Livre Blanc

Présentation de la méthode OpenUp 1.0

Ce livre blanc vise à présenter le processus de développement logiciel OpenUP et se veut une synthèse augmentée des différents documents disponibles actuellement en anglais.

Degré de difficulté : avancé

Suivi des révisions

EDIS Consulting
53-55 rue du Capitaine Guynemer
92400 Courbevoie

Tel : 01 47 89 63 10
Fax : 01 47 89 59 04
http://www.edis-consulting.com

Xavier Méhaut

Indice	Date	Auteur	Observations
1.11	22/10/2007	Xavier MEHAUT	– Corrections du document

Xavier Méhaut

Sommaire

Table des illustrations

Xavier Méhaut

I - INTRODUCTION

OpenUP est une méthodologie définie dans le cadre du sous-projet EPF (Eclipse Process Framework) de la fondation Eclipse, modélisée puis générée via l'outil de définition de processus EPFC (Eclipse Process Framework Composer).

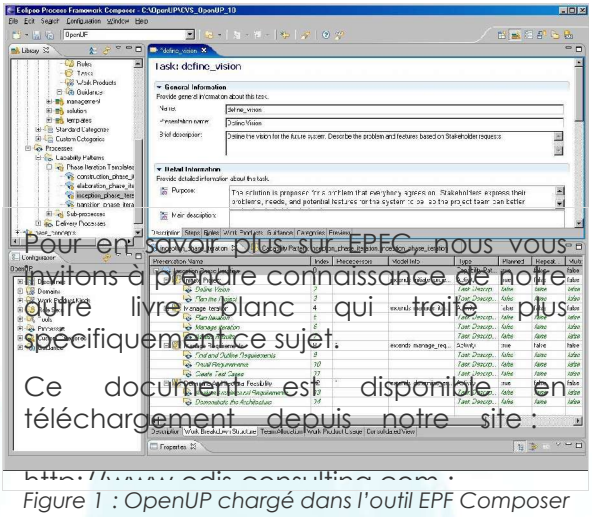

Figure 1 : OpenUP chargé dans l'outil EPF Composer

Pour en savoir plus sur EPFC nous vous invitons à prendre connaissance de notre autre livre blanc qui traite plus spécifiquement ce sujet.

Ce document est disponible en téléchargement depuis notre site :

http://www.edis-consulting.com

EPFC a permis la formalisation de la méthode ainsi que la génération du site WEB décrivant la méthode ; l'intranet ainsi généré rend la navigation dans la méthode plus interactive.

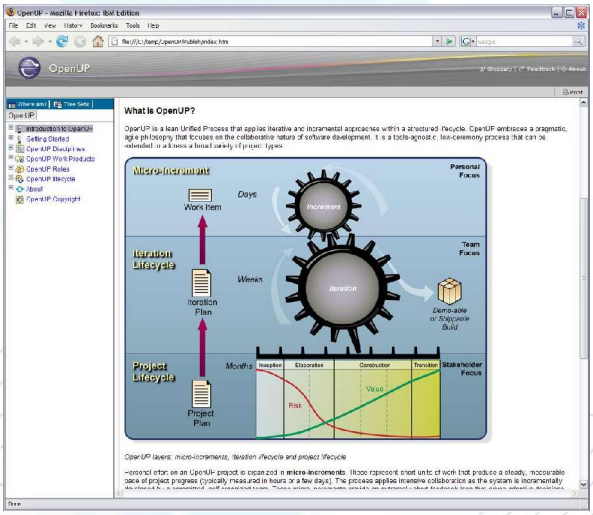

Figure 2 : Site WEB généré pour OpenUP à partir d'ECPF

Xavier Méhaut

II - Qu'est-ce qu'OpenUP ?

Pour paraphraser la définition donnée par les concepteurs de la méthode, nous pouvons dire qu'OpenUP est un Processus Unifié Ouvert léger (Open Unified Process en anglais) qui s'appuie sur une approche, aujourd'hui classique, itérative et incrémentale se situant à l'intérieur d'un cadre bien formalisé, structuré et reconnu pour sa pertinence.

OpenUP est pragmatique et adopte une philosophie de type agile qui se focalise sur la nature collaborative du développement logiciel. La méthode est agnostique en ce qui concerne les outils. Elle peut être étendue ou dérivée pour décrire des processus particuliers et spécifiques.

OpenUP est livrée sous licence Eclipse Public License v1.0[1].

Il semble que la méthode ait été dimensionnée pour de petits ou moyens projets, ce qui ne l'empêche pas d'être utilisable dans le cadre de projets plus importants. Dans ce dernier cas, le projet est géré de manière classique, les sous-systèmes (ou lots) constituants alors des projets OpenUP.

OpenUP trouve ses gènes dans le processus unifié de rational (RUP). Le RUP a été allégé et adapté aux contraintes de développement de type agile. OpenUP bénéficie aussi des travaux en méta-modélisation de sous-projets Eclipse comme UML2, EMF, etc. ainsi que des travaux sur la méta-modélisation des processus comme SPEM[2] (Software Process Engineering Metamodel), projet de recherche appliquée de l'OMG[3].

