DOCUMENTATION INFORMATIQUE.
MÉTHODOLOGIE DE DÉVELOPPEMENT

MÉTHODES AGILES
XP ET SCRUM

Par: Bernard GIACOMONI

VERSION	OBJET	DATE
1.0	Création	13/11/2019

Table des matières

I.AVANT PROPOS:

Ce document s'adresse à des informaticiens ayant une connaissance au moins théorique des pratiques de base du développement de logiciels. Son but est de leur transmettre une connaissance «synoptique» des principes généraux des méthodes de conduite de projets appelées «Méthodes Agiles»

Beaucoup d'auteurs opposent ces méthodes aux méthodes classiques, souvent dites «en cascade», dont la plus utilisée est le «cycle en V», encore largement utilisé de nos jours pour les réalisations de gros volume. Néanmoins, le but de cet ouvrage n'est pas de promouvoir une pratique par rapport à une autre, mais d'aider les responsables de projets à choisir la pratique la plus adaptée aux problèmes qu'ils doivent résoudre.

Une connaissance au moins superficielle des concepts liés à la conduite de projets informatique et du «cycle en V» est donc recommandée pour la compréhension de ce document. De ce fait, les premiers chapitres de cet ouvrage en font un rapide exposé. Les lecteurs ayant une bonne connaissance de ces notions pourront parcourir rapidement, voire ignorer cette partie de l'ouvrage.

L'ouvrage aborde ensuite la présentation des différents principes et des pratiques caractéristiques des méthodes agiles, de leurs avantages et de leurs inconvénients vis à vis des méthodes «classiques».

La dernière partie est consacrée à une présentation des méthodes SCRUM et XP, les méthodes agiles les plus utilisée actuellement.

II.RAPPELS SUR LES CONCEPTS DE BASE:

II.1.DIFFÉRENTES ACTIVITÉS D'UN PROJET INFORMATIQUE:

II.1.1.LES ACTIVITÉS DU CYCLE DE DÉVELOPPEMENT:

Un PROJET informatique dont la finalité est la création d'un PRODUIT NOUVEAU (Logiciel ou autre) intègre toujours les sept types d'activités suivantes:

- **Spécification du Besoin**;
- **Conception préliminaire**;
- Pour chacun des composants identifiés en conception préliminaire:
  - **Conception détaillée**;
  - **Codage**;
  - **Tests unitaires**;
- **Intégration** (chacun des composants développés dans les trois phases précédentes);
- **Réception-Qualification**.

II.1.2.MAÎTRISE D'OUVRAGE ET MAÎTRISE D'ŒUVRE:

NOTION DE MAÎTRISE D'OUVRAGE:

Les activités de Spécification du Besoin et de Réception-Qualification sont du ressort du CLIENT. En effet, elles permettent à celui-ci de garantir ses intérêts dans le projet:

- Avant la réalisation, s'assurer que le FOURNISSEUR est bien informé de son BESOIN et des conditions (techniques, environnementales, réglementaires, etc.) lesquelles il sera confronté;
- En fin de réalisation, s'assurer que le produit livré correspond bien à ce qu'il a commandé et qu'il peut être utilisé dans les conditions opérationnelles.

Ces activités font partie de la MAÎTRISE D'OUVRAGE. Elles conviennent aussi bien à la réalisation d'un nouveau produit qu'à l'achat d'un produit existant.

- Le résultat de l'activité de spécification du besoin est en général un document appelé «Cahier des Charges» ou «Spécifications de besoin».
- L'activité de Réception consiste à dérouler des tests de réception habituellement consignés dans un ou des documents appelés «Cahiers de Tests de Réception» ou «Cahiers de Réception Détaillés». Ceux-ci explicitent les procédures de test à dérouler pour effectuer la réception:
- Souvent, les méthodes prévoient l'élaboration de deux autres documents préparatoires à la réception, qui sont également du ressort de la Maîtrise d'Ouvrage:
  - Un Plan de Réception, qui définit le périmètre de la réception (liste des fournitures à tester, responsable de chaque fourniture: le client ou le fournisseur) et planifie les opérations de réception.
  - Un Cahier de Réception Général qui définit pour chaque fourniture les points à tester et la méthode de test;

NOTION DE MAÎTRISE D'ŒUVRE:

- Les activités liées au développement du produit (Conception, codage et tests, intégration), sont évidemment sous la responsabilité du FOURNISSEUR.
- Celui-ci assume également la responsabilité de la planification et de la conduite de ces phases du projet. Pour ce faire, il doit élaborer un PLAN DE DÉVELOPPEMENT qui identifie toutes les taches à effectuer (Certaines tâches, comme la livraison de matériels ou les formations nécessaires ne font pas partie du cycle de développement mais doivent être intégrées dans le planning).
- L'intégration doit aussi être planifiée par le fournisseur (dans quel ordre intègre-t-on?).

Toutes ces activités font partie de la MAÎTRISE D'ŒUVRE.

REMARQUE: MOA, MOE ET AMOA:

La Maîtrise d'ouvrage est souvent désignée par le sigle MOA (Maîtrise d'OuvrAge), tandis que la Maîtrise d'œuvre est souvent désignée par le sigle MOE (Maîtrise d'OEuvre). MOA et MOE désignent des personnes morales (l'entreprise cliente, l'entreprise chargée de la fourniture).

L'entreprise Maître d'Ouvrage ne possède en général pas en son sein les compétences nécessaires pour assumer en totalité les tâches qui incombent à la MOA (spécification du besoin, planification, validation du produit). Dans ce cas, elle délègue totalement ou partiellement ces tâches à un Assistant à la Maîtrise d'OuvrAge (AMOA).

L'activité de l'AMOA consiste donc à assister la MOA dans sa tâche en mettant en œuvre les moyens et les compétences nécessaires.

II.1.3.SPÉCIFICATION DU BESOIN :

DESCRIPTION DE L'ACTIVITÉ:

cette activité concerne en premier lieu:

- la définition et la caractérisation des exigences que le produit doit satisfaire;
- la description des interfaces externes du produit.

Dans la plupart des méthodes «classiques», le résultat de cette activité est consigné dans le document appelé **Spécifications Techniques de Besoin** (STB). Il résulte du recueil des besoins des différents utilisateur vis à vis du produit, de leur analyse et de leur traduction en termes d'EXIGENCES (fonctionnelles, opérationnelles, techniques, etc.). Son contenu est composé (au minimum):

- De la liste des EXIGENCES des utilisateurs vis à vis du produit, **caractérisées** et **ayant un sens au niveau technique**;
- De la description des INTERFACES EXTERNES du produit: interfaces techniques avec l'environnement matériel et définition des interfaces des utilisateurs avec le produit (IHM).

REMARQUES:

- Ces travaux sont en théorie de la responsabilité du CLIENT. En fait, celui-ci a rarement l'expertise méthodologique pour les mener à bien sans aide extérieure;
- Si le client veut mettre en concurrence plusieurs fournisseurs pour la réalisation du produit, la S.T.B devient un élément de l'APPEL d'OFFRES. La STB ne peut donc pas être élaborée avec l'aide du fournisseur qui n'est pas encore choisi. Si le client veut une assistance, il devra donc choisir une entreprise expressément pour cette prestation d'assistance méthodologique;
- Le client peut aussi choisir de confier dès le départ une prestation globale comprenant l'aide aux spécifications et la réalisation à un seul fournisseur (avec ou sans mise en concurrence).

LA NOTION D'EXIGENCE:

Une exigence doit être décrite par une phrase simple, compréhensible par l'utilisateur. Par exemple:

1. *L'**administrateur du site** doit pouvoir visualiser le nombre de **visiteurs connectés**.*
2. *Les **formation à l'utilisation** du produit devront être dispensées aux différents **utilisateurs** avant la date prévue pour la réception sur le site.*

Si ces formulations ont un sens pour le client ou l'utilisateur, elle n'ont encore qu'une signification assez vague du point de vue du réalisateur. C'est pour cela que dans une STB, les exigences sont accompagnées de **critères évaluables** qui les caractérisent. Par exemple, si l'on veut caractériser l'exigence N°1:

CRITÈRES	NIVEAU	FLEXIBILITÉ	JUSTIFICATION
Quand ?	A tout moment, sur activation d'un bouton situé sur une page d'administration.	Délais < 3 s.	La surveillance de la fréquentation du site est du ressort de l'administrateur système.
Comment?	Dans un champ situé sur la même page d'administration.	Sans objet	Le nombre de visiteur connecté ne doit pas être visible par les internautes sans privilèges administrateurs.
Etc...			

REMARQUES:

- Un document listant les exigences sans inclure ni critères de caractérisation ni description des interfaces techniques est souvent appelé CAHIER DES CHARGES FONCTIONNEL (CDCF).
- Dans un marché public ou privé concernant la fourniture de logiciels, le document STB est très souvent annexé au CAHIER DES CHARGES (celui-ci comprend également des spécifications administratives ou financières);
- La liste des exigences FONCTIONNELLES est souvent appelée «Cahier des Charges Fonctionnel» (CDCF);
- Dans les méthodes classiques, la STB est un document CONTRACTUEL. Ce n'est pas le cas dans le contexte des méthodes agiles.

REMARQUES CONCERNANT LA NOTION DE FONCTION:

Très souvent, dans les documents traitant des méthodes classiques comme des méthodes agiles, les EXIGENCES à caractère FONCTIONNEL sont appelées FONCTIONS (ou sous-fonctions) du produit. En effet, une exigence du genre:

- L'**administrateur du site** doit pouvoir visualiser le **nombre de visiteurs connectés**.

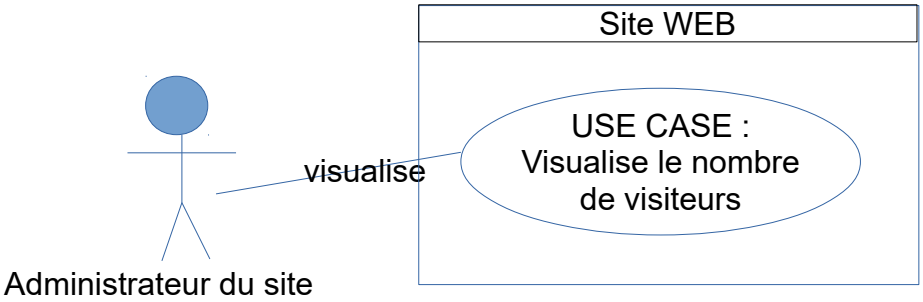
est équivalente à la formulation:

- Le **produit** permet à l'**administrateur du site** de visualiser le **nombre de visiteurs connectés**.

Qui définit indubitablement une FONCTION du produit.

Cependant, les méthodes de conception dites «guidées par les données» (en général basés sur les notion d'OBJET et de CLASSE), proscrivent l'utilisation de la notion de FONCTION, lui préférant la notion de CAS D'UTILISATION (USE CASE), tirée du «langage» UML:

Exemple de cas d'utilisation:



La différence entre la notion de FONCTION et celle de CAS D'UTILISATION peut être expliquée ainsi:

- Une FONCTION est une notion centrée sur le PRODUIT à réaliser. De ce fait, Il est très facile (et tentant) de passer directement de la FONCTION à l'ALGORITHME qui la réalise. C'est ce qu'on appelle une conception GUIDÉE PAR LES TRAITEMENTS. Ce type de démarche, si elle est intuitive et simple à mener, possède beaucoup d'inconvénients: en particulier, elle néglige l'étude des données et défavorise la réutilisation. De ce fait, on la réserve aux prototypages ou aux logiciel sans exigence de qualité ou de performances;
- Un CAS D'UTILISATION décrit en revanche le comportement d'un utilisateur. En conception objet, on cherche à définir à partir d'un USE CASE les DONNÉES qui participent au cas, puis les CLASSES qui vont les encapsuler et offrir des méthode de traitement au reste de l'application. Ceci initie plutôt une conception GUIDÉE PAR LES DONNÉES.
- Remarquons que la notion d'EXIGENCE, centrée sur l'utilisateur, est très proche de la notion de CAS D'UTILISATION et qu'une FONCTION n'est rien d'autre qu'une entité logique «résolvant» un certain nombre de cas d'utilisation.

ACTIVITÉS ANNEXES:

Accessoirement, certains travaux de préparation de la VALIDATION du produit peuvent être réalisés EN AVANCE en parallèle aux SPÉCIFICATIONS. Il s'agit du PLAN DE RÉCEPTION (ou encore plan de validation, plan de recette) qui:

- Détermine le PÉRIMÈTRE de la validation (ce qui doit être validé, ce qui n'a pas besoin de l'être);
- Détermine les ressources nécessaires (environnement, fournitures, moyens humains);
- Planifie les opérations de validation.

En effet, ces travaux sont également de la responsabilité du CLIENT. La plupart du temps, celui-ci a besoin d'une aide méthodologique qui peut être apportée par une société spécialisée.

II.1.4.CONCEPTION PRÉLIMINAIRE:

DESCRIPTION DES ACTIVITÉS:

La finalité première de l'activité de conception préliminaire est de déterminer l'architecture globale du produit la plus adaptée aux exigences. Le résultat de ces travaux est en général consigné dans un Document de Conception Préliminaire (DCP).

Accessoirement, les résultats de la conception préliminaire permettent d'affiner l'estimation du nombre et du volume des tâches à réaliser et la planification de ces tâches. Le résultat de ces travaux est consigné dans le Plan de Développement (PDev) et l'Organigramme des Tâches (OT).

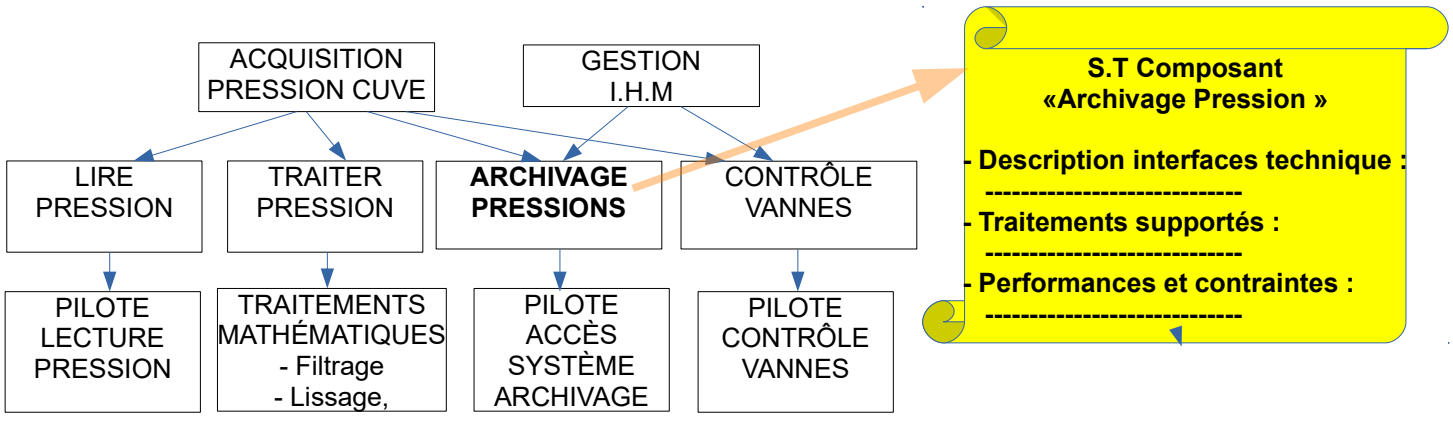
Ces travaux permettent également d'élaborer le Plan d'Intégration (PI), qui défini l'ordre d'intégration des composants produits.

L'architecture globale des logiciels comporte deux aspects:

- L'ARCHITECTURE STATIQUE, qui traite de la décomposition de l'application en différents COMPOSANTS et des relations entre ces composants (leurs INTERFACES TECHNIQUES). Cette démarche est souvent appelée ANALYSE ORGANIQUE, par opposition à l'ANALYSE FONCTIONNELLE qui est utilisée pour l'étude du besoin;
- L'ARCHITECTURE DYNAMIQUE qui traite des problèmes d'ordonnancement temporel ou de parallélisme des tâches et d'accès partagé aux ressources communes.

L'architecture globale doit également définir les SPÉCIFICATIONS TECHNIQUES de chacun des COMPOSANTS: interfaces, traitements encapsulés, performances, normes techniques, etc.

