreur : module verrouillé par X	22/03/07	OK		
User 2 S	FLE	message d'erreur : module verrouillé par X	14/03/07	OK		
User 1 ZI	GBN	masque saisie commentaire	14/03/07	OK		
User 1 saisie commentaire	GBN	checkin : mire gcl (retour au même module)	14/03/07	OK		
User 2 ZC	FLE	affichage liste des verrous --sans-- module	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZV	FLE	affichage liste version + version +1	14/03/07	OK		
User 2 S	FLE	affichage de la version + 1	14/03/07	OK		
User 2 F3	FLE	affichage liste version	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZO	FLE	affichage liste version + version +1	14/03/07	OK		
User 2 F3	FLE	mire gcl	14/03/07	OK		
User 2 ZL	FLE	affichage liste des verrous --avec-- module	14/03/07	OK		
User 2 ZU	FLE	affichage liste des verrous --sans-- module	14/03/07	OK		

Figure 35 : Scénario de test « User 1 Checkout Checkin »

Les pas de tests marqués User 1 sont les pas de test exécutés par Gérard Besson.

Certaines pas de test ont été rajoutés après coup car on n'y avait pas pensé avant d'où une date d'exécution plus tardive.

Le deuxième exemple présenté dans la Figure 35 est le scénario « Checkout Checkin ». Ce scénario permet de tester les accès concurrents. C'est pourquoi, il faut 2 testeurs User1 et User2. User1 pose un verrou Checkout par la commande de Checkout ZO. Gérard Besson a joué le rôle d'User1. J'ai ensuite joué le rôle d'User2 pour vérifier qu'il ne pouvait pas modifier le fichier source à cause du blocage par le verrou Checkout. Une fois ces vérifications faites, User1 modifie le fichier source. User2 vérifie alors qu'il est toujours bloqué par le verrou Checkout. Une fois ces vérifications faites, User1 fait un Checkin du fichier source par la commande de Checkin ZI. Ce Checkin lève le verrou Checkout. User2 vérifie alors qu'il peut modifier le fichier source.

Les tests de recette ont été effectués dans l'environnement de développement. Comme avec les premiers tests, les temps de réaction de LDD+ étaient satisfaisants, il n'a pas été fait de test de performance.

3.4.3 Les résultats de la recette

La recette des fonctions de versionning n'ont pas produit d'anomalies. Les anomalies ont été découvertes surtout lors des tests d'accès concurrents : l'utilisateur qui était censé être bloqué par un verrou, pouvait toujours modifier le fichier source. Suite à la recette de l'outil LDD+, nous avons constaté la nécessité d'évolutions pour faciliter l'utilisation de l'outil.

La première évolution concerne le moment du verrouillage. Dans la version V1 (voir 3.3), le verrou Checkout n'était posé que si le fichier source était modifié. Au début cela semblait suffisant car le système Mainframe empêche deux utilisateurs d'intervenir en même temps sur le même fichier source. Avec les tests de recette on a constaté que cela n'était pas suffisant, car pendant qu'un utilisateur travaillait sur un fichier source, un autre pouvait poser un verrou sur ce fichier source par exemple avec la commande de verrouillage ZL (voir 2.3.7.4). Une fois le verrou posé, le premier utilisateur ne pouvait plus travailler sur le fichier source alors que c'était lui qui l'avait modifié. C'est pour cela qu'on a décidé que le verrou Checkout serait posé par défaut au moment où un utilisateur accède au fichier source. Tant que le fichier source est verrouillé, aucun autre utilisateur ne peut intervenir ou le verrouiller. Si le fichier source n'est pas modifié alors le verrou est levé quand l'utilisateur quitte le fichier source. Par contre le fichier source reste verrouillé par l'utilisateur si le fichier source est modifié. Dans ce cas la seule façon de lever le verrou est que cet utilisateur enregistre une nouvelle version du fichier source par Checkin.

La deuxième évolution consiste à créer un verrou Transfert. Ce verrou est posé au moment où le fichier source est transféré en recette ou en production pour éviter des interventions sur le fichier source pendant le processus de transfert. Sans ce verrou, on risquait de transférer un fichier source différent de la dernière version enregistrée. On perdait alors la traçabilité des versions.

Ces deux évolutions ont été réalisées par Gérard Besson dans la foulée de la recette. Je les ai testées et on a alors obtenu la version V2 de LDD+ qui a été déployée pour trois projets pilotes. La Figure 36 montre les différences entre la version V1 et la version V2.

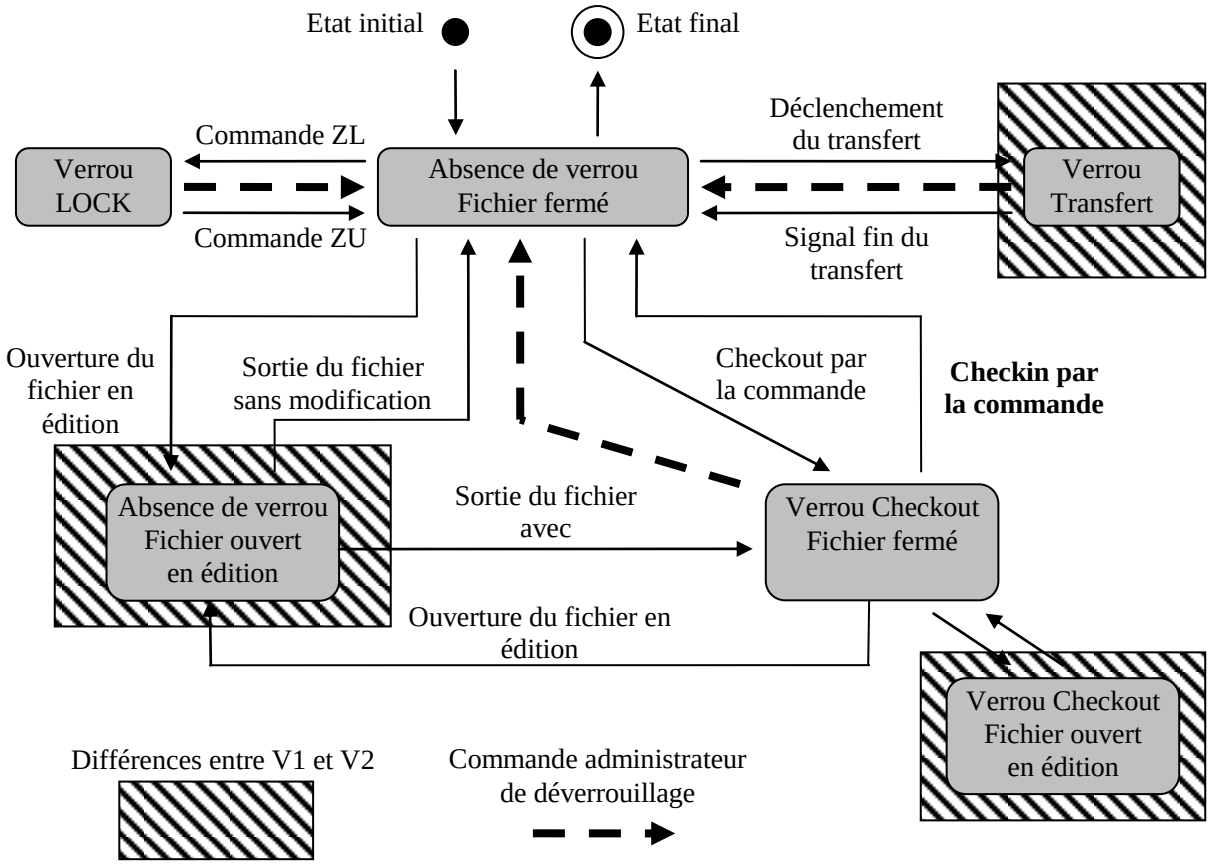


Figure 36 : Diagramme état-transition des verrous LDD+ dans la version V1

Par rapport à la Figure 32, il manque le verrou Transfert et le verrouillage Checkout dès l'ouverture en édition du fichier.

3.5 La migration des projets pilotes

Une migration consiste à remplacer LDD par LDD+ pour un projet donné. Une fois la recette terminée, nous avons migré 3 projets pilotes qui ont servi de cobayes. Les projets informatiques Renault sont regroupés en 3 domaines : le domaine GRM orienté gestion, le domaine COM orienté commerce, le domaine FLI orienté production. Chaque responsable de domaine a choisi un projet pilote. Les 3 projets choisis sont : PRF (GRM), PVS (FLI), TIC (COM). Ces projets devaient avoir une activité ni trop forte car la migration les aurait gêné dans leur planning, ni trop faible pour que l'équipe projet puisse utiliser LDD+ dans des conditions normales.

Ces projets pilotes nous ont permis :

- d'élaborer et de tester la procédure de migration de LDD vers LDD+
- d'élaborer les sessions de formation à l'utilisation de LDD+
- de détecter et de corriger les anomalies qui n'avaient pas été détectées pendant la recette.
- d'anticiper les difficultés que les équipes projet pourraient rencontrer dans l'utilisation de LDD+

Les trois projets pilotes ont migré fin mars 2007. Il n'y a pas eu de grosses difficultés dans la migration et dans la prise en main de l'outil. Des anomalies sont apparues plus tard au moment où les projets pilotes ont commencé à livrer en production. C'étaient des anomalies sur les transferts sur la mise à jour des bibliothèques de versionning et de la baseline de production. Ces anomalies ont été provoquées par une activation incomplète des paramètres de bascule de LDD vers LDD+. Cet oubli n'avait pas pu être détecté au moment de la recette car nous n'y avions pas testé de migration. Cet oubli a été corrigé par la formalisation des procédures de migration et en y incluant des tests de transfert.

3.6 Le périmètre de déploiement

Le déploiement de LDD+ consiste à remplacer LDD par LDD+ pour les projets utilisant l'environnement Mainframe. Pour un projet on parle de migration de LDD vers LDD+. Chez Renault, on a entrepris de recenser tous ses projets informatiques. Puis on les a classés en trois catégories selon leur criticité :

- les projets critiques ; un projet critique est tel que son arrêt a un impact direct sur l'activité de l'entreprise, exemple la gestion des usines de production.
- les projets stratégiques ; un projet stratégique est tel que son arrêt a un impact important sur les résultats financiers de l'entreprise. Ils sont donc particulièrement surveillés par la direction, exemple la gestion des concessionnaires.
- les projets standards ; un projet standard est un projet qui ne relève pas des deux précédentes catégories.

Dans le cadre du projet PRIMA et pour l'année 2007, les projets critiques et stratégiques étaient prioritaires. Ils ont donc constitué l'ensemble des projets cibles. Chez Renault, chaque projet informatique est identifié par un nom et un IRN (Identifiant Renault Normalisé). L'IRN est un numéro d'identification utilisé en particulier par la comptabilité. Donc comme point de départ, nous avions la liste des 302 projets cibles et de leurs IRN. Puis nous avons recherché pour chacun de ces projets un correspondant pour leur demander s'ils utilisaient l'environnement Mainframe. Au total, nous avons eu 63 projets mainframe à migrer dont 22 étaient critiques et 5 stratégiques.

Nous avons vu dans les principes de LDD (voir 2.2.1) que les projets qui utilisent l'environnement Mainframe sont identifiés par leur SIA. Un SIA (Système d'Information Autonome) est un code à 3 lettres. Ce code est utilisé dans le nommage des bibliothèques du projet Mainframe. Un projet Mainframe est donc identifié par 2 codes : l'IRN et le SIA. Dans la grande majorité des cas, un IRN correspond à un SIA. Par exemple pour le projet TIC (IRN 12031) correspond le SIA CRM. Exceptionnellement, il y a quelques cas où un IRN est associé à plusieurs SIA. Cela correspond à de très gros projets ou à des fusions de projet. De même il y a des cas où un SIA correspond à plusieurs IRN. Cela correspond à des traitements Mainframe communs à plusieurs projets ou à des séparations de projets sans qu'il y ait eu de séparation effective dans l'environnement Mainframe.

Au niveau communication, le déploiement de LDD+ était suivi par l'IRN des projets. Au niveau technique, le déploiement de LDD+ était suivi par le SIA. Le premier travail du déploiement a donc été de recenser les IRN et les SIA des projets et d'en établir les correspondances. Aux 63 IRN des projets Mainframe à migrer, nous avons établi une liste de 51 SIA à migrer. Ce recensement a été fait par Marie-Laure de Taffanel en mars 2007 (voir 1.4.6).

Le déploiement de LDD+ se fait par la migration des SIA. Dans le cas où un projet possède plusieurs SIA, on procédera à la migration SIA par SIA. On évitera néanmoins qu'il y ait trop d'écart entre 2 migrations pour éviter à l'équipe projet de jongler entre les 2 outils. Dans le cas où un SIA correspond à plusieurs projets, on contactera tous les projets pour qu'ils se mettent d'accord sur la date de migration. A cette date, la migration du SIA impactera tous les projets qui l'utilisent.

Le recensement a été utile pour faire un état des lieus des SIA encore actifs. En effet, certains SIA ont été abandonnés soit parce qu'ils étaient remplacés par d'autres SIA soit parce que le projet avait changé de technologie.

3.7 Les étapes d'une migration d'un projet

Il faut se souvenir que la migration d'un projet s'inscrit dans le déploiement du processus GCL (Gestion de Configuration Logicielle) (voir 1.3 et 1.4). Ce déploiement fait intervenir 3 équipes :

- l'équipe projet responsable du projet
- l'équipe VPMQ (voir 1.4.6) qui va accompagner l'équipe projet dans la mise en place du processus Gestion de configuration logicielle
- l'équipe SOAC (voir 1.4.6). Dans notre cas, il s'agit de l'équipe LDD+ dont je fais partie.

