# DÉVELOPPEURS, FAITES-VOUS PLAISIR !

JEAN-FRANÇOIS LÉPINE

This Page Intentionally Left Blank

# Table des matières

**Communiquez avec vos développeurs ..... 5**

Chapitre 1 **Le besoin métier. ..... 9**

**1.1** Ayez une vision de votre produit ..... 9

**1.2** Le bon produit est celui qui répond à votre vision ..... 10

**1.3** Vous attendez des résultats métiers ..... 11

**1.4** Re-priorisez souvent, changez souvent ..... 12

**1.5** Gérez votre besoin de changements ..... 13

Chapitre 2 **Communiquez ..... 17**

**2.1** Adoptez une Langue Commune ..... 17

**2.2** Laissez la technique aux développeurs ..... 18

**2.3** Racontez votre produit ..... 19

**2.4** Identifiez le Domaine fonctionnel ..... 20

**2.5** Faites-vous interviewer ..... 21

**2.6** Les tâches ne servent à rien ..... 22

**Développeurs, faites-vous plaisir ! ..... 25**

Chapitre 1 **Le client ne sait pas ce qu'il veut. Sauf si... vous communiquez. ..... 29**

**1.1** Le client doit vous fournir sa Vision ..... 29

**1.2** Votre Client ne parle pas la même langue que vous ..... 30

**1.3** Engagez-vous à livrer régulièrement ..... 31

**1.4** Adoptez le point de vue de l'utilisateur final ..... 34

Chapitre 2 **Votre code doit refléter le Besoin fonctionnel ..... 37**

**2.1** La programmation par Contrat ..... 37

**2.2** Le Domain Driven Design ..... 37

**2.3** Coder une règle métier efficacement : le Pattern Specification ..... 37

**2.4** Représenter une information : le Pattern Object-Value ..... 37

**2.5** Tout objet a une identité : le Pattern Entity ..... 37

**2.6** Manipulez les objets métiers : le Pattern Repository ..... 37

**2.7** Superposez le besoin métier à votre code source : le pattern Service ..... 37

Chapitre 3 **Comprenez (enfin!) ce que votre client vous demande ..... 39**

**3.1** Le besoin fonctionnel changera. Et c'est normal ! ..... 39
**3.2** Facilitez la communication, acceptez le changement ..... 39
**3.3** Le besoin doit être exprimé par des Fonctionnalités ..... 39
**3.4** Chaque Fonctionnalité peut être découpé en Scénarios ..... 39
**3.5** Demandez (exigez) des exemples précis ..... 39

Chapitre 4 **Automatisez votre recette. . . . . 41**

**4.1** Installez et utilisez Behat en PHP ..... 41
**4.2** Traduire une Fonctionnalité en code source ..... 41
**4.3** Testez dans un vrai navigateur ..... 41
**4.4** Organisez vos Contextes de tests ..... 41
**4.5** Réutilisez vos précédents tests ..... 41

Chapitre 5 **Optimisez vos tests fonctionnels. . . . . 43**

**5.1** Ne misez pas sur l'Interface graphique ..... 43
**5.2** Créez une couche d'isolation de l'IHM ..... 43
**5.3** Exploitez les compte-rendus de tests (html, xml, txt) ..... 43
**5.4** Anticipez les problèmes ..... 43
**5.5** Ne jetez pas le "duck typing" : tout ne pourra pas être testé ..... 43

# Communiquez avec vos développeurs

This Page Intentionally Left Blank

# Mettez toutes les chances de votre côté

Imaginez : voilà un bon père de famille qui se décide, enfin, a changer sa vieille voiture. Bien décidé à investir dans du neuf, il se rend chez le concessionnaire du coin, et explique ce qu'il veut… oh, rien de bien compliqué : il doit pouvoir aller faire ses courses, partir en vacances de temps en temps, et, ah si, aller au travail tous les jours. "Pas de problème, je vous livre votre nouvelle voiture dans deux semaines" répond le concessionnaire.

Trois semaines plus tard ("ah oui, désolé il y a eu un peu de retard, mais c'est normal" le rassure le concessionnaire), une petite surprise attend notre père de famille : ce n'est pas la voiture familiale qu'il espérait, mais une camionnette, 12m cube, 45 000 € ("on a rencontré des difficultés imprévues"), 2m10 de haut (la porte de son garage mesure 1m90, mais le concessionnaire est convaincu que "c'est quand même plus pratique d'avoir une bonne hauteur de coffre. Ah, et je vous ai même ajouté un volant de course, le top du top !").

Pas très agréable, non ? Ce n'est pas du tout ce que notre père de famille souhaitait ! Pourtant… Pourtant il peut faire ses courses, partir en vacances… Tout ce qu'il a demandé au concessionnaire est là. Le concessionnaire était plein de bonne volonté, il a fait son maximum pour satisfaire son client, a fait de son mieux, l'a écouté, a tenté d'y mettre du sien et d'ajouter des options qu'il pense indispensables ou confortables. On ne va pas l'en blâmer !