Exemple
d'Architecture STATIQUE :

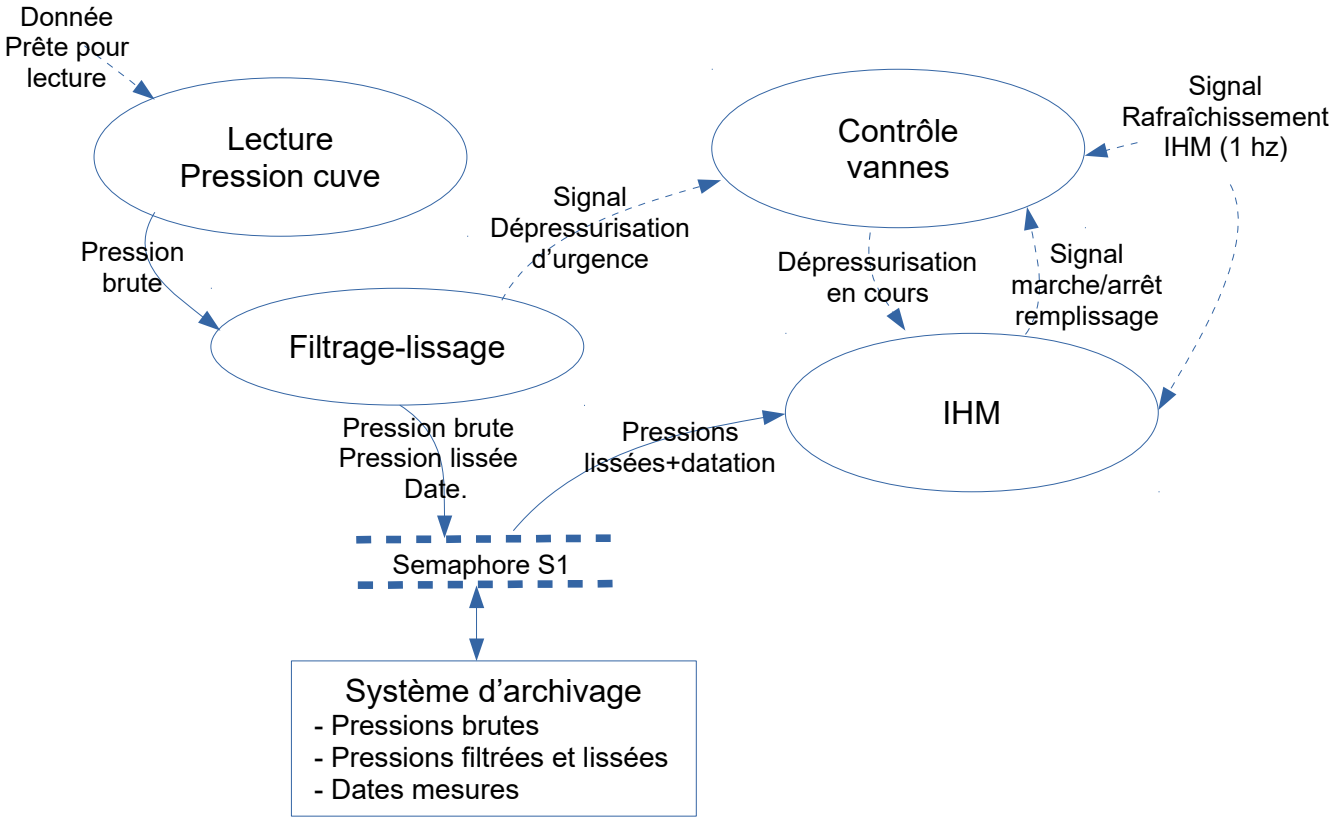


COMMENTAIRES:

- Les entités présentées sous forme de rectangles sont des COMPOSANTS de l'architecture statique. Il peut s'agir de CLASSES ou de BIBLIOTHÈQUES DE FONCTIONS;
- Les flèches en bleu signalent la relation d'UTILISATION d'un composant de niveau inférieur par un composant de niveau supérieur. Elles ne préjugent pas de la manière dont le composant de niveau inférieur est sollicité (appel procédural, messages, signaux, interruptions, déblocage de sémaphores, etc.);

- Les composants de niveau inférieur à 1 offrent aux composants des niveaux supérieurs des SERVICES et non forcément des données (par exemple, le composant LIRE PRESSION lit une pression (sur le capteur de pression) pour le composant ACQUISITION PRESSION CUVE: il peut soit transmettre la mesure au niveau supérieur, soit l'encapsuler pour la délivrer plus tard sur activation d'une autre de ses méthodes);
- Les composants du premier niveau ORDONNANCENT les traitements offerts par les niveaux inférieurs.

Exemple d'architecture dynamique



COMMENTAIRES:

- Les entités figurées sous forme d'ellipses représentent des TÂCHES, c'est à dire des algorithmes séquentiels exécutés par un FIL D'EXÉCUTION (thread) particulier de la machine;
- Deux tâches peuvent donc être exécutées SIMULTANÉMENT (cette simultanéité peut être réelle ou apparente). Elles peuvent également être CONCURRENTES pour l'accès à certaines ressources (voir ici la concurrence des tâches IHM et FILTRAGE-LISSAGE sur la ressource Système d'Archivage);
- Les flèches bleues représentent les relations entre tâches. il peut s'agir d'appels procéduraux, de messages, signaux, ou interruptions, d'accès concurrents a des ressources.

NOTION DE COMPOSANT:

Un COMPOSANT est un élément constitutif BIEN IDENTIFIÉ de l'ARCHITECTURE STATIQUE d'une logiciel. Ceci revient à dire qu'il est défini par des SPÉCIFICATIONS TECHNIQUES précises (interface externe, traitements supportés, procédures d'activation et de communication). Ceci peut donc lui conférer la capacité d'être RÉUTILISÉ dans un autre logiciel.

Il peut ainsi être incorporé dans l'application comme une «pièce détachée» peut l'être dans un système mécanique ou électronique. Les bibliothèques de fonctions logicielles, les classes, les fichiers, les bases de données, les fichiers de données ou de configuration sont des types de composants «de base», mais un composant peut également être constitué d'une architecture agrégeant plusieurs classes (correspondant par exemple à un «design pattern»), plusieurs bibliothèques, etc.

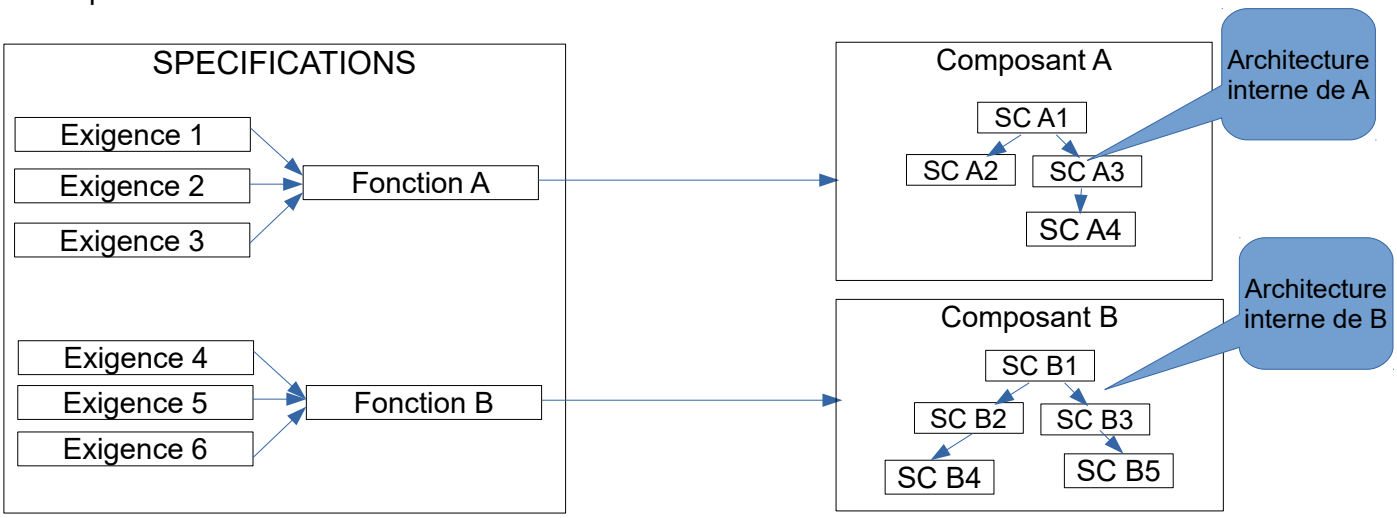
NOTION DE COMPOSANT FONCTIONNEL: un composant d'une application est dit FONCTIONNEL lorsqu'il implémente à lui seul un ensemble d'exigences fonctionnelles (ou de cas d'utilisation). Ceci revient à dire que le composant n'a pas besoin des autres composants de l'application pour être utilisable: il peut donc être livré au client EN AVANCE par rapport à la fin du projet. Cette capacité est très utilisée dans le cadre des méthodes agiles.

LES STYLES DE CONCEPTION ARCHITECTURALE:

Le passage des SPÉCIFICATIONS d'une application à son ARCHITECTURE GÉNÉRALE (c'est à dire le passage des exigences du client aux composants d'architecture) est un des sujets les plus problématiques du génie logiciel. Sans entrer dans le détail, on peut distinguer deux types de démarches: la conception **guidée par les fonctions** (ou par les traitements) et la conception **guidée par les données.** Toutes les méthodes existantes se rattachent à l'un ou l'autre de ces types.

LA CONCEPTION GUIDÉE PAR LES FONCTIONS:

Le schéma ci-dessous illustre le principe de cette démarche: a partir de la liste des exigences, on élabore la liste des FONCTIONS du produit. A chaque fonction est associé un composant qui l'implémente.



- L'avantage de cette démarche est d'être SIMPLE et INTUITIVE. Les composants issus de

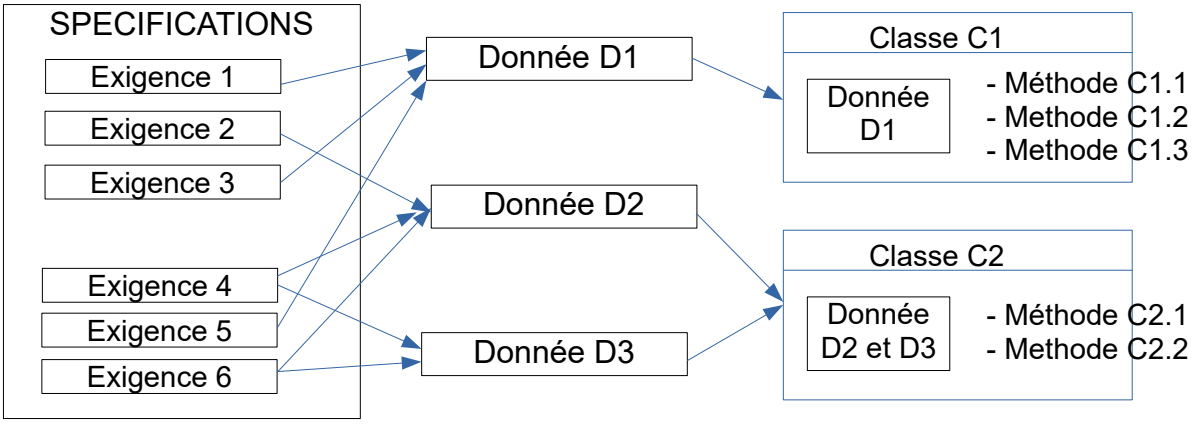
cette démarche sont, à priori, FONCTIONNELS;

- L'inconvénient principal de la démarche est que les architectures internes des composants sont élaborées indépendamment l'une de l'autre, peut-être par des équipes différentes, sans qu'il leur soit possible de détecter des opportunités de réutilisation entre les sous-composants (exemple: le sous-composant SC B4) peut très bien être identique au composant SC A2) ou des incompatibilités entre composants (exemple: le fonctionnement du sous-composant SC A4 peut perturber le fonctionnement de SC B3).

REMARQUE: en fait, la démarche est toujours plus complexe qu'une simple identification fonction-composant: le développeur déduit de chaque FONCTION les TRAITEMENTS qui vont permettre de les réaliser, puis il cherche à encapsuler ces traitements dans des COMPOSANTS (ici, SC A1, SC A2, etc.) et à les faire communiquer entre eux pour réaliser la fonction. Il serait donc plus judicieux de parler de conception guidée par les traitements.

LA CONCEPTION GUIDÉE PAR LES DONNÉES:

Dans cette démarche, on tente d'abord de déterminer quelles DONNÉES devront être manipulées pour implémenter chacune des exigences. Ces données seront alors encapsulées dans des CLASSES qui elles-mêmes seront munies de méthodes permettant de manipuler ces données:



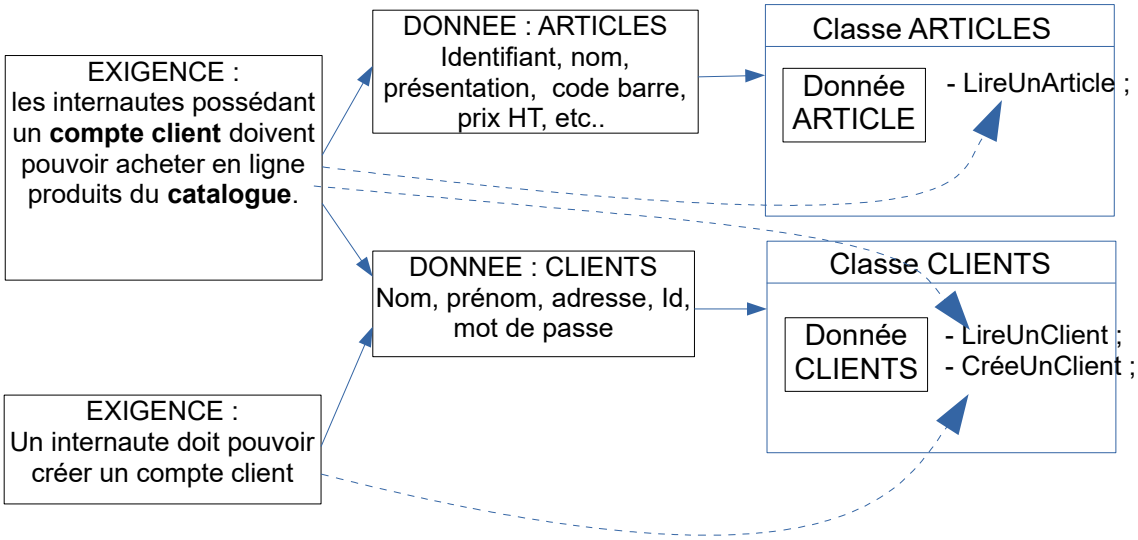
EXEMPLE:

Pour implémenter l'exigence «les internautes possédant un compte client doivent pouvoir acheter en ligne produits du catalogue», il va falloir:

- Déterminer quelles données devront être manipulées. Ici, ce sont les données CLIENTS (comptes clients) et ARTICLES (liste des produits à la vente);
- Encapsuler ces données dans des CLASSES (par exemple: CLIENTS et ARTICLES);
- Munir chacune de ces classes des méthodes qui permettent de manipuler ces données (Ici, ce seront les méthodes ARTICLES.LireUnArticle et CLIENTS.LireUnClient).

Pour implémenter la nouvelle exigence «Un internaute doit pouvoir créer un compte client», il suffira d'ajouter la méthode CLIENTS.CreerUnClient dans la classe CLIENTS.

Et ainsi de suite pour chacune des exigences (ou cas d'utilisation). A la fin de ce processus, tous les composants (classes) auront été identifiés d'une manière logique, en réduisant au minimum les risques de doublons et d'effets de bord.



REMARQUE: Les composants créés (ARTICLES et CLIENTS) ne sont évidemment pas FONCTIONNELS.

II.1.5.CONCEPTION DÉTAILLÉE, CODAGE ET TESTS UNITAIRES:

REMARQUE PRÉLIMINAIRE:

III faut bien garder à l'esprit que les activités de ces phases traitent de **chacun des composants** identifiés lors de la conception préliminaire. Les activités décrites ci-après s'adressent donc à chacun de ces composants.

DESCRIPTION DES ACTIVITÉS:

- La **CONCEPTION DÉTAILLEE** d'un composant intervient lorsque le codage de ce composant ne paraît pas «trivial» (évident). C'est le cas le plus fréquent. Il consiste en la détermination d'une l'architecture interne pour celui-ci. En effet, un composant peut être constitué de plusieurs éléments de base (classes, fonctions, fichiers, structures de données rémanentes, etc.), organisés autour d'une architecture interne. Par exemple, il peut être identifié à un Modèle de Conception (Design Pattern) constitué de plusieurs classes. La conception détaillée d'un composant est donc une analyse organique qui aboutira à des sous-composants «de base», pouvant être codés directement;
- Le **CODAGE** d'un composant peut intervenir lorsque, grâce aux travaux de conception détaillée, l'algorithmique de tous les composants et sous-composants apparaît clairement aux développeurs (la compétence de ces développeurs intervient évidemment dans le niveau d'analyse nécessaire);
- Les **TEST UNITAIRES** sont destinés à détecter les dysfonctionnements et non conformité des composants développés, afin d'aboutir à leur VALIDATION par rapport à leurs SPÉCIFICATIONS TECHNIQUES.

Les résultats de ces activités sont:

- Les CODES OBJETS des composants développés;
- Les PROCÉDURES DE TESTS UNITAIRES (celles-ci seront réutilisés lors de L'INTÉGRATION des composants).

DÉROULEMENT DE CES ACTIVITÉS:

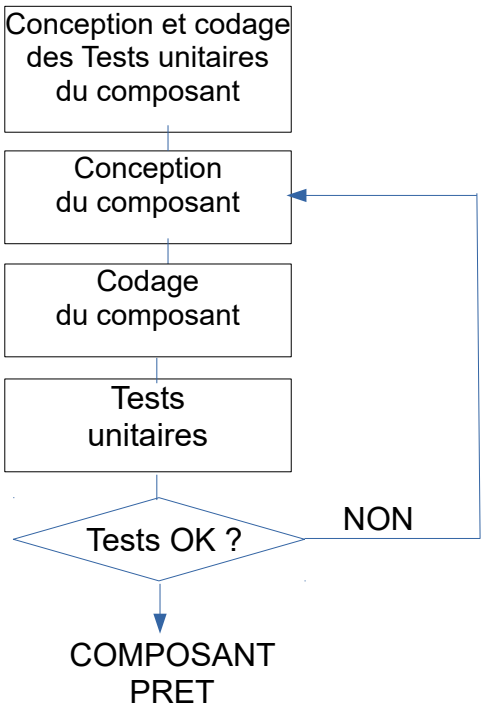
Très souvent, les représentations schématiques des méthodologies classiques présentent les trois phases Conception Détaillée, Codage et Tests Unitaires comme étant CONSÉCUTIVES. Cette manière de présenter les choses induit donc l'idée que l'on effectue **d'abord** la conception préliminaire de tous les composants, **ensuite** le codage de tous ces composants et **enfin** les tests pour tous ces composants.

Or, même dans les projets menés selon les méthodes les plus «traditionnelles», les équipes de développement ne procèdent PRATIQUEMENT JAMAIS de cette manière, pour les raisons suivantes:

- Contrairement à l'architecture globale élaborée en conception préliminaire, l'architecture interne d'un composant est établie «en interne» par l'équipe chargée de la réalisation. Elle n'engage que ce composant. Elle peut donc être très facilement «ajustée» (également «en interne») pour pallier des difficultés survenant lors du codage ou des tests ou pour améliorer le fonctionnement;
- Beaucoup de spécialistes du génie logiciel recommandent d'élaborer les tests AVANT le composant lui-même. Cette pratique, qui a pour avantage de provoquer une relecture critique des spécifications techniques de ce composant, offre aux développeurs de «mettre

au point» progressivement le composant à chacune des étapes de son développement.

De ce fait, la pratique réelle correspond plutôt à l'une des procédures suivantes:



REMARQUE :
La procédure ci-contre ne décrit pas forcément un développement incrémental tel qu'il est pratiqué dans les méthodes agiles : en effet, pour qu'il en soit ainsi, il faudrait que le codage soit effectué en plusieurs incréments, les tests n'étant à chaque itération effectués que sur les incréments effectivement réalisés.

