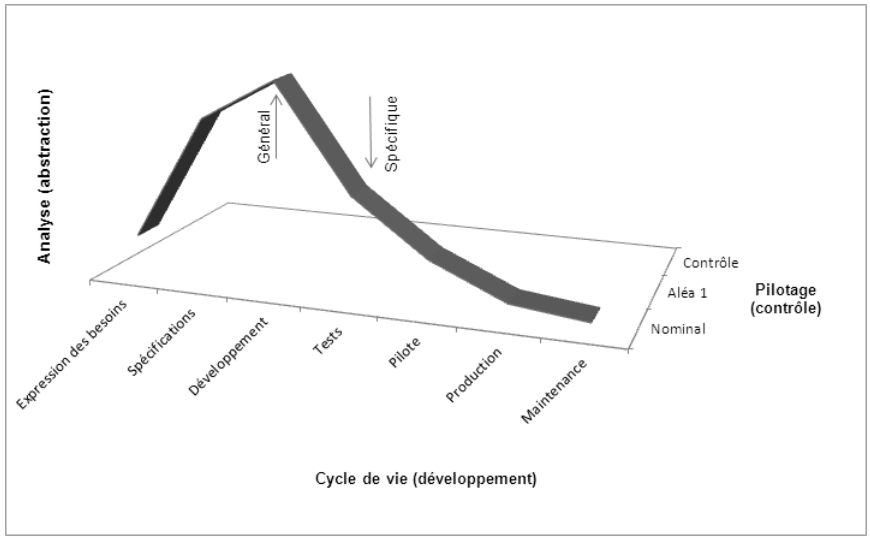
Chapitre 3
La prise en compte du risque

1. Les trois axes

Le chef de projet en devenir est parfois perplexe face au nombre des éléments qu'il doit organiser pour aboutir. Dans quelle direction partir ? Par quoi commencer et comment continuer ?

Pour répondre à ces questions, il faut d'abord se représenter le processus du projet tel que nous l'avons décrit au chapitre Démarrer un projet informatique. Cette fois-ci, nous choisissons un espace structuré par trois axes, le temps (cycle de vie), l'analyse et le pilotage.

Chaque dimension de cet espace correspond elle-même à une méthode plus ou moins standard, qui donnera lieu à un processus que le chef de projet devra suivre.



Le premier axe (en abscisse horizontale) est celui qui décrit les phases du projet, tel que l'on peut le préfigurer. Le schéma ci-dessus propose un modèle de développement assez standard, partant de l'expression des besoins, évoluant jusqu'à la livraison (production) et la maintenance du projet. En toute rigueur, on devrait introduire la mort du projet, car nous savons que le cadre support n'est pas infini. Nous le verrons par la suite, d'autres modèles de développement existent et proposent des organisations différentes.

Vient ensuite l'axe d'analyse (en ordonnée, vertical). Il indique le niveau d'abstraction du projet. Plus l'ordonnée est haute, plus grande est l'abstraction. Inversement, une faible ordonnée équivaut à un niveau de détail très important. Dans l'exemple figurant sur le schéma, on voit nettement l'abstraction grandir au début du projet avant de redescendre progressivement. Ce type de courbe est assez caractéristique des méthodes d'analyse basées sur UML (*Unified Modeling Language*) et Merise. Là encore, d'autres schémas sont envisageables et ils traduisent d'autres façons de procéder.

Sur le dernier axe sont représentés les aléas, les points de contrôles, les prises de décision. Il s'agit de l'axe de pilotage (cycle de décision). Par rapport aux deux autres axes, c'est celui qui réserve le plus de « surprises », mais il ne faut pas non plus le laisser totalement libre. Des méthodes se sont forgées au fil des projets pour organiser le pilotage de manière méthodique et efficace.

Ainsi, le « secret » du démarrage d'un projet résiderait-il dans le choix des méthodes associées à chaque dimension ? Ce n'est pas tout à fait suffisant, car il faut encore s'assurer d'un mélange harmonieux de ces trois composantes, d'une communication efficace, bref d'un sens pratique sans lequel les meilleures recommandations du monde resteront vaines. Autrement dit, l'application isolée, cloisonnée, d'une méthode sur un seul axe ne donne pas de meilleurs résultats qu'un projet dans lequel on oublierait l'existence des autres axes.