Imaginez ça maintenant : **ce père (ou mère) de famille, c'est VOUS** ! Oui, c'est bel et bien vous, vous qui souhaitez concevoir un produit informatique, un logiciel, un intranet, une application mobile… Vous voilà parti avec une superbe idée, rentable, révolutionnaire ! Et vous vous retrouvez finalement avec un produit coûteux, livré en retard, voire parfois totalement inadapté au marché !

Allez-vous vous en prendre au développeur ? A la société qui a conçu votre produit ? Au prestataire de service ? N'est-ce pas plutôt un problème plus général de communication ? Après tout, chacun est convaincu d'avoir fait de son mieux… N'y avait-il pas un moyen de faire mieux ? De mieux faire comprendre votre besoin ? De mieux vous préparer aux éventuels retard, de les limiter ? De réorienter votre demande en cours de route ?

C'est justement l'objet de ce livre. Bon, nous n'allons certainement pas vous offrir un moyen magique de réussir un projet informatique. Non, ça n'existe pas, il n'y a jamais de *silver bullet*. Rien n'est magique : **c'est à vous, et à vous seul, de faire réussir vos projets informatiques**. Par contre, ce livre peut peut-être vous donner quelques pistes de réflexion, afin de mettre plus de chances de votre côté.

This Page Intentionally Left Blank

Chapitre 1

# Le besoin métier

## 1.1 Ayez une vision de votre produit

Votre idée est peut-être géniale, révolutionnaire, utile ou juste rentable (aucun mal à cela). Elle va peut-être changer le monde, ou plus modestement faciliter le quotidien de pas mal de monde.

Pensez au premier téléphone portable : en voilà un produit qui a changé bien des choses; impossible aujourd'hui d'imaginer vivre sans : sans accès à votre boîte mail, sans appareil photo, sans sms, sans carnet de contacts... Mais, attendez, le premier téléphone, il ne faisait pas tout ça ! Non, rappelez-vous, un téléphone portable, à la base, ça sert à pouvoir téléphoner n'importe où.

Tout le reste, c'est utile, pratique, génial, sans doute indispensable au quotidien, bref, c'est tout autant révolutionnaire. Mais si on doit définir le téléphone portable en quelques mots, c'est "juste" un moyen d'échanger vocalement et instantanément de l'information entre deux intervenants, où qu'ils soient dans le monde.

Votre produit est pareil : **il doit pouvoir se définir en une courte phrase**, et en fait ne sert qu'à une chose. Si si, tout le reste est utile commercialement, esthétiquement, fonctionnellement... Mais votre produit possède un coeur, et la première chose à faire est de l'identifier.

Cela vous permettra de savoir vraiment ce dont vous avez besoin. Si vous demandez au père de famille. dont nous parlions plus haut, de décrire sa voiture, il va peut-être dire qu'il veut un break Citroen de 120 chevaux gris métallisé, avec 5 portes. Il sait ce qu'il veut... Pourtant il oublie l'essentiel : la voiture doit lui permettre de se rendre d'un point A à un point B.

Oui, c'est évident : une voiture permet de se déplacer. Mais pourtant, évident ou pas, une voiture, ce n'est rien d'autre, c'est ce qui reste une fois qu'on a enlevé tout le reste : retirez la climatisation, la peinture métallisée, le cuir… Il nous reste le coeur de la voiture, ni plus ni moyen qu'un moyen de locomotion.

Allez même plus loin, et imaginez une voiture sans moteur ; est-ce toujours une voiture ? Oui, sans aucun doute, le moteur n'est qu'un moyen pour cette fin, ultime, de permettre aux gens de se déplacer : une voiture à cheval n'en reste pas moins une voiture. Certes, elle n'est peut-être pas optimale, pas rapide, pas jolie, tout ce que vous voulez… Mais C'EST UNE VOITURE. Et peut-être moins cher en plus !

Pour votre logiciel, votre intranet, votre application mobile, c'est pareil : **identifiez le coeur de votre produit, ce qui fait son essence même, ce sans quoi votre produit n'est plus votre produit, et là vous aurez de bonnes bases pour commencer**.

Comme le dit Liz Keogh, **chaque projet informatique doit poursuivre une vision**. Cette vision répond souvent à un besoin économique (réduire des coûts, améliorer la productivité), voire parfois à des souhaits différents (améliorer le quotidien des gens, changer les mentalités…). Dans tous les cas, tout ce que vous allez mettre dans votre produit doit poursuivre cette vision.

