DÉVELOPPEURS, FAITES-VOUS PLAISIR !

JEAN-FRANÇOIS LÉPINE

This Page Intentionally Left Blank

Table des matières

Le développement est un plaisir

"Le client ne sait jamais ce qu'il veut". Comment-pouvez-vous dans ce cas faire en sorte qu'il soit satisfait ? Etes-vous condamné à délivrer des logiciels, sites web ou applications dont le client ne sera pas réellement satisfait ?

Et vous-même, êtes-vous vraiment obligé de subir les contraintes fonctionnelles, de brider vos compétences techniques ? Après tout, le métier de développeur est passionnant, et mérite qu'on y prenne le maximum de plaisir ! Et c'est tellement décevant d'entendre cette fameuse phrase : "ce n'est pas ce que je voulais"...

Il m'arrive de poser cette question aux personnes que je rencontre : "avez-vous, au moins une fois, participé à un projet informatique où le projet a été livré à l'heure, et où, non seulement le client, mais aussi le développeur, ont été pleinement satisfaits?" . A l'heure où j'écris ces lignes, seule une personne m'a répondu "oui".

Quoi ?! Une seule personne ? Mais pourtant tous les projets devraient se passer comme ça. Tous les projets devraient être livrés à l'heure, devraient être intéressants, enrichissants... Toutes les personnes qui travaillent sur un projet devraient y prendre plaisir. Sinon, à quoi bon travailler ?

L'émergence des méthodes agiles, ces dernières années, a permis de remettre l'humain, et ses valeurs, au centre des projets. Je vous propose de découvrir ici l'autre versant, la face technique, des méthodes agiles : le Développement piloté par le Comportement.

This Page Intentionally Left Blank

Chapitre 1

Le client ne sait pas ce qu'il veut. Sauf si... vous communiquez.

1.1 Le client doit vous fournir sa Vision

J'ai souvent été confronté, en tant que développeur, et sur des projets de toutes tailles (très grosses applications financières, petits intranet, sites web...) à cette situation : des clients étaient incapables de me décrire ce qu'ils voulaient que je réalise pour eux.

A chaque fois, il m'a fallu moi-même interpréter leurs souhaits, à partir de captures d'écrans de sites concurrents, de discussions interminables... Et pour un résultat pas toujours très heureux.

A quoi la faute ? Au client ? Pas forcément : après tout, ce n'est pas parce qu'on est patron d'entreprise ou que l'on a un besoin bien précis que l'on sait comment résoudre ce besoin, ou l'exprimer clairement.

Est-ce la faute du développeur, qui semble incapable de comprendre ce qu'on lui demande ? Personnellement j'ai toujours essayé de faire de mon mieux, même si je suis conscient de ne pas toujours été à la hauteur. Et je crois que la plupart des développeurs fait de même.

Alors ? Et si personne n'était en tort ? En réalité, la plupart des projets informatiques échouent car ils ne sont pas motivés par une Vision. Certes on veut délivrer un super site web, avec plein de fonctionnalités qui vont tuer la concurrence, on veut faire mieux que son voisin... Mais on oublie l'essentiel : un projet doit voir un but.

Quel est ce but ? Peu importe : changer le monde, rendre service à une catégorie de personne, se faire de l'argent... Tout est bon à prendre, du moment que cet objectif, cette Vision, reste le seul et unique maître du projet.

On le comprend bien alors : ce n'est pas parce qu'un concurrent propose un service qu'il faut le recopier. Non, on propose un nouveau service parce qu'il sert la Vision du projet.

Vous l'aurez compris : la Vision est essentielle, et c'est justement le rôle de votre client de la fournir. Sans Vision, le projet est condamné à être un échec, ou au mieux une semi-réussite.

Votre client doit impérativement vous fournir cette Vision. S'il en est incapable, insistez pour qu'il soit aidé : faites lui lire le premier tome de ce livre, faites le coacher par une société spécialisée... Sans cela, vous ne pourrez pas travailler efficacement avec lui et ne prendrez pas autant de plaisir que vous le mériter dans votre travail.

> La plupart des projets informatiques échouent car ils ne sont pas motivés par une Vision.

