APES (APES is a Process Engineering Software)

Plan du Cycle de Vie

Version 4.0

Auteur : Isabelle Cassagneau

Table des Révisions

Révision	Date	Auteur(s)	Description
4.0	20/01/2004	Isabelle Cassagneau	Modification des produits de travail concernant les tests
2.0	18/11/2003	Isabelle Cassagneau	Regroupement des disciplines "Implémentation", "Test" et "Déploiement", et description de cette nouvelle discipline.
1.0	05/11/2003	Isabelle Cassagneau	Création du document

Table des Matières

Chapitre 1
Introduction

1. Objectif

L'objectif du plan de cycle de vie est de définir la configuration du processus pour le projet APES. Ce processus est inspiré du processus RUP−F. Ce document présente les parties du processus RUP−F qui sont prises en compte pour le projet, les parties qui ne le sont pas, et les parties adaptées.

2. Portée

Le plan de cycle de vie est destiné aux membres de l'équipe et aux superviseurs de projet.

3. Références

- RUP−F
- L'eXtreme Programming (XP)
- Glossaire

Chapitre 2
Vue d'ensemble du Plan de cycle de vie

1. Modèle de cycle de vie

Le cycle de vie utilisé est conforme à celui décrit dans le RUP−F. Il est composé de quatre phases et chacune d'entre elles se termine par un jalon.

1.1. Phase de lancement

C'est la phase initiale du projet. Elle a pour objectif de définir une solution au problème et de détecter les risques qui peuvent en découler. Dans cette optique, il y sera écrit la liste des risques, le document vision, définissant les principales fonctionnalités du projet, le plan de développement logiciel, et le plan de cycle de vie.

1.2. Phase d'élaboration

Le but essentiel de cette phase est de produire une architecture stable du système pour avoir une base solide pour la phase de construction. Les cas d'utilisation les plus importants y sont définis et un prototype de l'architecture est réalisé.

1.3. Phase de construction

Elle a pour objectif de réaliser un logiciel utilisable basé sur l'architecture précédemment définie. Le découpage en itérations permet de répartir le travail et de le valider au fur et à mesure de l'avancement du projet. Chaque itération définit les cas d'utilisation à réaliser et comporte une partie de conception, une partie de tests unitaires, une partie de tests fonctionnels et une partie de codage.

A intervalles réguliers, des versions du logiciel (incomplètes mais utilisables) sont livrées au client, qui fournira ainsi un retour d'expérience.

1.4. Phase de transition

Le but de cette phase est de s'assurer que le logiciel est disponible pour les utilisateurs finaux. Cette phase peut s'étaler sur plusieurs itérations, inclure le test du produit avant sa sortie et les ajustements mineurs basés sur les remarques faites par les utilisateurs. A ce stade du cycle de vie, les remarques faites par les utilisateurs ne doivent viser qu'à de petites améliorations du produit, concernant la configuration, l'installation et les problèmes d'utilisation ; les problèmes de structure ayant été revus beaucoup plus tôt.

2. Disciplines

Configuration de la discipline (du PCDV)	Discipline de base (ce qui est prévu dans le RUP−F)	Commentaires
Expression des exigences	Discipline RUP−F : Expression des exigences	Comme défini dans le RUP−F
Analyse et conception	Discipline RUP−F : Analyse et conception	Production d'un sous−ensemble
Gestion de projet	Discipline RUP−F : Gestion de projet	Production d'un sous−ensemble
Gestion de processus	Discipline RUP−F : Gestion de processus	L'adaptation fait l'objet de ce plan du cycle de vie
Implémentation et Test	Pas dans le RUP−F	Regroupe les disciplines Implémentation, Test et Déploiement

3. Configuration des disciplines

Ce chapitre présente les produits de travail utilisés dans chaque discipline du processus. Les tableaux ci−dessous précisent comment ces disciplines et leurs produits associés seront décrits dans ce document.

3.1. Produits de travail

Chaque produit est représenté par une ligne du tableau suivant :

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples

Voici la description de chacune de ses colonnes :

Nom de la colonne	Objectif	Contenu / Commentaires
Produit	Le nom du produit de travail.	Une référence au produit de travail dans le processus.