> Tout ce que vous mettez dans votre produit doit poursuivre une vision

## 1.2 Le bon produit est celui qui répond à votre vision

Quand ils ont conçu Basecamp, un logiciel de gestion de projets aujourd'hui reconnu et plébiscité, Jason Fried et David Heinemer avaient une vision : faciliter la vie des différents intervenants lors de leurs créations de projets. Cette vision est visible partout dans Basecamp : simple, facile d'accès, rapide et ergonomique, il facilite réellement la vie de celui qui l'utilise.

Ils auraient pu choisir de concevoir un logiciel riche, multi-fonction et polyvalent ; ils ne l'ont pas fait. Pourquoi ? Tout simplement parce que cela n'aurait pas servi leur vision. On pourrait vouloir un CRM dans Basecamp, ou un intranet, ou tout ce que vous voulez. Tous ces modules pourraient être forts agréables, utiles et d'excellents arguments commerciaux. Mais ils ne servent pas la vision du produit. Et c'est justement pour ça que Basecamp plaît autant : il ne fait qu'une chose, mais il le fait bien !

Pensez également à ceci : certes Basecamp n'est pas riche fonctionnellement, mais

- Il est largement suffisant dans 90% des cas
- Il a été d'autant moins coûteux à concevoir

- Il a été d'autant plus rapide à développer
- Chaque fonctionnalité a pu être testée en profondeur et est fiable
- Il ne souffre d'aucune complexité inutile et est rapide
- Les équipes de développement se sont focalisés sur un domaine fonctionnel simple
- Tout changement ou évolution est simple à mettre en place

Tout cela serait impossible si Jason Fried et David Heinemer ne s'étaient pas focalisés sur leur vision du produit et n'avaient pas refusé systématiquement d'ajouter des fonctionnalités qui ne répondent pas à cette vision initiale.

Rappelez-vous : vous êtes le seul maître à bord. C'est à vous de vous servir de votre vision du produit pour limiter le superflu et focaliser toute l'énergie disponible (développeurs, serveurs, graphistes, ergonomes…) sur ce qui répond à cette vision, autrement dit sur ce qui est vraiment utile.

## 1.3 Vous attendez des résultats métiers

Vous avez probablement souvent entendu cette expression : ce qui compte, ce sont les résultats !

Bon, je ne vais pas vous dire que la fin justifie les moyens, pas du tout. Non, ce que je veux dire, c'est qu'en investissant du temps et de l'argent dans votre projet, vous ne devez jamais oublier que ce qui compte ce sont les résultats métiers (ou fonctionnels) de votre projet. Ces résultats qui justement servent votre vision.

Pensez-y : chaque fonctionnalité, chaque élément d'interface, chaque bouton, chaque champ de saisie… tout cela doit fournir un bénéfice à l'utilisateur de votre produit !

Ca peut paraître évident… Mais alors pourquoi consacrer autant d'énergie sur des choses inutiles fonctionnellement ? Si si, ça arrive tout le temps ! En tant que développeur j'ai souvent eu l'occasion de le voir : une grosse partie de mon temps était consacré à du superflu (gestion des profils utilisateurs hyper poussée, interfaces graphique à la iGoogle, optimisation prématurée des performances…), alors même que les fonctionnalités métier n'étaient pas encore finies, voire même pas spécifiées !

L'énergie dépensée sur un projet doit être corrélée aux bénéfices que votre produit en tirera pour satisfaire votre vision. Si, alors que les fonctionnalités métiers ne sont pas pleinement finies, testées, fiables, éprouvées et ouvertes aux changements, l'énergie dépensée ne sert qu'à "vendre du rêve", vous risquez la catastrophe.

Concevez votre produit, non pas pour le vendre, mais pour qu'il réponde à un but (votre vision). Si vous y arrivez, vous aurez à ce moment là en main les meilleurs atouts pour le vendre.

\

Le bénéfice sera d'autant plus grand que, au lieu de perdre du temps sur des choses moins utiles, vos développeurs, prestataires ou la société qui a en charge votre projet vont mieux comprendre votre besoin, sans se perdre en détails inutiles pour le moment.

En vous focalisant sur les fonctionnalités métiers, vous allez aussi pouvoir plus rapidement vous confronter au feedback de vos futurs utilisateurs. Les changements fonctionnels de votre produit seront ainsi réalisés plus tôt, et donc moins coûteux.

> Ne concevez pas votre produit pour le vendre, mais pour qu'il réponde à un but. Vous aurez alors en main les meilleurs atouts pour le vendre

## 1.4 Re-priorisez souvent, changez souvent

Vous l'avez vu : votre produit doit se focaliser sur ce qui apporte à un métier, sur ce qui répond à votre vision.

C'est bien beau, mais on en est tous conscients : un aspect métier peut être essentiel aujourd'hui, mais totalement inutile ou différent demain ! Comment gérer ces changements dans votre projet ?