Le tableau suivant donne un aperçu du résultat lorsque les méthodes sont appliquées seules (indépendamment les unes des autres) ou lorsqu'elles font défaut :

Méthode/axe	Application seule ou indépendante	Non-application (absence)
Développement	En général, la méthode d'analyse est censurée ou ne donne aucun résultat probant.	Non-conformité (l'expression des besoins est prise en compte tardivement). Dérive du projet (difficulté à stabiliser la solution par manque de tests).
Analyse	Vision beaucoup trop conceptuelle ; le projet devient techniquement risqué car les tests d'intégration sont oubliés.	Non-conformité (focus sur des fonctions mineures, alors que les fonctions majeures manquent). Code non optimal entraînant des difficultés à le faire évoluer (pas de factorisation).

Pilotage	Inadéquation du pilotage par rapport aux caractéristiques du projet.	Non tenue des objectifs, manque de contrôle. Explosion de l'équipe projet en cas de montée en charge. Conduite du changement inexistante : rejet de l'application livrée.

Ce tableau récapitule donc six constructions (*antipatterns*) qu'il convient absolument d'éviter. Et pourtant, les occasions de se tromper ne manquent pas, soit parce que les habitudes ont la vie dure, soit parce qu'une nouvelle méthode n'est pas toujours à même de s'intégrer facilement dans le paysage d'un référentiel de conduite de projets.

Remarque

Les exemples d'achoppement sont légion. Le chef de projet doit toujours garder en tête que les trois composantes sont indispensables et qu'elles doivent coopérer pour être efficaces.

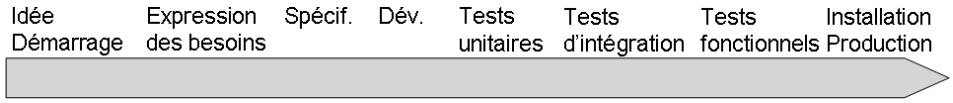
2. Le modèle de développement

Le modèle de développement constitue l'un des premiers choix que doit effectuer le chef de projet. Il façonne de manière explicite l'organisation temporelle du projet et a une grande influence sur le résultat final.

Il n'y a pas de bon ou de mauvais modèle, du moins parmi les grandes classes dont l'énumération suit. Cependant, chaque projet a des caractéristiques qui rendent l'application d'un modèle judicieuse ou au contraire inefficace.

2.1 Le modèle cascade

On l'appelle aussi modèle linéaire ou modèle nominal. C'est certainement le modèle le plus simple, ce qui ne le déprécie pas pour autant !



Dans ce modèle, chaque étape suit la précédente, sans autre nécessité que « d'attendre » son issue ; dès que l'expression des besoins est terminée, les spécifications sont écrites. Au terme de ce travail, le développement est réalisé jusqu'à son achèvement. Les tests unitaires s'enchaînent, et ainsi de suite.

Avant d'expliquer l'appellation cascade, intéressons-nous aux différentes étapes prévues par ce modèle.

Idée - démarrage	C'est le point de départ du projet, le moment où les conditions matérielles sont remplies et où la vision du projet est partagée par l'équipe.
Expression des besoins	Après la décision d'enclencher un projet, on refait le tour des utilisateurs pour recueillir plus en détail leurs exigences fonctionnelles. Ces requêtes sont consignées dans un classeur (papier ou tableur), identifiées, et finalement regroupées ou recoupées.
Spécifications	L'ensemble des requêtes est considéré selon différentes approches dictées par le modèle d'analyse : importance, priorité, difficulté technique, risques… Il ressort des spécifications un découpage technico-fonctionnel de l'application.
Développement	Les spécifications sont traduites en code, en projetant le découpage modulaire (composants) sur la plate-forme choisie par l'architecte.