1.2 Votre Client ne parle pas la même langue que vous

Vous l'avez vécu : votre client et vous ne vous comprenez pas toujours. Pour vous, un "réseau social" c'est une plate-forme communautaire, avec des amis, relations, des groupes, des flux de messages... Bref c'est Facebook.

Pour votre client, un "réseau social" c'est (par exemple) un espace où des employés font des demandes de documents, découvrent les actualités de l'entreprises, échangent des informations sur les commandes... Bref, c'est un intranet.

Attendez... Mais comment donc voulez-vous réussir à satisfaire votre client si vous ne vous n'employez pas le même vocabulaire sur des choses si fondamentales ?

Ajoutez à cela que votre client va toujours avoir tendance à faire deux choses : + employer des acronymes, sigles et autre vocabulaire fonctionnel + employer des termes techniques ("base de données", "formulaire", "sauvegarder") sans savoir précisément ce qu'ils signifient

La première chose à faire lorsque vous démarrez un projet avec un client, c'est donc de vous créer un vocabulaire commun. Il va falloir inventer une nouvelle langue, qui ne laisse plus la place aux ambiguïtés, où chaque mot n'a qu'une seule signification.

J'ai une mauvaise nouvelle pour vous : ce nouveau vocabulaire, c'est à vous de l'apprendre, pas à votre client. Cela vous évitera des débats interminables et vous permettra de mieux comprendre le besoin du Client.

Si votre client, lorsqu'il dit "bus", parle en réalité d'un véhicule avec des ailes et que vous retrouvez dans un aéroport, ne lui dites pas qu'il a tort. Non, désormais, lorsque vous parlerez avec lui, vous emploierez le mot "bus" pour désigner ce que LUI entend par "bus".

Bien entendu, ce vocabulaire commun, vous devez le constituer ensemble. Si votre client parle de "bus", et que c'est important dans le projet, discutez-en ensemble, et mettez noir sur blanc une définition simple de ce qu'il entend par là.

Faites de même pour chaque expression que votre client emploie régulièrement. De cette manière vous n'aurez plus aucune ambiguïté dans votre quotidien.

C'est le postulat de base du Développement piloté par le Comportement : vous devez élaborer avec votre client une Langue Commune.

1.3 Engagez-vous à livrer régulièrement

Quoi de plus frustrant, après de longues semaines de travail acharné, d'entendre un si triste "Mais ce n'est pas du tout ce que j'avais demandé" ?

Et même un simple "ne pourrait-on pas juste changer..." peut s'avérer totalement démoralisant, compte-tenu de tout ce qu'il faudra refaire techniquement pour gérer cette demande, et sachant surtout combien cela aurait été simple si seulement on vous avait prévenu quelques semaines à l'avance.

Comment éviter d'être touché par ce malheureux constat d'échec, ou de semi-réussite ?

La réponse en théorie simple : il suffit de livrer plus régulièrement au client, de sorte qu'il puisse rapidement s'apercevoir le plus tôt possible que le projet n'est pas parti dans la direction à laquelle il s'attendait.

Cette démarche est celle des méthodes agiles, particulièrement de Scrum. Le principe est le suivant : plutôt que de livrer votre projet une bonne fois pour toute à la fin, vous vous engagez à livrer des bouts de fonctionnalités très régulièrement (une fois toutes les deux semaines par exemple).

C'est ce qu'on appelle le développement par itérations, par opposition au cycle en V. Vous développez grossièrement une fonctionnalité, puis vous l'affinez, l'affinez encore si besoin, jusqu'à arriver au résultat final. De cette manière le client peut réorienter le projet à chaque itération sans que n'en soyez affecté négativement.

Imaginez, par exemple, que votre travail consiste à peindre le célèbre tableau La Joconde. Avec la méthodologie classique, votre devriez travailler de manière à finir chaque fonctionnalité de bout en bout, puis, une fois qu'elle est entièrement terminée, passer à la suivante.