Vous vous en doutez, de nombreuses personnes ont tenté de répondre à cette question. Parmi elles, les (nombreux) auteurs du Manifeste Agile. Voici certains des principes de ce Manifeste tel qu'ils sont présentés sur http://agilemanifesto.org, qui nous intéressent particulièrement :

> "Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
>
> Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agiles exploitent le changement pour donner un avantage compétitif au client.
>
> Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts."

C'est assez clair : si l'on suit ces principes, largement reconnus aujourd'hui dans le monde de l'édition logicielle, **le produit doit être confronté le plus tôt possible au monde réel pour obtenir un feedback régulier des potentiels utilisateurs**.

Ce feedback régulier va vous permettre de réorienter votre produit afin de supprimer, modifier ou ajouter des comportements dans celui-ci. Vous avez donc besoin de pouvoir re-prioriser certains éléments, de changer en cours de route le parcours de développement de votre site, logiciel, application ou intranet.

En un mot : vous avez besoin de confronter votre produit aux utilisateurs. Plus vous le confronterez souvent, plus votre produit sera compétitif, et plus vous aurez besoin de faire évoluer votre produit.

Vous l'avez compris : vous avez besoin de changements tout au long de la phase de conception de votre produit. Cependant, vous l'avez sans doute déjà vécu, **tout changement dans un produit informatique est généralement :**

- long
- coûteux
- source d'erreurs et de bugs
- difficile et démoralisant pour les équipes

L'idée pour résoudre ces difficultés est assez simple, mais parfois difficile à mettre en place, comme nous allons le voir.

## 1.5 Gérez votre besoin de changements

L'idée du Manifeste Agile pour gérer le changement peut paraître simple parce qu'il "suffit" (entre autres) de modifier les cycles de développement de votre produit. Généralement, un produit est conçu d'un seul bloc : on développe un produit non fini pendant une période assez longue, et on livre un produit fini (enfin, ça c'est qu'on espère !) à la fin. Cette méthode implique un effet tunnel assez long et souvent dévastateur.

Qui ne connaît pas le célèbre tableau de Léonard de Vinci : La Joconde ? Que se serait-il passé si Léonard de Vinci devait peindre Mona Lisa comme on travaille généralement sur un projet informatique ? Cela donnerait à peu près ceci :

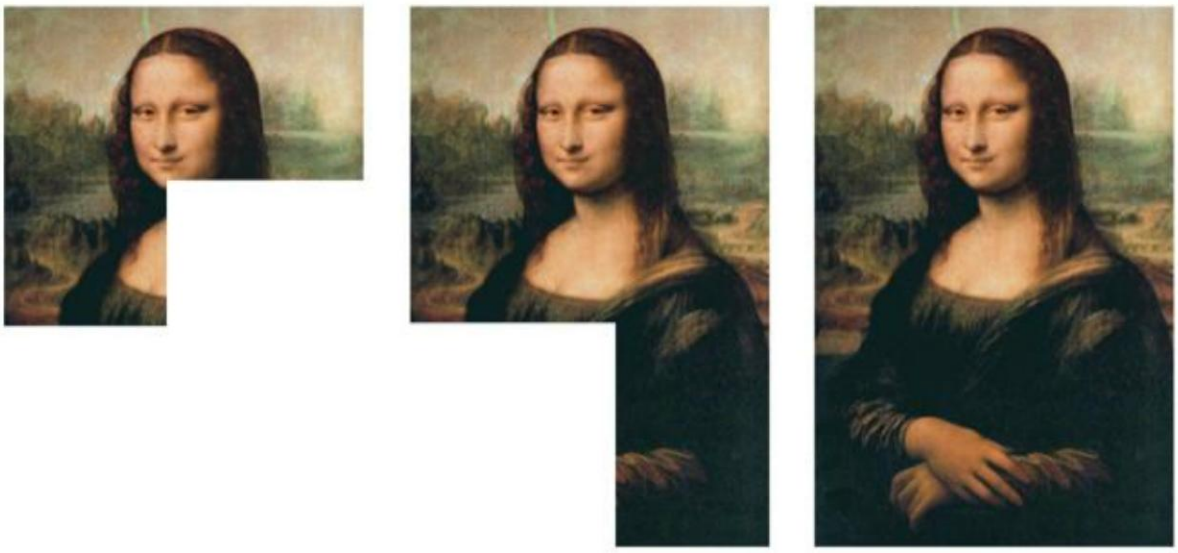

**Figure 1.1** Méthodologie classique : le travail et découpé en lots, chaque lot est totalement réalisé avant de passer à la suite. Le changement fonctionnel en cours de route est difficile.

Le travail est tout de suite très fin, et on ne peut le livrer qu'à la fin… Difficile dans ces conditions d'accepter le changement !