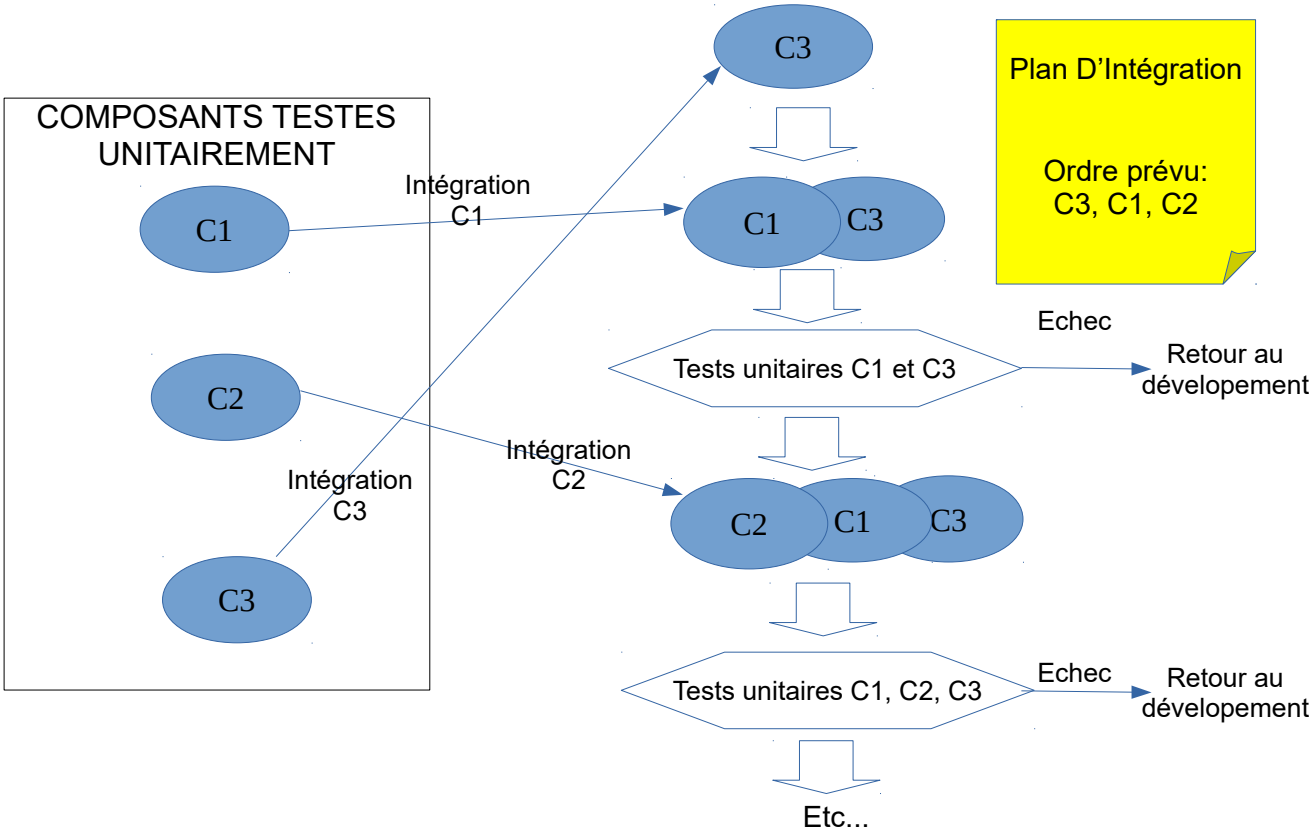
II.1.6.INTÉGRATION:

DESCRIPTION DES ACTIVITÉS:

Les différents composants développés au cours de la conception détaillés sont «agrégés» dans l'ordre prévu par le PLAN D'INTÉGRATION (PI). A la fin du processus, l'application est entièrement construite.

DÉROULEMENT DE CES ACTIVITÉS:

Chacun des composants ayant satisfait aux tests unitaires est intégré dans l'agrégat des composants déjà intégrés. Après intégration de chacun des composant, les tests unitaires de tous les composants déjà intégrés doivent être appliqués sur l'agrégat pour détecter d'éventuels EFFETS DE BORD ou des défauts de conception. L'impossibilité d'intégrer un composant peut amener une reprise de sa conception, ou même de la conception préliminaire:



REMARQUES:

- L'intégration peut commencer avant la fin de la conception détaillée de tous les composants: il suffit qu'à l'instant où l'on se situe dans l'intégration, le composant prévu par le plan d'intégration pour être intégré soit prêt;
- Si un des agrégats obtenus successivement au cours de l'intégration est FONCTIONNEL, rien n'empêche de le livrer en avance. Le plan d'intégration peut être élaboré en tenant compte d'un impératif de ce type (ce n'est pas forcément possible quand la conception préliminaire est orientée «données»).

II.1.7.RÉCEPTION-QUALIFICATION:

DÉFINITION DES ACTIVITÉS DE RÉCEPTION:

La RÉCEPTION d'un logiciel (on dit également RECETTE) est l'action par laquelle, en fin de développement, le CLIENT ACCEPTE ou REFUSE le produit développé.

Pour motiver son refus, le client doit pouvoir prouver que certaines de ses exigences (listée dans la S.T.B) ne sont pas respectées par le produit et que ce non respect entraîne pour lui une impossibilité de l'exploiter (c'est une RÉSERVE MAJEURE. Sinon, on parle de RÉSERVE MINEURE). La RÉCEPTION est donc la VALIDATION du logiciel par rapport aux exigence du client (SPÉCIFICATIONS de besoins fonctionnels et techniques)

DÉROULEMENT DES ACTIVITÉS DE RÉCEPTION:

Pour étayer sa décision, le CLIENT doit tester systématiquement chacune de ces exigences. Il s'agit là de TESTS D'EXIGENCES et non de TESTS UNITAIRES. Nous avons vu que divers documents ayant trait à la VALIDATION ont été élaborés sous la responsabilité du client au cours du projet:

- PLAN DE RÉCEPTION
- CAHIER DE RÉCEPTION GÉNÉRAL
- CAHIERS DE RÉCEPTION DÉTAILLES

Ces documents, approuvés par le fournisseur, servent de guide pour les opérations de tests.

Lorsqu'une ou des RÉSERVES MAJEURES sont exprimées par le client, le fournisseur doit s'efforcer de les lever en corrigeant le logiciel, puis en reprenant le processus de réception. Si le fournisseur n'arrive pas à lever ces réserves majeure dans les délais impartis pour la validation, le client peut rompre le contrat (avec indemnités) ou accepter d'abandonner ses réserves moyennant une diminution du prix.

Les RÉSERVES MINEURES (non bloquantes) exprimées par le client sont également relevées.

La VALIDATION ne se déroule généralement pas dans l'environnement opérationnel du produit. Un environnement «représentatif» est convenu dans le PLAN DE RÉCEPTION.

DESCRIPTION DES ACTIVITÉS DE QUALIFICATION:

L'activité de QUALIFICATION consiste à tester le produit validé dans son environnement opérationnel. Le produit doit avoir été VALIDÉ (réceptionné) par le CLIENT, ce qui implique qu'il ne reste plus de réserves majeures à lever.

DÉROULEMENT DES ACTIVITÉS DE QUALIFICATION:

En général, la qualification d'un produit de complexité moyenne est prévue pour durer un à deux mois. Au cours de cette période:

- Le CLIENT s'engage à mettre en œuvre le plus possible toutes les fonctionnalités du produit, dans toutes les conditions prévues par la STB.
- Le FOURNISSEUR s'engage à lever les RÉSERVES MINEURES qui subsistent.

Si, au cours de la période de qualification, des RÉSERVES MAJEURES nouvelles sont levées par le client et non résolues avant la fin de la qualification, le client peut, comme pour la validation,

rompre le contrat (avec indemnités) ou accepter d'abandonner ses réserves moyennant une diminution du prix.

Sans réserve majeure, la fin de la période de qualification marque la fin du contrat et le prix prévu doit être payé par le client.

II.1.8.REMARQUE:

Ces 7 activités se retrouvent dans toutes les méthodes de conduite de projets. En revanche, la manière d'enchaîner ces activités diffère. De ce point de vue, on peut classer ces méthodes en deux grandes catégories:

- Les méthodes «classiques», généralement basées sur le concept de «CYCLE EN V». Dans ce cas, ces différentes activités se succèdent sous la forme de 7 PHASES temporelles, en principe sans possibilité de retour en arrière;
- Les méthodes dites AGILES, basées sur des cycles itératifs dits «en spirale»: les 7 activités définies précédemment sont abordées au cours de chacun des CYCLES de la spirale de développement. Chaque itération permet la réalisation et l'intégration au logiciel d'une partie des fonctions prévues par les spécifications et permet ainsi de «converger» vers le produit final.

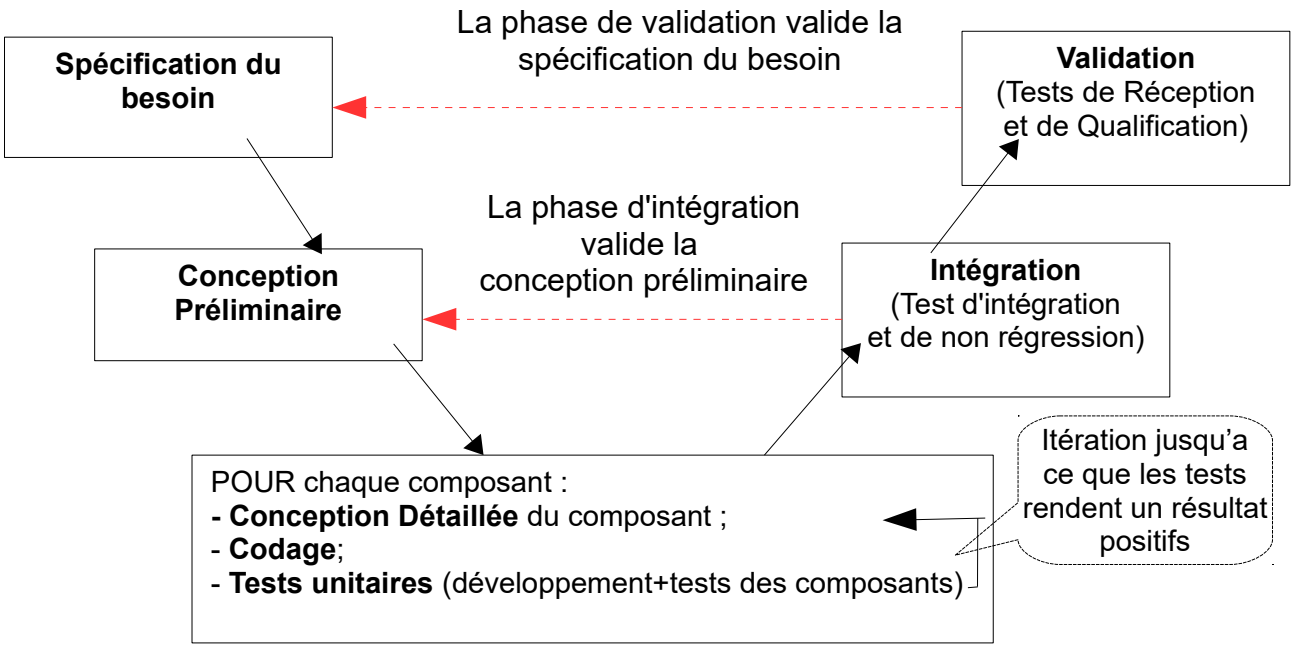
II.2.LE CYCLE EN V:

II.2.1.PRINCIPES GÉNÉRAUX:

PHASAGE DU CYCLE EN V:

Le cycle en V encapsule les 7 activités définies plus haut dans 7 CYCLES PHASES s'enchaînant en principe sans retour en arrière ni itération. Cependant, à l'intérieur de chaque phase, la validation des résultat peut être obtenue après plusieurs itérations du type:

création-tests-correction ou amélioration-tests- etc...



REMARQUE: le cycle en V est caractérisé par le fait qu'il se décompose en deux «mouvements» très distincts dans leurs principes:

- La partie gauche du schéma représente une démarche que l'on peut qualifier de «top-down» (ou d'analyse) : on part du besoin de l'utilisateur pour aboutir aux composants nécessaires à l'implémentation des exigences;
- La partie droite du schéma est, au contraire, une démarche de synthèse (down-top): on assemble (intègre) les composants afin de construire le produit final.

De plus, les phases de la partie droite assurent la VALIDATION des phases correspondantes de la partie gauche (c'est la signification des flèches rouges). Quand à la conception détaillée, ses produits sont validés immédiatement par les tests unitaires. On peut noter que la validation des spécifications intervient très tardivement: c'est un des défauts du cycle en V;

LE RÉFÉRENTIEL DOCUMENTAIRE:

Les méthodes appuyées sur le cycle en V sont en général très exigeantes en matière de documentation. Le tableau suivant liste les principaux documents exigés:

DOCUMENT	ÉLABORATION	RESPONSABLE	CONTENU
Spécifications Techniques de Besoin (STB))	En phase de spécification, après le recueil du besoin client.	CLIENT	**Présente la liste des EXIGENCES du client vis à vis du produit et décrit des interfaces externes.** • Les exigences peuvent être fonctionnelle, opérationnelle, techniques, qualitatives, etc. • Une exigence s'exprime simplement par une seule phrase. • Une exigence peut être caractérisée par différents critères (précision de calcul, mode de représentation, etc).
Plan de Réception (PR) et **Cahier de Réception Général (CRG)**	En phase de spécification, après finalisation de la STB.	CLIENT	• Le PR définit le périmètre de la réception (ce qu'on va tester) et établit une planification des actions de réception. • Le CRG définit comment on va tester chacune des exigences exprimées dans la STB (procédure générale de test et résultats attendus).
Document de Conception Préliminaire (DCP)	En phase de conception préliminaire.	FOURNISSEUR	• Définit l'architecture globale statique et dynamique du système: composants et interactions entre composants. • Définit pour chaque composant ses interfaces externes et les traitements qu'il encapsule (ses SPÉCIFICATIONS TECHNIQUES).
Plan de Développement et organigramme des tâches (PDv).	En phase de Conception Préliminaire (dépend du DCP)	FOURNISSEUR	• Définit et caractérise les différentes tâches à accomplir, les relations de précédences qui existent entre elle et les ressources matérielles et humaines nécessaires à leur accomplissement. • Permet de construire le PERT du projet.
Plan d'Intégration (PI)	En phase de conception Préliminaire.	FOURNISSEUR	Décrit la procédure d'intégration (ordre d'intégration des composants) et les procédures de tests à utiliser.
Documents de conception des composants (DCC)	En phase de conception détaillée.	FOURNISSEUR	Ces documents décrivent • L'architecture interne de chaque composant; • La description des tests unitaires applicables; • L'algorithmique interne (codes sources) en cas de projet informatique.
Cahiers de Réception Détaillés (CRD)	Avant la phase de réception	FOURNISSEUR	Décrivent dans le détail les tests de réception à employer.
Compte rendu de réception	Pendant la phase de réception	CLIENT +FOURNISSEUR	• Collationne les résultats des tests de réception (réserves majeures et mineures) • Indique si le client accepte ou refuse la réception.

Documents de conception des composants (DCC)	En phase de conception détaillée.	FOURNISSEUR	Ces documents décrivent • L'architecture interne de chaque composant; • La description des tests unitaires applicables; • L'algorithmique interne (codes sources) en cas de projet informatique.
Compte rendu de qualification	Pendant la phase de qualification	CLIENT	• Collationne les résultats de la période de qualification (réserves majeures et mineures, durée d'indisponibilité, difficultés d'exploitation); • Indique si le client accepte ou refuse la qualification.

II.2.2.CRITIQUE DU CYCLE EN V:

AVANTAGES:

- Les méthodes basées sur le cycle en V sont souvent qualifiées de PRÉDICTIVES. En effet, une fois la phase de spécification du besoin bouclée, la planification du projet est assez facile à évaluer, puisque ces spécifications deviennent contractuelles et le client ne peut plus les modifier. D'autre part, le référentiel documentaire, très fourni, assure une excellente TRAÇABILITÉ du déroulement du projet;
- Pour les mêmes raisons, le développeur peut se faire une idée très précise du produit qu'il doit réaliser et des moyens humains et matériel qu'il doit mettre en œuvre pour y arriver (planification des ressources);
- Le cycle en V n'exige que peu d'investissement de la part du CLIENT: il ne participe vraiment qu'à l'élaboration des SPÉCIFICATIONS GÉNÉRALES, des CAHIERS DE RÉCEPTION (recueil des tests de réception du produit), à la RÉCEPTION elle-même (validation du produit) et à la QUALIFICATION OPÉRATIONNELLE;
- Le fait d'effectuer une phase de conception préliminaire PRÉALABLEMENT À LA RÉALISATION DE TOUT COMPOSANT permet d'une part d'élaborer une architecture du produit très adaptée à l'ensemble des contraintes et d'autre part de bien repérer les CAS DE RÉUTILISATION, ce qui peut considérablement minimiser les COÛTS et les DÉLAIS.

INCONVÉNIENTS:

Cependant, ces avantages ont des revers:

- Les procédures sont très lourdes à gérer pour de petites équipes. Pour des réalisations de faible volume, l'effort consommé pour appliquer la méthode apparaît souvent comme disproportionné par rapport à l'effort consacré directement au développement du produit;
- Le client n'intervient qu'en phase de spécification et en phase de validation. Il perd donc contact avec l'équipe de réalisation pendant les trois autres phases, ce qui diminue son implication dans le projet;
- Pour la même raison, si le recueil du besoin a été incomplètement réalisé ou mal analysé, les défauts n'apparaîtront qu'en fin de projet (phase de validation), bien trop tard pour espérer les corriger sans trop de dégâts (dépassement des délais ou des coûts, abandon d'exigences, etc.).
- Si le client s'aperçoit en cours de projet que certaines des spécifications sont devenues inutiles, ou au contraire qu'il aurait besoin d'en rajouter d'autres, il est difficile de modifier

les spécifications sans recourir à un AVENANT AU CONTRAT: cette procédure est en général très lourde car elle a des implication COMMERCIALES et JURIDIQUES qui échappe largement aux équipes de développement proprement dites;

- La livraison du produit est prévue pour être effectuée en un seul bloc: il est très difficile de livrer en avance certains composants produits car ces composants n'encapsulent que très rarement des fonctions complètes: la livraison «en avance» de certains composant n'a donc en général aucun intérêt pour le client.