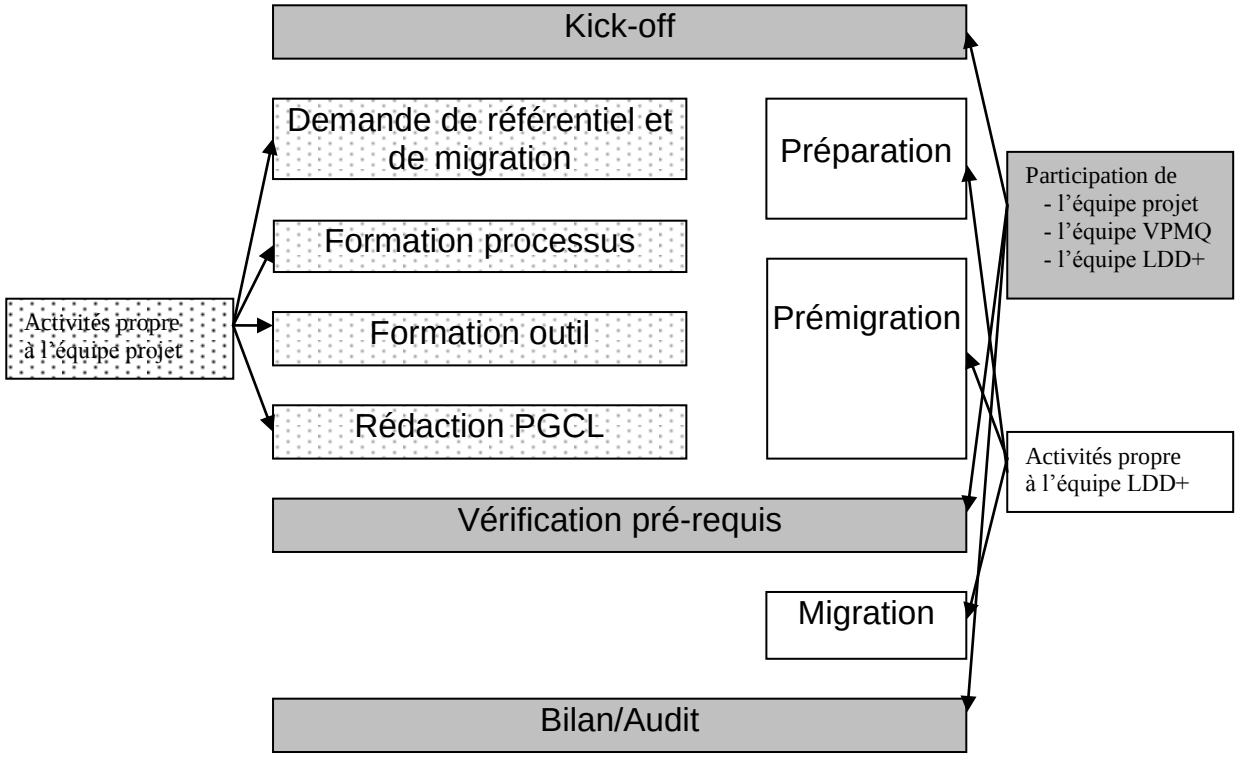


Figure 37 : Les étapes de la migration d'un projet

Les étapes de l'équipe LDD+ sont en blanc. Les étapes de l'équipe projet sont en pointillés. Les étapes de l'équipe VPMQ avec les 2 autres sont en gris.

Les étapes du déploiement du processus GCL sont présentées dans la Figure 37. Ces étapes sont les mêmes quel que soit l'outil GCL qui sera utilisé. Ici nous parlerons de LDD+ car c'est l'outil sur lequel porte ce mémoire. Les étapes peuvent se regrouper en trois catégories :

- les étapes qui font intervenir toutes les équipes (étapes en gris clair). Ce sont les étapes de kick-off, de vérification des pré-requis, de bilan/audit de migration.
- les étapes à la charge de l'équipe projet (étapes avec motif en pointillé) : demande de référentiel, formation au processus, formation à l'outil, rédaction du PGCL, plan de configuration logicielle.
- les étapes à la charge de l'équipe LDD+ (étapes en blanc) : préparation, prémigration, migration. Il s'agit d'interventions techniques.

Le kick-off est la réunion de démarrage qui réunit les 3 équipes. L'équipe VPMQ présente le processus GCL que l'équipe projet doit mettre en place et appliquer. Elle présente aussi les étapes de la migration et les planifie avec l'équipe projet. L'équipe LDD+ explique les nouveautés de LDD+ par rapport à LDD et les impacts de la migration. Les impacts sont de deux types :

- les impacts techniques : il faut expliquer l'apparition de nouvelles bibliothèques et de nouveaux fichiers et rassurer sur le fait que l'existant, les fichiers sources et les exécutables ne sont pas modifiés.
- les impacts méthodologiques : au-delà des nouvelles commandes, il faut expliquer les nouvelles façons de travailler. Nous avons bien insisté pour que la saisie du commentaire du Checkin soit bien faite pour pouvoir remonter à la demande à l'origine de la modification. C'est pourquoi, ce commentaire du Checkin est la principale chose qu'on vérifie quand on audite un projet pour vérifier qu'il applique correctement le processus GCL.

Après la réunion de kick-off, l'équipe projet fait une demande de création de référentiel et une demande de migration grâce à des formulaires dédiés. Une fois remplies, l'équipe projet envoie les formulaires à l'équipe LDD+. Elle suit ensuite deux formations, l'une sur le processus GCL, l'autre sur le ou les outils selon qu'ils utilisent Subversion, LDD+ ou les deux. L'équipe projet désigne le responsable de l'application du processus, le Responsable de Gestion de Configuration Logicielle (RGCL). Si l'équipe projet est grande et qu'elle possède un chef de projet, c'est lui qui est désigné comme RGCL sinon c'est l'une des personnes qui s'occupe des analyses. Le RGCL rédige ensuite le Plan de Gestion de Configuration Logicielle (PGCL). Ce document décrit comment le processus sera appliqué par le projet. Il décrira en particulier les objets gérés, les emplacements de stockages, la méthode de marquage des objets, l'outil utilisé LDD+ et/ou Subversion.

Pendant ce temps, l'équipe LDD+ prépare le ou les SIA pour la migration. C'est l'étape de préparation. Elle consiste à étudier l'existant du projet et à le préparer pour rendre les opérations suivantes possibles. Puis la phase de prémigration simule une migration du ou des SIA pour déterminer sa durée et anticiper d'éventuelles difficultés techniques.

Une fois que toutes ces opérations ont été menées à bien, les trois équipes se réunissent pour vérifier les pré-requis à la migration finale : un PGCL validé par l'équipe VPMQ, les formations suivies, la ou les prémigrations réussies. Les équipes choisissent alors une date de migration.

Quand arrive la date de migration, l'équipe LDD+ réalise la migration. L'équipe projet applique alors le processus GCL et utilise LDD+ pour ses développements. Au bout d'un certain temps, en général après la livraison en production d'une version majeure du projet, l'équipe VPMQ mène un audit pour vérifier que le processus est correctement appliqué tel qu'il est décrit dans le PGCL. C'est aussi l'occasion pour les équipes de faire un bilan du processus : constater une amélioration des méthodes de travail, relever les difficultés et les demandes d'évolution soumises par l'équipe projet.

3.8 Les sessions de formation

Nous avons organisés une vingtaine de sessions de formation, une douzaine par moi et le reste par mon collègue André Moreau. Elles ont eu lieu en général sur mon lieu de travail, le site Novadis de Renault. J'ai aussi organisé des sessions de formation à Bordeaux, Lyon et Douai et André à Courcouronnes, Meconsa et Valladolid, ces deux derniers lieux sont situés en Espagne.

Pour préparer ces sessions de formation, j'ai rédigé un support de formation sur Powerpoint, la check-list de préparation, la procédure de formation. Dans ce dernier document, je décrivais le matériel nécessaire, le déroulement de la formation et en particulier la préparation des machines et de la salle.

Une session de formation dure une demi-journée et elle s'adresse à des gens qui connaissent déjà LDD. Elle est composée de 2 parties : la première partie correspond à la présentation théorique de LDD+, la deuxième partie correspond à la partie pratique. Dans cette partie pratique, les personnes manipulent l'outil pour découvrir les nouvelles commandes. Pour cela un environnement de formation, appelé « bac à sable » a été mis en place. Cet environnement est accessible à toute personne ayant accès à l'environnement Mainframe. Il peut donc être réutilisé pour s'entraîner à reproduire les commandes étudiées pendant la formation. Pour la partie pratique, le support de formation reproduit des copies d'écran pour que l'utilisateur puisse facilement se retrouver dans l'interface.

Une session de formation peut accueillir au maximum 8 personnes. La formation se déroule dans une salle dédiée aux formations où les participants ont accès à un ordinateur. En général il y en a un par participant. Avant chaque session, le formateur prépare l'environnement. Au début de la session, je fais l'appel et je distribue le support de formation et le formulaire d'évaluation. Je m'appuie ensuite sur le support de formation sur Powerpoint que je projette à l'écran. Dans la partie pratique, les personnes manipulent sur des ordinateurs préparés pour l'occasion. A la fin de la formation, je réponds aux questions et je ramasse les formulaires d'évaluation remplis. Enfin je range la salle et j'éteins les ordinateurs. Une séance de formation dure environ 3 heures avec une pause entre la partie théorique et la partie pratique.

Toutes les personnes qui travaillent avec l'environnement Mainframe et qui donc vont utiliser LDD+ sont concernées par cette formation. Ce sont les analystes, les développeurs et les industrialisateurs. Ces derniers sont les personnes qui mettent en place les traitements et leur ordonnancement dans les environnements de recette et de production.

La première séance correspondant aux projets pilotes a été un peu difficile à cause du grand nombre de participants et d'un environnement défaillant. Les séances suivantes ont posé moins de problème. Les participants ont été satisfaits des sessions de formation. Les deux principales raisons de leur satisfaction sont :

- une partie pratique de la formation où ils manipulent eux-mêmes l'outil comme en situation de travail
- la faible différence entre LDD et LDD+. Parmi la dizaine de nouvelles commandes, il fallait surtout retenir la commande de Checkin car sans Checkin, il est impossible de transférer les fichiers sources.

Vous trouverez en annexe A un extrait du support de formation. Les 3 premières diapositives sont extraites de la partie théorique, les 3 suivantes de la partie pratique.

3.9 Les opérations menées par l'équipe LDD+

Nous allons détailler les étapes de la migration à la charge de l'équipe LDD+. Ce sont les étapes représentés en blanc dans la Figure 37. L'étape de préparation consiste étudier l'existant du projet et à le préparer pour rendre les opérations suivantes possibles. L'étape de prémigration simule une migration du ou des SIA pour déterminer sa durée et anticiper d'éventuelles difficultés techniques. L'étape de migration ajoute aux opérations de la prémigration des opérations qui rendront la migration définitive. Elle comporte aussi des tests pour vérifier que le SIA est totalement opérationnel.

3.9.1 Les outils de migration.

Toutes les opérations techniques décrites ont été réalisées grâce à des scripts générés par un menu spécifique de LDD+. Ce menu n'était accessible qu'à l'équipe LDD+. Dans la Figure 38, on voit le premier écran affiché une fois lancé LDD+. En haut de l'écran, la liste des choix accessibles est affichée. Parmi eux, il y a le choix N utilisé pour les opérations de migrations. Il n'est très visible à cause des couleurs employées. En dessous, il y a une zone de saisie dans laquelle on va saisir le choix voulu. Sous la zone de saisie, sont affichées des informations techniques.

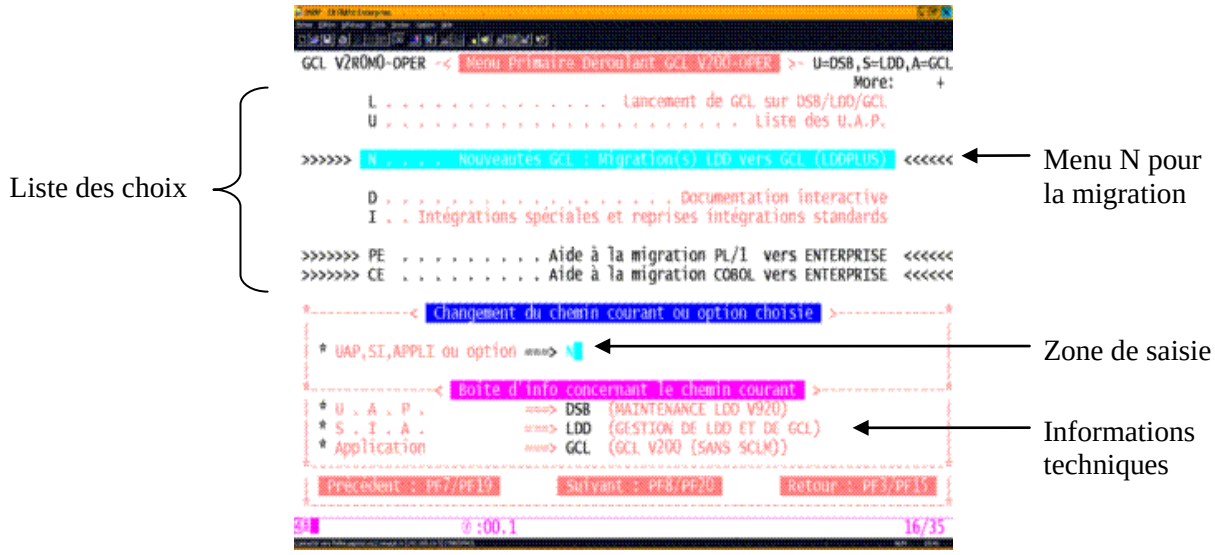


Figure 38 : Les outils de migration

On voit ici le premier écran de LDD+.
On accède au menu des opérations de migration par le choix N.

Dans la Figure 39, on voit l'écran affiché une fois que le choix N a été sélectionné. En haut de l'écran, la liste des choix accessibles est affichée. On y voit 7 choix MI1 à MI7. Tous les choix ne sont pas utilisés. On utilise surtout les choix MI1, MI2, MI4 et MI7. En dessous, on trouve une zone de saisie où on saisira le choix et le SIA qu'on voudra migrer. Une fois, les saisies effectuées, l'outil de migration va générer les scripts de traitement.



Figure 39 : Menu des opérations de migration

Tous les choix du menu ne sont pas utilisés.
On utilise MI1, MI2, MI4 et MI7. Avec la simplification de la procédure de préparation, on utilise essentiellement MI4 au lieu de MI3.