Maintenant examinez la méthode proposée par les fondateurs du Manifeste Agile. Les équipes techniques vont travailler en cycles, itérativement, de manière à livrer régulièrement un produit, certes moins complet, mais exploitable et sur lequel il est possible de fournir un feedback.

Le rythme de ces itérations, et leurs objectifs, sont fixés en accord avec tous (équipes techniques, fonctionnels…), et chacun s'engage à faire le maximum pour livrer un produit exploitable. Voici ce que cela donnerait cette fois-ci pour notre cher Léonard :

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

Bien sûr, un logiciel qui prendra un an à être développé ne sera pas exploitable commercialement au bout de trois itérations de deux semaines ! Par contre vous aurez moyen d'obtenir des premiers retours utilisateurs, et surtout vous allez pouvoir reprioriser et changer certains éléments en cours de route : la forme du visage ne vous convient pas ? Faites la changer lors d'une prochaine itération ! Les couleurs sont trop ternes ? Priorisez un changement des couleurs pour voir ce qu'en pensent vos futurs utilisateurs. Bref, vous avez compris l'idée !

Les principes agiles sont simples et évidents, mais ils peuvent être étrangement contre-intuitifs à mettre en place : souvent, les équipes (techniques et fonctionnels) ont du mal à se détacher de leurs anciennes pratiques, quant bien même elles adhèrent aux concepts agiles.

J'ai à plusieurs reprises eu l'occasion de voir des entreprises adopter des méthodologies agiles trop brusquement, ou d'une manière inadaptée. Les méthodes agiles ne sont pas un remède miracle à tous les maux, il serait faux de croire que du jour au lendemain des équipes peuvent oublier toutes leurs anciennes pratiques et que par magie vous serez livré à temps et avec un produit fini.

Mais le changement vers l'agilité peut être simple et rapide, sous deux conditions :

**vous DEVEZ vous impliquer dans votre projet !** C'est à vous de prioriser, de dire ce qui va, ne va pas, régulièrement.

Vous devez accepter de vous faire aider : **les changements à adopter ne sont pas que méthodologiques, ils sont aussi conceptuels**. De nombreuses sociétés agiles sauront vous guider et vous conseiller. Et croyez moi, l'investissement en vaut la peine, largement !

Chapitre 2

# Communiquez

## 2.1 Adoptez une Langue Commune

Ce que vous entendez en utilisant un mot peut parfois (souvent !) être totalement différent de ce qu'une autre personne entend pour ce même mot. Et c'est encore pire avec les développeurs !

Gardez cela à l'esprit : les développeurs ne parlent pas la même langue que vous. Lorsque vous parlez "stock", "produit" ou "commande", un développeur pensera "base de données", "enregistrement" et "sauvegarde en base".

La première chose à faire est donc de constituer avec vos équipes une Langue Commune. Peu importe le moyen : une feuille excel, un document word ou un bout de papier feront l'affaire ! L'important est de créer un référentiel commun, souvent désigné par "Ubiquitus Language", c'est-à-dire **un moyen d'utiliser une terminologie dont on sait qu'elle est comprise par tous de la même manière**.

Attention, cet Ubiquitus Language, **cette Langue Commune ne doit PAS être élaborée unilatéralement !** Instaurez un dialogue entre toutes les parties prenantes du projet, jusqu'à déterminer des termes plus importants que les autres. Identifiez les, puis, en commun, définissez-les. Ne l'oubliez pas : une définition que vous aurez mis en place tout seul ne voudra peut-être rien dire pour un développeur.

Dernière chose : ce référentiel commun ne doit porter que sur le domaine fonctionnel. Il est inutile de créer un référentiel technique (du moins pas en dehors des équipes techniques) : ce qui compte ce n'est pas de faire du développement logiciel ; non, ce qui compte c'est votre produit, vos bénéfices fonctionnels et votre vision !

Pour synthétiser cette Langue Commune en quelques mots, elle permet :

- d'éviter un travail de traduction entre les tâches techniques et les besoins fonctionnels (le jargon métier)
- que tout le monde "parle" votre projet comme vous le "pensez"
- d'éviter de vous retrouver avec des surprises idiotes, dues à des incompréhensions sémantiques ou des ambiguïtés.

> les développeurs ne parlent pas la même langue que vous"

## 2.2 Laissez la technique aux développeurs

Laissez-moi vous rassurez tout de suite : **non, vous n'avez pas besoin de savoir comment fonctionne techniquement votre produit**. Non, vous n'avez pas à comprendre la terminologie des développeurs.

Mieux ! Vous ne devez SURTOUT PAS comprendre ce que font les développeurs.