Ce sont ces défauts que les méthodes agiles sont sensé atténuer.

II.3.ATTÉNUATION DES INCONVÉNIENTS DU CYCLE EN V:

II.3.1.CYCLE EN V AVEC DÉVELOPPEMENT INCRÉMENTAL:

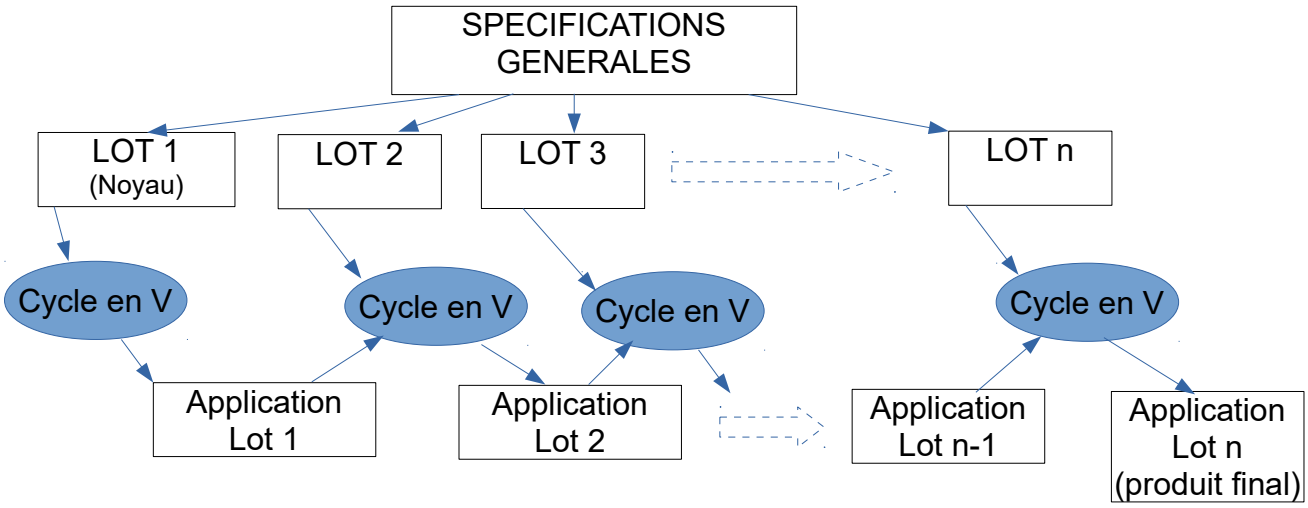
Pour atténuer les inconvénients du cycle en V, il est possible d'utiliser une variante de celui-ci, souvent appelée appelée CYCLE EN V INCRÉMENTAL.

Dans cette démarche, après l'élaboration de SPÉCIFICATIONS classiques, on sépare (en accord avec le client), les exigences fonctionnelles en plusieurs LOTS.

Le premier lot correspond au NOYAU de l'application. Celui-ci implémente les mécanismes généraux de l'application et les fonctions qui paraissent essentielles aux yeux du client. Ce lot est donc FONCTIONNEL. C'est ce lot que l'on réalise et qu'on livre en premier suivant la démarche du cycle en V.

Puis on relance un cycle en V pour réaliser le deuxième lot de spécifications, et ainsi de suite jusqu'à ce que le produit implémente toutes les exigences listées par les SPÉCIFICATIONS.

Il est possible de rendre les lots de rang supérieur à 1 CONDITIONNELS. Dans ce cas, après la livraison de chaque lot, le client se réserve le droit, de sa seule initiative, de terminer le processus de développement.



Cette démarche de réalisation pallie un certain nombre de défauts du cycle en V. En effet, du fait de la livraison en plusieurs lots opérationnels, elle IMPLIQUE BEAUCOUP PLUS LE CLIENT dans le processus de développement et PERMET LA LIVRAISON EN AVANCE de certaines fonctions. Nous verrons qu'elle se rapproche sensiblement des méthodes agiles.

D'autre part, cette démarche de réalisation est compatible avec le type de marché public appelé MARCHÉ A TRANCHES CONDITIONNELLES.

II.3.2.IMPLICATION DU CLIENT DANS LES PHASES DE DÉVELOPPEMENT:

Beaucoup de clients, avec l'accord du fournisseur, mettent en place une SURVEILLANCE INDUSTRIELLE pendant la partie MAÎTRISE D'ŒUVRE du projet (conception, développement, intégration). Cette surveillance peut se traduire par:

- Leur participation à certaines réunions de PLANNING, à titre consultatif;

- La communication aux clients de documents intermédiaires dès leur validation (ce qui leur permet de surveiller leur conformité avec le PLAN D'ASSURANCE QUALITÉ);
- La présentation au client de certains composants correspondant à des exigences fonctionnelles (un IHM, par exemple), afin de recueillir ses impressions et de détecter des dérives dues à de mauvaises spécification).

III.LES MÉTHODES AGILES:

III.1.HISTORIQUE:

Les premières tentatives de gestion de projets basées sur le modèle ITÉRATIF (ou INCRÉMENTAL), principe de base des méthodes agiles, datent du milieu des années 1980. L'expression MÉTHODES AGILES a été employé pour la première fois en 2001, date à laquelle des spécialistes du développement logiciel ont entrepris d'unifier leurs méthodes respectives. C'est à cette occasion que le MANIFESTE AGILE, a été élaboré.

Les méthodes agiles ne sont donc pas des techniques nouvelles: elles ont été largement confrontées aux réalités du développement de logiciel.

III.2.LES VALEURS DU «MANIFESTE AGILE»:

Les partisans des méthodes agiles ont élaboré un «MANIFESTE AGILE» qui met en avant les quatre valeurs fondamentales suivantes:

- Les individus et leurs interactions sont plus déterminants pour la gestion d'un projet que les processus et les outils utilisés;
- Il est plus important que les logiciels créés soient opérationnels que d'élaborer une documentation exhaustive;
- Une collaboration étroite avec les clients est plus efficace qu'une négociation contractuelle;
- Il est plus important de savoir s'adapter au changement que de suivre rigoureusement un plan préétabli.

De ce fait, contrairement au cycle en V où la création du logiciel se fait par une succession de phases sans retour en arrière (du moins en principe...), les méthodes agiles reposent sur une remise en cause systématique des résultats de chaque étape, à la fois par les clients et les développeurs, avec de fréquentes itérations. On peut dégager deux principes essentiels:

- Le DÉVELOPPEMENT ITÉRATIF;
- La LIVRAISON AU PLUS TÔT.

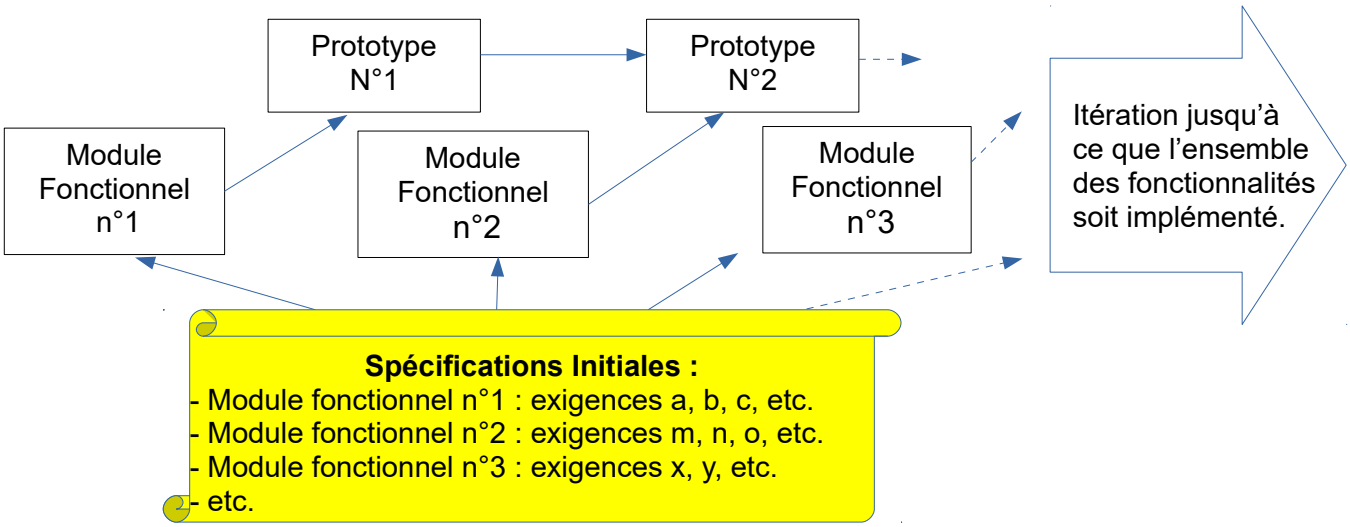
III.3.PREMIER PRINCIPE: DÉVELOPPEMENT ITÉRATIF:

III.3.1.PRÉSENTATION:

DESCRIPTION:

1. A partir des exigences (fonctionnelles, opérationnelles, etc.) exprimées par le client, des SPÉCIFICATIONS sont élaborées en concertation avec celui-ci. Contrairement au cycle en V, ces exigences n'ont pas un caractère contractuel;
2. A partir de ces spécifications, le développement se déroule sous la forme d'un certain nombre d'ITÉRATIONS. Chacune de ces itérations concerne le développement complet d'un sous-ensemble des exigences prévues par le document de spécifications initial. Nous appellerons cet ensemble d'exigences un MODULE FONCTIONNEL;
3. Un premier MODULE FONCTIONNEL fait donc l'objet d'un développement complet qui aboutit à la création d'un premier PROTOTYPE. A l'issue de cette première itération, les logiciels développées doivent être OPÉRATIONNELS (voir plus loin la définition).
4. Ce prototype fait l'objet d'une validation par le CLIENT. Si le client valide le prototype, celui-ci peut lui être livré;
5. On choisit alors un deuxième MODULE FONCTIONNEL comme objectif de la deuxième itération. Les logiciels correspondant à ce module fonctionnel sont développés;
6. Le premier prototype est alors complété et modifié pour intégrer les fonctions spécifiées par le deuxième MODULE FONCTIONNEL. On aboutit ainsi à un deuxième prototype;
7. Ce deuxième prototype fait également l'objet d'une évaluation par le client. Cette évaluation peut conduire celui-ci à modifier ou compléter les nouvelles spécifications;
8. On choisit alors un troisième MODULE FONCTIONNEL qui conduira à un troisième prototype. Si le client valide ce prototype, celui-ci peut lui être livré;
9. Et ainsi de suite, l'application se construit par itérations, chacune de ces itérations incrémentant dans le prototype de nouvelles fonctions.

Le schéma ci-après résume le principe des itérations:



REMARQUES:

- Les contenus et la priorité des modules fonctionnels à réaliser à chaque itération sont

décidés par le représentant du CLIENT;
- Entre deux itérations le CLIENT peut modifier les SPÉCIFICATIONS du produit en accord avec le RÉALISATEUR. En revanche, pendant une itération, le sous-ensemble de fonctions qui constitue l'objectif de cette itération ne peut être modifié: il est «sanctuarisé»;
- Les itérations s'arrêtent dans deux cas:
  - L'ensemble des exigences fonctionnelles a été implémenté;
  - A un moment donné, le client se déclare satisfait du prototype tel qu'il est et abandonne ce qui reste à faire.

III.3.2.AVANTAGES:

Cette procédure a l'avantage de maintenir le client au plus près de la réalisation et de lui permettre d'affiner progressivement ses exigences en collaboration avec le développeur. Elle minimise donc les risques d'incompréhension ou d'imprécision dans l'expression des exigences.

III.3.3.INCONVÉNIENTS:

Les inconvénients sont de deux ordres:
- La procédure exige beaucoup d'engagement du client tout au long du projet en termes de mise à disposition de ses personnels;
- La procédure permet difficilement de prévoir la durée du projet: la plus ou moins grande disponibilité des personnels du client, mais aussi leur connaissance plus ou moins grande de l'environnement d'utilisation du produit interviendront beaucoup dans cette durée.

III.4.DEUXIÈME PRINCIPE: LIVRER AU PLUS TÔT CE QUI PEUT L'ÊTRE:

III.4.1.PRÉSENTATION:

Contrairement au cycle en V classique où l'ensemble du logiciel est dans la plupart des cas livré en une seule fois, les méthodes agiles conseillent de livrer en plusieurs lots, chaque lot étant FONCTIONNEL, c'est à dire implémentant complètement une ou plusieurs des FONCTIONS du produit.

Exemple: *Supposons que le produit final soit un site marchand supportant les fonctions suivantes:*

1. *Présentation du catalogue des produits;*
2. *Gestion du panier d'achat;*
3. *Gestion des comptes clients;*
4. *Gestion facturation et expédition;*
5. *Etc.*

Il peut être prévu de livrer un premier lot supportant la fonction n°1 dès que le logiciel correspondant sera entièrement réalisé et validé, alors que les autres lots seront toujours en développement.

III.4.2.AVANTAGES:

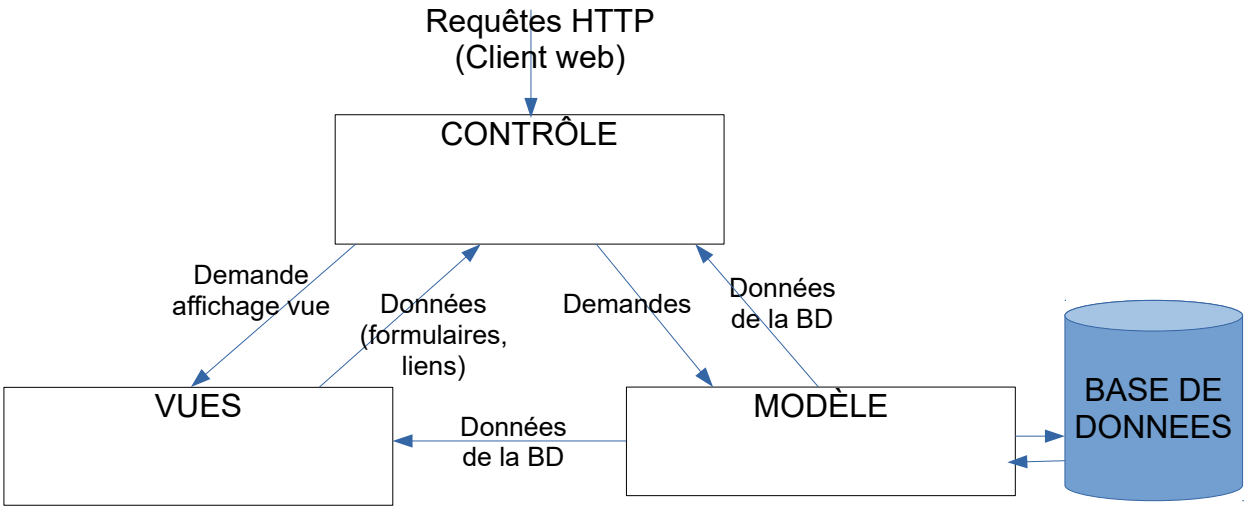
Ces livraisons «dès que possible» permettent d'entretenir la relation client-fournisseur, de repérer le plus tôt possible les difficultés éventuelles et de former «en avance» les équipes d'utilisateurs.

III.4.3.INCONVÉNIENTS:

Développer séparément les différentes fonctions d'une application sans avoir au préalable étudié l'architecture générale du système pose fatalement de nombreux problèmes, dont les plus courants sont:

- Les incompatibilités et effets de bord entre modules fonctionnels (conflits de ressources);
- Les redondance: certains mécanismes communs risquent d'être développés plusieurs fois ou bien refondus et complétés à chaque introduction d'une nouvelle fonctions;
- Une architecture générale peu efficiente car résultant d'adaptations successives.

EXEMPLE: Prenons le cas du site marchand évoqué plus haut. Le schéma ci-dessous représente une architecture de principe, basée sur le DESIGN PATTERN MVC:



Si l'intention du client est de disposer au plus vite de la fonction «**présentation du catalogue des produits**» il nous faudra pour cela disposer d'un certain nombre de composants comprenant au minimum:

- La VUE qui permet d'afficher la liste des articles disponibles du catalogue ainsi que celle qui permet d'afficher le détail d'un article sélectionné;
- Les fonctions (ou méthodes) du MODÈLE qui permettent de récupérer les informations d'un article dans la TABLE DES ARTICLES de la base de données;
- Les méthodes du CONTRÔLEUR qui permettent d'activer les différentes fonctions ou méthodes du MODÈLE et d'afficher les VUES.

Pour «livrer au plus vite», ces composants seront élaborés en tenant compte uniquement des contraintes liées à la fonction «présentation du catalogue des produits». **C'est en général ce que recommandent les principales méthodes agiles**. Les conséquences prévisibles sont les suivantes:

- L'étude de l'architecture générale statique et dynamique qui supportera les différents traitements liés à l'exécution de cette fonction sera réduite à la résolution des contrainte induites par celle-ci. Il faudra donc s'attendre à devoir adapter cette architecture aux prochaines itérations du processus de développement, avec les effets de bord et les régressions qui peuvent en résulter;
- Il est probable que certains composants développés dans le cadre de l'implémentation de cette fonction pourraient être réutilisés pour l'implémentation d'autres fonctions de l'application. Malheureusement, ces composants, n'ayant été développés qu'en tenant compte des contraintes imposées par la fonction «présentation du catalogue des produits», ils ne pourront probablement pas être réutilisés **tels quels** pour d'autres fonctions. Il faudra donc
  - soit **les modifier pour les rendre réutilisables** (on appelle cette pratique la REFACTORISATION: elle implique des retours en développement fréquents),
  - soit créer des **doublons**, ce qui augmente inutilement le volume de réalisation. Dans les deux cas, il y a augmentation des coûts et délais.

III.5.MÉTHODES AGILES- DÉROULEMENT GÉNÉRAL:

III.5.1.INITIALISATION DU PROJET ET PREMIÈRE ITÉRATION:

Les développements de logiciels menés suivant les principes de l'agilité débutent en général par une première itération qui concerne:

- L'étude du besoin initial aboutissant (souvent, mais pas toujours) à un document de SPÉCIFICATIONS non contractuel ;
- L'élaboration d'un PLAN DE RÉCEPTION indicatif et de CAHIERS DE RÉCEPTION;
- Une PLANIFICATION INITIALE. Cette planification est basée sur l'évaluation des charges de travail respectives liées au développement des fonctions à réaliser. La procédure est détaillée plus loin.