OpenUP repose par ailleurs sur trois pierres angulaires :

- Les cas d'utilisation (use case) et les scénarios
- La gestion des risques
- Une approche centrée sur l'architecture

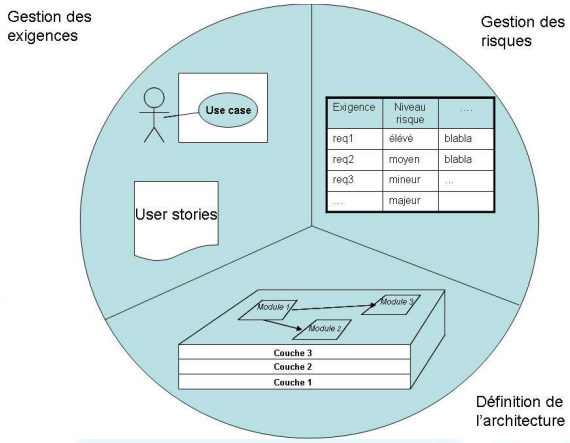

Figure 3 : Les 3 pierres angulaires de la démarche OpenUp

Pour résumer, OpenUP est :

- Minimale et efficiente : seul ce qui est fondamental y est décrit
- Complète : est prête à l'emploi sous sa forme actuelle
- Extensible : peut être utilisée comme base pour la définition de tout autre processus/méthode ou bien peut être personnalisée pour des besoins spécifiques

[1] http://www.eclipse.org/org/documents/epl-v10.php

[2] http://www.omg.org/technology/documents/formal/spem.htm

[3] http://www.omg.org/

Xavier Méhaut

III - OpenUP

Nous décrivons plus précisément dans ce chapitre ce que recouvre la méthode OpenUP en se focalisant principalement sur les points clés de la méthode.

Les acteurs, les rôles dans le projet

Nous poursuivons cet article de vulgarisation en vous présentant les divers intervenants identifiés dans le processus décrits par OpenUP, auxquels s'ajoutent d'autres intervenants dans le cas d'OpenUP/DSDM. Il est à noter que chaque membre de l'équipe projet peut, le cas échéant, jouer plusieurs rôles :

- Référent métier (Stakeholders) : il représente un groupe d'intérêts dont les besoins peuvent être satisfaits par le projet. C'est un rôle qui peut être joué par n'importe qui (ou du moins potentiellement) et qui est matériellement affecté par le résultat du projet. Habituellement, on réserve ce privilège aux utilisateurs finaux, la MOA, les experts métier du client, les commerciaux, le marketing, …
- Analyste : il représente les problématiques client en formalisant les données fournies par les référents métier afin de permettre la compréhension du problème à résoudre et de prioriser les exigences
- Architecte : il est responsable de la conception de l'architecture logicielle, ce qui inclut les décisions techniques clefs qui peuvent contraindre in fine la conception générale et l'implémentation
- Développeur : il est responsable du développement d'une partie du système, incluant la conception, l'implémentation, la conduite des tests de vérification[4], la gestion des sources et l'analyse des résultats
- Testeur : il est responsable des activités centrales de test et de validation[5], à savoir l'identification, la définition, l'implémentation, et la conduite des divers tests de validation, ainsi que la gestion de configuration des résultats des tests et de leur analyse
- Chef de projet : il conduit le développement du projet en collaboration avec les référents métier et l'équipe de développement, coordonne les interactions entre référents métier et s'assure que l'équipe est focalisée sur les objectifs du projet.
- Tout rôle : représente n'importe qui dans l'équipe qui peut réaliser une tâche

Armés de ces informations, nous pouvons maintenant découvrir le cycle de vie proposé par OpenUP.

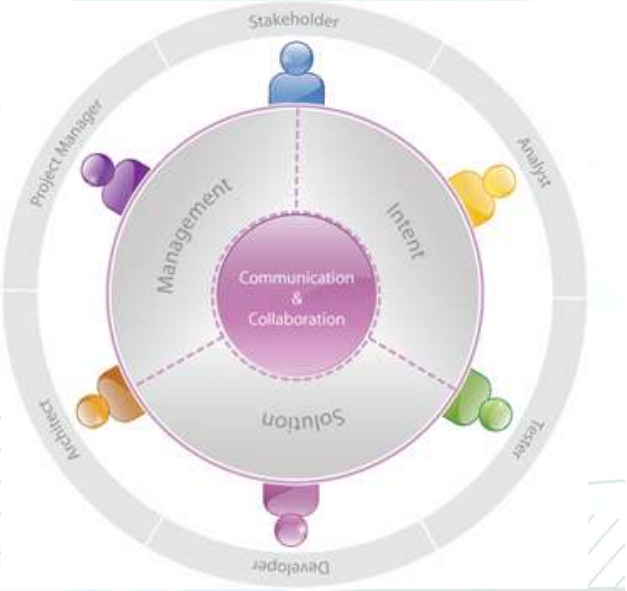

Figure 4 : Relations entre intervenants du proje

[4] Les tests de vérifications visent à montrer que l'application est robuste et que les API du système se comportent comme elles sont spécifiées. Dans un cycle en V traditionnel, cela correspond aux tests unitaires et d'intégration.

[5] On valide que l'application est conforme aux exigences.

Xavier Méhaut

Les principes fondateurs

OpenUp se base sur quatre principes pour interpréter :

- les rôles joués par les divers intervenants dans l'application de la méthode,
- les divers éléments produits par la méthode,
- et l'exécution des diverses tâches inhérentes à l'application d'un processus méthodologique.

Organiser les priorités concurrentes

L'objectif est de promouvoir les pratiques qui permettent aux participants d'un projet et aux référents métier[6] de développer une solution qui maximise les apports, les informations, le temps, bref la valeur ajoutée de ces référents, et qui est compatible avec les diverses contraintes encadrant le projet.