Ça fait maintenant quelques années que je conduis une voiture. Je sais passer mes vitesses, je comprend comment lire mon tableau de bord... Mais je ne sais pas, mais alors pas du tout, comment fonctionne ma voiture. Et croyez-moi, je n'en ai rien à faire !

Je dois vous avouer une chose : quand j'ai une panne, je vais voir mon garagiste et je lui fais confiance. Quand il m'explique le problème, je dis des "ah oui ?" avec un air curieux. Mais je ne comprend rien ! Et tant mieux !

Pourquoi serait-ce différent pour l'informatique ? Mettre les mains "dans la cambouis" risque :

- de brider vos demandes quand vous allez imaginer qu'un point va être techniquement coûteux alors qu'il n'en n'est rien
- de vous faire adopter une angle de vue différent (celui des
- développeurs) au détriment de votre vision initiale
- de laisser le code dominer le domaine fonctionnel
- de vous faire perdre votre force fonctionnel

Je peux vous le dire : les projets les plus coûteux et les plus en retard sur lesquels j'ai travaillé ont été initié par d'anciens développeurs devenus chefs d'entreprise ou fonctionnels. Naturellement, et on ne peut pas les en blâmer, ils ont tendance à vouloir comprendre l'implémentation technique de leur produit.

Cependant, cela est très coûteux : tout d'abord l'équipe technique doit passer beaucoup de temps à expliquer ce qu'elle fait au lieu d'avancer réellement. Ensuite, elle doit passer du temps également à monter en compétence le fonctionnel. Enfin, et ce n'est pas le moindre, cela introduira un sentiment de défiance entre vous et l'équipe technique, sentiment qui va ralentir et limiter vos possibilités de créer une Langue Commune et freiner l'investissement de vos équipes.

Vous ne devez surtout pas comprendre ce que font les développeurs"

## 2.3 Racontez votre produit

Combien de documents sont passés entre vos mains sans que vous ne les lisiez totalement, partiellement ou pas du tout ? Avouez-le, cela nous est tous arrivé au moins une fois.

Pourquoi le document que vous fournissez à votre prestataire ne serait-il pas justement celui-là même qu'il va survoler, ou mal comprendre ?

Un document word, par exemple, est une excellente base de travail. Vous pouvez y noter vos réflexions, vous en servir comme brouillon... Mais pas comme spécification !

Comprenez : votre pensée est ce qu'elle est : mouvante, changeante, et surtout personnelle. Avec tous les efforts du monde, vous n'arriverez pas à transmettre l'ensemble de votre pensée par écrit. Il subsistera toujours de l'interprétation, des non-dits, de l'implicite... Voire même parfois des contradictions qu'il faudra lever. C'est normal, comme un projet, une pensée vit et change.

Votre vision, vos besoins, vos spécifications... tout cela doit faire l'objet d'un échange oral. Racontez votre produit plutôt que de le décrire. Qu'en pensez-vous ?

**Votre projet doit être le résultat d'une conversation** : impliquez vous, mais impliquez aussi le testeur, le fonctionnel, l'ergonome, le graphiste et le développeur. Racontez leur votre produit, et dialoguez, de telle manière que :

- Ils **s'approprient votre besoin**
- Ils **mettent en évidence certaines incohérences** le plus tôt possible
- Ils vous proposent des changements (vous restez le seul maître de les accepter ou non)
- Ils soient impliqués et aient le sentiment d'être partie prenante de votre projet
- Ils ne perdent pas de temps à traduire votre message

C'est d'ailleurs comme ça que vous allez vous rendre compte que certains points demeurent obscurs, malgré tous vos efforts. Ce sera peut-être que ces points sont plus importants que vous ne l'estimiez, et il sera peut-être utile d'y consacrer plus de temps.

Oubliez donc les spécifications fonctionnelles détaillées (SFD), et autres joyeusetés qui font la joie du développeur quand il doit lire un document de 200 pages pour comprendre un besoin qu'il mettra 20 minutes à implémenter. La documentation est utile, les SFG, SFD, etc. aussi, mais uniquement quand elle est réservée aux cas qui le nécessitent vraiment !

Oubliez cette masse de documentation d'autant plus qu'elle va être un frein à votre besoin de changements : très vite, avec de nombreux documents, vous allez :

ou bien avoir un décalage conséquent entre vos spécifications et votre produit (quid des nouveaux arrivants sur le projet ?),

ou bien devoir passer un temps précieux à mettre à jour vos spécifications à chaque changement. Et ça, croyez-moi, dans la vraie vie ça n'arrive jamais si vous n'avez pas une ressource dédiée à cette tâche, à temps plein !

Vous allez le voir, il existe d'autres moyens de préciser votre besoin. En attendant, oubliez les documents word, pdf, rtf... Et Initiez la conversation.