Dans la Figure 40, on voit un exemple de script généré. Du fait des restrictions d'affichage dans l'environnement Mainframe, on ne voit ici qu'une petite partie de ce script. Pour exécuter le script, on saisie la commande SUB dans la zone de saisie Command.

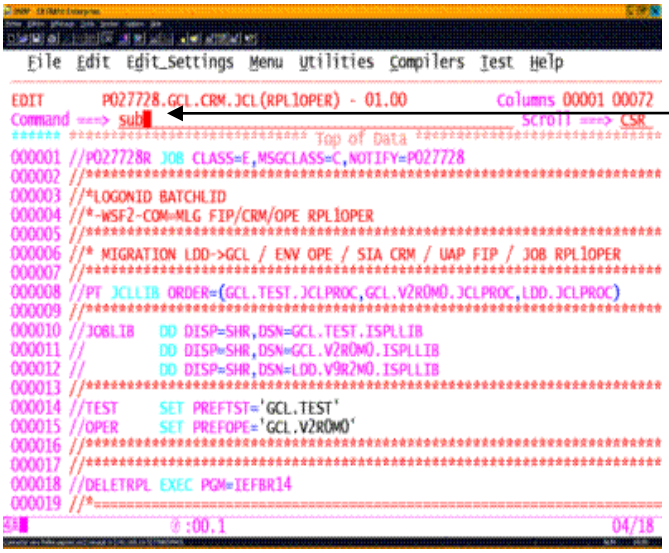


Figure 40 : Exemple de script généré par le menu

Pour déclencher le script, on lance la commande SUB.

3.9.2 Rappel sur l'architecture de LDD+

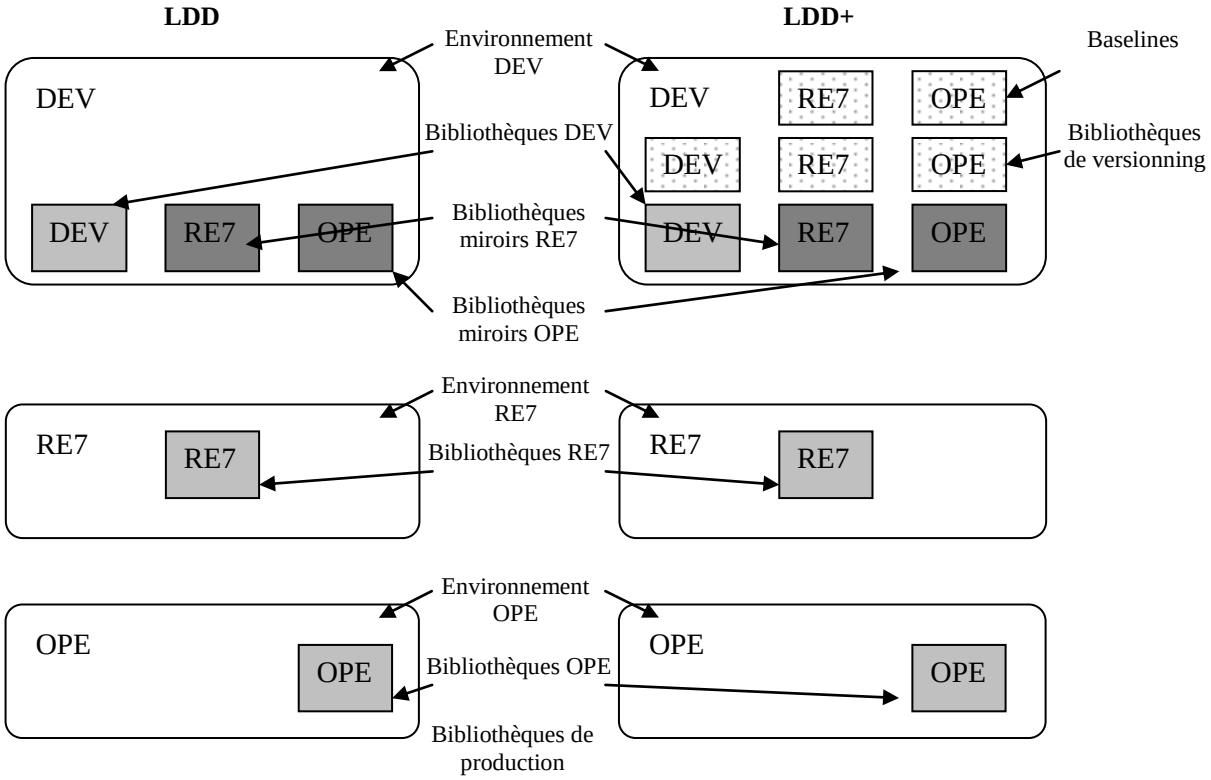


Figure 41 : Comparaison de l'architecture LDD/LDD+

Par rapport à LDD, LDD+ possède en plus les bibliothèques de versionning et les fichiers baseline.

Dans la Figure 41, nous comparons l'architecture de LDD+ avec celle de LDD. Cela nous permettra de mieux comprendre l'aspect technique d'une migration de LDD vers LDD+. En comparant LDD+ par rapport à LDD, nous constatons que les différences se situent uniquement dans l'environnement de développement. Il y a deux types de fichiers supplémentaires :

- les bibliothèques de versionning qui vont stocker les différentes versions des fichiers sources
- les fichiers baselines qui vont enregistrer les transferts entre les différents environnements. Il n'y a pas de fichier baseline pour le développement car il n'y a pas de transfert vers l'environnement de développement.

Nous en déduisons deux points importants :

- il n'y a pas de changement au niveau des environnements de recette et de production. C'est très important car cela signifie que la migration de LDD vers LDD+ n'a pas d'impact ni sur l'environnement de recette ni sur l'environnement de production. Cela a rassuré les équipes projet car elles n'avaient pas besoin de procéder à des relivraisons ni en environnement de recette ni en environnement de production. Et cette absence de relivraison signifie pour eux pas de risques opérationnels.
- la migration de LDD vers LDD+ ne provoque pas de modification dans les fichiers sources. Il n'y a pas besoin de les adapter ni de les recompiler. L'intervention de l'équipe projet dans la migration est donc minimale.

3.9.3 L'étape de prémigration

Pour rappel, l'étape de prémigration se situe après l'étape de préparation et avant l'étape de migration (voir 3.7). C'est l'étape la plus délicate techniquement et qui prend le plus de temps. La prémigration consiste à créer et à remplir les bibliothèques de versionning et les fichiers baselines sans activer LDD+. Comme LDD+ n'est pas encore activé et que les fichiers créés ne sont pas utilisés par LDD, l'équipe projet ne subit aucun impact. Comme il n'y a pas d'impact pour l'équipe projet, on peut exécuter les traitements sans l'avertir et sans la gêner. La prémigration est exécutée avant la migration finale pour anticiper d'éventuelles difficultés et avoir une estimation de sa durée. C'est pour cela qu'on parle de prémigration.

Pressés par le temps et en absence d'un environnement de test adéquat, les opérations de prémigration n'ont pas été recettées comme pour LDD+. Elles ont été validées avec la migration des projets pilotes.

Les résultats de la prémigration sont consignés dans un compte-rendu de prémigration. Ce document est ensuite présenté à l'équipe projet lors de la planification de la migration définitive. L'information la plus importante est la durée des opérations car elle nous donne un aperçu de la durée de la migration finale. Vous trouverez un exemple de compte-rendu dans l'annexe B du mémoire.

Avant mi-octobre 2007, la prémigration nécessitait une étape de préparation complexe du SIA à migrer. A la mi-octobre 2007, nous ne pouvions plus garder cette étape de préparation à cause de sa durée et des réticences de l'équipe projet. J'ai alors proposé une simplification des opérations de préparation et de prémigration. Cette proposition a été acceptée et validée par Renault.

Dans ce qui suit, j'appellerai « ancienne étape de préparation et de prémigration » les opérations de préparation et de prémigration avant mi-octobre 2007 et « nouvelle étape de préparation et de prémigration » les opérations de préparation et de prémigration après mi-octobre 2007 qui prennent en compte mes propositions de simplification. Je commence par présenter l'ancienne étape de prémigration parce que cela nous permettra de comprendre l'ancienne étape de préparation. Je présenterai alors la nouvelle étape de préparation et de prémigration. Cette nouvelle étape de préparation et de prémigration grâce à sa simplicité nous a fait gagner beaucoup de temps.

3.9.4 L'ancienne étape de prémigration

L'ancienne étape de prémigration comporte 5 opérations.

La première opération consiste à extraire les fichiers sources RPL. RPL est un langage de programmation spécifique à Renault. Tous les projets Mainframe de Renault ne l'utilisent pas. L'une des particularités de RPL est que ses fichiers sources ne sont pas stockés dans des bibliothèques telles qu'elles ont été décrites dans la présentation de l'environnement Mainframe (voir 2.1.3). Donc pour pouvoir les lire et les utiliser dans la migration, il faut au préalable les extraire pour les mettre dans des bibliothèques.

La deuxième opération consiste à créer les fichiers de versionning de production (OPE). A partir du fichier source contenu dans la bibliothèque miroir de production, on crée la version 1.00 qui est stockée dans le fichier versionning de production qui est lui-même stocké dans la bibliothèque de versionning de production.

La troisième opération consiste à créer les fichiers de versionning de recette (RE7). A partir du fichier source contenu dans la bibliothèque miroir de recette, on crée la version 1.01 puis on recopie la version 1.00 à partir du fichier versionning de production. Le nouveau fichier versionning qui comporte les versions 1.01 et 1.00 est stocké dans la bibliothèque de versionning de recette. On stocke la version 1.00 parce qu'elle a été transférée vers l'environnement de recette avant d'avoir été transféré en environnement de production.

La quatrième opération consiste à créer les fichiers versionning de développement (DEV). A partir du fichier source contenu dans la bibliothèque de développement, on crée la version 1.02 puis on recopie les versions 1.01 et 1.00 à partir du fichier versionning de recette. Le nouveau fichier versionning qui comporte les versions 1.02, 1.01 et 1.00 est stocké dans la bibliothèque de versionning de développement.

Comme pour un Checkin, la création d'une version se fait par écriture d'un entête et recopie du fichier source (voir 2.3.4.1). Les figures suivantes montrent ces trois opérations de remplissage des bibliothèques de versionning. La Figure 42 montre la création du fichier versionning correspondant à l'environnement de production (OPE). Ce fichier versionning contient la version 1.00 obtenue à partir de la bibliothèque miroir OPE.

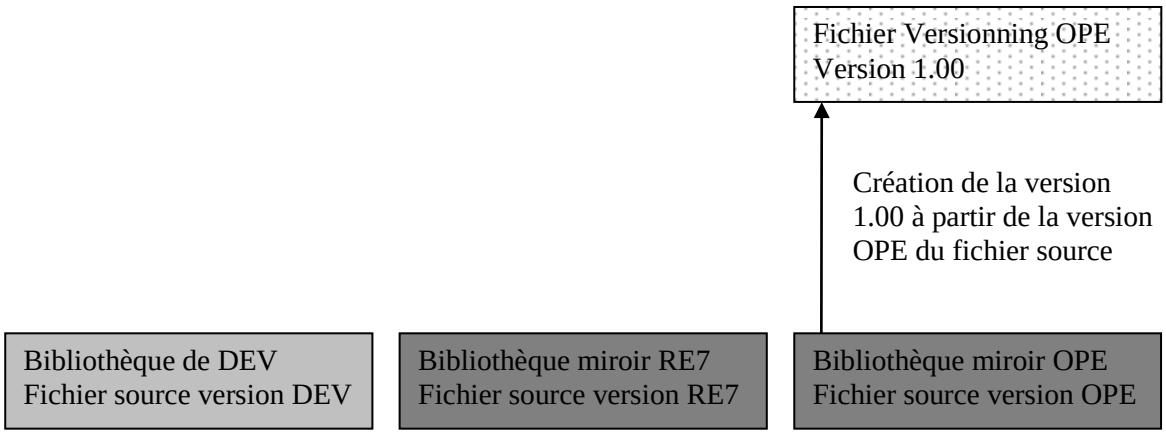


Figure 42 : Deuxième opération de l'ancienne étape de prémigration : création du fichier versionning OPE

La version 1.00 correspond à la version qui est dans la bibliothèque miroir OPE, ce qui correspond à la version de production.

La Figure 43 la création du fichier versionning correspondant à l'environnement de recette (RE7). Ce fichier versionning contient la version 1.00 et la version 1.01 obtenue à partir de la bibliothèque miroir RE7.

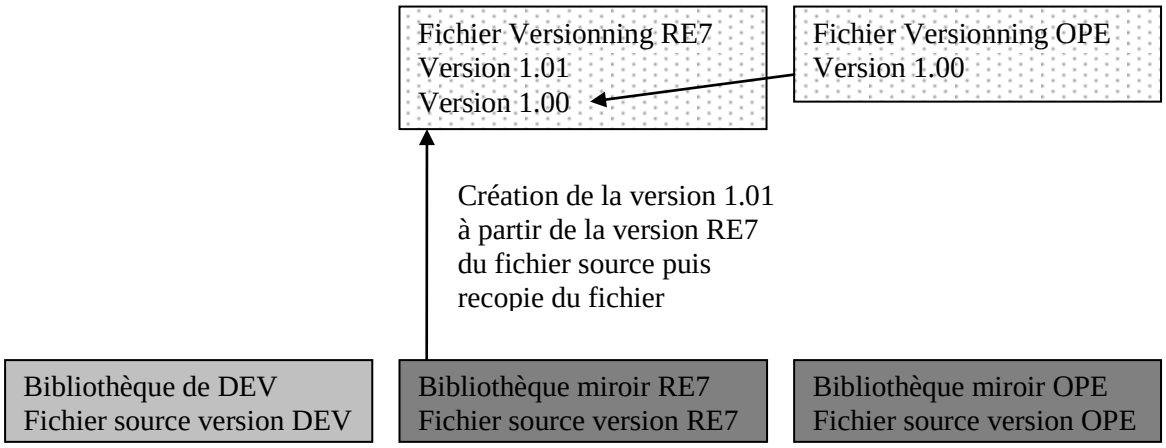


Figure 43 : Troisième opération de l'ancienne étape de prémigration : création du fichier versionning RE7

La version 1.01 correspond à la version qui est dans la bibliothèque miroir RE7, ce qui correspond à la version de recette.