REMARQUE: *des travaux de CONCEPTION PRÉLIMINAIRE destinés à élaborer des principes d'architecture globaux peuvent être effectués pendant cette première itération. Cependant, certains spécialistes considèrent qu'une telle démarche est inutile: pour eux, l'architecture globale doit émerger par retouches et adaptation successives des différentes itérations. Cette pratique est appelée REFACTORING (ou réusinage, remaniement, refactorisation).*

Les travaux liés à cette phases initiales constituent une première itération. Après celle-ci, le processus de construction incrémentale des fonctions se déroule comme décrit plus haut.

III.5.2.ITÉRATIONS SUIVANTES:

Chaque itération, qui en principe doit durer entre deux et quatre semaines, correspond en fait à un cycle complet de développement comprenant:

- Une phase de **spécifications** (choix des fonctions à réaliser par l'itération, planning et estimation du volume de réalisation de chaque tâche parcellaire);
- Une phase de **Conception préliminaire** de l'incrément;
- Une phase de **Conception détaillée** de chacun des composants qui composent cet incrément (conception, codage, tests unitaires);
- Une phase d'**Intégration** des composants au prototype en cours (avec tests d'intégration);
- Une phase de **Réception** du nouveau prototype (test des fonctions ajoutées par l'itération et tests de non régression, validation par le client).

A la fin du développement, une RÉCEPTION GLOBALE est effectuée avant de réaliser la QUALIFICATION opérationnelle du produit.

III.5.3.RECUEIL DES EXIGENCES DU CLIENT:

Le CLIENT exprime ses exigences (fonctionnelles, opérationnelles, méthodologiques, normatives, techniques, etc.) sous la forme de phrases simples, éventuellement accompagnées de critères de réalisation. Par exemple:

Exigence:

L'Administrateur du système doit pouvoir visualiser le nombre de visiteurs du site.

Critères:

- Nombre de visiteurs connectées;
- Nombre de premières connexion par heures, par journées et par semaines;
- Sous forme numérique et graphique.

Éventuellement, ces exigences sont analysées avec le concours du réalisateur. Elles peuvent être regroupées ou scindées suivant les besoins.

Dans certaines démarches agiles, cette activité n'aboutit pas à la rédaction d'un document formel de SPÉCIFICATIONS, mais à de simples FICHES.

III.5.4.CONSTITUTION DES CAHIERS DE RÉCEPTION:

Pour chaque exigence du client une PROCÉDURE DE TESTS DE RÉCEPTION est élaborée. Autant que possible, ces procédures doivent être automatisées sous la forme de logiciels de test, mais ce n'est pas toujours possible. La procédure peut alors s'appuyer sur:

- Pour les fonctions fournissant des résultats numériques ou alphanumériques: un TABLEAU fournissant pour chaque jeu de données d'entrée les données de sorties correspondantes (tableur excell ou autre).
- Pour les autres fonctions (IHM, sorties graphiques, sonores ou autres): des scripts peuvent parfois être utilisés (par exemple, pour simuler des interactions de l'utilisateur avec un interface graphique).

III.5.5.NÉGOCIATION DES EXIGENCES:

Avant chaque itération, client et réalisateurs peuvent NÉGOCIER les exigences contenues dans le recueil initial des spécifications (STB). Ceci concerne:

- L'abandon d'exigences par le client;
- La modification de la caractérisation de certaines exigences;
- L'introduction d'exigences supplémentaires, soit avec augmentation du prix, soit en compensant par l'abandon d'autre exigences (TROC des exigences).

III.5.6.DÉMARCHE DE PLANIFICATION:

POSITION DU PROBLÈME:

Dans le contexte des méthodes classiques, la planification initiale du projet est effectuée à la suite de l'activité de conception préliminaire. C'est en effet cette activité qui permet de déterminer l'ensemble des composants logiciels à réaliser et d'estimer l'effort de développement exigé par chacun d'eux. La réalisation d'un composant constitue une TÂCHE unitaire dont on peut estimer la durée en fonction de l'effort exigé et des ressources pouvant lui être consacrées (exemple: si l'effort est estimée à 6 «développeurs.jours», la durée sera de 3 jours pour un binôme).

D'autre part, dans un projet, quelle que soit la méthode utilisée, certaines tâches ne correspondent pas à la réalisation de composants: il en est ainsi des commandes de fournitures ou des formations nécessaires. Ces tâches doivent être prises en compte car elles impactent le planning.

Dans le contexte des méthodes agiles, nous ne disposons pas, en général de résultats de conception préliminaire initiale (celle-ci n'est pas faite ou bien elle est forcément incomplète). En revanche, la liste des exigences exprimées par le client à ce stade du projet est connue, même si elle peut évoluer. La planification se fait alors à partir de cette liste:

ÉTABLISSEMENT DE PLAN DE DÉVELOPPEMENT:

Le plan est donc établi à partir de la liste des exigences du client et avec sa collaboration. La procédure est la suivante:

- Pour chaque exigence, un COÛT D'IMPLÉMENTATION est évalué en POINTS par les réalisateurs. Le nombre de point attribué à chacune des exigences ne correspond pas à une quantité: il tente simplement de représenter les coûts d'implémentation RELATIFS des exigences (exemple: l'exigence la plus facile à implémentée sera notée 3 point, la plus difficile sera notée 15 points, les autres exigences seront notées en fonction de ces deux premières).
- Éventuellement, chaque exigence est affectée par le client d'une PRIORITÉ qui caractérise l'importance qu'elle revêt pour celui-ci;
- Ces informations seront collationnées sous la forme d'un tableau dont chaque ligne correspondra à une exigence. Ce tableau sera alors retravaillé par client et réalisateurs pour:
  - Éventuellement regrouper les exigences trop «légères» ou scinder les exigences trop «lourdes»;
  - Puis reclasser les exigences par ordre de priorité décroissante.

CALCUL DE LA VÉLOCITÉ DE L'ÉQUIPE DE RÉALISATION:

Une fois ces travaux effectués le maître d'œuvre essaiera d'évaluer la durée d'implémentation d'une exigence relativement lourde, mais dont on maîtrise bien l'implémentation. A partir de cette durée D et de la note T attribuée à cette exigence, on pourra définir une «durée du point» (D/T), qui caractérisera la «vélocité» V = D/T de l'équipe de réalisation. Cette valeur permettra d'évaluer la DURÉE réelle d'implémentation de chaque exigence par la formule:

Durée = <Nombre de points attribuée à l'exigence>*V

La vélocité sera réévaluée à la fin de chaque itération en fonction du nombre de points traités pendant cette itération et de la durée de celle-ci.

UTILISATION DU PLAN DE DÉVELOPPEMENT:

Le plan de développement sera utilisé par le client tout au long de la réalisation pour choisir les exigences qui seront implémentées à chaque itération. En général, les exigences les plus prioritaires seront choisies en premier.

EXEMPLE DE PLAN DE DÉVELOPPEMENT:

EXIGENCE	COÛT D'IMPLÉMENTATION	PRIORITÉ
Les internautes doivent pouvoir visualiser la liste des articles à la vente.	25	1
Les internautes doivent pouvoir acheter un article de la liste visualisable	15	1
Les administrateurs doivent pouvoir ajouter ou retirer un article de la liste	20	2
Les internautes doivent pouvoir ouvrir et gérer un compte client	15	3
Les clients titulaires d'un compte doivent recevoir un newsletter.	12	3

III.6.DIFFÉRENTES MÉTHODES AGILES:

En fait, la différence essentielle entre méthodes traditionnelles (cycle en V) et méthodes agiles ne tient pas tellement dans les activités pratiquées ou les outils utilisés, qui restent sensiblement les mêmes, mais plutôt dans un CHANGEMENT DE PARADIGME, qui lui, est assez radical:

- Les méthodes traditionnelles cherchent à éliminer les risques liés à la VARIABILITÉ DES BESOINS DU CLIENT dans le temps et à la difficulté d'anticiper l'ÉVOLUTION DE CES BESOINS en portant l'effort sur les ÉTUDES PRÉLIMINAIRES (besoins, architecture). Ce parti pris permet d'amasser le maximum de connaissances sur le produit à livrer avant le début de sa réalisation effective. Le but est de rendre la démarche plus «PRÉDICTIVE» et d'éviter les retours en arrière;
- Les méthodes agiles, au contraire, se basent sur l'affirmation que, compte tenu de ces facteurs de variabilité, il est plus efficace de partir d'une connaissance incomplète du produit à livrer sans perdre de temps dans des travaux préliminaires et de compléter cette connaissance au fur et à mesure de la réalisation de ce produit, quitte à REFACTORISER périodiquement ce produit au cours de sa réalisation. La «PRÉDICTIVITÉ» du projet est sacrifiée au profit d'une plus grande «MALLÉABILITÉ» du produit.

De ce fait, les différentes méthodes agiles pratiquées actuellement (2018) ont des caractéristiques assez semblables, qui découlent de cette différence de paradigme:

- Les SPÉCIFICATIONS ne sont pas contractuelles: les exigences du client peuvent être renégociées tout au long du processus de développement (TROC d'exigences);
- Le développement du produit est réalisé en plusieurs ITÉRATIONS, chacune pouvant correspondre à la livraison d'un sous-ensemble FONCTIONNEL;
- La phase de CONCEPTION PRÉLIMINAIRE (conception de l'architecture globale) est abandonnée au profit d'une REFACTORISATION du logiciel à l'occasion de chaque itération;
- Les cycles de développement (itérations) sont très courts (deux à quatre semaines);
- Le CLIENT intervient tout au long de la réalisation, et en particulier **en début d'itération** (pour préciser les objectifs à atteindre) et **en fin d'itération** (pour accepter ou refuser le résultat de l'itération). La présence quasi permanente d'un représentant du client sur les lieux de réalisation est souvent OBLIGATOIRE;
- Les équipes de développement ont des effectifs faibles (moins de 10 personnes). Les développeurs travaillent en binômes (attention, les binômes ne sont pas figés, au contraire, ils changent très fréquemment). Il n'existe ni hiérarchie ni division en sous-groupes. La formule de l'OPEN SPACE est pratiquement incontournable;
- Le code est TESTÉ, AMÉLIORÉ et REFACTORISÉ tout au long du processus de développement. Les TESTS doivent être développés AVANT les logiciels à tester;
- Des indicateurs actualisés JOURNELLEMENT permettent de mesurer l'avancement du projet afin de mettre à jour le plan de développement.

Les méthodes les plus connues sont probablement XP (eXtreme Programming) et SCRUM. Nous pouvons citer également:

- RAD (Rapid Application Development - Développement rapide d'applications).
- DSDM (Dynamic Software Development Method)
- UP (Unified Process)
- RUP (Rational Unified Process).

IV.LA DÉMARCHE SCRUM:

IV.1.HISTORIQUE:

SCRUM a commencé à être appliquée à partir de 1993. C'est actuellement la méthode Agile la plus utilisée et la mieux documentée, avec Extrême Programming.
Le verbe anglais TO SCRUM peut se traduire par MÊLER. Ce vocable fait probablement référence à la manière de travailler préconisée par la méthode: tous les membres de l'équipe de développement sont sensés travailler aux mêmes tâches, dans la même pièce sans hiérarchie ni spécialisation. Les équipes de développement ne sont jamais fractionnée en sous-groupes affectés à des tâches différentes.

IV.2.PRINCIPES GÉNÉRAUX:

Plus qu'une méthode, SCRUM définit un schéma d'organisation adapté au développement de logiciels. SCRUM offre aux développeurs un cadre de travail (framework) pour la gestion de projet. Ce cadre définit des rôles et un mode de déroulement du projet, constitué par des phases de développement, des réunions, des fournitures et des produits à livrer.

IV.2.1.LES RÔLES:

Scrum définit ainsi 3 rôles principaux

RÔLE	DESCRIPTION
PRODUCT OWNER	Le Product Owner est le représentant du CLIENT. Issu des personnels du client, possédant une excellente connaissance de son métier et disposant de pouvoirs décisionnels sur le déroulement du projets, il veuille au respect des EXIGENCES du client. Ce rôle est analogue à celui du MAÎTRE D'OUVRAGE dans un processus classique.
SCRUM MASTER	Le Scrum Master veille à l'application de la méthode SCRUM. Son rôle est analogue à celui d'un RESPONSABLE QUALITÉ.
ÉQUIPES DE DÉVELOPPEMENT	• Une équipe est constituée de l'ensemble des personnels du réalisateur qui prennent en charge le développement d'un module fonctionnel (planification, conception, codage, tests, documentation). • La taille d'une équipe doit être réduite (moins de 10 personnes). • Il n'y a pas de spécialisation des rôles ni de hiérarchie: l'équipe se gère elle-même.

IV.2.2.LA PHASE DE SPÉCIFICATIONS INITIALES:

CONTENU:

Comme dans un projet classique, cette phase permet d'élaborer les SPÉCIFICATIONS initiales du produit, de concert avec le client. Cette phase conduit à l'élaboration d'un document listant et caractérisant les exigences initiales du client, appelé PRODUCT BACKLOG (Littéralement «Carnet de produit». Ce document est un simple document de travail à caractère NON CONTRACTUEL. Il peut être modifié tout au long de la réalisation avec accord des deux parties (client et réalisateur).

NOTION D'USER STORY:

Les méthodes agiles utilisent peu le vocable EXIGENCES (du client), probablement à cause du caractère impératif qui lui est attaché. Elles préfèrent pour cette raison utiliser l'expression USER STORY (littéralement histoire d'utilisateur) pour évoquer l'expression du besoin des utilisateurs.

Une USER STORY se matérialise par une phrase simple, utilisant un langage courant qui **évoque** (et non définit) une fonction de l'objet à développer. Elle contient généralement les trois informations suivante:

- Qui s'exprime? (cette fonction est utile à qui?);
- Que veut cette personne? (description de l'objet de la demande);
- Pourquoi le veut-elle? (Justification de la demande).

Le récit de l'utilisateur doit être précisé par des CRITÈRES d'ACCEPTATION qui permettront de CARACTÉRISER la demande et de concevoir des tests de VALIDATION.

EXEMPLE:
USER STORY:
En tant que simple internaute, je veux pouvoir visualiser la liste des articles proposés par le site en les sélectionnant suivant divers critères, afin de prendre connaissance de l'offre du site et de me décider pour un éventuel achat.

CRITÈRES *D'ACCEPTATION:*
Je veux pouvoir sélectionner les articles en fonction:

- De leur catégorie de produit;
- De leur marque;
- De leur prix.

Le PRODUCT BACKLOG rassemble toutes les USER STORY issues de l'activité de recueil et d'analyse du besoin.

REMARQUE: *Nous voyons aisément que la notion d'USER STORY est très proche de la notion d'EXIGENCE utilisée dans les méthodes traditionnelles, du moins si on excepte le caractère impératif lié à la notion d'exigence qui semble interdire toute négociation de leur contenu.*

IV.2.3.LES SPRINTS ET LES RÉUNIONS ASSOCIÉES:

Un SPRINT (ou ITÉRATION) est une période durant en principe entre deux et quatre semaines

pendant laquelle une équipe de développement doit réaliser une partie bien définie des EXIGENCES du client, exprimées dans le PRODUCT BACKLOG.

SCRUM définit également un ensemble de réunions liées à chaque sprint, dont les objet et les durées doivent être clairement définis et limités (c'est le principe du TIMEBOXING):

- La réunion de SPRINT PLANNING (planification du sprint) doit permettre d'élaborer la sélection d'exigences (issues du PRODUCT BACKLOG) que le sprint devra satisfaire. Cette sélection d'exigences est appelée SPRINT BACKLOG BACKLOG) est élaborée de concert avec le représentant du client (Product Owner). C'est durant cette réunion qu'est estimée la charge de développement correspondant à la réalisation du SPRINT BACKLOG;
- Le déroulement du sprint est contrôlé tous les jours par des réunions appelées DAYLY SCRUM (mêlées quotidiennes). Ces réunions permettent de synchroniser les travaux des différents agents de l'équipe de développement. Ces réunions, qui se passent DEBOUT et ne doivent pas durer plus de 15 minutes. Chaque participant fait le point sur l'avancement des tâches qui lui sont confiées et les difficultés qu'il rencontre. Le résultat de ces réunions permet au maître d'œuvre de mettre à jour les indicateurs d'avancement du projet;
- Les résultats d'un sprint sont évalués lors d'une réunions de REVUE DE SPRINT. Cette réunion, à laquelle le PRODUCT OWNER participe, a pour but de décider si l'objet du SPRINT a été atteint et de prendre les mesures adaptées;
- Enfin, les RÉTROSPECTIVE DE SPRINT permettent, après la Revue de Sprint, de faire le bilan du sprint et d'en tirer des enseignements et des voies d'amélioration.

RÉUNIONS LIÉES A UN SPRINT		
RÉUNION	**QUAND? COMMENT**	**OBJET**
SPRINT PLANNING (planification du sprint)	- Avant le SPRINT - En présence du Product Owner	doit permettre d'élaborer le SPRINT BACKLOG (sélection d'exigences ou d'user stories à satisfaire pendant le sprint).
DAYLY SCRUM (mêlée quotidienne)	- Pendant le sprint. - Tous les jours. - Debout et en moins de 15 mn.	- Permet à chaque participant de faire le point sur l'avancement des tâches qui lui sont confiées et les difficultés qu'il rencontre; - Permet au responsable de mettre à jour les indicateurs d'avancement du sprint.
SPRINT REVIEW (revue de sprint)	- Après le sprint. - En présence du Product Owner.	Vérifie que les objectifs du sprint ont été atteints et prend des mesures en conséquence.
RÉTROSPECTIVE DE SPRINT	Après la revue de sprint.	Fait le bilan du sprint, tire des enseignements et voies d'amélioration.

IV.3.DÉROULEMENT GÉNÉRAL D'UN PROJET SCRUM:

IV.3.1.TRAVAUX PRÉPARATOIRES:

RECUEIL ET ÉTUDE DES EXIGENCES:

Si l'on excepte la phase de constitution des équipes et de nomination des responsables, un projet SCRUM commence par un classique RECUEIL DES EXIGENCES du client, exprimées sous la forme d'USER STORIES.
Cette activité va donner naissance, sous la responsabilité du PRODUCT OWNER, à un premier document de spécifications appelé PRODUCT BACKLOG.
La différence avec un cycle en V est que ce recueil n'a pas vocation à être exhaustif ni contractuel, puisque son contenu pourra tout au long du projet être corrigé et complété par le client, en accord avec le réalisateur.