Pour que le développement soit un succès, les référents métier ainsi que les équipes projet doivent converger vers une compréhension claire de la problématique de l'application attendue et doivent préciser et définir sans ambiguïté au moins trois points :

- Le problème à solutionner
- Les contraintes projets, i.e. placées sur l'équipe de développement : coûts, échéances, ressources, réglementations à suivre, …
- Les contraintes environnementales, i.e. placées sur la solution : infrastructure technique préexistante ou imposée, technologies imposées (langage, librairies, patterns, framework, etc.), logiciel préexistant et devant être modifié, …

Voici les points clefs pour trouver ce consensus :

- Connaître le(s) interlocuteur(s) : ceci est vrai aussi bien pour les référents métier que pour les équipes projet
- Séparer le problème de la solution : il est important de bien distinguer ce qui est du domaine du problème (cahier des charges, spécifications, architecture générale, validation, recette) de ce qui est du domaine de la solution (architecture des modules, conception, codage, tests de vérification, etc.) et de respecter les différents domaines de responsabilités
- Créer une compréhension partagée du métier, du domaine sur lequel s'applique l'application
- Utiliser des cas d'utilisation (use case), des scénarios, user stories pour capturer les exigences
- Etablir et maintenir dans le temps l'accord sur les priorités fixées en commun au préalable
- Faire des arbitrages sur les objectifs pour maximiser le développement et les chances de réussite
- Gérer le périmètre fonctionnel ou technique du projet : ce périmètre est forcément flottant à travers le temps. Cette notion de périmètre flottant doit être comprise et acceptée par toutes les parties du projet, et doit permettre de reconsidérer, voire adapter les objectifs initiaux sans heurt ni tension
- Savoir quand arrêter le développement : quand la qualité visée est atteinte, la qualité étant la conformité aux exigences exprimées pour le système.

Collaborer avec tous les intervenants dans le projet

L'objectif est de promouvoir les pratiques qui encouragent un environnement d'équipe sain, permettent la collaboration et développent une même compréhension du projet et de ses objectifs ;

Un rappel toujours utile : un logiciel est créé par des personnes qui ont des intérêts et des compétences différents et qui doivent

[6] Référents métier : client, MOA, utilisateurs finaux, experts métiers, commerciaux, marketing, etc.

Xavier Méhaut

travailler ensemble de manière efficace pour atteindre leurs objectifs respectifs.

Pour arriver à cette collaboration efficace, un certain nombre de règles sont à respecter :

- Maintenir une compréhension commune des objectifs à chaque instant du projet, mais aussi de la méthode utilisée, des outils mis en place et de leur raison d'être, des priorités fixées, etc. La clef du succès est la communication continue et transparente.
- Instituer un climat de confiance élevé entre collaborateurs et avec le client au travers d'actions du type :
  - Manager par la confiance, la délégation : créer un environnement dans lequel les équipes se gèrent elles-mêmes, et les managers servent de leaders et de référents aux équipes afin d'atteindre les objectifs. Écrouler les barrières physiques et mentales qui inhibent la communication, l'initiative, etc.
  - Pratiquer la seconde position, c'est-à-dire se mettre à la place de l'autre afin de comprendre son point de vue avant de critiquer ses idées, ou de répondre à ses critiques
  - Répondre aux conversations avec pertinence en développant et en encourageant les initiatives valorise la curiosité et la pertinence plutôt que l'argumentation et les désaccords.
  - Faire son autocritique d'abord en cas de problèmes de communication
  - Comprendre les contraintes de la culture du lieu de travail
- Partager les responsabilités, savoir déléguer et responsabiliser chacun sur l'atteinte de ses objectifs
- Apprendre continuellement en pratiquant une veille technologique active
- Faire de l'architecture logicielle la plaque tournante (voir section suivante), le point de synchronisation de toute l'équipe

Se focaliser sur l'architecture tôt dans le développement

L'objectif est de promouvoir les pratiques qui permettent à l'équipe de développement de se focaliser sur l'architecture pour minimiser l'effort et organiser le développement.

L'architecture d'un système logiciel est l'organisation ou la structuration des composants significatifs d'un système à travers ses interfaces, avec des composants eux-mêmes constitués de composants plus petits et d'interfaces.

Sans une base architecturale de qualité, un système évoluera de manière inefficace et hasardeuse. Il est de plus difficile d'organiser les équipes de développement, de définir le graphe des dépendances des composants et donc de planifier le travail ainsi que de définir ce que l'on appelle communément le chemin critique.

Se focaliser très tôt sur l'architecture permet de minimiser grandement les risques, d'organiser, de rationaliser et d'optimiser le développement !

Les règles à appliquer sont les suivantes :

- Créer une architecture qui corresponde à la connaissance à un moment donné de la problématique, et non une architecture « générique » qui adresserait des besoins futurs mal ou encore non définis précisément. Une architecture plus générale peut tout de même se justifier si elle fait partie des exigences exprimées par le client.
- Développer l'architecture de manière agile et collaborative, c'est-à-dire en commun, et pas à pas avec des cycles courts tout en gardant une vision précise de l'objectif final. En effet la connaissance à un instant t du problème est forcément partielle et doit s'enrichir par cycles successifs avec l'appui de tous
- Traiter la complexité en faisant appel à la modélisation qui permet d'augmenter le

Xavier Méhaut

degré d'abstraction. Il faut d'abord simplifier la problématique avant de se replonger ensuite dans les subtilités de celle-ci. UML ou d'autres formalismes de ce genre sont utilisés pour décrire cette modélisation.