La Figure 44 montre la création du fichier versionning correspondant à l'environnement de développement (DEV). Ce fichier versionning contient la version 1.00, la version 1.01 et la version 1.02 obtenue à partir de la bibliothèque de développement.

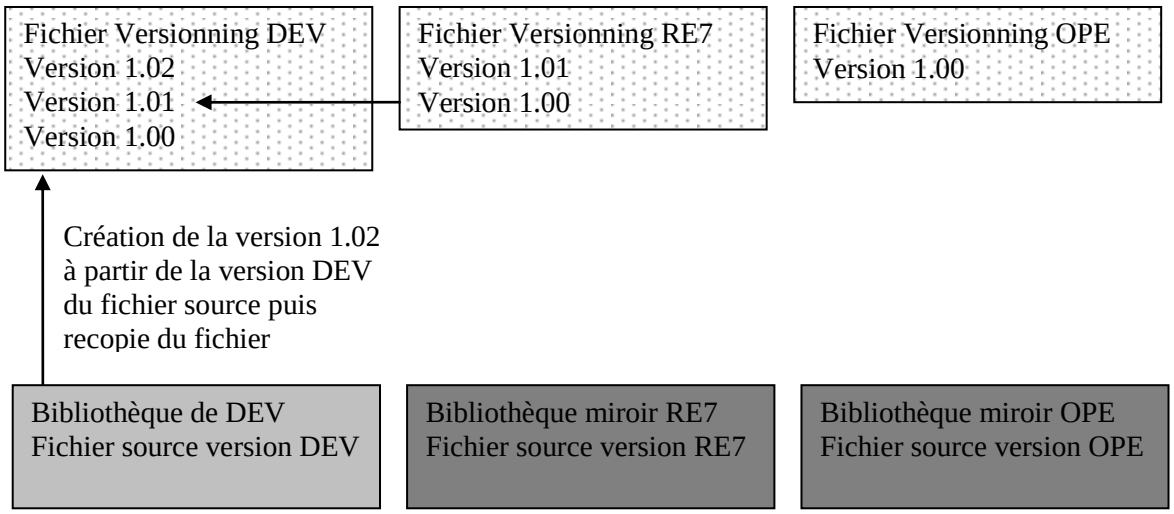


Figure 44 : Quatrième opération de l'ancienne étape de prémigration : création du fichier versionning DEV
La version 1.02 correspond à la version qui est dans la bibliothèque de développement, ce qui correspond à la version de développement.

La cinquième opération consiste à créer et à renseigner les fichiers baselines (voir 2.3.5.2). Pour le fichier baseline OPE, on créera un enregistrement pour chaque fichier versionné. Même chose pour le fichier baseline RE7. On obtient donc à la fin des opérations de prémigration dans chaque fichier baseline une photo de chaque environnement, c'est-à-dire la liste de tous les fichiers sources présents soit en OPE soit en RE7 avec un numéro de version.

3.9.5 L'ancienne étape de préparation

Dans la présentation de l'ancienne étape de prémigration (voir 3.9.4), les deuxième, troisième et quatrième opérations consistent à enregistrer des versions. Pour enregistrer les versions de production et de recette, il n'y a pas de problème car pour un SIA donné et pour un type de fichier source donné, la bibliothèque des fichiers sources est unique et son nom est codé de façon standard. Par contre ce n'est pas vrai pour l'environnement de développement : pour un type donné on peut avoir une ou plusieurs bibliothèques de développement pour un même SIA. Cette subdivision de SIA s'appelle dans le vocabulaire Renault une application. Un SIA est donc composé de plusieurs applications dont les bibliothèques de développement peuvent être distinctes. Il s'agit en général de vieux SIA. La Figure 45 présente la situation.

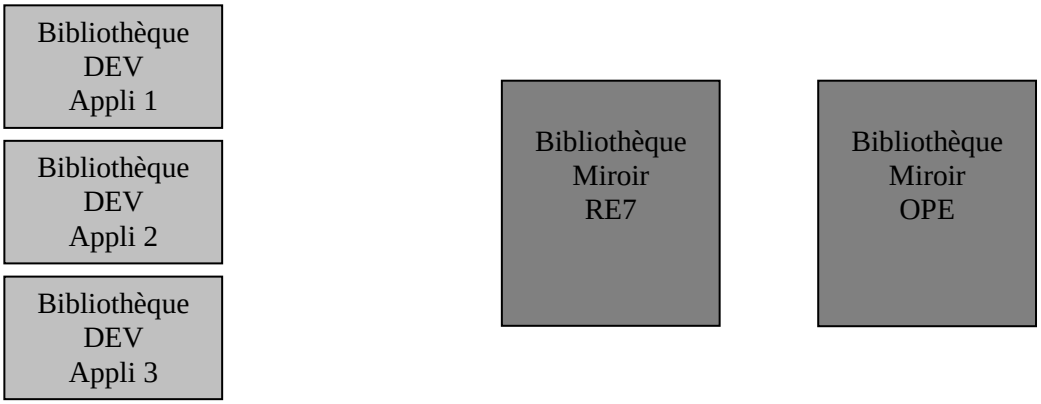


Figure 45 : Etat des bibliothèques des fichiers sources avant fusion
Les bibliothèques miroirs RE7 et OPE sont uniques mais pas les bibliothèques DEV de ce SIA qui comporte 3 applications.

Les opérations d'enregistrement de version ne fonctionnent qu'avec une unique bibliothèque de développement. C'est pour cela qu'avant la prémigration, il fallait fusionner les bibliothèques de développement dans une bibliothèque de développement unique. C'est l'étape de préparation. Dans cette fusion, si deux fichiers sources étaient situés dans deux bibliothèques de développement distinctes mais qu'ils portaient le même nom, alors on gardait la version la plus récente. La Figure 46 présente la situation après fusion.

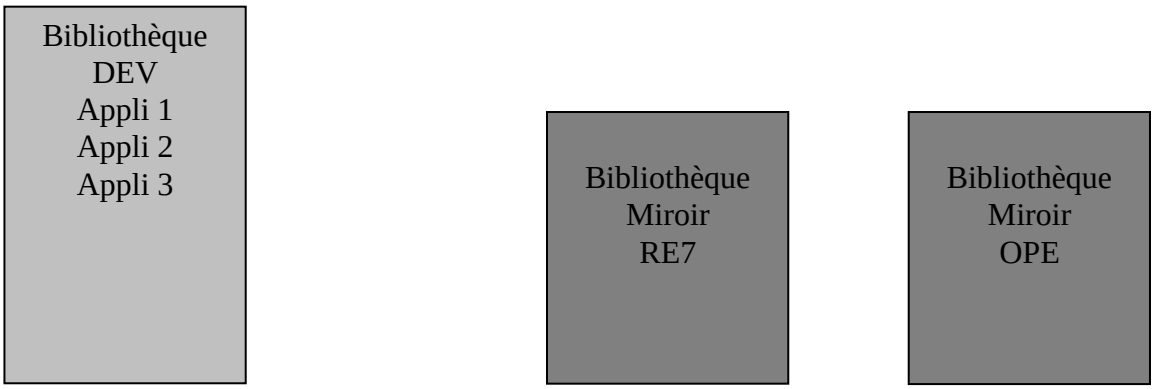


Figure 46 : Etat des bibliothèques des fichiers sources après fusion
La nouvelle bibliothèque DEV est unique et reprend
tous les fichiers sources des trois applications.

L'étape de préparation correspond à cette fusion des bibliothèques de développement. Cette fusion était délicate, longue et risquée. La fusion était délicate car elle devait se faire manuellement et les doublons devaient être traités au cas par cas. Pour ces doublons, on gardait en général le fichier le plus récent. La fusion était longue car nous étions obligés de recenser toutes les bibliothèques de développement. De plus, comme nous modifions les bibliothèques du projet, il fallait attendre une validation de l'équipe projet. Enfin la fusion était risquée car qui dit suppression de doublon dit risque de perte de données. Pour minimiser ces risques de perte, nous créions de nouvelles bibliothèques afin de conserver les anciennes bibliothèques. Il nous fallait ensuite les déclarer comme bibliothèque de travail. Ce remplacement de bibliothèques créait un risque supplémentaire car il pouvait perturber le travail de l'équipe projet.

3.9.6 Les nouvelles étapes de préparation et de prémigration

Au début du déploiement de LDD+, les fusions étaient rares et quand elles étaient nécessaires, elles étaient simples à réaliser avec deux ou trois bibliothèques à fusionner avec peu de doublons. On a ensuite rencontré des projets beaucoup plus complexes comme le SIA GAR. Ce SIA possède 24 applications. Nous pouvions nous attendre à une préparation longue et complexe. De plus quand nous avons présenté le processus de migration à l'équipe projet, celle-ci a catégoriquement refusé de fusionner les bibliothèques de développement car chaque application de ce SIA correspond à des développements spécifiques que l'équipe projet ne voulait pas voir mélangé. C'est pour cela que nous avons mis au point une nouvelle procédure de migration sans phase de préparation.

Après une longue réflexion et en discutant avec plusieurs projets, j'ai constaté que l'enregistrement de la version de développement d'un fichier source n'est pas obligatoire car cette version serait de toute façon enregistrée à la prochaine livraison. J'ai donc proposé de ne plus enregistrer la version de développement lors de la migration. Nous avons alors élaboré une procédure de migration sans enregistrement de la version de développement. Puisqu'on n'enregistrait plus la version de développement, l'unicité des bibliothèques de développement pour un même SIA n'était plus nécessaire. La phase de préparation devenait alors inutile et on gagnait sur deux aspects : les migrations prenaient moins de temps et les risques étaient diminués.

Dans la nouvelle étape de prémigration, les opérations 1, 2, 3 et 5 (voir 3.9.4) ne changent pas par rapport à l'ancienne étape de prémigration. Dans la nouvelle étape de prémigration, comme nous n'enregistrons plus la version de développement, nous ne recopions que le fichier versionning RE7 dans le fichier versionning de développement. La Figure 47 montre le principe de l'opération 4 de la nouvelle étape de prémigration.

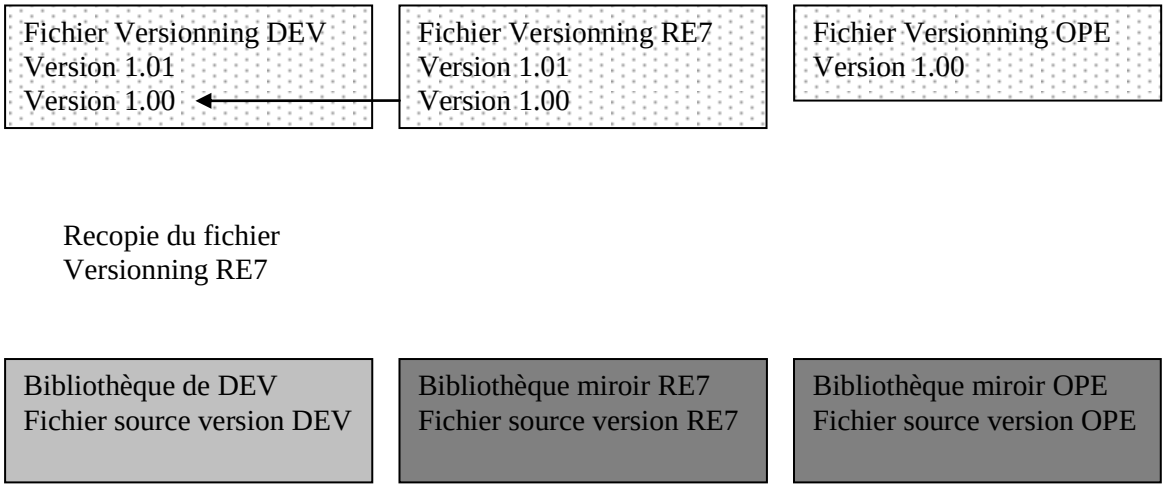


Figure 47 : Nouvelle étape de prémigration

Dans la quatrième opération, il n'y a plus que la copie du fichier versionning RE7. Il n'y a plus de création de la version 1.02 à partir du fichier source version DEV.

3.9.7 L'étape de migration

Dans les opérations de migration, il y a les opérations de prémigration qui enregistrent les fichiers sources dans les bibliothèques de versionning. Donc pendant une migration, il ne faut pas que le projet ne soit pas en train de modifier les fichiers sources car sinon nous ne sommes pas sûrs d'enregistrer la dernière version des fichiers sources. Une migration doit donc être planifiée soigneusement en accord avec l'équipe projet qui doit temporairement suspendre ses activités sur LDD. La durée de la prémigration nous permet d'évaluer cette durée de suspension. En général, les opérations de migration durent une demi-journée.

Une fois que la préparation et la prémigration effectuée, l'équipe LDD+ prend contact avec l'équipe projet pour se mettre d'accord sur une date de migration. Une fois celle-ci fixée, l'équipe LDD+ envoie un mail officiel de planification. Elle s'assure aussi que Gérard Besson est disponible à ce moment là car lui seul est autorisé pour finaliser la bascule de LDD vers LDD+. Voici le modèle du mail de planification.