ESTIMATION DES TRAVAUX A FAIRE:

Dans le cadre d'un projet SCRUM, du fait du déroulement par itérations, il est pratiquement impossible d'élaborer un planning classique. Tout au plus peut-on tenter d'évaluer la charge de travail correspondant à la réalisation des différentes demandes du client et de repérer les dépendances temporelles entre les différentes tâches à accomplir.

SCRUM conseille d'évaluer les demandes exprimées dans le BACKLOG en considérant chacune des demandes et en les rapprochant de réalisations comparables sur les plans du contenu fonctionnel, du volume et de la complexité. La COMPLEXITÉ est matérialisé par un NOMBRE ENTIER arbitraire sensé être proportionnel à cette complexité.

EXEMPLE:

Éléments du BACKLOG (demandes des utilisateurs-user stories)	Estimation de la complexité
En tant que simple internaute, je veux pouvoir visualiser la liste des articles proposés par le site en les sélectionnant suivant divers critères, afin de prendre connaissance de l'offre du site et de me décider pour un éventuel achat.	4
En tant qu'administrateur du site, je veux pouvoir créer, modifier ou supprimer un article, afin de pouvoir mettre à jour la liste des articles.	8
En tant qu'utilisateur possédant un compte client, je veux pouvoir récupérer par e-mail mes identifiants de connexion, afin de pallier la perte de ceux-ci.	3

Le Product Owner ordonnance ensuite la liste du BACKLOG en fonction de différents critères:

- La valeur du service rendu à l'utilisateur;
- Les dépendances existantes entre les diverses exigences.
- Les contraintes des équipes de développement.
- Le coût estimé;
- Les risques identifiés.

PLANIFICATION GÉNÉRALE DES SPRINTS:

- Cet ordonnancement va aider le PRODUCT OWNER à établir la liste des SPRINTS (itérations) qui permettront de réaliser l'application, en tenant compte de l'objectif principal de SCRUM: produire le plus tôt possible la plus grande valeur possible, afin de créer des opportunités d'accélération du «Time to market» (dans ce sens, time to market désigne simplement la durée qui reste avant de pouvoir proposer le produit sur le marché).
- Dans le même esprit, le PRODUCT OWNER va également procéder au regroupement de sprints qui aboutissent à un ensemble de fonctionnalités pouvant faire l'objet d'une livraison (RELEASE).

IV.3.2.ENCHAÎNEMENT DES SPRINTS:

LE SPRINT PLANNING (Durée < 4 heures):

En préalable à chaque sprint se tient une réunion de SPRINT PLANNING (planification du sprint). C'est au cours de cette réunion que sera déterminé le contenu du SPRINT BACKLOG.

- Le PRODUCT OWNER peut affiner ou compléter son besoin, ce qui permet à l'équipe de développement d'estimer plus précisément la charge de travail du sprint;
- L'équipe de développement détermine alors avec le Product Owner les USER STORIES du Product Backlog qui seront prises en compte dans le sprint;
- L'équipe de développement analyse alors les user stories prises en compte dans le SPRINT et les traduit en un certain nombre de TÂCHES à accomplir. Puis, elle estime le temps nécessaire à la réalisation de chacune de ces tâches.
- L'ensemble de ces travaux est consigné dans le SPRINT BACKLOG. Un TABLEAU DES TÂCHES est également produit (Tache, Contenu, Estimation de durée, Pourcentage de réalisation, Durée restante). Ce tableau permettra de visualiser l'AVANCEMENT du SPRINT.

DÉROULEMENT DU SPRINT:

- Un SPRINT dure normalement entre deux et quatre semaines;
- Pendant le sprint, le SPRINT BACKLOG est «sanctuarisé» (son contenu ne peut être modifié);
- Un sprint comporte toutes les étapes classiques de la conception d'un logiciel: Conception, codage et tests unitaires, intégration.
- Des réunions quotidiennes (DAYLY SCRUMS) sont tenues chaque jour pour faire le point sur l'avancement des travaux (réunions DEBOUT, pas plus de 15 minutes). A l'occasion de ces réunions, le tableau d'avancement des tâches est actualisé.

LA SPRINT REVIEW (Durée < 2 heures):

A la fin de chaque sprint se tient une réunion de revue du sprint (SPRINT REVIEW), en présence du PRODUCT OWNER. Cette réunion a pour objectif de vérifie que les objectifs du sprint ont été atteints et de prendre des mesures en conséquence.
Si le dernier sprint complète un ensemble livrable (RELEASE), cet ensemble est l'objet d'une VALIDATION avant livraison.

LA RÉTROSPECTIVE DE SPRINT:

Tenue après la SPRINT REVIEW, cette réunion a pour objectif de faire un bilan du sprint et d'en

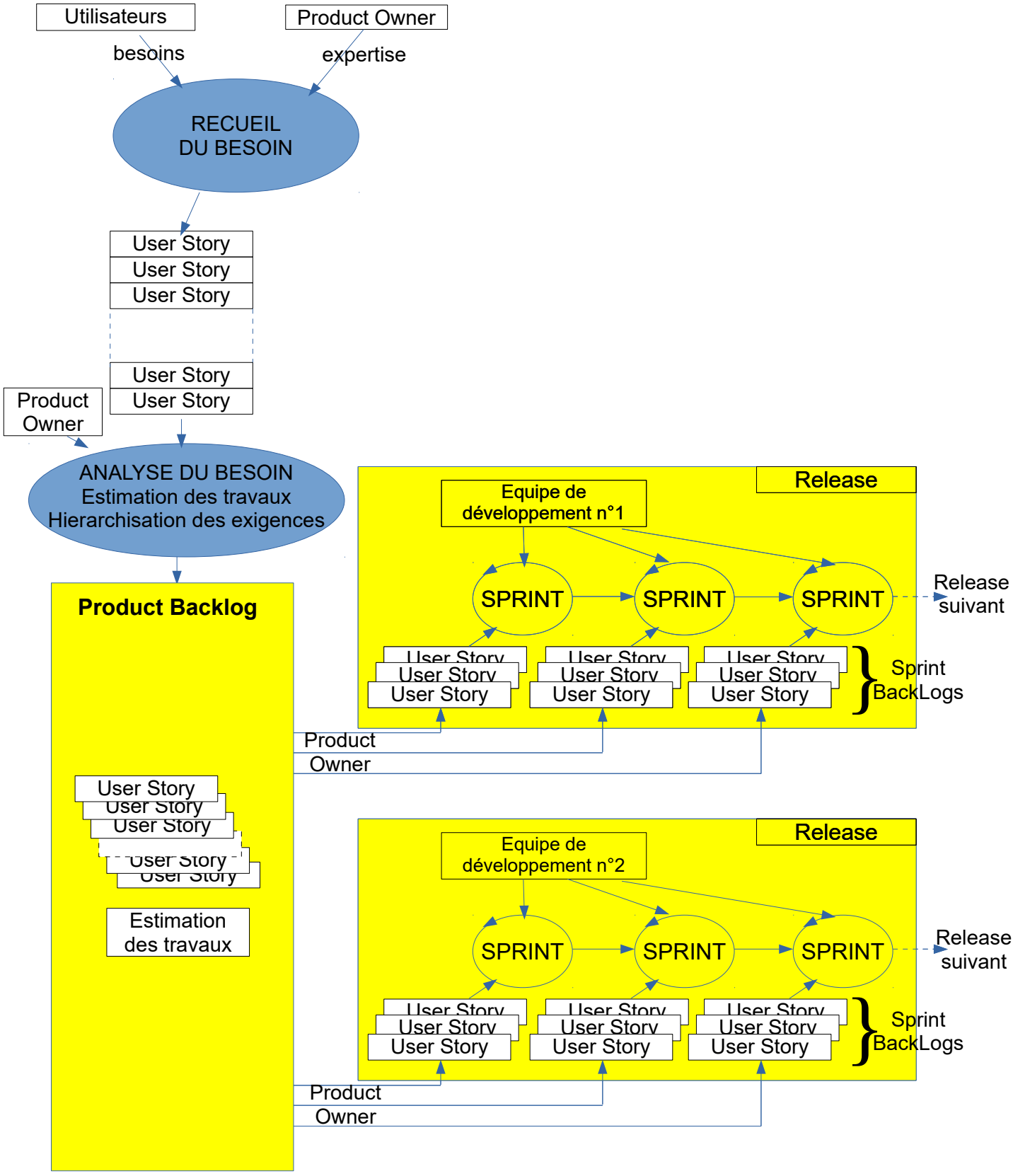
tirer des enseignements et des voies d’amélioration.

PASSAGE AU SPRINT SUIVANT:

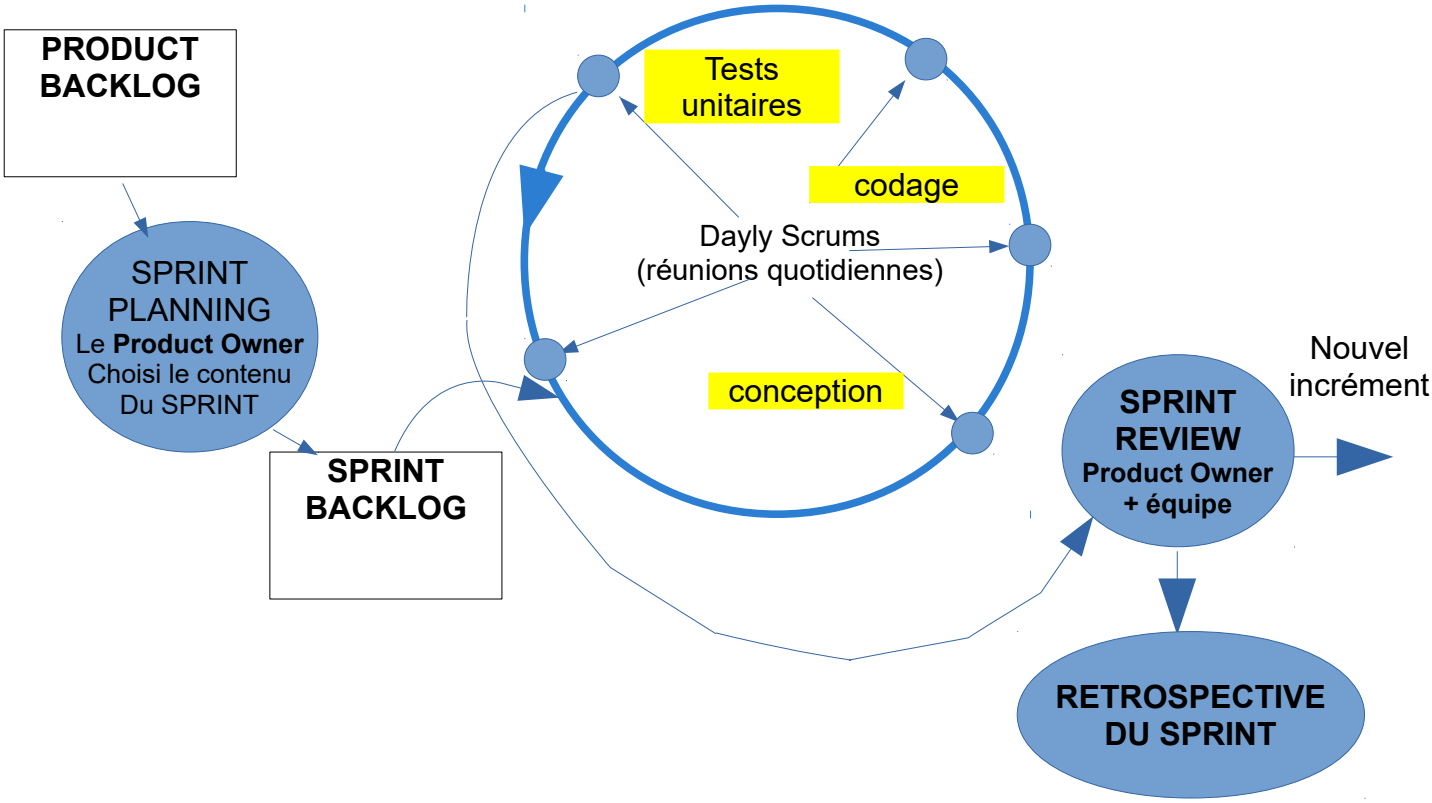
Le sprint suivant s’enchaîne à la suite selon le même cycle et ainsi de suite jusqu'au dernier sprint de la release.

IV.3.3.SCHÉMAS GÉNÉRAUX:

ORGANISATION GÉNÉRALE:



ORGANISATION D'UN SPRINT:



V.L'EXTRÊME PROGRAMMING:

V.1.CARACTÉRISTIQUES GÉNÉRALES:

L'eXtrême Programming (XP) prône un ensemble de pratiques dont le but principal est d'organiser le travail d'une équipe de développement de logiciel. Cette activités est du ressort de la MAÎTRISE D'ŒUVRE. Cependant, comme dans toute méthode agile, les travaux liés à la MAÎTRISE D'OUVRAGE (client) sont très fortement imbriqués avec ceux de la maîtrise d'œuvre, et ces deux entités dialoguent tout au long du projet. De ce fait, XP traite également des activités de maîtrise d'ouvrage, quoique d'une façon plus schématique.
La caractéristique principale de XP est de pousser à l'extrême (d'où le nom) les pratiques agiles qui sont en rupture avec les méthodes traditionnelles, comme le montre le tableau suivant:

Méthodes traditionnelles	XP
A chaque étape du projet, un DOCUMENT consigne les travaux réalisés et permet ainsi d'assurer la TRAÇABILITÉ de la démarche.	Une communication constante avec le client est privilégiée par rapport à l'établissement de documents d'étapes.
Le document consignant les SPÉCIFICATIONS du produit à réaliser est CONTRACTUEL. Il précède toute activité de conception ou de réalisation. Son contenu ne peut être modifié sans un AVENANT au contrat initial.	Les SPÉCIFICATIONS du produit ne font pas l'objet d'un document contractuel préalable au développement: elle sont censées émerger progressivement du dialogue avec le client. A l'extrême, il est admis qu'on puisse se passer d'établir un document de spécification: celui-ci peut être remplacé par les procédures de TEST DE RÉCEPTION, qui décrivent implicitement la réalité des interfaces et des scénarios d'utilisation.
L'architecture globale du produit est définie en préalable à toute activité de développement, pendant la phase de conception préliminaire. Elle fait l'objet d'un document qui s'impose à toutes les équipes de réalisation. Elle est déduite du document de spécifications et de l'étude des contraintes impliquées par ces spécifications. L'architecture globale doit être étudiée en vue de supporter toutes les fonctions résultant des spécifications et toutes les contraintes déduites de celles-ci.	L'architecture générale de l'application n'est pas définie à priori: elle est censée se dégager progressivement des travaux de développement, par REFACTORISATION et ADAPTATION du logiciel déjà développé dans le but d'accueillir les nouveaux composants. Une équipe de développement doit toujours choisir la solution la plus simple RÉSOLVANT UNIQUEMENT les problèmes posés par le module logiciel dont elle est chargée. Si des incompatibilités «incontournables» avec le reste de l'application apparaissent, l'architecture globale est améliorée, voire REFACTORISÉE pour résoudre le problème.

Cette comparaison met en évidence une volonté de rompre radicalement avec les méthodes classiques en abandonnant leurs principes essentiels:

- L'engagement contractuel est remplacée par une négociation permanente entre client et fournisseur sur les objectifs à atteindre ;
- Le cheminement «rectiligne» du général vers le particulier: conception préliminaire => conception détaillée des composants => Codage et tests => intégration, est remplacée par un cheminement par ITÉRATIONS successives convergeant vers le résultat final, chaque itération correspondant à l'implémentation de nouvelles exigences (cycle «en spirale»);
- L'exigence de TRAÇABILITÉ (écrite) est nettement atténuée au profit d'un ENGAGEMENT PERMANENT du client auprès des développeurs, qui fait de ce client un ACTEUR ENGAGÉ du développement.

Pour le reste, nous verrons par la suite que les pratiques XP sont globalement celles qui ont été décrites précedemment pour les méthodes agiles.

V.2.PRINCIPES FONDAMENTAUX:

V.2.1.GÉNÉRALITÉS:

Comme il se doit dans toute démarche agile, un projet XP avance par ITÉRATIONS successives: à chaque itération, de nouvelles exigences du client sont implémentées et intégrées au reste de l'application. Ces itérations doivent être relativement courtes (deux semaines en moyenne).

Il n'y a pas de phase exclusivement consacrée à des TRAVAUX PRÉALABLES (analyse fonctionnelle, spécification des exigences): en effet, la liste des exigences du client et l'architecture d'ensemble sont toujours susceptibles d'évolutions en cours de projet: elles sont sensé se dégager et se préciser au cours des itérations:

- Du fait de la négociation Client-Réalisateur en cours de projet (pour les exigences);
- Du fait de remaniements (refactorisations) successifs des logiciel déjà créés nécessités par de nouvelles implémentations (pour l'architecture).

Cependant, la première itération est forcément différente des autres car il faut bien mettre en place et initialiser les outils de suivi du projet. De ce fait, elle est en général nettement plus longue que les autres (jusqu'à un mois).

V.2.2.LES RÔLES:

LE CLIENT XP:

Le CLIENT XP est une personne physique qui assure le PILOTAGE du projet pour le compte du CLIENT. Il représente la MAÎTRISE D'OUVRAGE. Dans un projet classique, il pourrait s'agir de l'Assistant à la Maîtrise d'OuvrAge (AMOA). Le Client XP est responsable de:

- La définition et la caractérisation des exigences du client (appelées SCÉNARIOS CLIENTS en XP), et de la détermination de leurs priorités relatives;
- L'établissement du PLAN DE DÉVELOPPEMENT et de la mise à jour du planning;
- La détermination du contenu de chacune des itérations;
- La prononciation de l'ACCEPTATION (ou "acceptance") des livraisons (releases) issues des itérations.

Le Client XP représente le GROUPE DE PILOTAGE du projet, formé par le CLIENT, et qui peut comprendre des représentants de la direction, des finances, des futurs utilisateurs ou des experts et consultants employés sur le projet.