- Définir l'architecture du systèmes en un ensemble de sous-systèmes à forte cohésion interne et par ailleurs faiblement couplés entre eux. Cela rend le système véritablement extensible, voire réutilisable d'une part, et plus facilement testable d'autre part.
- Réutilisez au maximum les composants qu'ils soient développés, téléchargés ou achetés. L'important est de concentrer ses efforts sur le métier, ce qui est à faire et non sur ce qui est somme toute annexe.

Évoluer continuellement

Il n'est habituellement pas possible au démarrage du projet de connaître tous les besoins du client, d'être conscient de tous les risques du projet, de maîtriser toutes les technologies impliquées dans le projet, ou même de savoir comment travailler efficacement avec vos collègues. Même si cela était possible, tout ceci est susceptible d'évoluer en cours de développement, d'autant plus si ce dernier est long.

C'est pour cela qu'il faut promouvoir des pratiques qui permettent à l'équipe d'obtenir très tôt et de manière continue le retour des référents métier (contre-réaction) sur ce qui est produit, et de démontrer de manière incrémentale que l'on est en phase avec ceux-ci.

Les points importants de cette démarche sont :

- Développer le projet sous forme d'itérations successives bornées dans le temps et, autant faire se peu, régulières. Cette démarche donne de la visibilité sur le projet, tant pour le client que pour les équipes impliquées. Cela permet aussi de détecter très tôt les problèmes éventuels. Il est important de noter que les premières itérations se focaliseront d'abord sur les points à risque (tant techniques que fonctionnels) du chemin critique.
- Concentrer ses efforts à un instant t sur les objectifs fixés de la prochaine échéance temporelle définie dans le plan de gestion de projet. Ni plus, ni moins.
- Gérer les risques en recherchant continuellement l'accord de votre client
- Prendre à bras le corps les changements et les gérer efficacement. Ceux-ci sont inévitables et doivent donc être pris en compte dans la vie du projet
- Mesurer objectivement et avec sincérité la progression du projet
- Continuellement réévaluer ce qui doit être fait en accord avec le client ; des livraisons rapprochées sont une manière naturelle d'établir des points de suivi.

Le cycle de vie général

Nous avons vu plus haut[7] que le processus peut être décrit comme itératif et incrémental. Le schéma ci-dessous vous montre que ce cycle de vie peut être décomposé en trois sous ensembles projet, d'itération et de micro-incrément.

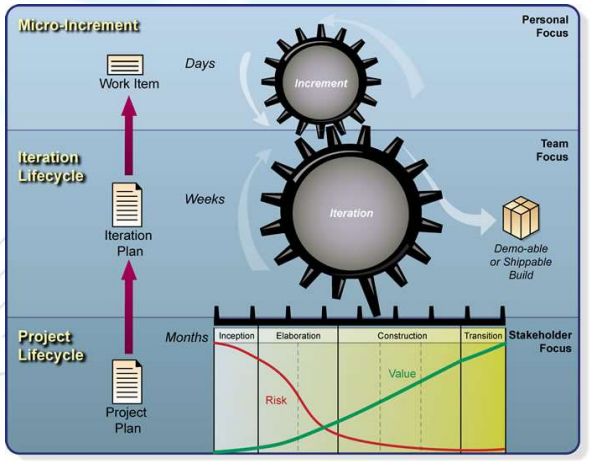

Figure 5 : Les trois cycles de vie du processus

[7] Cf. , page

Xavier Méhaut

Cycle de vie projet

Le cycle de vie Projet peut être décomposé en quatre phases. Les questions à se poser en fin de chaque phase sont :

- Démarrage du projet (*Inception*) : est-on d'accord avec le périmètre du projet et les objectifs ; le projet doit-il être ou non engagé ?
- Élaboration : est-on d'accord sur l'architecture exécutable et estime-t-on que le produit fourni jusqu'à présent et les risques restants sont acceptables ?
- Construction : trouve-t-on que l'on a une application qui est suffisamment proche de ce que l'on envisageait, et que faire si ce n'est pas le cas ?
- Transition : est-ce que l'application est prête à l'emploi ?

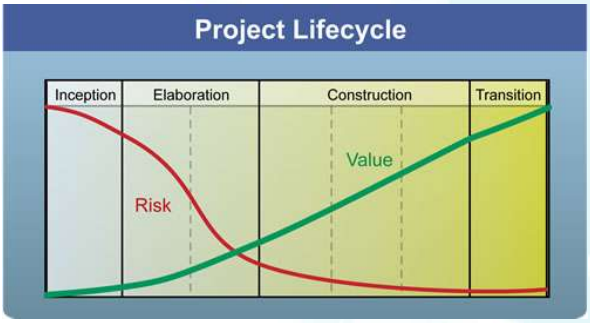

Figure 6 : Les quatre phases du cycle de vie projet

Ce cycle de vie de haut niveau fournit au client et aux équipes projet la visibilité et les points de décisions tout au long du projet. Ceci permet d'avoir une vue efficace du projet, et de permettre des prises de décisions intermédiaires de type « go, no-go » au moment opportun.

Ce cycle de vie est décrit dans un plan projet (Project plan), le résultat final étant l'application livrée.

Démarrage du projet (*Inception*)

Le but dans cette phase est d'obtenir l'accord entre tous les acteurs sur les objectifs du projet, et de savoir si l'on continue ou non le projet… ensemble.