Tests unitaires	Chaque composant est testé indépendamment des autres. Son comportement doit être conforme aux spécifications, dans la plage de fonctionnement prévue.
Tests d'intégration	Les composants sont le plus souvent assemblés sur une plateforme techniquement hétérogène. La lecture d'enregistrements peut provoquer des problèmes de sécurité, de charge, de robustesse (gestion des valeurs NULL par exemple…). Les tests d'intégration s'assurent que l'édifice final est stable dans toutes les occasions prévues par le cahier des charges.
Tests fonctionnels	Toutes les fonctionnalités de l'application sont testées pour s'assurer du respect du cahier des charges (logique, résultat d'exécution, temps de réponse...).
Installation	L'application est déployée sur sa plateforme définitive (cette étape peut être précédée d'une phase d'industrialisation pour les logiciels packagés). Des ajustements de sécurité sont parfois nécessaires, des chaînes de connexion sont à redéfinir, etc.

Ce modèle est nominal ; chacun est libre d'y ajouter d'autres étapes ou de le remanier, c'est une base de travail. Il est aussi linéaire, aucune étape n'est traitée parallèlement à une autre.

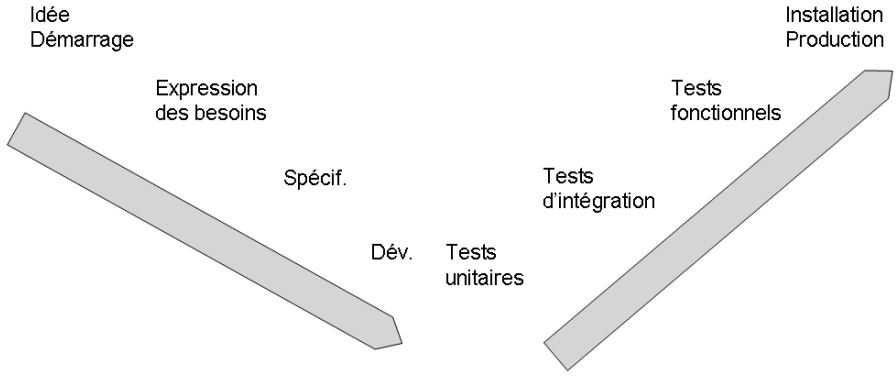
D'où vient l'appellation cascade ? La remontée est difficile, comme pour des poissons qui cherchent à gagner les sources d'une rivière. Les écluses, barrages, ou cascades sont difficilement franchissables. Dans le modèle cascade, un coût unitaire « à la descente » est décuplé s'il faut faire machine arrière. Autrement dit, une erreur de conception se paye de manière exponentielle ! Quelque chose qui a oublié d'être spécifié coûtera 10 fois plus à développer. Si l'oubli est réparé en phases de tests unitaires, c'est 100 fois plus…

2.2 Le modèle en V

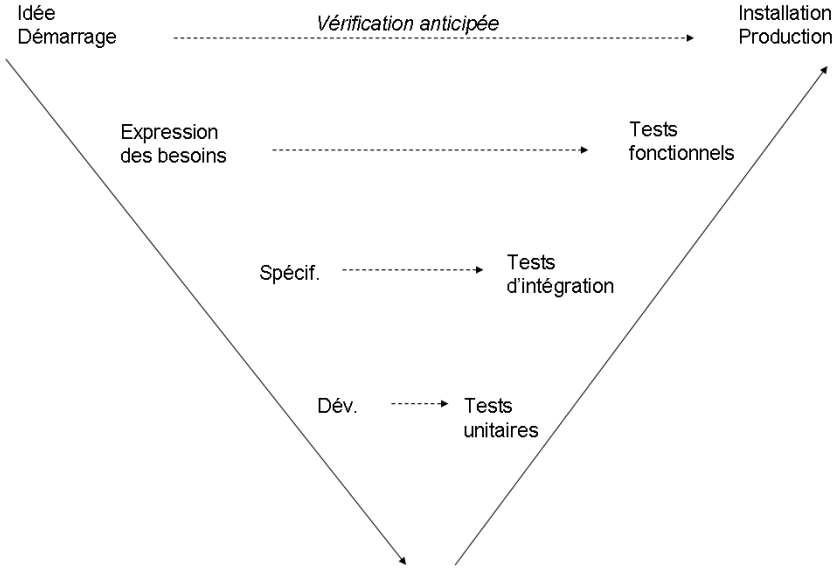
À regarder de plus près le modèle cascade, on remarque une symétrie entre les différentes étapes qui jalonnent le processus. Cette symétrie est justement l'un des points faibles du modèle, puisque les phases de validation sont finalement extrêmement éloignées des phases de spécifications. Chaque acteur du projet se trouve de ce fait piégé dans un tunnel, il est forcé d'attendre la fin du processus pour vérifier la conformité du produit.