PRIMA GCL : Planification du déploiement de LDD+ (SIA :) (IRN :)

Thème : GCL
Objet : Planification du déploiement de LDD+
Attente : Votre information

Synthèse :
1- Suite à votre demande, le déploiement de LDD+ sur le SIA référencé est planifié, conformément aux informations fournies dans le formulaire ci-joint.
2- Le début du déploiement commencera le xxxx à (Paris)
3- La préparation de ce déploiement a fait l'objet entre autre :
a- d'une analyse qualitative de l'existant
b- d'une simulation du déploiement (prémigration)
Les résultats sont consultables sur l'Intranet PRIMA (http://baselinesvn.mc2.renault.fr:9090/Portal/Documents/DeploymentLDD.html)
4- Nous vous rappelons que le déploiement concerne l'intégralité du SIA, et tous acteurs habilités à y intervenir
5- Pendant l'opération, Gérard BESSON devra être disponible
6- Le déploiement de LDD+ fera l'objet, en collaboration avec le projet, d'une série de contrôles qui seront consignés dans un PV à valider par le responsable du projet.
7- Le déploiement effectif de LDD+ sur le SIA sera notifié par mail.
Détails et attendus :
Votre participation au déploiement de LDD+
Description des pièces jointes :
PJ1: Formulaire de demande
PJ2: Compte-rendu de prémigration

Faire simple, agir : e-Mail formaté suivant les préconisations Simplicité & Vitesse

Lorsque le moment de la migration est arrivé, l'équipe LDD+ procède aux opérations :

- les opérations de prémigration
- la bascule LDD vers LDD+. La bascule bloque pour un SIA l'utilisation de LDD et impose l'utilisation de LDD+. Cela se fait en modifiant un fichier paramètre.
- la déclaration des administrateurs habilités à utiliser la commande UNLOCK. Cette commande permet à l'administrateur de lever un verrou même si ce n'est pas lui la personne à l'origine du verrou. Cela permet d'intervenir sur un fichier que quelqu'un d'autre a commencé à modifier. En général les administrateurs sont déclarés par SIA, mais certaines personnes comme Gérard Besson et l'équipe LDD+ étaient déclarés comme administrateurs sur tous les SIA grâce à un code spécial.
- la mise à jour de la procédure d'intégration. La procédure d'intégration est la procédure qui finalise le transfert des fichiers sources dans l'environnement de production. La mise de cette procédure à jour permettait à LDD+ de mettre à jour les fichiers versionning et les baselines lors des transferts en environnement de production. C'est la seule étape que l'équipe LDD+ ne pouvait exécuter elle-même. Il fallait qu'elle demande l'intervention de Gérard Besson car il était habilité à intervenir sur les environnements de production.

- le test de LDD+. Une fois LDD+ installé, nous procédions à un test d'utilisation. Ce test vérifie la gestion des versions et les transferts en recette et en production. Dans ce test d'utilisation, un transfert dans l'environnement de production est initialisé. Ce transfert ne sera finalisé que le lendemain lorsque la procédure d'intégration présentée précédemment s'est exécutée. Une fois le test terminé, l'équipe LDD+ prend contact avec l'équipe projet pour l'avertir qu'elle peut reprendre son travail sur le projet.

- la vérification de la procédure d'intégration. Cette vérification a été ajoutée systématiquement quand il est apparu des problèmes de transfert vers l'environnement de production pour les projets pilotes. Les projets pilotes ont été les premiers projets qui ont migré vers LDD+ en tant que projets « cobayes ». L'équipe projet vérifie que le transfert en production initialisé la veille s'est correctement déroulé avec l'exécution de la procédure d'intégration.

- l'envoi du mail officiel de fin de migration. Ce mail de fin de migration est accompagné d'un PV de validation. Dans ce PV, le test d'utilisation était décrit. L'équipe projet devait renvoyer ce PV pour constater l'absence ou non d'anomalie dans l'utilisation de LDD+. Si le PV n'était pas renvoyé sous 15 jours, la validation était acquise d'office. Vous trouverez un exemple de PV dans l'annexe C du mémoire. Cette validation automatique évitait à l'équipe LDD+ d'attendre et de relancer l'équipe projet. Voici le modèle du mail de fin de migration.

Thème : GCL
Objet : Livraison de LDD+
Attente : Votre information

Synthèse :
1- Suite à votre demande, le SIA référencé est maintenant géré sous LDD+.

2- Conformément au processus mis en place pour le déploiement de LDD+, les contrôles effectués sont décrits dans le PV ci-joint, à valider par le responsable du projet. En l'absence de réponse sous 15 jours, la validation sera considérée comme effective.

3- **Formation**
Les formations au processus de GCL et à la solution LDD+ sont disponibles sur le portail de GCL (http://www.intra.renault.fr/projet/prima/).
Un eLearning sur le processus de GCL est accessible (http://campus.mc2.renault.fr/default_fr.asp?IDFICHE=CRS_PRIMA)

4- **Contact et support**
Support à l'utilisation des outils : HelpDesk, F Le(01 76 84 72 29)
Support au processus de GCL: D. Chaniat pour Renault
M. Besson pour Atos
Support au déploiement des solutions de GCL: Contacter votre process champion
Détails et attendus :
Merci de me retourner le PV validé et de me faire part de toute remarque éventuelle
Description des pièces jointes :
PJ1: PV à Valider
Faire simple, agir : e-Mail formaté suivant les préconisations Simplicité & Vitesse

- la mise à jour du fichier EXCEL de suivi de la migration. Ce fichier est converti en page HTML pour publication. Vous trouverez un extrait de cette page HTML dans l'annexe D du mémoire.

Une fois la migration faite, l'équipe LDD+ restait disponible pour le support soit pour répondre aux questions de l'équipe projet soit pour intervenir en cas d'anomalie. Nous avons aussi prévu une procédure de retour arrière. Elle était utilisée si une anomalie bloquante était constatée suite à la migration ou si le projet avait besoin d'une intervention urgente pendant le processus de migration. Cette procédure permettait aussi de rassurer les équipes projets.

Du fait de la nature de la migration qui consiste essentiellement à créer et à remplir des fichiers non utilisés par LDD, cette procédure consistait à annuler la modification dans le fichier paramètre de bascule. Si nécessaire, on annulait la mise à jour de la procédure d'intégration. Une fois le blocage levé, on pouvait recommencer la migration. Cette procédure n'a été utilisée qu'une seule fois, quand nous avions constaté pour un SIA des particularités qui nécessitaient une adaptation de la procédure d'intégration. Ces particularités n'ont pas pu être détectées lors de la prémigration car celle-ci ne faisait pas intervenir sur la procédure d'intégration.

3.10 Le suivi du déploiement de LDD+

Le déploiement de LDD+ était suivi dans un fichier EXCEL. Pour chaque SIA, ce fichier EXCEL était mis à jour à deux moments :

- la première fois quand la prémigration est terminée (voir 3.9.3)
- la deuxième fois quand la migration est terminée (voir 3.9.7)

A chaque fois que le fichier EXCEL était modifié, il était converti en page HTML pour être publié. Vous trouverez un extrait de cette page HTML dans l'annexe D du mémoire. Il était très important que l'avancement du déploiement soit facilement accessible. Cela rassurait d'une part nos responsables hiérarchiques, d'autre part les équipes projets qui n'avaient pas encore migré vers LDD+. Ils pouvaient constater que le déploiement se déroulait sans souci et ils pouvaient contacter si nécessaire les équipes qui avaient déjà migré. Grâce à des liens hypertextes, on pouvait accéder au compte-rendu de prémigration et au PV de validation.

3.11 Le bilan du déploiement de LDD+

Tous les 51 SIA qui devaient migrer ont migré avant fin 2007. Seuls 3 projets ont fait l'objet d'une surveillance spécifique à cause de leur criticité, taille et complexité : GAR, GPI, DPR. Seul le SIA CKD a eu besoin d'un retour arrière avant la migration définitive à cause de ses spécificités techniques. Les principales interventions que j'ai du faire après la migration d'un SIA sont :

- Absence de bibliothèque de versionning : certaines bibliothèques de versionning n'étaient pas présentes après une migration. Elles ont été crées puis remplies manuellement. Cela est apparu quand l'équipe projet utilise un type de fichier source qui n'était pas encore utilisé au moment de la migration.

- Problème de Checkin : certains utilisateurs ne pouvaient pas faire de Checkin sur un fichier source alors qu'il était modifié. Cela concernait des fichiers sources qui avaient été modifié en dehors de LDD+, exemple par un traitement de génération. Ces personnes ne pouvaient pas faire de Checkin à cause de l'absence au préalable de verrou Checkout. Pour poser ce verrou, il faut d'abord sauvegarder le fichier source dans LDD+.

- Bibliothèque de versionning trop petite. Dans certains cas, en particulier pour les programmes, la taille choisie par défaut des bibliothèques de versionning était trop petite. On a donc agrandi ces bibliothèques manuellement selon les besoins.

- Explication de la commande UNLOCK et déclaration des personnes autorisées à l'utiliser. Ce sont des personnes qui ont besoin de lever des verrous posés par d'autres personnes.

- Problème de transfert d'un programme DB2. DB2 est un Système de Gestion de Bases de Données (SGDB). Un SGDB permet de stocker et de restituer des données sans que l'utilisateur n'ait à s'occuper de la gestion physique de ces données comme par exemple le nom et l'emplacement des fichiers qui les contiennent. Quand on parle de « programme DB2 », on parle d'un programme capable d'accéder au SGBD pour lire ou mettre à jour des données. Quand un programme DB2 est transféré, le fichier source de type PACK (voir 2.2.1) qui lui est associé est transféré automatiquement. Pour que ce fichier source PACK soit transféré, il faut qu'il soit versionné correctement. Un problème de transfert apparaissait quand l'utilisateur n'avait pas correctement versionné le fichier source de type PACK au moment du transfert du programme.

D'après les avis des équipes projets, LDD+ a été facilement accepté par les équipes projets pour deux principales raisons :

- sa fiabilité (on n'a pas eu d'incident bloquant en 2008) et des temps de réponses satisfaisants
- son utilisation pratique très proche de LDD car la seule commande obligatoire à retenir est la commande de Checkin. LDD+ ne changeait pas beaucoup les habitudes des développeurs.

LDD+ apporte aussi des informations supplémentaires qui peuvent faciliter les analyses :

- un historique des versions pour chaque fichier source
- un historique des livraisons

LDD+ apporte également un système de verrou qui permet de mieux gérer les développements concurrents.

Les industrialisateurs (voir 3.8) ont réclamé un Checkin de masse. Chaque fichier source qui doit être transféré doit être enregistré par un Checkin. A chaque Checkin, il faut saisir un commentaire qui justifie la modification. De plus pour certains objets, on ne peut pas faire de Checkin directement, il faut d'abord sauvegarder le fichier source pour mettre en place un verrou Checkout. Si on a qu'une vingtaine de fichiers sources à livrer, ce genre d'opération (sauvegarder, Checkin avec saisie du justificatif fichier source par fichier source) n'est pas trop gênant, mais si on a plusieurs centaines d'objets cela devient très gênant. C'est pour cela que les industrialisateurs ont demandé un Checkin de masse pour qu'un ensemble de fichier source soient sauvegardés et enregistrés par un Checkin en une seule opération.

4 QUATRIEME PARTIE : BILAN DU PROJET

Dans cette partie je commencerai par donner mon point de vue sur l’outil LDD+. Puis je présenterai mon bilan personnel sur ce projet.

4.1 Mon point de vue sur LDD+

4.1.1 La qualité de LDD+

Les informations qui suivent sont issues du document suivant :
EELES P., 2005. Capturing Architectural Requirements, IBM.
http://www.ibm.com/developerworks/rational/library/4706.html

LDD+ est un outil de GCL qui a été développé à partir de l'outil LDD. Ce développement interne est moins couteux et plus facile à mettre en place que des outils commerciaux comme ENDEVOR et CHANGEMAN. Par rapport à LDD, il apporte un vrai progrès dans la Gestion de Configuration Logicielle et dans le suivi des transferts. LDD+ est fiable et donne des temps de réponses satisfaisants. Pour juger de la qualité de LDD+, nous pouvons utiliser le modèle FURPS+. L'acronyme FURPS+ correspond aux qualités suivantes :

- Functionality : Les fonctions du produit doivent correspondre bien aux besoins voulus. Malgré ses limites, LDD+ permet d'appliquer le processus de GCL en mettant en œuvre un référentiel de versions, une baseline et un système de verrous pour gérer les développements concurrents.

- Usability : Le produit doit être utilisable. LDD+ est simple à utiliser. La grande majorité des utilisateurs ont appris à manipuler LDD sur le tas à cause de la faible documentation disponible. Le passage à LDD+ était simple puisqu'il n'y avait que la commande de Checkin à retenir obligatoirement.

- Reliability : Le produit doit être fiable. LDD+ est fiable car il y a eu très peu de blocages généraux. La plupart des blocages sont limités à un projet et ils sont souvent dus à des bibliothèques trop petites qui sont très faciles à agrandir.

- Performance : Le produit doit être performant. Les temps de réponse de LDD+ sont satisfaisants.

- Supportability : Le produit doit être maintenable. C'est le plus gros point faible de LDD+. Il sera décrit plus loin (voir 4.1.3)

- « + » : Le produit doit répondre à des contraintes ou des besoins qui ne font pas partie des cinq précédemment cités. On peut par exemple citer des contraintes de conception (design requirements), exemple utilisation d'un SGBD ou des contraintes de développement (implementation requirements), exemple utilisation d'un langage spécifique de programmation. LDD+ n'est pas directement concerné par ce genre de contraintes et besoins.

4.1.2 Les limites fonctionnelles de LDD+

LDD+ permet d'appliquer le processus de GCL en mettant en œuvre un référentiel qui gère les différentes versions des fichiers, une baseline qui enregistre les évolutions des environnements RE7 et OPE, et un système de verrous pour gérer les développements concurrents. Néanmoins LDD+ possède de nombreuses limitations fonctionnelles.

LDD+ ne permet pas de retour arrière. Comme l'outil n'intervient pas directement sur l'environnement de recette ou sur l'environnement de production, on ne peut pas faire de retour arrière dans ces environnements. Il faut retransférer les fichiers voulus dans l'environnement cible.

LDD+ ne gère pas la notion de version d'un projet. A un date donnée, on peut connaître l'état d'un projet mais on ne peut pas distinguer un état parmi d'autre. Il n'existe pas l'équivalent de la notion de tags dans Subversion (voir 2.4.2.3). Le retour à un état antérieur du projet est plus difficile car il faut alors restaurer les fichiers un par un.

LDD+ n'intervient que sur des fichiers sources dont la structure est spécifique. Il faut que ces fichiers sources soient contenus dans des bibliothèques (voir 2.1.3) et la longueur de chaque ligne est de 80 caractères. LDD+ ne peut donc pas gérer les fichiers qui ont une autre structure comme par exemple les fichiers paramètres qu'il serait intéressant de gérer en Gestion de Configuration Logicielle.