> Racontez votre produit plutôt que de le spécifier"

## 2.4 Identifiez le Domaine fonctionnel

Vous l'avez vu, vous avez besoin de constituer une Langue Commune avec vos équipes. Vous le savez également, une image vaut souvent mieux qu'un long discours... Pourquoi ne pas concilier les deux ?

Il existe une approche, mise en évidence par Eric Evans, pour faire en sorte le code source (ce qui permet à votre produit de fonctionner) corresponde de près aux fonctionnalités métiers. Cette approche, c'est le Domain Driven Design, ou Conception Pilotée par le Métier.

En général, au fil du temps, les développeurs qui auront travaillé sur votre projet vont mettre en place ce qu'on appelle familièrement des "rustines", des morceaux de code source rapides à mettre en place mais peu maintenables, destinés à gérer vos demandes de changements.

Pire, dès le départ, les développeurs vont se focaliser sur les aspects techniques du projet. Normal me direz-vous ? Non, car s'ils font cela, l'énergie à dépenser pour comprendre les liens entre le code source et le besoin fonctionnel risque d'être très importante, voire tel-

lement importante que votre projet risque de vous mettre sur la paille à la moindre demande de changement.

Le Domain Driven Design propose une approche différente : les développeurs vont se focaliser uniquement sur le besoin métier, et concevoir leur code source autour de ce besoin. Ca peut paraître une évidence, mais c'est loin d'être le cas, croyez-moi ! Le code source sera alors le reflet du besoin métier.

Si vous et vos équipes souhaitez mettre en place cette orientation (du Domain Driven Design), **la première chose dont vous aurez besoin, après une Langue Commune, sera une cartographie du domaine fonctionnel couvert par votre application**.

Rassurez-vous, rien de bien compliqué. Oubliez les logiciels de modélisation, oubliez Visio, oubliez l'UML, Merise (si vous les avez déjà utilisés), voici venue l'ère ... du papier et du crayon !

Listez, puis représentez sur un dessin simple, l'ensemble des grands concepts fonctionnels liés à votre produit (ceux qui sont présents dans votre glossaire, votre Langue Commune). Reliez-les entre eux par de flèches annotées, qui représenteront les différentes interactions de ces concepts entre eux. S'il y a trop de flèches à un endroit, c'est que vous vous attardez trop sur des détails. Si vous pensez que ce ne sont pas des détails, prenez une autre feuille et redémarrez, mais cette fois en zoomant sur la partie concernée.

Cette cartographie, qui doit être simple (en caricaturant un peu, disons que votre neveu de huit ans devrait pouvoir la comprendre s'il connaît votre Langue Commune !), sera le **référentiel du Comportement de votre Produit**.

Ce comportement étant désormais identifié, il sera plus facile pour les équipes d'appréhender les changements et d'en identifier les conséquences.

Cette cartographie vous sera utile à vous aussi : elle va vous permettre d'identifier l'indispensable, l'utile et le pratique dans votre produit. Vos équipes ne doivent commencer à travailler sur l'indispensable. L'utile et le pratique viendront après.

> Prenez un papier et un crayon, puis faites la cartographie fonctionnelle de votre Produit"

## 2.5 Faites-vous interviewer

Mais attention, cette cartographie fonctionnelle, en réalité... ce n’est pas à vous de la faire !

Pourquoi donc cet air surpris sur votre visage ? Si si, je vous assure, ce n’est pas à vous de la réaliser, mais rassurez-vous, vous en serez le moteur.

**Concrètement, vous devez dans un premier temps discuter avec vos développeurs, mais surtout ceux-ci doivent réaliser un travail d'enquête**
vos développeurs doivent vous interviewer.

Ou plutôt ils doivent interviewer le Produit. Après tout… vous êtes le porte-parole de votre produit.

Cet interview, comme une interview classique va consister en différents moments :

- présentation générale d'un aspect fonctionnel
- une question de l'équipe technique sur un point abordé
- votre réponse fonctionnelle
- une nouvelle question de l'équipe technique
- etc.

Attention, l'interview ne doit jamais dériver sur des aspects techniques. Il sera largement temps plus tard de voir comment résoudre techniquement tel ou tel point. Pour l'instant, ne discutez QUE du fonctionnel.

Cette démarche a au moins trois avantages :

- impliquer les équipes techniques sur votre besoin métier
- mettre en évidence des aspects implicites, qui sont évidents pour une personne du métier, mais totalement inconnus des équipes techniques
- vous permettre de prendre du recul sur votre métier et votre besoin en l'expliquant à des néophytes.