Nom de la colonne	Objectif	Contenu / Commentaires
Comment l'utiliser ?	Décrire comment le produit est utilisé tout au long du cycle de vie.	Décider de l'utilisation du produit au cours de la phase : – Doit (impérativement être utilisé) – Peut (impérativement être utilisé) – Case vide (ne sera pas utilisé)
Statut	Décider du statut du produit de travail.	– Formel–Externe – Formel–Interne – Informel – Aucun
Outils employés	Définition de l'outil utilisé pour créer le produit.	Références aux détails des outils utilisés pour créer et maintenir les produits de travail.
Plans types / Exemples	Les plans types à utiliser et les exemples de produits construits à partir de ces plans types.	Références aux plans types et exemples du RUP–F ou aux plans types et exemples locaux. Cette colonne peut également contenir des références à des produits de travail courants qui peuvent fournir une aide complémentaire aux membres du projet.

Les versions des produits de travail sont définies de la manière suivante : version X.Y, où X identifie l'itération au cours de laquelle la version a été créée, et Y permet de distinguer les différentes révisions du produit au cours de l'itération.

3.2. Notes sur les produits de travail

Le tableau ci–dessous est donné pour chaque discipline : il contient d'une part la liste de tous les produits proposés par le processus RUP–F et non utilisés dans le cycle de vie, d'autre part des modifications ou des renseignements supplémentaires sur des produits utilisés. Ces choix sont tous justifiés dans la colonne *Raison*.

Produit	Notes	Raison

4. Classification des produits de travail

Un produit de travail est une fourniture du processus. Il est souvent produit dans le cadre d'une discipline, bien qu'il existe des exceptions. Les produits de travail sont organisés dans la discipline où ils sont crées. Pour décrire comment un produit de travail sera utilisé, on emploie le schéma de classification suivant :

Classification	Explication
Doit	Ce produit est obligatoirement utilisé. C'est un produit clé. Cela peut causer des problèmes plus tard dans le développement s'il n'est pas produit.
Peut	Ce produit n'a pas à être utilisé et produit. Il est produit seulement dans les cas où cela apporte de l'intérêt et s'il y a assez de temps.
Case vide	Ce produit ne sera pas utilisé.

5. Procédures de revues

Le projet utilise les niveaux de revue suivants :

Classification	Explication	Commentaires
Formel – Externe	Le produit arrivé à un jalon spécifique est une partie de la distribution finale et nécessite l'approbation du client.	Par exemple, le document Vision est un produit formel–externe, car il doit être présenté et approuvé par le client lors de la revue de fin de phase d'élaboration.
Formel – Interne	Le produit est approuvé par les membres du projet lors d'une revue interne.	Par exemple, le modèle de conception est un produit formel–interne.
Informel	Le produit est utilisé lors de revues internes, mais n'a pas besoin d'être approuvé formellement par les membres du projet.	Le produit est créé puis mis à jour tout au long du projet. Il s'agit le plus souvent d'un produit inclus dans un produit formel plus important. Le sous–système de conception est un exemple typique d'un produit qui n'est pas approuvé formellement.
Aucun	Le produit n'est pas forcément utilisé lors de revues, et il n'a pas besoin d'être approuvé.	Le produit est utilisé comme support de travail. Il est souvent temporaire et aide à une meilleure compréhension par les membres du projet de certains autres produits.

Chapitre 3
Disciplines

1. Expression des exigences

1.1. Objectifs de la discipline

- Se mettre d'accord avec le client et les autres intervenants sur les principales fonctionnalités du logiciel,
- Fournir aux développeurs une bonne définition des besoins du logiciel,
- Définir les limites du projet,
- Fournir une base pour la planification du contenu technique des itérations,
- Fournir une base à l'estimation des coûts et délais pour le développement du logiciel,
- Définir l'interface utilisateur, en se concentrant sur les besoins des utilisateurs.