Ainsi, l'utilisateur espère que le produit final répondra à ses attentes. Le concepteur ne peut pas vérifier ses hypothèses avant la réalisation des tests fonctionnels. L'architecte ne peut pas intégrer des composants qui n'existent pas encore… Le développeur n'est pas mieux loti puisqu'il y a de grandes chances, vu le manque de recul et de perspective d'ensemble, que ses réalisations soient à reprendre ou à abandonner.

Au lieu de différer au maximum les tests fonctionnels, le cycle en V les anticipe en déroulant le processus de telle manière que ces tests sont sous-traités, réalisés en parallèle, et finalement appliqués au plus tôt.



La sous-traitance des tests est une nécessité, sans quoi leur objectivité serait discréditée. Le parallélisme vise à vérifier « sur le papier » que la composition coïncide avec les besoins exprimés. De cette façon, on anticipe une erreur flagrante de couverture fonctionnelle et on épargne un travail inutile ou défectueux aux développeurs, voire aux analystes.



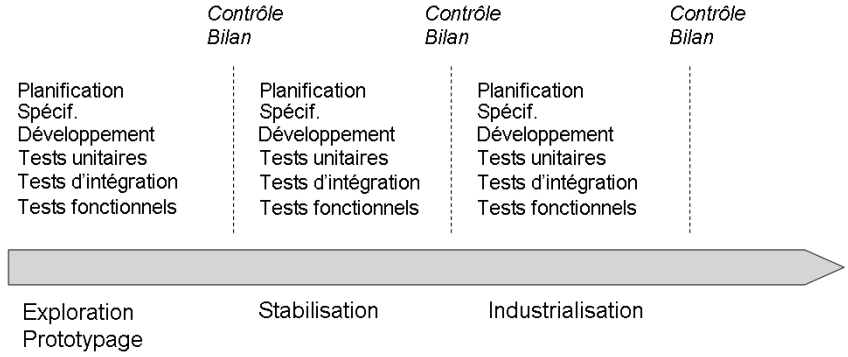
Quels sont les impacts de ce modèle sur la durée du projet ? Le logiciel n'est pas disponible deux fois plus rapidement ! Mais la durée est au contraire rallongée d'environ 30 à 40 %. En effet, les tests sont joués deux fois. La première, de manière fictive, à des fins de contrôle. La seconde, en réel, sur la base d'un logiciel opérationnel.

Le modèle en V constitue donc une amélioration du modèle cascade sur le plan des tests fonctionnels, au prix il est vrai d'une augmentation de la durée du projet.

2.3 Le modèle itératif

À l'inverse du modèle en V, le modèle itératif va anticiper les tests d'intégration, c'est-à-dire s'attacher à diminuer le risque technique. Ce cycle est une succession de cascades dans lesquelles les périmètres fonctionnels et techniques évoluent constamment.

On commence par les parties les plus difficiles pour évacuer le risque d'abandon du projet. Le développement du produit se poursuit par des phases de stabilisation et d'industrialisation.



2.4 Le modèle RAD

RAD est l'acronyme de *Rapid Application Development*. Ce modèle fonctionne sur le principe de l'exploration fonctionnelle, avec une prédominance du maquettage graphique. Le client (ou le demandeur) est fortement impliqué dans les phases de délimitation du périmètre, un peu moins dans celles relevant de la stabilisation et de l'industrialisation.

Pour fonctionner, ce modèle nécessite l'emploi d'une technologie robuste et mature, dotée d'outils de développement de haut niveau. Une plateforme standard est tout à fait à même de supporter le développement et la production de logiciels conçus en mode RAD. Dans ce cas de figure, le risque technique est maîtrisé au travers de l'emploi de technologies simples mais éprouvées.