## 2.6 Les tâches ne servent à rien

Vous voulez que vos développeurs développent une nouvelle fonctionnalité ? Ne leur donnez pas de tâche à faire ! N'exprimez que votre besoin…

Imaginez un instant cette scène : vous re-voila chez votre concessionnaire automobile, prêt à signer le chèque de votre nouvelle voiture. A votre avis, des deux situations suivantes, laquelle est la plus adaptée :

```
+ Situation 1 :
Client : "J'ai besoin de la voiture le mois prochain. Il vous faudra
donc passer commande auprès du constructeur. Pour cela vous disposez du
logiciel mis à disposition par ce dernier. Entrez vos codes d'accès,
puis saisissez la référence de la voiture, puis cliquez sur "Commander".
```

```
Dès que vous recevez la voiture, faites un bilan complet, puis
appelez-moi."
+ Situation 2 :
Client : "J'ai besoin de ma voiture le mois prochain, je sais que vous
ferez le maximum. Tenez moi simplement régulièrement informé de
l'avancée de la commande..."
```

Alors, à votre avis, quelle situation est la plus adaptée ? La deuxième sans doute, non ? Examinons la Situation 1 : que se passe t-il si le concessionnaire appelle directement le constructeur pour commander votre voiture plutôt que de respecter les tâches qui lui sont assignées…

Quel en sera le résultat ?

- Vous serez livré plus vite
- **Les objectifs seront atteints**
- **Pourtant le concessionnaire n'aura pas accompli les tâches demandées**

Je pense que vous avez compris : peu importe ce que va faire votre concessionnaire, ce qui compte c'est d'être livré à temps. Pire, vous risquez d'embrouiller le concessionnaire ou de le mettre sur des fausses pistes. Avec une liste de tâches à suivre, le concessionnaire ne pourrait pas prendre l'initiative d'appeler directement le constructeur automobile.

Et bien pour le développeur c'est pareil. **Exprimez votre besoin, les délais souhaités, les modalités, mais ne lui dite pas comment réaliser ce besoin**, vous risquez de le mettre sur de fausses pistes.

Attention, les listes de tâches sont utiles bien entendus, mais pas pour communiquer. Vous pouvez avoir votre liste de tâches, les équipes de développement vont peut-être (sans doute !) transformer votre besoin en sous-tâches techniques… Mais en aucun cas vous n'avez besoin de communiquer avec des tâches à faire. Le dialogue ne doit porter que sur votre besoin métier.

> Communiquer en liste de tâches ne sert à rien, le dialogue ne doit porter que sur le besoin métier.

This Page Intentionally Left Blank

# Développeurs, faites-vous plaisir !

This Page Intentionally Left Blank

# 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.

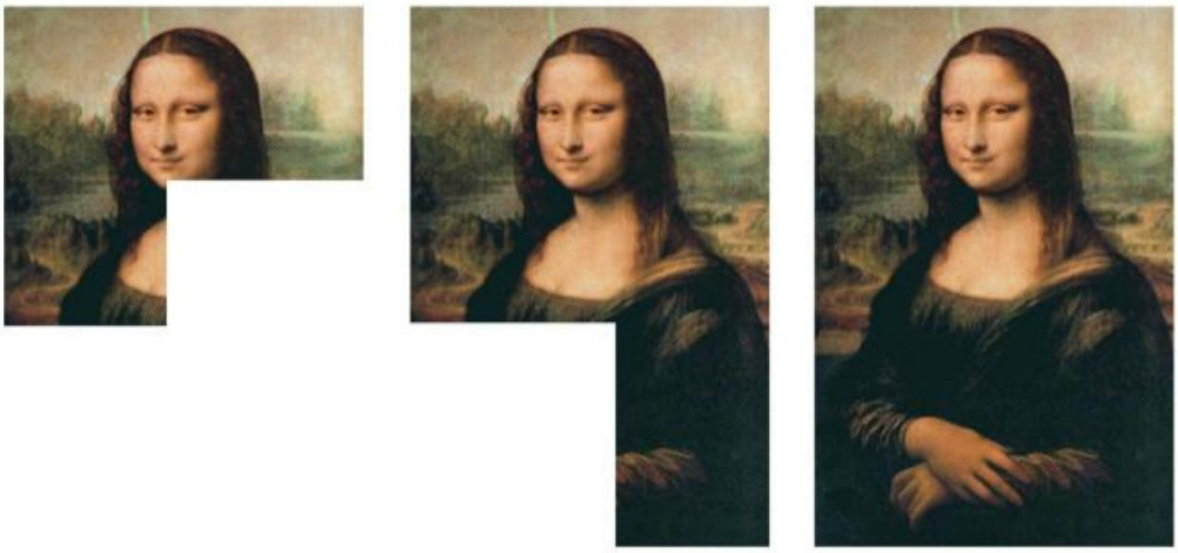

**Figure 1.1** Méthodologie classique : le travail et découpé en lots, chaque lot est totalement réalisé avant de passer à la suite. Le changement fonctionnel en cours de route est difficile.

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