1.2. Produits de travail

Produit	**Lanc.**	**Elab.**	**Const.**	**Trans.**	**Statut**	**Outils employés**	**Plans types / Exemples**
Vision	Doit	Doit	Doit	Doit	Formel – Externe	IPSdoc	rup_vision–.dot
Modèle de cas d'utili–sation	Doit	Doit	Peut		Formel – Externe	IPSdoc	
Spécifi–cations supplé–mentaires	Peut	Doit	Doit	Doit	Formel – Externe	IPSdoc	rup_sspec–.dot
Maquette de l'IHM		Doit	Doit		Formel – Externe	JAVA, Eclipse	
Glossaire	Doit	Doit	Peut	Peut	Formel – Interne	IPSdoc	rup_gloss–.dot, Glossaire (APES1)
Acteur	Doit	Doit	Doit	Doit	Informel		
Modèle	Doit	Doit			Informel		

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
objet du domaine							
Classe d'interface	Peut	Doit	Peut	Peut	Informel		

1.3. Notes sur les produits de travail

Produit	Notes	Raison
Modèle de cas d'utilisation	On ne fera pas apparaître de diagrammes, mais simplement une liste des cas d'utilisation et leurs descriptions.	Allègement du processus
Paquetage de cas d'utilisation	Inclus dans le produit "Modèle de cas d'utilisation".	Allègement du processus
Spécifications des cas d'utilisation	Inclus dans le produit "Modèle de cas d'utilisation".	Allègement du processus
Scénarios des cas d'utilisation	Inclus dans le produit "Modèle de cas d'utilisation".	Allègement du processus
Maquette de l'IHM	L'interface est implémentée par le prototype.	

2. Analyse et conception

2.1. Objectifs de la discipline

- Transformer les exigences exprimées en conception du système à réaliser,
- Elaborer une architecture robuste,
- Adapter la conception pour tenir compte de l'environnement d'implémentation.

2.2. Produits de travail

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
Document d'architecture		Doit	Doit	Doit	Formel –	IPSdoc	rup_sad.dot, Document

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
logicielle					Externe		d'architecture logicielle (APES1)
Sous−système de conception		Peut	Doit	Doit	Informel		
Paquetage de conception		Doit	Doit	Doit	Informel		
Classe d'interface	Peut	Doit	Doit	Doit	Informel		
Classe de conception	Peut	Doit	Doit	Doit	Informel		
Réalisation de cas d'utilisation		Peut	Doit	Doit	Informel		

2.3. Notes sur les produits de travail

Produit	Notes	Raison
Modèle de conception	Inclus dans le produit "Document d'architecture logicielle".	Allègement du processus
Modèle d'analyse	Inclus dans le produit "Modèle de conception".	Allègement du processus
Classe d'analyse	Inclus dans le produit "Classe de conception".	Allègement du processus
Modèle de données	Non utilisé.	Pas de base de données
Modèle de déploiement	Non utilisé.	Le système est simple et sans distribution des traitements.

3. Gestion de projet

3.1. Objectifs de la discipline

- Fournir des guides pratiques pour planifier, former les équipes, exécuter et contrôler les projets,
- Prévoir les risques et les maîtriser s'ils apparaissent,
- Contrôler l'avancement des itérations.

3.2. Produits de travail

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
Plan de développement logiciel	Doit	Doit	Doit	Doit	Formel – Externe	IPSdoc	rup_sdpln.dot
Liste des risques	Doit	Doit	Peut	Peut	Formel – Interne	IPSdoc	rup_rsklst.dot
Liste des problèmes	Peut	Peut	Peut	Peut	Formel – Interne	IPSdoc	
Plan d'itération	Doit	Doit	Doit	Doit	Formel – Interne	IPSdoc	rup_itpln.dot
Compte−rendu de revue	Doit	Doit	Doit	Doit	Formel – Interne	IPSdoc	
Evaluation de l'itération	Doit	Doit	Doit	Doit	Informel	IPSdoc	rup_itass.dot

3.3. Notes sur les produits de travail

Produit	Notes	Raison
Liste des problèmes	Ce produit ne doit être créé qu'en cas d'apparition de problèmes importants, d'ordre humain ou technique.	Allègement du processus
Plan d'itération	Un nouveau produit est créé pour chaque itération.	Ce produit ne concerne qu'une itération.
Compte−rendu de revue	Un nouveau produit est créé pour chaque itération.	Ce produit ne concerne qu'une itération.

Produit	Notes	Raison
Evaluation de l'itération	Un nouveau produit est créé pour chaque itération.	Ce produit ne concerne qu'une itération.

4. Gestion de processus

4.1. Objectifs de la discipline

- Définir le processus qui va être utilisé sur le projet,
- Préparer l'environnement de développement du projet,
- Evaluer le processus en vue de son amélioration.