LDD+ gère les versions par copie intégrale des fichiers sources et non par delta (voir 2.4.2.2). Cela consomme beaucoup d'espace disque. Pour certains projets, la taille maximale des bibliothèques de versionning a été atteinte. Dans ce cas, on a choisi des fichiers sources dont on a éliminé les versions les plus anciennes. A cause de ces manipulations, l'intégrité du référentiel n'est plus assurée mais cela ne crée pas de difficultés dans le travail des équipes projets car celles-ci utilisent très peu les anciennes versions d'un fichier.

LDD+ gère les bibliothèques de versionning RE7 et OPE par copie intégrale de la bibliothèque de versionning DEV (voir 2.3.6.5). Non seulement cela pose un problème d'espace disque pour les bibliothèques de versionning RE7 et OPE, mais en plus on retrouve dans ces bibliothèques versionning des versions qui n'ont pas été réellement transférées.

Comme on le voit, le fonctionnement actuel de LDD+ est satisfaisant mais on peut y apporter de nombreuses améliorations.

4.1.3 La très faible maintenabilité de LDD+

Le plus gros défaut de LDD+ est sa très faible maintenabilité. Les causes sont nombreuses :

- Les langages de programmation utilisés CLIST et REXX sont de moins en moins connus.
- Gérard Besson est parti à la retraite fin mars 2008. Or c'était la seule personne qui connaissait vraiment LDD+ de l'intérieur.
- Les documents de spécifications et les documents techniques sont quasiment inexistants.
- L'architecture et la programmation de LDD+ sont peu lisibles. La programmation s'est faite à l'ancienne sans beaucoup de commentaires et de standardisation.
- Beaucoup de traitements spécifiques ont été codés. Cela entraîne que certaines modifications doivent être reportées à des endroits difficiles à trouver sous peine d'incohérence de code.

LDD+ est un outil maison développé à l'ancienne sans ou avec très peu de documentation et en particulier des spécifications. Les délais et la méthode de travail de Gérard Besson n'ont pas permis de mettre en place une vraie documentation avec en particulier une analyse sous forme de spécifications avant toute réalisation. Néanmoins grâce à la recette et au support de formation, nous disposons de documents fonctionnels qui n'existaient pas chez LDD le prédécesseur de LDD+. Nous avons eu de la chance que Gérard Besson ait toujours été disponible car sans ses connaissances personne d'autre ne pouvait intervenir sur LDD+. Il faut saluer son engagement dans le développement de LDD+ car sans son travail accompli chez lui pendant ses soirées, LDD+ n'aurait jamais été prêt à temps.

4.2 L'avenir de LDD+

4.2.1 Les demandes d'amélioration

Au début de l'année 2008, les projets cibles ayant migré vers LDD+, nous avons proposé les cinq améliorations suivantes :

- l'ajout de la version du fichier source dans le suivi des opérations pour les opérations de Checkin et de Checkout (voir 2.3.5.3 colonne ACTION)
- le Checkin de masse (voir 3.11)
- l'obligation de saisir le motif de transfert (voir 2.3.5.2). Dans la présentation du suivi de transfert, nous voyons dans la Figure 24, qu'il existe des transferts dont le motif de transfert n'a pas été saisi. Cette évolution rendra systématique cette saisie
- la mise à jour des bibliothèques de versionning RE7 et OPE restreintes aux versions réellement transférées (voir 2.3.6.5). Au lieu de faire une copie intégrale du fichier de versionning au moment du transfert, on ne recopierait que la version effectivement transférée.
- La limitation du nombre de versions dans la bibliothèque de versionning. Actuellement, les versions s'accumulent dans les bibliothèques de versionning qui risquent d'atteindre leur limite physique. Une fois ces limites atteintes, LDD+ serait bloqué car les Checkin seraient impossibles.

Pour certains SIA, ces limites ont déjà été atteintes et il a fallu éliminer des anciennes versions. Pour ces SIA, les versions s'accumulent très vite car leur équipe projet est obligée de transférer un programme en environnement de recette pour pouvoir le tester. Elles le font parce qu'il n'y a qu'en environnement de recette qu'elles trouvent un ensemble homogène et cohérent de données de test. Elles n'ont pas fait l'effort ou n'ont pas eu les ressources pour créer cet ensemble homogène et cohérent en environnement de développement.

Pour éviter que les limites des bibliothèques de versionning soient atteintes, nous proposons de ne garder :

- pour l'environnement de développement les dix dernières versions
- pour l'environnement de recette les six dernières versions
- pour l'environnement de production les six dernières versions.

Ces chiffres ont été choisis empiriquement et seront ajustés après consultation des équipes projets. Comme cela a déjà été dit (voir 4.1.2), ces éliminations et restriction de version dégradent l'intégrité du référentiel puisque certaines versions seront absentes du référentiel. Mais cela a été accepté car cela ne gêne pas l'activité des équipes projets qui consultent très peu les vieilles versions. Les deux dernières demandes d'évolution sont liées car elles concernent toutes les deux la saturation des bibliothèques de versionning.

Ces demandes d'amélioration ont été transmises. Elles ne changent pas l'utilisation de LDD+ excepté le Checkin de masse. Seule la première a pu être réalisée avant le départ à la retraite de Gérard Besson. La réalisation des autres demandes d'amélioration dépendra des ressources allouées et de la disponibilité de son remplaçant.

4.2.2 La pérennité de LDD+

On peut voir qu'à l'exception de sa très faible maintenabilité, LDD+ est un outil de GCL de bonne qualité malgré ses limites fonctionnelles. C'est un outil très spécifique à l'environnement Mainframe de l'entreprise Renault. Malgré quelques besoins d'amélioration, les utilisateurs ne lui ont pas trouvé de défauts rédhibitoires. Néanmoins la pérennité de LDD+ se pose pour plusieurs raisons :

- L'environnement Mainframe est une technologie obsolète. Son déclin est inéluctable car seules les entreprises qui utilisent déjà l'environnement Mainframe (Banques, Assurances, Grosses Industries) développent encore des projets dans cette technologie. Son déclin est néanmoins lent à cause des coûts de migration très élevés vers d'autres technologies.
- Sa très faible maintenabilité.
- Le manque de volonté politique. Les restrictions de budget font que l'amélioration de LDD+ ne constitue pas une priorité.

LDD+ perdurera tant qu'il existera un environnement Mainframe à Renault. Il n'y a pas de volonté à le remplacer parce qu'il répond à l'essentiel des besoins et que les coûts de remplacement sont trop élevés.

4.3 Les difficultés rencontrées

4.3.1 Les délais

Quand je suis arrivé sur le projet, l'outil LDD+ n'était pas encore prêt. Or l'objectif était de déployer le processus et l'outil avant la fin de l'année pour les projets cibles. Le moment-clé est arrivé après la migration des 3 premiers projets pilotes. A ce moment, LDD+ fonctionnait correctement et l'équipe maîtrisait suffisamment le processus de déploiement pour pouvoir planifier son avancement. On savait alors que le projet pouvait aboutir. Le projet a connu une grande accélération quand l'étape de préparation était devenue inutile avec la nouvelle étape de prémigration (voir 3.9.5 et 3.9.6).

4.3.2 L'opposition IRN/SIA

Le déploiement du processus et de l'outil étaient sous la responsabilité de 2 équipes : l'équipe VPMQ et l'équipe SOAC (voir 1.4.6). Une des difficultés a été de rapprocher les points de vue en particulier sur la désignation des projets. L'équipe des PCAQ désigne les projets par leur code IRN. C'est le code de désignation officiel. L'équipe LDD+ travaille par SIA puisque les migrations se font SIA par SIA. Il a donc fallu jongler entre les IRN et les SIA, connaître les SIA correspondants à un IRN donné et inversement connaître les IRN correspondants à un SIA donné. Le recensement des projets utilisant l'environnement Mainframe s'est fait avec deux méthodes complémentaires.

La première méthode se base sur les IRN. L'équipe VPMQ a recensé les projets par leur IRN puis les a interrogés pour savoir s'ils utilisaient l'environnement Mainframe. L'équipe LDD+ a ensuite interrogé les projets utilisant l'environnement Mainframe pour connaître le ou les SIA utilisés. Ces informations n'étaient parfois pas connues du premier interlocuteur. Et souvent, il a fallu interroger d'autres personnes pour obtenir l'information voulue.

La deuxième méthode se base sur les SIA. L'équipe LDD+ a recensé tous les SIA. Cette méthode était nécessaire pour ne pas oublier de SIA. Elle a permis de détecter et de neutraliser les SIA qui n'étaient plus actifs depuis plus de deux ans. Cette méthode a permis aussi de détecter des projets utilisant l'environnement Mainframe qui n'étaient pas apparus dans le recensement des projets par IRN. Pour trouver l'IRN d'un SIA, on demandait cette information aux dernières personnes ayant utilisé LDD avec ce SIA.

Les cas les plus délicats étaient les SIA utilisés par plusieurs projets car souvent dans ce cas, les projets ignoraient l'existence des autres projets. Pour ces SIA, les migrations est plus difficiles à planifier car il fallait mettre d'accord tous les projets sur la date de migration.

4.3.3 La phase de préparation

Dans l'étape de préparation (voir 3.9.5), nous étions obligés de fusionner les bibliothèques de développement. Nous avons expliqué que c'était une opération manuelle, délicate longue et risquée. Heureusement, nous avons pu faire évoluer l'étape de prémigration en rendant inutile cette fusion.

4.3.4 L'évolution des procédures

Pour une tâche donnée, par exemple le recensement des projets, il faut harmoniser et faire évoluer les procédures. Si quelqu'un commence à travailler sur cette tâche, il va définir sa propre procédure. Supposons qu'une autre personne la rejoint plus tard pour accomplir la même tâche. Parce qu'elle possède une manière différente de procéder ou parce qu'elle dispose d'autres informations, la deuxième personne peut avoir du mal à appliquer telle quelle la procédure élaborée par la première personne. Il faut alors se réunir et se mettre d'accord sur une procédure commune.

Avec l'avancement du projet, on constate souvent qu'une procédure devient inadaptée. Il faut alors se réunir pour l'améliorer. Mais surtout, il faut souvent reprendre le travail déjà accompli pour le rendre cohérent avec le travail accompli avec la nouvelle procédure.

4.4 Mon bilan personnel

4.4.1 L'enrichissement personnel

Le projet a été très enrichissant dans de nombreux domaines.

Ce projet constitue ma première mission sans développement. Mes précédentes missions étaient des missions de développeur. J'ai donc développé mon expérience d'analyste dans la rédaction de la documentation et dans la spécification des tests. Néanmoins mon expérience de développeur m'a été très utile dans ce projet. Comme je connaissais déjà LDD et son utilisation, je comprenais mieux les appréhensions des équipes projets et donc mes explications étaient mieux comprises.

Ce projet ma donné une expérience dans le déploiement d'un projet à grande échelle. En suivant l'activité de M. Dominique Chaniat le responsable de l'équipe SOAC, j'ai relevé plusieurs éléments importants :

- il faut bien définir les actions qu'on s'engage à exécuter pour une échéance donnée. Il ne faut pas s'engager sans un minimum de maîtrise.
- il faut bien définir les procédures. Il ne faut pas avoir peur d'être bureaucratique en mettant en place des formulaires et des PV de validation. Cela permet d'avoir des résultats cohérents, reproductibles et prévisibles. Cela permet aussi d'impliquer les projets dans le déploiement.
- il faut bien communiquer. Cela permet à la hiérarchie de suivre l'avancement du projet et d'anticiper d'éventuelles difficultés. En cas de difficulté, il faut expliquer son impact sur le projet et des solutions pour la contourner. Les indicateurs d'avancement ainsi que les documents clés d'une migration ont été publiés dans un site intranet pour qu'ils soient consultables par tout le monde.
- il ne faut pas hésiter à mettre la pression sur les équipes projets qui doivent migrer car sinon le planning ne serait plus tenu. En même temps, il ne faut pas trop les brusquer au risque de provoquer un blocage.
- il faut se montrer disponible auprès des équipes projet pour leur apporter explication et assistance. C'est comme cela qu'on les rassure et qu'on gagne leur confiance. Si une migration se passe mal pour une équipe projet, on peut être sûr que les autres seront réticentes à migrer.

J'ai pu développer mes compétences dans la communication :

- pour élaborer des documents Powerpoint avec éventuellement des animations
- pour fournir assistance et explications aux équipes projets
- pour la préparation et l'exécution des séances de formation

Grâce à ce projet je connais à peu près tous les projets qui utilisent le Mainframe soit par les sessions de formation soit parce que j'ai participé à leur migration.

Enfin j'ai pu aussi développer mes compétences dans la qualité logicielle puisque grâce à la rédaction de ce mémoire, j'ai approfondi mes connaissances dans les domaines de la démarche qualité CMMI et de la gestion de configuration logicielle.

4.4.2 L'environnement de travail

Le projet s'est accompli dans de bonnes conditions matérielles. Il n'y a pas eu de problèmes d'ordinateur et le lieu de travail n'était pas trop éloigné de mon domicile. Je me suis bien entendu avec les différents intervenants du projet : l'équipe SOAC, l'équipe VPMQ, les équipes projets même si certaines étaient au début réticentes à migrer de LDD vers LDD+.

Je pense que ma bonne entente avec M. Gérard Besson a considérablement facilité le développement de LDD+. Je le remercie pour toutes les explications qu'il m'a fournies.

4.5 L'année 2008

Pendant l'année 2008, je suis resté dans l'équipe SOAC. Nous avons continué la migration des SIA qui n'avaient pas migré en 2007 dans un contexte de moindre urgence. Grâce au déploiement de LDD+, le déploiement du processus GCL qui n'avait au début qu'une très faible visibilité, a pu se faire dans les temps. En Avril 2008, la direction informatique de Renault a été certifié CMMI3 pour ses développements informatiques. J'ai participé à la migration des référentiels CVS et PVCS vers les référentiels Subversion. Suite à la politique d'externalisation vers l'Inde, j'ai aussi procédé à la traduction de LDD+ en anglais et à la formation d'informaticiens indiens.

Conclusion

Pendant l'année 2007, j'étais en mission chez Renault pour le compte de la SSII Atos-Origin. Dans cette mission j'ai participé au déploiement de la méthode qualité CMMI et en particulier au déploiement du processus Gestion de Configuration Logicielle (GCL). Pour la GCL, j'ai été chargé de recetter et de déployer l'outil LDD+ en remplacement de l'outil LDD dans l'environnement Mainframe.