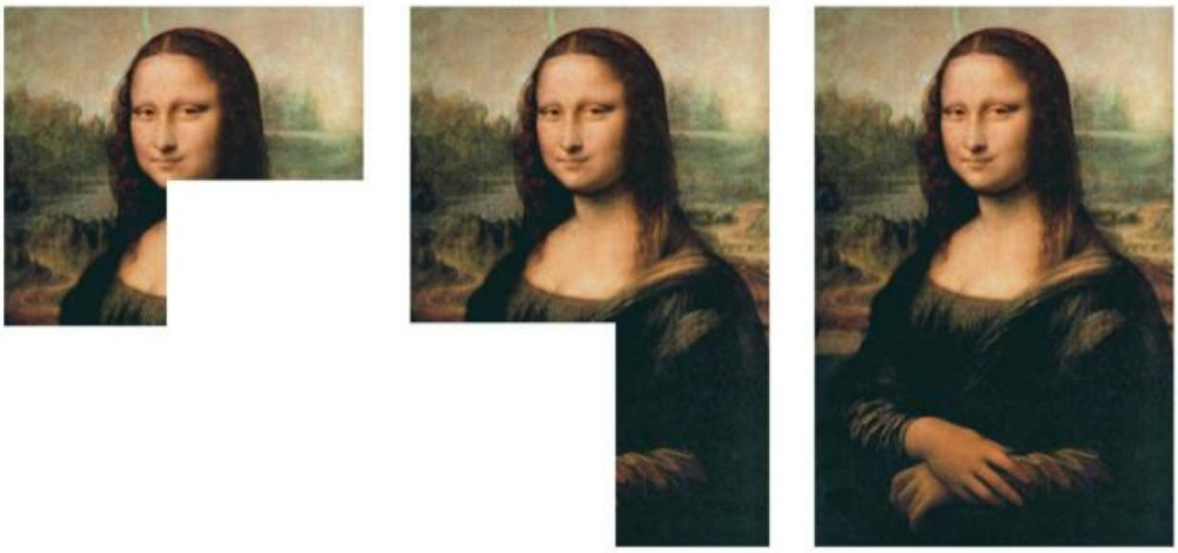

Figure 1.1 Méthodologie classique : le travail et découpé en lots, chaque lot est totalement réalisé avant de passer à la suite. Le changement fonctionnel en cours de route est difficile.

Avec les méthodes agiles la démarche est inverse : vous esquissez d'abord les traits de ce que vous avez à développer, afin de permettre au client d'avoir un aperçu du produit final et de bénéficier rapidement du feedback des utilisateurs, puis vous affinez, selon la priorité de chaque fonctionnalité : une fonctionnalité plus complète par là, un autre lors de l'itération suivante…

Figure 1.2 Méthodologie agile : le tableau s'affine petit à petit. Le changement fonctionnel est facile, même en cours de route

Bien entendu, cela ne vas pas sans un changement des méthodologies de travail : il va falloir découper les fonctionnalités pour faire en sorte qu'elle puisse être développées rapidement, il faut optimiser les recettes (pourquoi ne pas en profiter pour utiliser des tests automatisés, comme des tests unitaires ?)...

Mais, croyez-moi, ces efforts valent la peine : non seulement les relations avec votre client / patron seront de plus en plus saines (le projet devient totalement transparent pour tous), mais en plus vous arriverez vite à livrer du code fonctionnel régulièrement.

Et ô combien cela fait plaisir de voir que l'on avance ! Cela vous permettra même de gérer plus facilement les demandes de changements fonctionnels : il est toujours plus facile de revenir sur deux semaines de travail que sur trois mois, et c'est bien moins démoralisant.

Je vous conseille même de faire quelque chose qui peut sembler assez étrange : à chaque fois que vous livrerez un lot fonctionnel, n'hésitez pas à présenter, pendant une heure ou deux, le fruit de votre travail à votre client. Oui, comme un commercial ! Prenez même un vidéo-projecteur.

Calez systématiquement une réunion à chaque fin d'itération, et prenez la parole : montrez ce que vous avez fait, et soyez-en fier ! Après tout, si vous avez donné le meilleur de vous même, il est légitime que tout le monde sache de quoi il était question.

> Prenez la parole ! N'entendez plus jamais le fameux "Ce n'est pas ce que j'avais demandé".

1.4 Adoptez le point de vue de l'utilisateur final

Chaque projet informatique, qu'il s'agisse d'un site web, d'un intranet, d'une application... tout projet dessert un objectif : rendre service à quelqu'un. Il faut vous mettre dans la peau de cette personne.