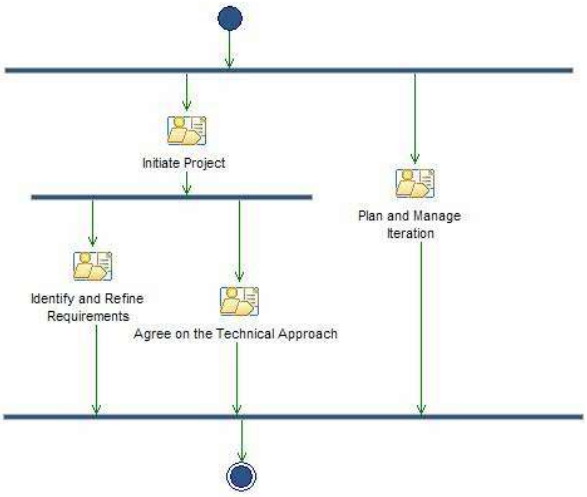

Figure 7 : Diagramme d'activité de la phase

Il y a quatre objectifs principaux dans cette phase :

- déterminer la vision, l'étendue du système visé ainsi que ses frontières, son périmètre. Il est important dans cette phase de déterminer qui fait quoi et pourquoi.
- identifier les fonctionnalités clefs du système, celles qui sont les plus critiques ou à risques
- déterminer au moins une solution possible, identifier au moins une architecture candidate et évaluer sa faisabilité.
- comprendre le coût, les échéances et les risques associés au projet

On peut très bien imaginer dans cette phase, pour aider à atteindre ces objectifs principaux, de se doter d'une maquette ou d'un prototype, ces derniers permettant de lever les zones d'ombres identifiées en commun avec le client.

Élaboration

Le but de cette phase est de discriminer les risques techniques des risques non techniques.

Les risques techniques sont typiquement identifiés en établissant les structures de base d'une architecture opérationnelle du système, fournissant ainsi une base stable pour l'essentiel des efforts de développement de la phase suivante.

Les risques non techniques sont circonscrits, en compagnie des référents métier, par une identification des informations clefs nécessaires.

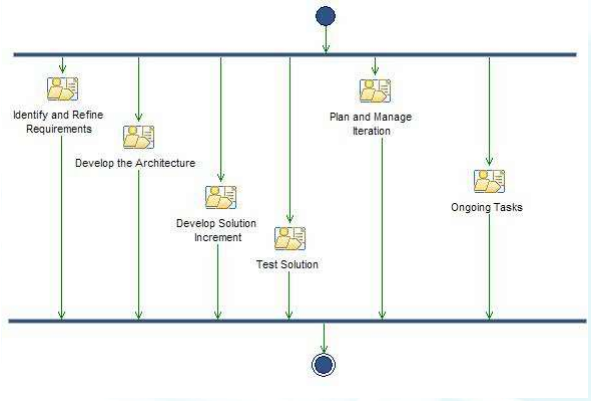

Figure 8 : Diagramme d'activités de la phase

Il y a des objectifs dans la phase d'élaboration qui vous aident à repérer les risques associés aux exigences, à l'architecture, aux coûts et aux échéances. On peut notamment citer :

- Avoir une meilleure compréhension des exigences ; cela ne veut pas dire que tous les use cases nécessitent d'être détaillés en profondeur mais que vous avez détaillé les use cases clefs et que vous avez compris de manière suffisante les autres exigences pour pouvoir les évaluer correctement ultérieurement
- Concevoir, implémenter, valider et établir une architecture de base ; le but est d'avoir un squelette d'architecture testable sur un petit nombre de scénarios critiques. In fine, il s'agit donc d'une architecture exécutable et testable
- Identifier les risques essentiels, produire un planning précis et estimer les coûts

Construction

L'objectif de cette phase est de développer au meilleurs coût un produit complet au niveau fonctionnalités et pouvant être déployé de manière opérationnelle.

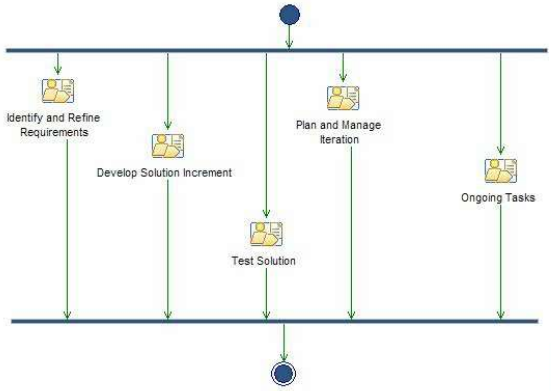

Figure 9 : Diagramme d'activités de la phase

Voici les deux grands points clefs de la phase :

- développer de manière itérative un produit complet qui est prêt à être validé par les utilisateurs
  - en décrivant les exigences qui n'ont pas été complètement décrites dans la phase précédente,
  - en affinant les détails de conception,
  - en finalisant l'implémentation
  - puis en effectuant les tests de vérification sur le logiciel produit. L'utilisation de versions alpha et bêta ou Milestones est de mise.
- Minimiser les coûts de développement et atteindre un certain degré de parallélisation des tâches entre développeurs ou équipes de développeurs en se basant sur le graphe de dépendance des composants qui aura été déterminé précédemment

Xavier Méhaut

Transition

La finalité de cette phase est de s'assurer que le logiciel est prêt à être livré aux utilisateurs finaux.