Le Client XP doit être à la fois expert dans le domaine d'activité du client, dans les activités de maîtrise d'ouvrage (recueil et expression du besoin, planification de projet, procédures de réception) et dans la relation client-fournisseur.

REMARQUE: *Dans un projet XP, le Client XP rejoint le projet à plein temps: il est donc difficile de le choisir à l'intérieur de l'entreprise cliente. Il s'agit souvent d'un CONSULTANT indépendant.*

MANAGER:

Le MANAGER est le chef hiérarchique et le gestionnaire de l'ÉQUIPE DE RÉALISATION. Il supervise la bonne marche de l'équipe et s'assure que ses résultats obtenus sont conformes aux prévisions. Il veille à ce qu'elle dispose des moyens est nécessaire à son fonctionnement. Il assure l'interface entre l'équipe et les autres entités du REALISATEUR.

COACH:

Dans un projet XP, l'équipe est censée gérer elle-même son activités. De ce fait, la fonction de chef de projet classique se trouve amputée de ses attributions concernant l'organisation et la surveillance des travaux. La fonction évolue donc vers un rôle différent résumé par l'appellation COACH (entraîneur). Le COACH assure donc:

- La coordination de l'équipe et la résolution des conflits;
- La formation «continue» aux pratiques de XP;
- L'amélioration des pratiques dans le sens d'une plus grande efficacité du groupe et d'une plus grande autonomie des individus, en exploitant le «feedback» (retour d'expérience) des itérations.

Le COACH doit donc avant tout avoir une excellente connaissance de la méthode XP et de sa pratique. Il doit également être formé à la dynamique des groupes et a leur animation. A la différence du SCRUM MASTER de la méthode SCRUM, il est immergé dans l'équipe de réalisation.

PROGRAMMEURS:

Les PROGRAMMEURS XP sont en fait des DÉVELOPPEURS capables de prendre en charge toutes les activités de développement des logiciels: conception, codage, tests unitaires et intégration, remaniement et amélioration des codes, documentation. Autonomes, ils travaillent le plus souvent en BINÔME.
Ces binômes sont créés pour une courte durée (de l'ordre de la journée). Les deux membres d'un binôme travaillent à LA MÊME TÂCHE. Dans ce cadre, ils occupent ALTERNATIVEMENT les fonctions de développement et de «relecture critique».

TRACKER:

Le TRACKER surveille l'avancement du projet et transmet les informations obtenues à toutes les personnes impliquées (manager, client, équipe de réalisation). Dans ce but, il suit le déroulement

des itérations et des tests de réception. Il s'assure du respect des règles de qualité de réalisation et des délais de livraison.

TESTEUR:

Les TESTEURS sont chargés de la PRÉPARATION et de la MISE EN ŒUVRE des tests de RÉCEPTION. Ce sont eux qui certifient les résultats des tests de réception (et non la réception toute entière qui est du ressort du Client XP). En principe, ils ne doivent pas faire partie de l'équipe. Idéalement, ils font partie de l'entreprise cliente, ou, mieux, appartiennent à une organisation tierce spécialisée.

V.2.3.LA PHASE EXPLORATOIRE:

Un projet XP commence par une phase exploratoire qui a pour buts:

- Recueillir et d'analyser les exigences du client afin d'établir une première liste de SCENARIOS CLIENTS (liste que l'on appelle souvent SPÉCIFICATIONS);
- Établir à partir de cette liste un PLAN DE DÉVELOPPEMENT initial;

Cette phase exploratoire est intégrée au début de la première itération de développement qui, de ce fait, dure plus longtemps que les autres (environ un mois).

V.2.4.LES ITÉRATIONS:

Chaque itération est censée implémenter un ensemble de scénarios clients choisis par le Client XP. Les résultats d'une itération sont immédiatement intégrés au produit en cours de réalisation. L'ensemble est validé par des TESTS D'INTÉGRATION (enchaînement des TESTS UNITAIRES de tous les composants créés). Une itération dure normalement 2 semaines.

V.2.5.LES LIVRAISONS (Releases):

XP, comme toutes les démarches agiles, recommande de faire le plus de livraisons partielles possibles du produit. Cependant, une livraison doit être validée par des TESTS D'ACCEPTATION (tests de réception), qui doivent correspondre à l'implémentation de nouvelles FONCTIONS ayant un sens pour les utilisateurs. De ce fait, une LIVRAISON concerne les résultats d'une ou plusieurs itérations (pas plus de 3).

V.3.DÉROULEMENT D'UN PROJET XP:

V.3.1.PREMIÈRE ITÉRATION:

EXPLORATION DES EXIGENCES:

Le CLIENT XP recueille et analyse les besoins des utilisateurs, puis établit une première liste de ses exigences (fonctionnelles, opérationnelles, environnementales, techniques, etc.), qu'il présente sous la forme de SCENARIOS CLIENT (équivalents aux USER STORIES de SCRUM). Cette liste n'a pas vocation à être exhaustive ni contractuelle: à priori, elle contiendra les exigences les plus importantes pour le client et permettra de définir la PREMIÈRE LIVRAISON attendue.

REMARQUE:

- Un scénario client se présente sous la forme d'une phrase simple définissant une attente d'un utilisateur vis à vis du produit. Par exemple: «Un utilisateur de base muni d'une carte de crédit bancaire doit pouvoir retirer de l'argent liquide en saisissant le montant désiré et son code secret.»;
- Un scénario doit être accompagné de CRITÈRES précis et VÉRIFIABLES (testables). Par exemple: «le retrait maximum sera de 300 euros par jour» ou bien «Les billets seront délivrés sous réserve d'autorisation de la banque de l'utilisateur». En effet, ces critères sont essentiels pour la création des procédures de tests de réception;
- Les scénarios sont rédigés «dans le langage du client». A priori, toute formalisation du genre UML ou autre doit être bannie à ce niveau;
- Un scénario doit pouvoir être implémenté en une seule itération (typiquement 2 semaines). Pour cela, il doit décrire sans ambiguïté les attentes du client (pour rendre inutile tout complément de spécifications) et ne pas être trop complexe (pour ne pas exiger un volume de codage trop important). Il est donc conseillé, lorsqu'un scénario ne remplit pas ces conditions, d'essayer de le décomposer en scénarios plus clairs et plus simples à réaliser.
- Inversement, des scénarios trop simples à réaliser doivent être fusionnés.

La Liste des scénarios obtenue constitue la première version de ce que l'on appelle souvent les SPÉCIFICATIONS du produit à réaliser. Elle n'est pas forcément consignée dans un document unique: parfois on se contente de noter chaque scénario sur une simple fiche. Dans certains projets, on considère que l'ensemble des TESTS DE RÉCEPTION (directement déduits des scénarios) représente les spécifications du produit.

PLANIFICATION DU PROJET:

Une fois la liste des scénarios établie et consolidée, un premier plan de développement peut être élaboré par le CLIENT XP, en collaboration avec l'ÉQUIPE DE RÉALISATION: La procédure est la suivante:

- Le CLIENT XP communique à l'équipe la liste des scénarios qu'il a établie;
- L'ÉQUIPE tente d'évaluer le coût d'implémentation de chaque scénario. Ce coût est représenté par un nombre de points qui est sensé représenter ce coût, non dans l'absolu, mais RELATIVEMENT aux coûts des autres scénarios. Ainsi, un scénario que l'on évalue à 6 points est sensé avoir un coût d'implémentation double d'un scénario à 3 points.
- L'équipe tente également d'évaluer sa VÉLOCITÉ, c'est à dire le nombre de points qu'elle est capable d'implémenter en une itération «standard» (en général, 2 semaines). Là encore, cette VÉLOCITÉ n'a qu'une valeur RELATIVE: elle sera constamment réajustée en cours de projet. C'est surtout sa VARIATION qui a une importance comme révélateur de difficultés que le groupe est en train de rencontrer.
- Le CLIENT XP effectue alors un classement des scénarios en fonction de leur IMPORTANCE pour les utilisateurs. Les scénarios les plus importants on vocation à être traités en premier.
- Le CLIENT XP planifie également les ITÉRATIONS et les LIVRAISONS au client.

REMARQUE: *à la lumière de ce que nous venons de lire, il est évident que le coût d'implémentation représente une DURÉE et non un EFFORT, comme c'est le cas des «hommes*mois» classiques. Cela s'explique par le fait que dans XP, l'équipe de développement ne se scinde pas en plusieurs sous-groupes: l'ensemble des développements d'une itération est*

placé sous la responsabilité COLLECTIVE des membres de l'équipe.

Ces travaux de planification sont effectués lors d'une réunion bipartite (CLIENT XP - ÉQUIPE). Le résultat peut être enregistré sur un TABLEUR, ce qui facilitera sa mise à jour par la suite. Il constituera le PLAN DE DÉVELOPPEMENT du produit. Ce document sera mis à jour JOURNELLEMENT pendant tout le projet.

ÉLABORATION DE LA PREMIÈRE LIVRAISON:

L'ÉQUIPE dispose alors d'une première liste de scénarios et d'un PLAN DE DÉVELOPPEMENT initialisé. Elle peut donc commencer le développement d'un premier ensemble de scenarios qui seront choisis par le CLIENT XP.
Celui-ci doit essayer de choisir cet ensemble de manière à ce qu'il représente pour les utilisateurs une sorte de NOYAU FONCTIONNEL minimal de l'application, LIVRABLE au client.
La démarche est alors exactement la même que pour les itérations suivantes. Se reporter au paragraphe suivant «LES ITÉRATIONS» ;

V.3.2.LES ITÉRATIONS:

REMARQUE PRÉLIMINAIRE:

Souvent, la littérature traitant de XP, mentionne que toute ITÉRATION doit aboutir à une LIVRAISON au client. Cependant, la durée recommandée pour les itérations en XP étant très courte (2 semaines), il est difficile de développer pendant cette durée des incréments ajoutant des fonctionnalités complètes et représentant un intérêt pour les utilisateurs.

De ce fait, il est souvent admis que les LIVRAISONS puissent correspondre au produit de plusieurs ITÉRATIONS (pas plus de 3, toutefois). De ce fait, la planification des LIVRAISONS doit être dissociée de celle des ITÉRATIONS.

LE CYCLE XP:

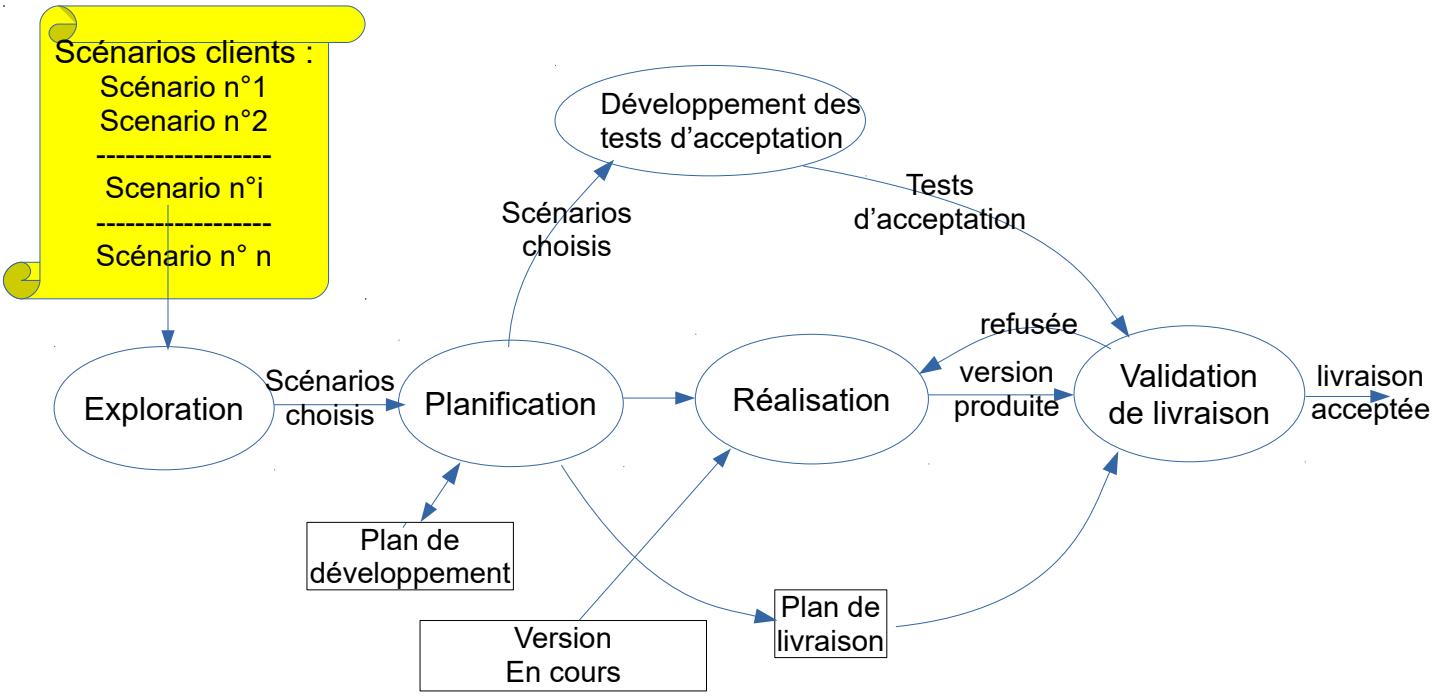
Un cycle (ou itération) eXtrême Programming peut se décomposer en 3 ou 4 PHASES, suivant qu'une LIVRAISON est prévue ou non à son issue. Ces phases sont :

1. La phase d'EXPLORATION, pendant laquelle le CLIENT XP choisit les SCÉNARIOS CLIENTS qui doivent être implémentés par l'itération;
2. La phase de PLANIFICATION pendant laquelle le CLIENT XP et L'EQUIPE déterminent et évaluent les différentes TÂCHES qu'il faut accomplir pour implémenter ces scénarios. Si l'itération se termine par une livraison, celle-ci sera préparée (préparation et planification des tests). Le développement des TESTS D'ACCEPTATION de chacun des scénarios choisis est alors lancé, en parallèle de la phase de réalisation (ce sont les TESTEURS qui les conçoivent et les réalisent);
3. La phase de RÉALISATION pendant laquelle ces différentes tâches sont réalisées par les développeurs (réalisations en binômes). L'équipe est censée s'administrer elle-même: le COACH assure la coordination, la résolution des conflits et le support méthodologique. Le TRACKER assure la surveillance des travaux et fait remonter les informations au manager et au client XP. Cette phase comporte toutes les activités liées au développement (Conception, codage et tests unitaires, intégration, tests d'intégration. Les PROGRAMMEURS, en même temps qu'ils implémentent les scénarios prévus par l'itération, doivent également:
    - AMÉLIORER ET NETTOYER si nécessaire les codes qu'ils sont amenés à consulter ou à utiliser;
    - REMANIER (si nécessaire) la conception d'ensemble pour l'adapter aux contraintes de ces nouveaux scénarios ou pour la rendre plus efficiente

(refactoring);
- Détecter et utiliser les OPPORTUNITÉS DE RÉUTILISATION de codes objets.

4. La phase de VALIDATION (uniquement si l'itération se termine par une livraison). Pendant cette phase, les différents produits de l'itération sont TESTÉS par les TESTEURS pour déterminer s'ils peuvent être acceptés par le CLIENT XP. Si l'ensemble des produits est accepté, les produits sont livrés et on passe à une autre itération. Sinon, les produits non acceptés sont remis en production.

Le Schéma ci-après représente le déroulement d'une itération terminée par une livraison:



V.4.ÉTUDE DÉTAILLÉE DES PRATIQUES DE XP:

V.4.1.PRATIQUES RÉDUISANT LA DURÉE DU CYCLE DE DÉVELOPPEMENT:

NOTE: *on appelle DURÉE DU CYCLE DE DÉVELOPPEMENT d'une EXIGENCE le temps qui s'écoule entre le moment où l'on décide d'implémenter cette exigence et celui où l'implémentation de cette exigence dans le code objet est effective.*

Dans les méthodes de développement classiques, on peut considérer que la décision d'implémenter une exigence est prise au moment où la S.T.B est validée définitivement par le client, c'est à dire à la fin de la phase de SPÉCIFICATION DU BESOIN (en fait, toutes les exigences sont validées en même temps). Cette exigence n'est effectivement implémentée dans le produit qu'en phase d'INTÉGRATION. La durée du CYCLE DE DÉVELOPPEMENT d'une exigence comprend donc au minimum:

- La durée de la phase de CONTRACTUALISATION, qui peut être très longue si l'on y intègre le choix du maître d'œuvre;
- La durée de la phase de CONCEPTION PRÉLIMINAIRE;
- La durée de la phase de CONCEPTION DÉTAILLÉE-CODAGE ET TESTS UNITAIRES des composants implémentant l'exigence;
- Au moins une partie de la durée de la phase d'INTÉGRATION (l'implémentation est effective lorsque les tests d'intégration ont été effectués avec succès (y compris les tests de non régression).

Pour un projet de complexité moyenne, la durée du cycle de développement d'une exigence se compte en mois (trois mois doit être une valeur moyenne !).

Un des objectifs de XP est de réduire au minimum cette durée. On y parvient essentiellement par:

- Le découpage du projet en ITÉRATIONS de courte durée, la plupart d'entre elles s'attachant à obtenir un produit LIVRABLE AU CLIENT;
- L'absence d'une phase de CONCEPTION PRÉLIMINAIRE au profit de l'étalement des travaux de conception sur toutes les itérations, par la pratique du REMANIEMENT de la structure logicielle;
- La recherche de la SIMPLICITÉ en matière de conception et de réalisation;
- La prise en compte la plus rapide possible du RETOUR SUR EXPÉRIENCE (feed back) de chaque itération, ce qui permet d'ajuster les pratiques «en temps réel»;

V.4.2.LA RÉUNION DE PLANIFICATION D'ITÉRATION:

Cette réunion a lieu au début de chaque itération, après la PHASE EXPLORATOIRE. Elle réunit le CLIENT XP, L'ÉQUIPE DE DÉVELOPPEMENT et les TESTEURS. Son ordre du jour est le suivant:

- Retour d'informations (feed back) sur la dernière itération, recherche d'amélioration;
- Description des scénarios choisis par le client pour être implémenter au cours de l'itération en cours;
- Analyse de ces scénarios, recherche de solutions techniques, constitution de la liste des tâches à effectuer pendant l'itération.