4.2. Produits de travail

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
Plan du cycle de vie	Doit	Doit	Doit	Doit	Formel – Externe	IPSdoc	rup_devcs.dot
Plans types spécifiques au projet	Peut	Peut	Peut	Peut	Informel	IPSdoc	
Guides de programmation		Doit	Doit		Informel	IPSdoc	java codingstandard

5. Implémentation et Test

5.1. Objectifs de la discipline

Cette discipline regroupe les disciplines Implémentation, Test et Déploiement. L'objectif est de mettre en oeuvre l'implémentation du logiciel et d''élaborer des procédures de test afin de valider cette implémentation.

5.2. Produits de travail

Produit	Lanc.	Elab.	Const.	Trans.	Statut	Outils employés	Plans types / Exemples
Prototype et versions		Doit	Doit	Peut	Formel – Externe	Java, Eclipse	APES 1.0
Tests unitaires		Doit	Doit	Doit	Formel – Externe	JUnit	
Tests fonctionnels		Peut	Doit	Doit	Formel – Externe	Abbot	
Plan de validation du logiciel		Peut	Doit	Doit	Formel – Externe		
Cahier de validation du logiciel		Peut	Doit	Doit	Formel – Externe		
Manuel utilisateur			Doit	Doit	Formel – Externe		

5.3. Notes sur les produits de travail

Produit	Notes	Raison
Prototype et versions	Les différentes versions de APES doivent être développées à partir du code du prototype ou de la version précédente.	Réutilisation
Tests fonctionnels	Les tests fonctionnels sont en partie automatisés grâce à l'outil Abbot. Les tests fonctionnels non–automatisés seront définis dans le cahier de validation du logiciel.	Certaines opérations sont trop complexes pour être automatisées avec Abbot.

Chapitre 4
Rôles

Quatre catégories de rôles apparaissent dans le RUP−F : les analystes, les développeurs, les testeurs et les gestionnaires.

1. Analystes

Les analystes sont principalement impliqués dans l'expression des exigences. Ils conduisent et coordonnent le recueil des exigences et la modélisation des cas d'utilisation en définissant les fonctionnalités et les limites du système. Par exemple, ils établissent quels sont les acteurs et les cas d'utilisation, et comment ils sont liés les uns aux autres.

2. Développeurs

Les développeurs comptent le concepteur, l'architecte et les codeurs.

Le concepteur est la personne qui définit les responsabilités, les opérations, les attributs, et les relations d'une ou plusieurs classes, et détermine comment elle(s) va(vont) être ajustée(s) à l'environnement d'implémentation.

De plus, le concepteur peut avoir la responsabilité de création des paquetages de conception, ou sous−systèmes de conception, incluant toutes classes du paquetage ou sous−système.

L'architecte logiciel a la responsabilité de mener et coordonner les activités techniques et les produits de travail tout au long du projet. L'architecte logiciel établit la structure globale pour chauqe vue de l'architecture : la décomposition de la vue, le regroupement des éléments, et les interfaces entre ces différents regroupements.

Les codeurs doivent réaliser la partie codage en respectant les exigences déterminées par le concepteur et l'architecte.

3. Testeurs

Les testeurs sont chargés de tester l'application et de veiller à ce qu'elle réponde bien aux exigences du projet.

4. Gestionnaires

Il existe quatre types de gestionnaires :

- Le chef de projet alloue les ressources, définit les priorités, coordonne les actions entre clients et utilisateurs, et généralement s'assure que l'équipe projet reste concentrée sur le véritable objectif. Il établit également un ensemble de pratiques pour garantir l'intégrité et la qualité des produits de travail du projet.
- L'ingénieur processus est responsable de la gestion du processus lui–même. Cela inclut la configuration du processus avant le démarrage du projet et l'amélioration constante du processus au cours de l'effort de développement.
- Le spécialiste outils est responsable du support pour les outils utilisés au cours du projet. Il configure les outils et vérifie leur bon fonctionnement.
- Le superviseur de projet est responsable de l'évaluation des produits de travail au cours des revues majeures du cycle de vie du projet. Ces revues de projet sont des points très importants dans la mesure où elles peuvent aboutir à la décision d'abandonner le projet si la planification est inadéquate ou si la progression est trop faible.