Malgré les délais serrés, les objectifs du déploiement de LDD+ ont été atteints. L'outil LDD+ a été facilement été accepté par les équipes projets grâce à sa fiabilité et sa simplicité d'utilisation.

Dans ma contribution à ce projet deux éléments importants sont à noter :

- La recette de l'outil LDD+
- L'évolution de la procédure de prémigration.

Grâce aux tests de recette, j'ai mis en évidence deux défauts dans le système de gestion des verrous. Le premier défaut permettait de bloquer un fichier alors qu'une autre personne était en cours de travail dessus. Le deuxième défaut permettait de modifier un fichier alors qu'il était en cours de transfert. Le fichier transféré aurait été alors différent de la version qu'on avait prévue de transférer. Ces défauts ont été corrigés dans la version V2 de LDD+ qui a été déployée pour les trois projets pilotes.

La première version de la procédure de prémigration pouvait être longue, délicate et risquée à cause de l'étape de préparation qui consistait fusionner les bibliothèques de développement. Suite à ma réflexion, je me suis appuyé sur le fait qu'il était inutile d'enregistrer la version de l'environnement DEV d'un fichier au moment de la migration vers LDD+. C'était inutile car cette version est obligatoirement enregistrée quand le fichier est transféré. En supprimant cette étape de préparation, nous avons gagné un temps considérable. Cela nous a permis de migrer tous les projets cibles dans la limite de temps qui nous était imposée.

LDD+ est un outil de GCL très spécifique à l'environnement Mainframe de Renault. Il remplit toutes les fonctions nécessaires. Il possède un référentiel pour gérer les différentes versions d'un fichier, une baseline pour enregistrer toutes les évolutions des environnements RE7 et OPE et un système de verrous pour gérer les développements concurrents. Les défauts de LDD+ sont quelques limites fonctionnelles et sa très faible maintenabilité.

Du fait de sa très faible maintenabilité et d'un manque de volonté politique, les améliorations des fonctions GCL de LDD+ seront lentes à arriver. La pérennité de LDD+ est liée à celle de l'environnement Mainframe chez Renault.

J'ai été très content d'avoir participé au projet « recette et déploiement de l'outil LDD+ » pour les expériences qu'il m'a apportées et pour les relations humaines que j'ai pu établir. Je suis aussi très satisfait d'avoir trouvé une mission qui m'a apporté suffisamment de substance pour constituer un mémoire d'ingénieur. Je n'aurai que deux petits regrets :

- que le départ à la retraite de M. Gérard Besson ait eu lieu trop tôt. J'aurai bien voulu apprendre davantage sur l'aspect technique de LDD+,
- que Renault ne m'ait pas fait de proposition d'embauche alors que les compétences motivées sur le Mainframe deviennent rares.

Finalement grâce à ce mémoire, j'ai pu exercer une réflexion critique et formalisée du travail accompli et l'expérience que j'en ai retirée me sera utile pour mes futurs projets.

Annexe A : Extrait du support de formation

Voir 3.8 Les sessions de formations

Agenda de la formation



- **Présentation de la formation**
  - *Rappel sur la GCL*
- **Présentation générale**
  - *LDD*
  - *LDD+*
  - *Le déploiement*
- **Utilisation de LDD+**
  - *Les commandes LDD+*
  - *Création d'une version*
  - *Récupération d'une version*
  - *Extraction des versions*
- **Conclusion**

P / 2

Positionnement de la formation

Pré requis :

- *LDD*
- *formation PRIMA au « Processus GCL »*

Projets :

- **tous les projets qui utilisent MVS**

Public :

- *Back-Office*
- *Front-Office*
- *Département industrialisation*

P / 3

Intérêts de la Gestion de Configuration Logicielle

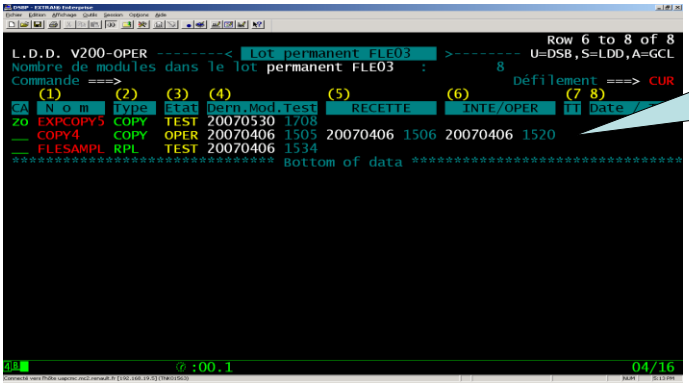
La GCL permet répondre aux questions suivantes

- ***Pour quelle raison un fichier a-t-il été modifié ?***
- ***Qui est l'auteur de cette ligne de code ?***
- ***Quels sont les changements / différences entre deux versions ?***

La GCL permet de reconstituer une version précédente pour faire de la maintenance

La Gestion de Configuration Logicielle met à disposition un REFERENTIEL permettant d'identifier de manière complète et univoque l'ensemble des constituants d'une VERSION d'un produit livré ou en cours de fabrication

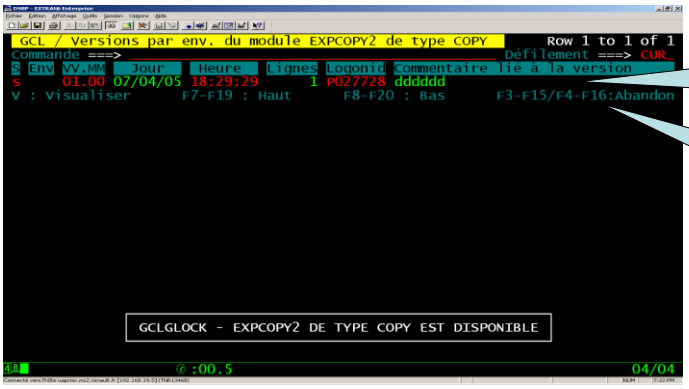
Récupération d'une version (1)



Liste des modules d'un lot

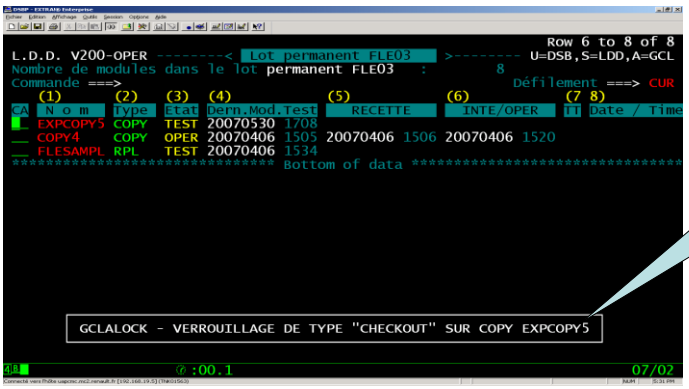
Pour récupérer une version d'un module, taper ZO devant son nom
Le module doit être au préalable libre de tout verrou

Récupération d'une version (2)



***V** pour visualiser une version*
***S** pour récupérer une version par Checkout*
Le Checkout provoque
- *le remplacement dans la bibliothèque de DEV par la version choisie*
- *le dépôt d'un verrou Checkout permanent*

Récupération d'une version (3)



Seul un Checkin fait par la même personne pourra lever le verrou Checkout

Annexe B : Compte-rendu de prémigration

Voir 3.9.3 L'étape de prémigration

PREMIGRATION DU SIA DAV

IRN : 6690

Version V1.0

13/08/2007

Synthèse sur la prémigration	
PB RPL à migrer DEV.DAV.RPL.RLOCP0D	:
Nombre total de modules RPL	: 958
Nombre total de modules à transférer	
- Opérationnel	: 4622
- Recette	: 4650
- Développement	: 5835

Temps d'exécution de la prémigration	
Temps total de prémigration	: 0,50 j (1)
Temps ELAPSE pour tous les traitements	: 20 minutes

(1) arrondi à 0,25j/h

	DATE	NOM	FONCTION (SOCIETE)	COMMENTAIRES
AUTEUR	13/08/2007	Marie-Laure De TAFFANEL	ATOS ORIGIN	
VERIFICATEUR(S)	24/08/2007	Francis LÊ	ATOS ORIGIN	
APPROBATEUR(S)	27/08/2007	Dominique Chaniat	Renault	

Historique des évolutions

VERSIONS	DATE	COMMENTAIRES - MODIFICATIONS	AUTEUR

Références

Documents	Version N°

OBJET

Ce document est un compte-rendu de l'exécution de la prémigration.

Généralités sur le SIA DAV

Informations recueillies sous LDD

U . A . P .	APV	(APRES_VENTE)
S . I . A .	DAV	(DIRECTION APRES VENTE)
Application	BSM	(BOURSE ECHANGE PIECES RECHANGE)
Application	OCP	(VENTILATIONS COMERCIALES)
Application	TAR	(TARIFICATION P.R.)

Seuls les modules de l'application OCP ont été pris en compte dans la prémigration. Les autres applications ne sont plus utilisées. Pour ces applications, aucune livraison n'a été effectuée depuis 2001.

Préparation a la prémigration effectuée

Temps total de la prémigration

Opérations de prémigration	Résultat
Vérification des opérations de normalisation	0,5 h
Création et exécution des jobs de prémigration (simulation)	2 h
Contrôles de l'exécution des jobs	1,5 h

Résultat des jobs de préparation a la migration

Opérations de prémigration	Résultat
MI1 Création BATCH des référentiels GCL	OK
MI2 Eclatement BATCH des PB RPL dans des PO MVS (OPE, RE7, DEV)	OK
MI3 Migration BATCH LDD->GCL proprement dite (OPE, RE7, DEV)	OK

Durée d'exécution

Opérations de migration	Environnement	Temps CPU minutes	Temps ELAPSE minutes
MI1			
		Non significatif	Non significatif
MI2			
	opérationnel	.03	.35
	recette	.03	.61
	développement	.03	.84
MI3			
	Opérationnel	.15	5.44
	recette	.20	6.09
	Développement	.25	7.50

Annexe C : PV de validation pour la migration de LDD vers LDD+

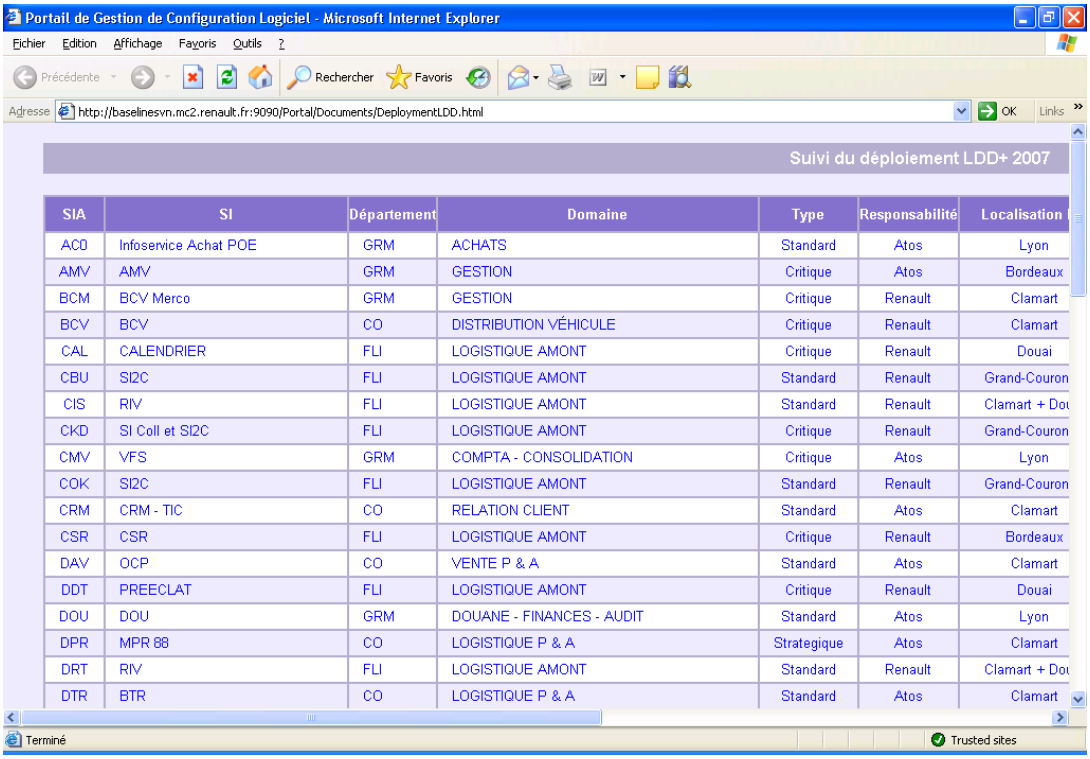
Voir 3.9.7 L'étape de migration

Système d' Information	*Demandeur*
Nom du SI : OCP IRN : 6690 SIA : DAV	Nom : MAGNUS eMail : francis.magnus-renexter@renault.com Téléphone : 01 41 28 09 92