Une description plus fine des scénarios et une meilleure compréhension des mécanismes qu'ils impliquent est recherchée. Le dialogue entre les différents participants doit s'appuyer sur des MÉTAPHORES appartenant à la vie courante (Par exemple, le mécanisme d'interaction entre un utilisateur et un logiciel peut être assimilé à celui d'un AUTOMATE BANCAIRE: identification, authentification, saisie de la demande, délivrance des billets ou refus).

Au terme de la réunion, les développeurs choisissent eux-mêmes les tâches qu'ils accompliront en cours d'itération. Ce choix n'empêche pas le fait que la RESPONSABILITÉ DU CODE EST COLLECTIVE: chaque développeur est concerné par l'ensemble des tâches et peut être amené à participer à chacune d'entre elles du fait du travail en BINÔME.

V.4.3.AUTOMATISATION DES TESTS DE RÉCEPTION:

Les tests de réception doivent, dans l'idéal, vérifier de manière AUTOMATIQUE (sans intervention ou interprétation humaine) chacune des fonctionnalités demandées par le client.

Ces tests exploitent des JEUX D'ESSAI qui peuvent se présenter sous diverses formes:

- Feuilles de tableur ou fichiers XML renfermant des données d'entrées et les données de sortie correspondantes ;
- Scripts simulant des séquences d'interactions de l'utilisateur avec l'interface graphique du produit ;
- Etc.

En réalité, il est très difficile de définir des tests entièrement automatiques pour des fonctions INTERACTIVES comme les IHM.

Dans un projet XP l'ensemble des tests de réception est parfois considéré comme équivalent aux SPÉCIFICATIONS DU PRODUIT. Dans ce cas, aucun document de spécifications n'est élaboré. Cependant, il est rare que le client se contente de cette solution.

L'ensemble des tests de réception est lancé très souvent, et au minimum après chaque intégration d'un incrément (d'où l'intérêt de l'automatisation).

V.4.4.ORGANISATION DES ÉQUIPES DE DÉVELOPPEMENT:

ORGANISATION «TRADITIONNELLE»:

Une équipe de projet «traditionnelle» possède une structure fonctionnelles bien déterminée:

- Le CHEF DE PROJET organise et coordonne les travaux de l'équipe pour le compte de la MAÎTRISE D'ŒUVRE. Les autres membres de l'équipe lui sont subordonnés, au moins pendant la durée du projet. Il supervise les travaux de conception et d'intégration. Il est garant du respect de l'architecture générale du produit ainsi que de la qualité de réalisation (il s'appuie parfois pour cette dernière tâche sur un «responsable qualité», distinct de l'équipe);
- L'équipe est souvent organisée en petits groupes ayant chacun la responsabilité d'un sous-ensemble des tâches à effectuer et travaillant indépendamment l'un de l'autre à partir de documents de conception préliminaire élaborés sous la direction du chef de projet. Ceci ne veut pas dire que deux groupes ne peuvent pas collaborer ponctuellement, mais chaque

groupe n'est responsable que des tâches qui lui sont confiées.

ORGANISATION D'UNE ÉQUIPE XP:

Dans une équipe XP, au contraire, il n'existe pas de rapport de subordination, même sur le plan fonctionnel. Chaque développeur peut être amené à intervenir dans toutes les tâches de l'itération. Chaque développeur est garant de la qualité de la réalisation de l'ensemble des tâches. De ce fait, il a le devoir de remanier si besoin le code objet qu'il est amené à connaître afin d'augmenter son efficience ou sa qualité du point de vue de la conception ou du codage et de faciliter le travail des développeurs qui interviendront par la suite sur ces mêmes codes. La RESPONSABILITÉ du code produit est donc COLLECTIVE.

Le seul membre de l'équipe qui se différencie des autres est le COACH, qui, nous l'avons vu précédemment, exerce surtout les rôles d'animateur, de facilitateur et de formateur (formateur XP).

Chacun des développeurs d'une équipe XP est sensé travailler **EN BINÔME**. Pendant la durée de vie du binôme, les deux agents travaillent sur le même poste et aux mêmes tâches, sans hiérarchie. Le but recherché est la confrontation permanente des points du vue et des expériences. Les binômes peuvent changer très souvent (tous les jours est un bon rythme), afin d'amener chaque agent à travailler avec tous les autres, à avoir une vision d'ensemble de la réalisation et à minimiser l'impact de l'absence d'un agent.

SURVEILLANCE DES TRAVAUX EN COURS D'ITÉRATION: STAND-UP MEETINGS:

Un STAND-UP MEETING est une réunion JOURNALIÈRE d'une quinzaine de minutes où les participants se tiennent debout (pour éviter que cela ne traîne). Au cours de cette réunion, chacun des membres de l'équipe fait le point sur l'avancement des tâches auxquelles il collabore. En fonction des renseignements recueillis, les travaux des journées suivantes sont réajustés et le planning mis à jour. Cette réunion est assez semblable au DAYLY SCRUM de la démarche SCRUM.

V.4.5.APPROCHE DE LA CONCEPTION BASÉE SUR LES MÉTAPHORES:

Dans le domaine de l'architecture des logiciels, les mécanismes décrivant les interactions entre les différentes entités sont très souvent suggérés par des MÉTAPHORES prises dans la vie courante. Par exemple:

- Le modèle CLIENTS-SERVEURS désigne une architecture dans laquelle une entité informatique (le SERVEUR) se tient prête en permanence à recevoir des REQUÊTES de la part d'entités CLIENTES (qui, elles, peuvent apparaître et disparaître à tout moment) et à fournir à ces entités les SERVICES répondant aux requêtes reçues.
- La notion de FILE D'ATTENTE désigne un mode de communication dans lequel les messages en attente de traitement par une entité informatique réceptrice sont traités par celle-ci dans l'ordre de leur arrivée.

L'utilisation de métaphores permet une approche intuitive des mécanismes logiciels qui convient bien au dialogue utilisateurs-concepteurs. L'ensemble des métaphores couramment utilisées peut constituer un vocabulaire commun aux développeurs et au client qui permet une approche commode des concepts architecturaux.

V.4.6.SIMPLICITÉ DANS LA DÉMARCHE DE CONCEPTION:

Lorsqu'un binôme travaille sur l'implémentation d'un scénario client, il doit, dans un premier temps, prendre en compte dans sa démarche de conception uniquement les contraintes imposées par ce scénario. Ce n'est qu'en cas d'impossibilité d'implanter ce scénario sans modifier l'environnement d'implantation (la partie de l'application déjà intégrée) que l'on envisagera de remanier celui-ci pour le rendre compatible.

D'autre part, on s'attachera à répondre EXACTEMENT au scénario client, même si l'on détecte des possibilités d'amélioration ou d'optimisation.

V.4.7.IMPORTANCE DE L'ADOPTION DE RÈGLES DE CODAGE:

L'instauration de règles de codage permet de donner au code un aspect uniforme sur toute l'application. Ceci est très important en XP car tous les développeurs peuvent être amenés à travailler sur tout l'applicatif.

V.4.8.DÉMARCHE DE DÉVELOPPEMENT PILOTÉE PAR LES TESTS:

XP recommande la pratique du DÉVELOPPEMENT GUIDÉ PAR LES TESTS. Celle-ci consiste essentiellement à développer les TESTS UNITAIRES d'un composant AVANT de développer ce composant: de ce fait, ces tests unitaires peuvent être utilisés au cours du développement du composant lui-même pour valider partiellement ou totalement les ajouts effectués.

V.4.9.REMANIEMENT CONTINU DU CODE:

Nous avons vus qu'en XP, chacun des développeur est responsable de l'ensemble du code produit lors d'une itération. Le corollaire de cette règle est que chaque développeur SE DOIT d'améliorer le codage ou la conception de tous les éléments du produit dont il a connaissance.

Cette amélioration peut concerner:

- L'architecture globale (design refactoring);
- La qualité du code (respect des normes de codage, élimination de code mort, etc.);
- L'élimination des duplications par la recherche de la RÉUTILISATION de codes (remaniement de fonctions, méthodes ou classes pour les rendre réutilisables).

V.4.10.PRATIQUE DE L'INTÉGRATION CONTINUE:

En XP, dès qu'un binôme a terminé une tâche, le code produit est intégré à la version en cour. La nouvelle version est alors soumise a des tests d'intégration (ensemble des tests unitaires). La version "livrable" du logiciel est donc à jour de l'état d'avancement des développements.

V.5.LES VALEURS PRÔNÉES PAR XP:

En définitive, les méthodes agiles, et en particulier XP essaient de ramener l'élément humain au centre du processus de création en assouplissant, ou même en proscrivant les pratiques qui dans les méthodes traditionnelles tendent justement à minimiser l'influence de ces éléments humains (division du travail, accent mis sur la documentation et la traçabilité, hiérarchisation des responsabilités, planification stricte, contrôle de la circulation de l'information).

Les créateurs de XP mettent en avant les valeurs suivantes:

- IMPORTANCE DU CONTACT HUMAIN DIRECT par rapport à l'échange formel de documents: au cours d'une itération, tout le monde PARLE avec tout le monde, sans filtre hiérarchique;
- IMPORTANCE DES RETOURS D'EXPÉRIENCE: les itérations courtes, le travail en binômes non spécialisées, les équipes non morcelées permettent les améliorations par le «feed back»;
- IMPORTANCE D'AVOIR UNE DÉMARCHE SIMPLE en conception et en codage: ne s'attacher qu'aux problèmes qui se posent effectivement à un moment donné;

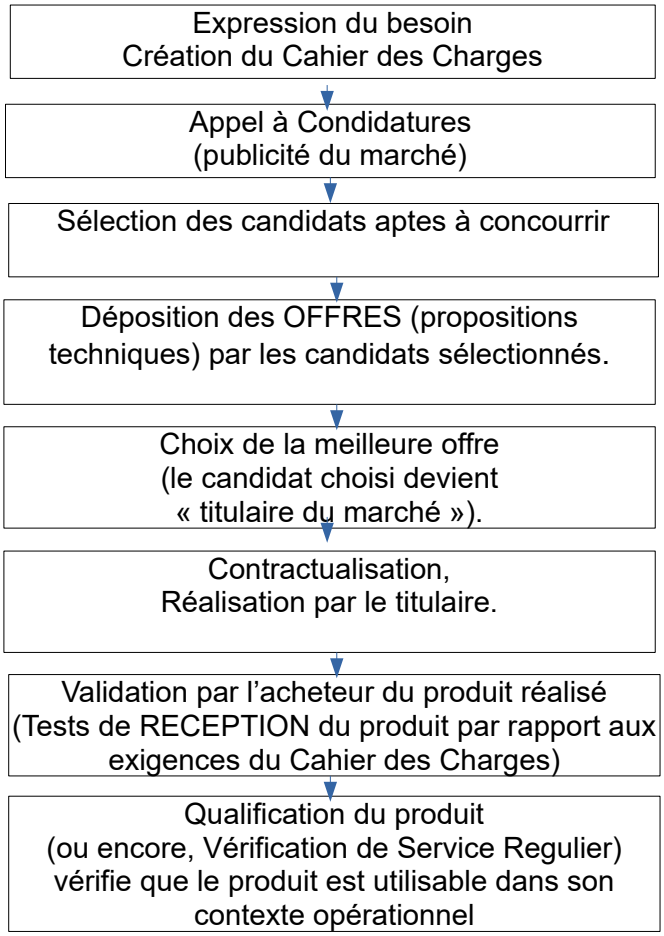
A ces trois valeurs, ils ajoutent le COURAGE de maintenir le cap malgré les difficultés d'une démarche qui s'appuie énormément sur la bonne volonté, la compétence et l'adhésion des acteurs.

VI.MÉTHODES AGILES ET MARCHÉS PUBLICS:

VI.1.GÉNÉRALITÉS:

Un marché public «CLASSIQUE», dit «marché à forme simple» ou «marché forfaitaire», est basé sur la procédure suivante:

DEROULEMENT :



PRINCIPES :

Garantir l'égalité de traitement des acteurs économiques vis à vis de l'accès aux marchés publics dans les domaines suivants:
- Informations sur les marchés;
- Dépot des chandidatures et des offres techniques.
- Choix des candidature ;
- Choix des offres ;

De ce fait, lorsqu'un titulaire a été choisi, sur la base de sa réponse à un Cahier des Charges donné, il est impossible de modifier les exigences contenues dans ce cahier des charges.
En effet, une telle pratique pourrait être considérée par les candidats non retenus comme du favoritisme vis à vis du titulaire : ils pourraient au minimum exiger l'annulation de l'appel d'offres et l'organisation d'un nouvel appel d'offre sur la base des nouvelles exigences.

L'utilisation de méthodes agiles dans le cadre de marchés publics de développement de logiciels se heurte donc à deux difficultés:

- Le fait que les spécifications de besoin initiales puissent évoluer en cours de réalisation peut poser le problème de l'égalité de traitement des candidats au marché. En effet, toute négociation avec le titulaire sur le contenu du Cahier des Charges peut être considérée comme une rupture d'égalité entre les candidats et qualifiée de favoritisme.
- La difficulté d'établir un planning de réalisation fiable et un prix de réalisation fixe empêchent également de finaliser le Cahier des Charges avant l'appel d'offres. Cemme précédemment, toute négociation avec le titulaire sur le contenu du Cahier des Charges peut être considérée comme une rupture d'égalité entre les candidats et qualifiée de favoritisme.

Il existe cependant certaines formes de marchés publics qui permettent de déroger en partie à ces principes et de s'adapter plus ou moins aux méthodes agiles. Les types de marchés suivants sont souvent évoqués:

- Les ACCORDS CADRES;
- Les MARCHES A BONS DE COMMANDE;
- Les DIALOGUES COMPÉTITIFS;
- Les MARCHÉ À TRANCHES CONDITIONNELLES.

VI.2.LES ACCORDS CADRES:

DESCRIPTION:

L'acheteur sélectionne un ou des candidats sur la base de PRINCIPES (et non d'un Cahier des Charges) qui présideront à la conclusion de marchés ultérieurs. Les marchés passés ultérieurement avec les candidats sélectionnés devront respecter les principes définis par l'accord cadre.

DEGRÉS D'ADAPTATION AUX MÉTHODES AGILES:

Les accords cadres permettent effectivement de développer un produit en plusieurs itérations, sans définir au départ un besoin et un planning figés. Cependant, chaque itération va correspondre au passage d'un marché public avec mise en concurrence des prestataires et toute la lourdeur administrative du processus. Or, une itération de méthode agile est prévue pour durer 2 à 4 semaines alors que le passage d'un marché public dure plusieurs mois, ce qui rallonge d'autant la durée du projet.

VI.3.LES MARCHÉS A BONS DE COMMANDES:

DESCRIPTION:

Ils permettent à l'acheteur de sélectionner un ou plusieurs fournisseurs pour un type de besoins, à partir d'un cahier des charges définissant des types de prestations, puis de passer de simples COMMANDES à ces fournisseurs à mesure que ces besoins apparaîssent.

DEGRÉS D'ADAPTATION AUX MÉTHODES AGILES:

Dans le cadre des marchés à bons de commandes, l'acheteur peut sélectionner **un seul candidat** sur la base d'un cahier des charges spécifiant que les offres concernent la réalisation d'un ou plusieurs lots définis dans le CCTP et la STB annexés à ce cahier des charges. Chacun de ces lots correspond à la réalisation d'une partie des exigences de l'application et doit être FONCTIONNEL.

Une fois le candidat sélectionné, l'acheteur lui passe une série de COMMANDES, chacune des commandes correspondant à la réalisation d'un lot.

Le passage d'une commande étant beaucoup plus simple que le passage d'un marché public, ce type de marché semble beaucoup plus adapté aux méthodes agiles, d'autant plus que l'acheteur est libre de passer le nombre de commandes qu'il estime nécessaire et que le contenu des différentes commandes peut être adapté en fonction de l'évolution des besoins de l'acheteur et des résultats des commandes précédentes.

VI.4.LES DIALOGUES COMPÉTITIFS:

DESCRIPTION:

- L'acheteur fait un appel d'offres sur la base d'un PROGRAMME FONCTIONNEL, dans lequel, selon la réglementation, «l'acheteur décrit en termes pratiques ses attentes et les résultats qu'il veut atteindre». Il ne s'agit donc pas d'un cahier des charges complet.
- Une fois les candidats retenus, l'acheteur va initier un DIALOGUE COMPÉTITIF avec chacun d'eux, par lequel il va tenter d'améliorer leurs offres respectives. A tout instant, l'acheteur peut décider que les négociations sont terminées et inviter les candidats à déposer leurs offres. La suite correspond à un marché classique.

DEGRÉS D'ADAPTATION AUX MÉTHODES AGILES:

Ce type de marché permet effectivement de ne figer ni les besoins ni les délais. Cependant, une fois que le titulaire du marché a été choisi, celui-ci est tenu de respecter son OFFRE, ce qui est peu compatible avec la prise en compte de l'évolution des besoins qui est un principe des méthodes agiles.

VI.5.LES MARCHÉ À TRANCHES CONDITIONNELLES:

DESCRIPTION:

Ils perment à l'acheteur de préciser dans le cahier des charges que la réalisation de l'application est divisée en plusieurs TRANCHES, la première étant «ferme», les autres étant «conditionnelles». Il peut ainsi décider de terminer le développement après chaque étape.

DEGRÉS D'ADAPTATION AUX MÉTHODES AGILES:

Ce type de marché permet effectivement de prendre en compte les différentes itérations prévues dans les méthodes agiles. Cependant, le marché est passé sur la base sur un cahier des charges entièrement finalisé, auquel l'acheteur et le titulaire peuvent difficilement se soustraire d'un point de vue juridique. La modification des exigences tout au long du projet semble donc difficile à mettre en œuvre.

VI.6.CONCLUSION:

La forme de marché qui semble s'adapter le plus facilement aux méthodes agiles tout en n'engendrant pas trop de tâches administratives semble être le MARCHE A BONS DE COMMANDES. Cependant, ce type de marché est plutôt réservé à des MARCHÉS DE FOURNITURES ou à des prestations d'ASSISTANCE ou de MAINTENANCE. Il convient donc, pour éviter tout problème juridique, de faire valider par un expert la procédure avant le début du marché.

A défaut, les MARCHES A TRANCHES CONDITIONNELLES peuvent convenir en partie, du moins en ce qui concerne le déroulement itératif. En revanche, la négociation des exigences semble plus difficile à mettre en œuvre sans faire intervenir des AVENANTS au contrat initial.