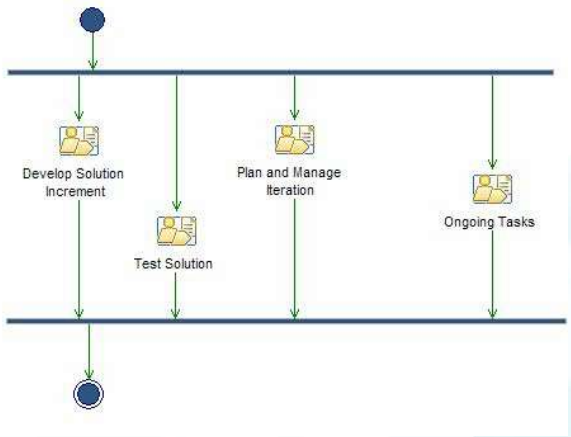

Figure 10 : Diagramme d'activités de la phase

Les moyens identifiés sont :

- Versions en Bêta test ou Release candidate (versions qui n'évoluent plus fonctionnellement) afin de valider en mode opérationnel que les attentes utilisateurs sont bien atteintes.
- Impliquer activement les référents métier (et surtout les utilisateurs) dans les tests de validation dans des conditions d'utilisation opérationnelles différentes afin d'atteindre le niveau d'acceptation final
- Améliorer les performances des futurs projets au travers des leçons apprises lors du développement du projet, tant au niveau processus qu'outils

Cycle de vie d'itération

Les itérations dans OpenUP gardent l'équipe focalisée sur la fourniture toutes les n semaines (à définir lors de la phase précédente) au client d'une application certes incomplète mais néanmoins pleinement testable voire présentable à des utilisateurs ou des référents métier choisis et qui sont au fait du processus de développement. On appelle, dans ce cas, l'application livrée un incrément produit.

Le développement itératif se concentre sur la production du code afin de réduire très tôt les risques de paralysie due à l'analyse. On appelle généralement ce type de développement un développement guidé par les livraisons ou mieux développement par intégration continue (Continuous integration development).

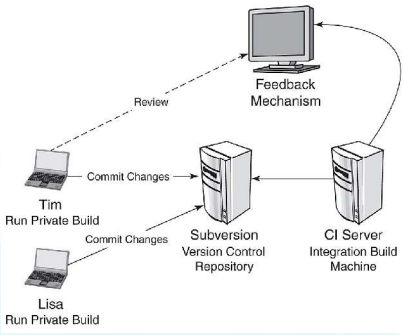

Figure 11 : Serveur d'intrégation continue (CI Server)

La planification des itérations, l'estimation et le suivi de la progression sont centrés sur les éléments qui sont produits. Le plan d'itération est créé en sélectionnant les éléments/items de plus haute priorité, i.e. les exigences, qui doivent être implémentées durant l'itération. Les techniques d'estimation de type agile sont utilisées pour comprendre combien d'items doivent être traités dans le cadre temporel alloué à l'itération.

Tout comme un projet suit un cycle de vie donné, les itérations possèdent leur propre cycle de vie se focalisant sur différents objectifs selon si l'équipe se trouve à la première ou à la dernière semaine de l'itération comme indiqué ci-dessous :

Xavier Méhaut

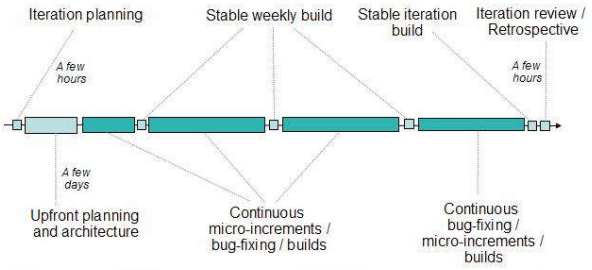

Figure 12 : Cycle de vie de l'itération

Une itération commence avec une réunion de planification qui dure quelques heures au plus. Les deux premiers jours sont typiquement centrés sur le planning futur et sur l'architecture pour, parmi d'autres choses, comprendre les dépendances et l'ordre logique des items sélectionnés pour l'itération en cours, ainsi que les impacts architecturaux du travail qui est à effectuer.

La majeure partie du travail durant une itération est passée à la réalisation de micro-incréments que nous présenterons dans la sous-section suivante. Chaque micro-incrément doit produire un code testé (test de vérification) et compilé, ainsi que les divers artéfacts (produits générés par l'application) qui auront été validés.

Par ailleurs, une version stable de l"application est fournie chaque semaine. On porte une extrême attention à ces « builds » hebdomadaires pour s'assurer qu'il n'y a pas de dérive dans la qualité du code produit. On utilise pour cela abondamment des frameworks de tests unitaires de type JUnit pour automatiser les tests, ainsi que des outils comme Maven ou Cruise control, pour produire automatiquement l'exécutable et fournir des comptes-rendus de processus de construction et des tests. C'est ce que l'on appelle aussi un développement guidé par les tests (*Test driven development*), complémentaire du *Continuous integration Development*.

L'itération se termine avec l'approbation par les référents métier de ce qui a été produit, ainsi qu'une rétrospective sur comment améliorer le processus à chaque itération.