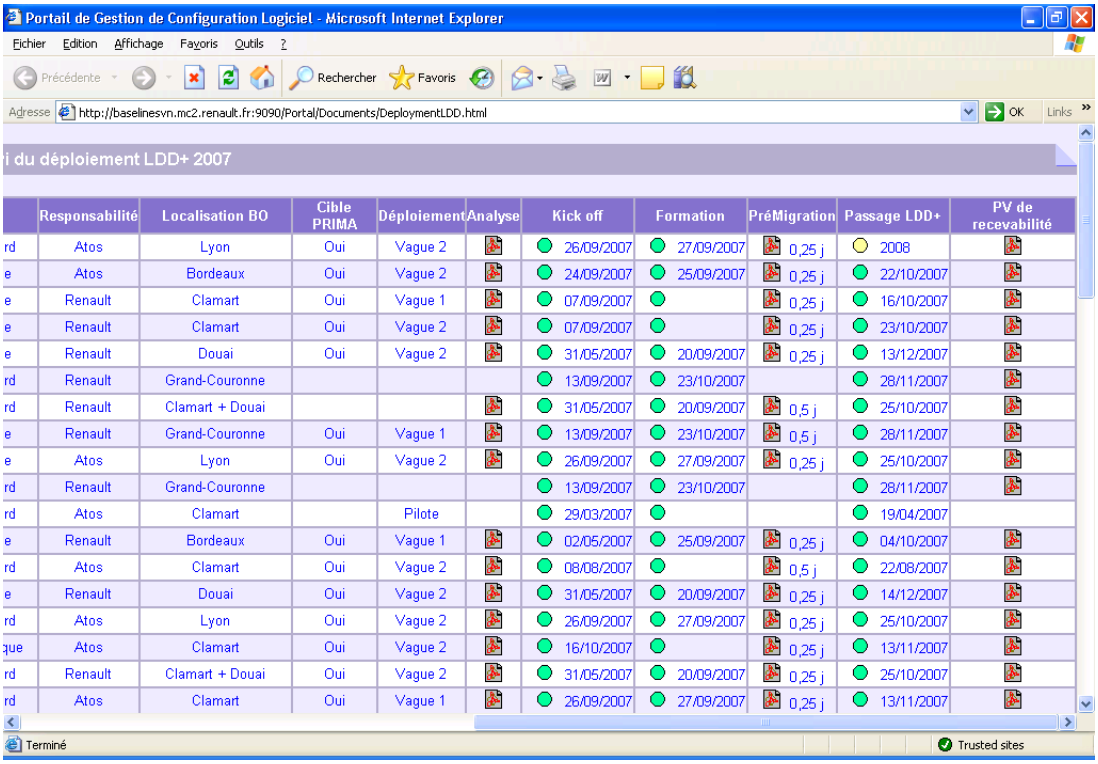
Projet : migration LDD +	**Visa du responsable : Francis MAGNUS** **Date : 22/08/2007**
Description des validations effectuées **-Vérification que les lots LDD sont conservés dans LDD+** **- Création d'un lot sous LDD+** **- Commande LDD+ : Checkout, Checkin** **Visualisation des verrous, Visualisation des versions** **Visualisation des Baselines après transfert** **- Compilation du programme PL1 P1700A5** **- Transfert en recette du programme PL1 P1700A5** **- Transfert en oper du programme PL1 P1700A5**	**Résultat de la validation** **OK** **OK** **OK** **OK** **OK** **OK** **OK** **OK** **OK** **OK**
Remarques **Code retour 8 pour le programme P1700A5** **Considéré comme normal par le responsable**	
Aspects non validés et raisons :	
Acceptation □ **Acceptation avec réserves** □ **Refus** □ **La période probatoire d'utilisation du SIA sous LDD+ est prévue pour une durée de 15 jours, à compter de la date de la migration.**	
Réserves :	

Annexe D : Suivi du déploiement de LDD+.

Voir 3.9.10 Le suivi du déploiement de LDD+



SIA	SI	Département	Domaine	Type	Responsabilité	Localisation
AC0	Infoservice Achat POE	GRM	ACHATS	Standard	Atos	Lyon
AMV	AMV	GRM	GESTION	Critique	Atos	Bordeaux
BCM	BCV Merco	GRM	GESTION	Critique	Renault	Clamart
BCV	BCV	CO	DISTRIBUTION VÉHICULE	Critique	Renault	Clamart
CAL	CALENDRIER	FLI	LOGISTIQUE AMONT	Critique	Renault	Douai
CBU	SI2C	FLI	LOGISTIQUE AMONT	Standard	Renault	Grand-Couron
CIS	RIV	FLI	LOGISTIQUE AMONT	Standard	Renault	Clamart + Dou
CKD	SI Coll et SI2C	FLI	LOGISTIQUE AMONT	Critique	Renault	Grand-Couron
CMV	VFS	GRM	COMPTA - CONSOLIDATION	Critique	Atos	Lyon
COK	SI2C	FLI	LOGISTIQUE AMONT	Standard	Renault	Grand-Couron
CRM	CRM - TIC	CO	RELATION CLIENT	Standard	Atos	Clamart
CSR	CSR	FLI	LOGISTIQUE AMONT	Critique	Renault	Bordeaux
DAV	OCP	CO	VENTE P & A	Standard	Atos	Clamart
DDT	PREECLAT	FLI	LOGISTIQUE AMONT	Critique	Renault	Douai
DOU	DOU	GRM	DOUANE - FINANCES - AUDIT	Standard	Atos	Lyon
DPR	MPR 88	CO	LOGISTIQUE P & A	Strategique	Atos	Clamart
DRT	RIV	FLI	LOGISTIQUE AMONT	Standard	Renault	Clamart + Dou
DTR	BTR	CO	LOGISTIQUE P & A	Standard	Atos	Clamart



	Responsabilité	Localisation BO	Cible PRIMA	Déploiement	Analyse	Kick off	Formation	PréMigration	Passage LDD+	PV de recevabilité
rd	Atos	Lyon	Oui	Vague 2		26/09/2007	27/09/2007	0,25 j	2008	
e	Atos	Bordeaux	Oui	Vague 2		24/09/2007	25/09/2007	0,25 j	22/10/2007	
e	Renault	Clamart	Oui	Vague 1		07/09/2007		0,25 j	16/10/2007	
e	Renault	Clamart	Oui	Vague 2		07/09/2007		0,25 j	23/10/2007	
e	Renault	Douai	Oui	Vague 2		31/05/2007	20/09/2007	0,25 j	13/12/2007	
rd	Renault	Grand-Couronne				13/09/2007	23/10/2007		28/11/2007	
rd	Renault	Clamart + Douai				31/05/2007	20/09/2007	0,5 j	25/10/2007	
e	Renault	Grand-Couronne	Oui	Vague 1		13/09/2007	23/10/2007	0,5 j	28/11/2007	
e	Atos	Lyon	Oui	Vague 2		26/09/2007	27/09/2007	0,25 j	25/10/2007	
rd	Renault	Grand-Couronne				13/09/2007	23/10/2007		28/11/2007	
rd	Atos	Clamart		Pilote		29/03/2007			19/04/2007	
e	Renault	Bordeaux	Oui	Vague 1		02/05/2007	25/09/2007	0,25 j	04/10/2007	
rd	Atos	Clamart	Oui	Vague 2		08/08/2007		0,5 j	22/08/2007	
e	Renault	Douai	Oui	Vague 2		31/05/2007	20/09/2007	0,25 j	14/12/2007	
rd	Atos	Lyon	Oui	Vague 2		26/09/2007	27/09/2007	0,25 j	25/10/2007	
que	Atos	Clamart	Oui	Vague 2		16/10/2007		0,25 j	13/11/2007	
rd	Renault	Clamart + Douai	Oui	Vague 2		31/05/2007	20/09/2007	0,25 j	25/10/2007	
rd	Atos	Clamart	Oui	Vague 1		26/09/2007	27/09/2007	0,25 j	13/11/2007	

Bibliographie

www.atosorigin.com voir 1.1.2
https://source.atosorigin.com/intranet/

www.renault.com voir 1.1.3
http://www.intra.renault.fr/corp/dsir/

Software Engineering Institute, 2006. CMMI for Development Version 1.2, Carnegie Mellon, Pittsburgh voir 1.2
www.sei.cmu.edu/cmmi/models/index.html
BASQUE R., 2009. CMMI 1.2 – Le Modèle. DUNOD, Paris

Site intranet du projet PRIMA voir 1.3
http://www.intra.renault.fr/projet/primab/

http://fr.wikipedia.org/wiki/Roue_de_Deming voir 1.3.3
http://www.hci.com.au/hcisite2/toolkit/pdcacycl.htm

Support de formation au processus GCL voir 1.4
PRIMA-SS#01-SF-393-Processus Gestion de configuration logiciel - V2.1.ppt

Support de formation LDD+ voir 2
Formation-LDD+-080314.ppt

PRINTZ J., 2004. Introduction à la gestion de configuration, CNAM. voir 2.3.7
Support de cours : Intro-GCONF-10.pdf

COLLINS-SUSSMAN B., FITZPATRICK B., PILATO C., 2008. Version control with Subversion. voir 2.4.2
http://svnbook.red-bean.com/svn-book.pdf

Scénarios de test voir 3.4
Test-LDD+-01.xls

Procédure de déploiement du processus GCL voir 3.7
Déploiement GCL.ppt

EELES P., 2005. Capturing Architectural Requirements, IBM.
http://www.ibm.com/developerworks/rational/library/4706.html voir 4.1.1

Table des figures

Figure 1 : Organisation de la société Atos-Intégration France 6
Figure 2 : Organisation de la DSIR 7
Figure 3 : Représentation étagée du modèle CMMI 8
Figure 4 : Logo du projet PRIMA avec les 8 domaines de processus 10
Figure 5 : Les processus du projet PRIMA 11
Figure 6 : Cycle de déploiement d'un processus PRIMA 13
Figure 7 : Roue de Deming de la méthode PDCA 13
Figure 8 : Ecran d'accueil de l'environnement Mainframe 21
Figure 9 : Liste des bibliothèques 22
Figure 10 : Liste de membres dans la bibliothèque DEV.SCRM.SOURCE.COB. 22
Figure 11 : Principe d'un lot 24
Figure 12 : Interface de LDD 25
Figure 13 : Architecture de LDD 26
Figure 14 : Les étapes des transferts dans LDD 28
Figure 15 : Diagramme état-transition d'un fichier source 29
Figure 16 : Comparaison de l'architecture LDD/LDD+ 31
Figure 17 : Principe de la fonction Checkin 33
Figure 18 : Contenu d'un fichier versionning 34
Figure 19 : Transcription d'un enregistrement entête 34
Figure 20 : Principe de la fonction Checkout 35
Figure 21 : Principe de la fonction Extraction des versions 36
Figure 22 : Bloc de transfert 36
Figure 23 : Contenu d'un fichier Baseline 37
Figure 24 : Liste des transferts 39
Figure 25 : Contenu d'un transfert 40
Figure 26 : Suivi des opérations 40
Figure 27 : Utilisation du Checkin et du Checkout 42
Figure 28 : Problème du développement concurrent 46
Figure 29 : Modèle « Verrouillage-Edition-Enregistrement » 48
Figure 30 : Modèle « Extraction-Edition-Fusion » 49
Figure 31 : Limite du modèle « Verrouillage-Edition-Enregistrement » dans l'environnement Mainframe. 52
Figure 32 : Diagramme état-transition des verrous LDD+ 54
Figure 33 : Contenu d'un référentiel Subversion 57
Figure 34 : Scénario de test « Création Checkin ». 65
Figure 35 : Scénario de test « User 1 Checkout Checkin » 68
Figure 36 : Diagramme état-transition des verrous LDD+ dans la version V1 70
Figure 37 : Les étapes de la migration d'un projet 72
Figure 38 : Les outils de migration 76
Figure 39 : Menu des opérations de migration 76
Figure 40 : Exemple de script généré par le menu 77
Figure 41 : Comparaison de l'architecture LDD/LDD+ 77
Figure 42 : Deuxième opération de l'ancienne étape de prémigration : création du fichier versionning OPE 80
Figure 43 : Troisième opération de l'ancienne étape de prémigration : création du fichier versionning RE7 80
Figure 44 : Quatrième opération de l'ancienne étape de prémigration : création du fichier versionning DEV 81
Figure 45 : Etat des bibliothèques des fichiers sources avant fusion 81
Figure 46 : Etat des bibliothèques des fichiers sources après fusion 82
Figure 47 : Nouvelle étape de prémigration 83

Table des tableaux

Tableau I : Représentation continue du modèle CMMI 9
Tableau II : Les activités PRIMA et les attentes CMMI pour la GCL 16
Tableau III : Chronologie du projet 61

Recette et déploiement de l'outil LDD+.
Mémoire d'Ingénieur C.N.A.M., Paris 2010.

RESUME

En 2005, Renault a choisi Atos-Origin comme prestataire pour ses développements informatiques. Pour améliorer leur qualité, ils ont décidé d'appliquer la démarche qualité CMMI. CMMI formalise les activités sous forme de processus.

Le processus Gestion de Configuration Logicielle suit les évolutions des composants informatiques d'un projet. Pour appliquer la GCL dans l'environnement Mainframe, Renault a choisi LDD+.

LDD+ est une évolution de l'outil de développement LDD. Il apporte un référentiel qui contient les versions des fichiers, une baseline qui enregistre les transferts en environnements RE7 et OPE, une gestion des développements concurrents par verrous.

Dans la recette, j'ai vérifié que LDD+ fonctionne sans anomalies. J'ai alors fait remonter la nécessité d'affiner la gestion des verrous. Nous avons ensuite déployé l'outil LDD+ à tous les projets cibles avec en parallèle la formation des utilisateurs.

Au début du déploiement, il était nécessaire de fusionner les bibliothèques de développement. Cette fusion devenait de plus en plus longue et difficile au point de devenir impossible. J'ai alors mis au point une nouvelle procédure qui contournait la fusion. Sans cette fusion, le déploiement s'est accéléré et s'est achevé fin 2007 pour tous les 63 projets cibles.

Mots clés : CMMI, GCL, Mainframe, LDD, LDD+, Recette, Migration, Formation.

SUMMARY

In 2005, Renault chose Atos-Origin as contractor for its computing developments. To improve their quality, they decided to apply the CMMI quality approach. CMMI formalizes the activities in the form of process.

The process Software Configuration Management reports the evolutions of the computing components of a project. To apply SCM in the Mainframe environment, Renault chose LDD+.

LDD+ is an evolution of the LDD development tool. It brings a repository which contains the versions of the files, a baseline which records transfers in RE7 and OPE environments, a management of the parralel developments with locks.

In the acceptance phase, I verified that LDD+ works without defects. I then raised the necessity of refining the management of the locks. Next, we deployed the LDD+ tool to all the target projects with in the same time the training of the users.

At the beginning of the deployment, it was necessary to merge the development libraries. These fusions became longer and longer and more and more difficult to the point to be impossible. I then designed a new procedure which bypassed the fusion. Without this fusion, the deployment accelerated and ended at the end of 2007 for every 63 target projects.

Keywords : CMMI, SCM, Mainframe, LDD, LDD+, Acceptance, Migration, Training.