Lorsque vous développez une boutique eCommerce, le service est rendu à un potentiel acheteur; lorsque vous développez un intranet, ce sont les employés qui utiliseront l'intranet qui sont les bénéficiaires du service.

Dans les deux cas, vous aurez besoin de comprendre comment le Produit va être utilisé. Finalement, à quoi sert cette fonctionnalité ? Pourquoi est-elle utile ? Comment va elle rendre service ? Va t-elle faciliter la vie de l'utilisateur ? Va t-elle lui permettre de réaliser quelque chose de nouveau ? Lui faire gagner du temps ? Bref, quel est le bénéfice métier de la fonctionnalité que je vais développer ?

Pourquoi se poser ces questions ? Tout simplement pour prendre plus de plaisir dans votre travail, et pour être plus efficace. Tant que ça ? Oui oui...

Après tout, si vous connaissez les personnes qui vont utiliser ce que vous êtes en train de développer, vous pouvez discuter avec elles, adopter leur vocabulaire, et donc profitez directement de leur feedback. Petit à petit, ce feedback sera de plus en plus positif ; ce sera alors de plus en plus agréable de voir que les gens sont contents de ce que vous leurs offrez.

Cela vous permettra aussi de quitter à l'occasion de casquette de développeur pour, pourquoi pas, proposer des améliorations, discuter de l'orientation du projet... Si vous savez comment va être utilisé le Produit, qu'est-ce qui vous empêche d'essayer de l'améliorer ?

Changer de casquette peut parfois être difficile. Vous pouvez très bien travailler sur des projets dont vous n'avez rien à faire ; c'est triste, mais ça arrive fréquemment. Il arrive de devoir travailler sur un projet pour des besoins alimentaires. C'est légitime. Mais autant faire en sorte que même ces projets, alimentaires, vous apportent de la satisfaction humaine. Adopter la vision de l'utilisateur final vous obligera toujours à discuter : discuter avec votre client, les utilisateurs finaux, vos collègues...

J'ai il y a quelques temps lancé un sondage auprès des développeurs de la société dans laquelle je travaille. Une des questions était à peu près la suivante : "Prenez-vous plaisir dans votre travail?". Près d'un quart des réponses était négatif. Quel dommage !

N'oubliez pas que vous travaillez dans un monde d'humains : plus vous interagirez avec eux, plus vous fournirez un travail qui leur procurera satisfaction, plus vous même éprouverez du plaisir à travailler. Le métier de développeur offre une chance unique : il permet de prendre plaisir en travaillant ; autant en prendre le maximum !

Changez de casquette pour prendre le maximum de plaisir dans votre métier de développeur.

This Page Intentionally Left Blank

Chapitre 2

Votre code doit refléter le Besoin fonctionnel

2.1 La programmation par Contrat

2.2 Le Domain Driven Design

2.3 Coder une règle métier efficacement : le Pattern Specification

2.4 Représenter une information : le Pattern Object-Value

2.5 Tout objet a une identité : le Pattern Entity

2.6 Manipulez les objets métiers : le Pattern Repository

Superposez le besoin métier à votre code source : le pattern Service

This Page Intentionally Left Blank

Chapitre 3

Comprenez (enfin!) ce que votre client vous demande

3.1 Le besoin fonctionnel changera. Et c’est normal !

3.2 Facilitez la communication, acceptez le changement

3.3 Le besoin doit être exprimé par des Fonctionnalités

3.4 Chaque Fonctionnalité peut être découpé en Scénarios

Demandez (exigez) des exemples précis

This Page Intentionally Left Blank

Chapitre 4

Automatisez votre recette

4.1 Installez et utilisez Behat en PHP

4.2 Traduire une Fonctionnalité en code source

4.3 Testez dans un vrai navigateur

4.4 Organisez vos Contextes de tests

Réutilisez vos précédents tests

This Page Intentionally Left Blank

Chapitre 5

Optimisez vos tests fonctionnels

5.1 Ne misez pas sur l'Interface graphique

5.2 Créez une couche d'isolation de l'IHM

5.3 Exploitez les compte-rendus de tests (html, xml, txt)

5.4 Anticipez les problèmes

Ne jetez pas le "duck typing" : tout ne pourra pas être testé