Cycle de vie de micro-incrément

Chaque itération est organisée en micro-incrément. Ce dernier représente le résultat de quelques heures ou de quelques jours de travail pour une ou plusieurs personnes collaborant pour atteindre les objectifs de l'itération en cours

Le concept de micro-incrément aide individuellement les membres de l'équipe à segmenter le travail en plus petites unités de travail ainsi qu'à fournir des éléments parfaitement mesurables à l'équipe. Les micro-incréments fournissent un retour très rapide (contre-réaction en automatique) sur le développement, ce qui permet d'éviter les dérives d'une part, et de donner la possibilité de réorienter les développements le cas échéant d'autre part.

Voici un exemple de micro-incrément :

- Identifier les référents métier compétents pour servir d'interlocuteur sur les exigences traitées dans l'incrément, et permettre la validation de ce qui a été produit
- Développer la solution, i.e. définir, concevoir, implémenter et tester les cas d'utilisation et les scénarios
- Se mettre d'accord sur l'approche technique de persistance des données
- Planifier l'itération

OpenUp fournit un ensemble d'activités pour réaliser cet incrément. Chaque activité est capturée comme un ensemble de tâches, étapes à l'intérieur d'une tâche, et conseils. OpenUP ne fournit pas une description complète des micro-incréments potentiels, chaque organisation devant se constituer au fur et à mesure de ses propres recettes.

OpenUP se veut une cartographie de l'ensemble des actions mises en œuvre dans le processus de développement et doit servir de référence sans pour autant être un carcan rigide intangible ; quand la réalité et le processus ne collent pas ensemble, faites confiance à la réalité !

Xavier Méhaut

Considérations sur le management

La gestion du projet dans OpenUP reprend un pattern de gestion de configuration connu, c'est-à-dire la notion de baseline, ou ligne de vie. Le projet sur cette baseline commence à un point que l'on appelle version source et se termine à un autre appelé version ciblée.

Le but d'un développement est donc de partir d'un état initial stable (vide ou non) et d'arriver à une version stable planifiée et identifiée par un numéro de version et dont le périmètre fonctionnel et technique est validé par toutes les parties du projet, en général le client et le chef/directeur de projet.

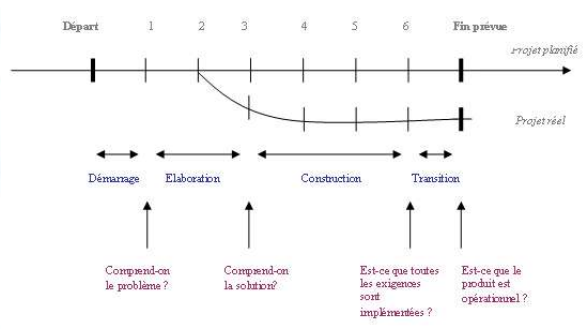

Figure 13 : Cycle de vie de l'itération

Le plan projet développé par le chef de projet défini le jalon de départ et le jalon d'arrivée du projet, ainsi que le nombre de jalons réguliers pour atteindre la version ciblée du logiciel. A chaque fin de phase, il est nécessaire de répondre de manière précise aux questions posées (voir diagramme ci-dessus).

Un certain nombre d'exigences à réaliser sont définies pour chaque jalon ; la somme de toutes les exigences de tous les jalons représent le logiciel dans sa totalité. Normalement, les fonctionnalités traitées dans une itération ne sont pas, hors corrections, retraitées ultérieurement dans une autre itération.

OpenUp, comme toute méthode agile, applique le principe de réalité, c'est-à-dire que l'on prend en compte l'imprévu et les changements au cours du développement. A chaque fin d'itération, on fait le bilan de ce qui a été réalisé, des problèmes rencontrés, du changement éventuel des conditions du projet, etc., et on réactualise les objectifs de l'itération suivante, voire des suivantes si on le peut dans le plan projet. Seul le planning initial des itérations est respecté[8].

Le plan projet définit pour chaque itération les exigences à implémenter et les hiérarchise. C'est ce mécanisme qui permet de suivre le principe de réalité.

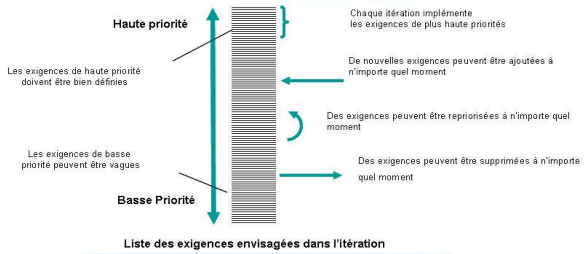

Figure 14 : liste des exigences dans l'itération

Considérations sur le développement de la solution

L'architecture

Comme déjà évoqué, un point essentiel de la méthode est de mettre l'accent sur l'architecture très tôt dans la vie du projet. Nous pouvons, comme le font *Krolls* et *Lyons*[9] dans leur description d'OpenUP, mettre en perspective la définition de l'architecture en regard des quatre objectifs principaux de la méthode.

- Collaborer : l'architecture permet de maintenir une compréhension technique commune entre tous
- Organiser les priorités : les décisions sur l'architecture servent aussi à bien définir les priorités, ne serait-ce qu'en

[8] Voir diagramme précédent.

[9] Article *OpenUp in a nutshell*.

permettant la définition du graphe de dépendance des modules/composants.

- Se focaliser sur l'architecture : cela a pour vertu de se focaliser sur les décisions techniques essentielles
- Savoir évoluer : des évolutions précoces garantissent la faisabilité du produit et permettent de réduire de manière significative les risques.

Pour décrire l'architecture cible, il n'est pas nécessaire de s'encombrer de tonnes de documents. On peut se baser sur les diagrammes UML idoines, ou mieux, se référer à des Architecture Patterns[10] connus ou encore à des documents décrivant une architecture existante dont on donnera la référence. Il faudra aussi argumenter les choix.