Chapitre 5
Pratiques XP

L'eXtreme Programming (XP) est une méthode de développement dédiée à de petites équipes confrontées à un environnement changeant ou des besoins mal connus.

Les pratiques XP sont pour la plupart des pratiques de "bon sens" et certaines, concernant la qualité du design et du code et le travail en équipe, pourront être appliquées au processus APES.

1. Qualité du design et du code

1.1. Conception simple

Le développeur devra réfléchir à la conception la plus simple possible, et pas forcément la plus facile, pour résoudre la tâche qui lui est affectée. Il ne devra pas perdre de vue que son architecture doit être aisément testable.

Par ailleurs, on ne fait de conception que pour les fonctionnalités existantes, pas pour les fonctionnalités futures. On n'introduit donc pas d'optimisations si elles ne sont pas demandées par le client.

En somme, plutôt que de préparer le logiciel pour le futur, on s'assure que le travail actuel soit bien fait (lisibilité du code, simplicité, tests) pour que les changements restent faciles et rapides.

1.2. Remaniement

Un des principes de base d'XP est la lisibilité du code, le remaniement est une pratique qui aide à cela. Quand, à force de fonctionnalités ajoutées au logiciel, le code devient trop illisible, ou que par exemple plusieurs fonctions réalisent la même tâche, il est du devoir des programmeurs de remanier le code pour le rendre plus lisible et plus simple.

Les remaniements consistent par exemple à renommer les variables, les méthodes, les classes, déplacer des attributs ou des méthodes vers d'autres classes, décomposer les grosses méthodes en des méthodes plus petites, factoriser des parties de code, etc.

C'est un moyen de faire émerger un design aussi adapté que possible aux besoins de l'application, en supprimant au fur et à mesure tout ce qui nuit à sa simplicité.

1.3. Tests fonctionnels

Ils sont définis par le client, au moyen de cas d'utilisation, puis codés et automatisés par le testeur. Ils servent à définir si le logiciel remplit bien les fonctionnalités demandées par le client. Les tests fonctionnels sont réalisés pour les premières fonctionnalités durant la phase d'élaboration, puis tout au long de la phase de construction.

2. Travail en équipe

2.1. Programmation en binôme

Les programmeurs travaillent en binôme, ceci a pour but d'améliorer la qualité du code. En effet, pendant qu'un des deux développeurs code, l'autre relie immédiatement le code, ce qui permet une forte réduction du nombre d'erreurs de codage. Ensuite cela permet d'obtenir une conception simple plus propre, basée sur deux avis plutôt qu'un seul. Les binômes ont pour vocation de changer fréquemment, ce qui a pour objectif d'améliorer la cohésion de l'équipe de développement et de faciliter un partage des connaissances.

2.2. Responsabilité collective du code

Chaque programmeur travaille sur l'ensemble du logiciel, et est donc responsable de l'ensemble du code. L'application n'est pas découpée en zones réservées à chaque développeur : chacun est susceptible de travailler sur toutes les parties de l'application. Chacun a donc la responsabilité d'écrire un code aussi clair que possible, sachant que d'autres y travailleront peu après.

D'autre part le fait qu'une même portion de code soit visitée par plusieurs développeurs diminue les risques d'erreur et augmente sa qualité. Cette pratique est encouragé par la programmation en binôme, qui fait varier les tâches sur lesquelles chacun travaille, et le remaniement.

2.3. Règles de codage

Dans l'optique de rendre le code lisible par tous, les programmeurs doivent respecter des règles de codage afin de rendre le code homogène dans tout le logiciel. Cette pratique est encouragée par le remaniement : si un programmeur voit une partie de code ne respectant pas les règles de codage, il doit le remanier pour qu'il les suive. Ainsi, naturellement, les programmeurs en viendront à tous utiliser les règles de codage.

2.4. Intégration quasi–continue

Dès qu'un développeur finit une tâche, il met à jour la version d'intégration en s'assurant que tous les tests de non–régression de l'application passent à 100%. La version "livrable" du logiciel évolue donc chaque jour, en reflétant fidèlement l'état d'avancement des développements.

Cela permet à l'équipe de développement d'être continuellement sûre de la fiabilité du logiciel, cela réduit le risque de problème d'intégration et cela permet à l'équipe d'avoir une bonne perception de l'avancée du projet, ce qui peut être très stimulant.