La conception

Concevoir signifie suivre étape par étape un chemin permettant d'obtenir au final à une modélisation réaliste et réalisable de l'application attendue :

- Comprendre les exigences formalisées, identifier un scénario
- Identifier les éléments manipulés
- Déterminer comment ces éléments peuvent/doivent collaborer
- Raffiner les choix de conception, le design interne
- Et surtout communiquer pour trouver les meilleures solutions au moindre coût et bien faire comprendre les choix de conception

Lors de cette phase, OpenUp n'impose pas un cadre rigide. La phase d'analyse[11] classique du cycle en V n'est pas obligatoire. Mais plus le design est important, plus cette phase d'analyse est conseillée.

[10] Architecture pattern, patron de modélisation décrivant une architecture type correspondant à une problématique donnée.

[11] L'analyse est une phase de modélisation du problème ne prenant pas en compte les contraintes de l'implémentation.

Une conception graphique ou la réalisation de maquettes dépendra du type de projet visé. L'utilisation d'outils de conception n'est pas non plus obligatoire, mais conseillée, etc.

On applique par contre le concept de *Test-First Design*, c'est-à-dire la faculté d'implémenter le test unitaire en même temps que les interfaces (API) du composant en cours de conception. On dit que l'on est en présence d'un développement guidé par les tests.

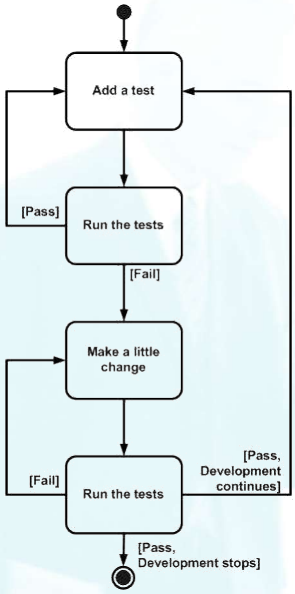

Figure 15 : Cycle de vie de l'itération

Les tests définis sont aussi bien des tests de succès que d'échec. Les tests sont généralement écrits via des frameworks de test de type *XUnit*.

Les rôles dans le développement de la solution

Les personnes impliquées dans le développement de la solution ont les responsabilités suivantes :

- Développeur : il modélise et implémente la solution
- Architecte: il prend les décisions techniques clefs et structure la solution. On peut aussi le cas échéant distinguer architecte fonctionnelle d'architecte technique

- Testeur : il implémente et conduit les tests et valide la solution
- Directeur/chef de projet : il pilote les développements
- Référent : il participe à la validation de la solution
- Analyste : il fournit une information additionnelle, une expertise

Xavier Méhaut

IV - Conclusion

Comme nous le disions plus haut dans cet livre blanc, l'objectif de ce dernier n'est pas de proposer un mode d'emploi d'OpenUP, mais de vous sensibiliser sur les choix faits par l'équipe qui a défini cette méthode, et surtout de vous imprégner de l'état d'esprit général de ce processus. « La lettre tue, l'Esprit vivifie », dirait Paul de Tarse.

OpenUP ne comporte pas de concepts révolutionnaires et représente par maints égards une synthèse de pratiques courantes dans le monde des méthodes agiles au travers de projets concrets comme par exemple la façon dont Eclipse est développée (même si Eclipse a sa propre méthode de développement agile adaptée à ses contraintes de développement).

OpenUp veut clairement être le pont entre des méthodes plus traditionnelles comme le RUP, le cycle en Y, en V, en W, …, et le monde des méthodes agiles. Le but officieux est sans doute de pouvoir combiner la confiance dans des processus de développement connus, mieux compris et déjà maîtrisés et les retours d'expérience élogieux dans l'application de méthodes plus légères et agiles. Ce n'est pas une rupture mais vraiment une synthèse intéressante.

Les concepts sont clairs, la documentation fournie, l'utilisation d'EPFC pour la formalisation rassurante, et les bases saines.

Quels sont les points obscurs pour l'instant ?

- Peu de recul sur des projets industriels pour le moment ;
- Une communauté encore embryonnaire, mais cet article va sans doute changer les choses ;)
- Une méthode pas encore outillée complètement, exceptée dans sa définition ;
- La concurrence de méthodes ayant des retours d'expérience plus nombreux comme XP, Scrum, Lean development, EclipseWay, …
- Une méthode de plus ?

L'adoption d'OpenUP n'induit pas à mon avis de risque important car la filiation et les fondamentaux sont bons… L'expérience nous montrera les limites et avantages de ce processus, qui pourra alors être amendé, enrichi, décliné grâce à EPFC, source d'un autre article à venir.

Documents de référence

Nous n'avons pas mis dans ce livre blanc toutes les références aux articles ou présentations nécessaires à l'écriture de celui-ci. Ces informations se trouvent sur http://eclipse.org/epf, et nous avons plus particulièrement pris comme source les articles ou présentations suivantes :

- Site WEB généré par EPFC pour la méthode OpenUP
- Présentation powerpoint OpenUP distilled
- Article Introduction to OpenUP
- Article OpenUP in a Nutshell
- Wiki sur OpenUP
- Articles sur ECPF part 1 et 2
- Article OpenUp vision
- Article Introducing continuous integration, by Paul Duvall, Steve Matyas, Andrew Glover, JavaWorld.com, 06/21/07