CODER
PROPREMENT

Robert C. Martin
Michael C. Feathers Timothy R. Ottinger
Jeffrey J. Langr Brett L. Schuchert
James W. Grenning Kevin Dean Wampler
Object Mentor Inc.

Traduit de l'anglais (États-Unis) par Hervé Soulard

nouveaux horizons

Publié par Pearson France
2 rue Jean Lantier
75001 Paris

Titre original :
Clean Code: a handbook of agile software craftsmanship

Mise en pages : Hervé Soulard, IDT

Traduit de l'anglais (États-Unis) par Hervé Soulard

Relecture technique : Éric Hébert

ISBN original : 978-0-13-235088-2 Copyright © 2009 Pearson Education, Inc. All rights reserved

Distribution Nouveaux Horizons – ARS, Paris, 2021, pour l'Afrique francophone et Haïti.

Votre avis nous intéresse ! Contactez-nous à arsnh@state.gov.

Nouveaux Horizons est la branche édition d'Africa Regional Services (ARS), qui fait partie du Bureau des affaires africaines du département d'État américain. Les éditions Nouveaux Horizons traduisent et publient en français des livres d'auteurs américains et les commercialisent en Afrique subsaharienne, au Maghreb et en Haïti. Pour connaître nos points de vente ou pour toute autre information, consultez notre site : https://fr.usembassy.gov/fr/ars-paris-fr/livres/nh.

ISBN : 978-2-35745-608-2

Table des matières

Préface

Les pastilles Ga-Jol sont parmi les sucreries préférées des Danois. Celles à la réglisse forte font un parfait pendant à notre climat humide et souvent froid. Le charme de ces pastilles réside notamment dans les dictons sages ou spirituels imprimés sur le rabat de chaque boîte. Ce matin, j'ai acheté un paquet de ces friandises sur lequel il était inscrit cet ancien adage danois :

> *Ærlighed i små ting er ikke nogen lille ting.*

"L'honnêteté dans les petites choses n'est pas une petite chose." Il présageait tout à fait ce que je souhaitais exprimer ici. Les petites choses ont une grande importance. Ce livre traite de sujets modestes dont la valeur est très loin d'être insignifiante.

Dieu est dans les détails, a dit l'architecte Ludwig Mies van der Rohe. Cette déclaration rappelle des arguments contemporains sur le rôle de l'architecture dans le développement de logiciels, plus particulièrement dans le monde agile. Avec Bob (Robert C. Martin), nous avons parfois conversé de manière passionnée sur ce sujet. Mies van der Rohe était attentif à l'utilité et aux aspects intemporels de ce qui sous-tend une bonne architecture. Néanmoins, il a choisi personnellement chaque poignée des portes des maisons qu'il a conçues. Pourquoi ? Tout simplement parce que les détails comptent.

Lors de nos "débats" permanents sur le développement piloté par les tests (TDD, *Test Driven Development*), nous avons découvert, Bob et moi-même, que nous étions d'accord sur le fait que l'architecture logicielle avait une place importante dans le développement, même si notre vision personnelle sur sa signification réelle pouvait être différente. Quoi qu'il en soit, ce pinaillage est relativement peu important car nous convenons que les professionnels responsables passent du temps à réfléchir sur l'objectif d'un projet et à le planifier. La notion de conception pilotée uniquement par les tests et le code, telle qu'elle était envisagée à la fin des années 1990, est désormais obsolète. L'attention portée aux détails est aujourd'hui une preuve de professionnalisme plus que n'importe quelle autre grande vision. Premièrement, c'est par leur participation aux petits projets que les professionnels acquièrent la compétence et la confiance nécessaires aux grands projets. Deuxièmement, les petits riens d'une construction négligée, comme une porte qui ferme mal ou un carreau légèrement ébréché, voire le bureau désordonné, annulent totalement le charme de l'ensemble. C'est toute l'idée du code propre.

L'architecture n'est qu'une métaphore pour le développement de logiciels, plus particulièrement en ce qui concerne la livraison du produit initial, comme un architecte qui livre un bâtiment impeccable. Aujourd'hui, avec Scrum et les méthodes agiles, l'objectif recherché est la mise sur le marché rapide d'un produit. Nous voulons que l'usine tourne à plein régime pour produire du logiciel. Voici les usines humaines : réfléchir, en pensant aux programmeurs qui travaillent à partir d'un produit existant ou d'un scénario utilisateur pour créer un produit. La métaphore de la fabrication apparaît encore plus fortement dans une telle approche. Les questions de production dans les usines japonaises, sur une ligne d'assemblage, ont inspiré une bonne part de Scrum.

Dans l'industrie automobile, le gros du travail réside non pas dans la fabrication, mais dans la maintenance, ou plutôt comment faire pour l'éviter. Dans le monde du logiciel, au moins 80 % de notre travail est bizarrement appelé "maintenance" : une opération de réparation. Au lieu d'adopter l'approche occidentale typique qui consiste à produire un bon logiciel, nous devons réfléchir comme des réparateurs ou des mécaniciens automobiles. Quel est l'avis du management japonais à ce sujet ?

En 1951, une approche qualité nommée maintenance productive totale (TPM, *Total Productive Maintenance*) est arrivée au Japon. Elle se focalise sur la maintenance, non sur la production. Elle s'appuie principalement sur cinq principes appelés les 5S. Il s'agit d'un ensemble de disciplines, le terme "discipline" n'étant pas choisi au hasard. Ces principes constituent en réalité les fondements du *Lean*, un autre mot à la mode sur la scène occidentale et de plus en plus présent dans le monde du logiciel, et ils ne sont pas facultatifs. Comme le relate l'Oncle Bob, les bonnes pratiques logiciels requièrent de telles disciplines : concentration, présence d'esprit et réflexion. Il ne s'agit pas toujours simplement de faire, de pousser les outils de fabrication à produire à la vitesse optimale. La philosophie des 5S comprend les concepts suivants :

- *Seiri*, ou organisation (pensez à "s'organiser" en français). Savoir où se trouvent les choses, par exemple en choisissant des noms appropriés, est essentiel. Si vous pensez que le nommage des identifiants n'est pas important, lisez les chapitres suivants.
- *Seiton*, ou ordre (pensez à "situer" en français). Vous connaissez certainement le dicton *une place pour chaque chose et chaque chose à sa place*. Un morceau de code doit se trouver là où l'on s'attend à le trouver. Si ce n'est pas le cas, un remaniement est nécessaire pour le remettre à sa place.
- *Seiso*, ou nettoyage (pensez à "scintiller" en français). L'espace de travail doit être dégagé de tout fil pendouillant, de graisse, de chutes et de déchets. Que pensent les auteurs de ce livre de la pollution du code par des commentaires et des lignes de

code mises en commentaires qui retracent l'historique ou prédisent le futur ? Il faut supprimer tout cela.

- *Seiketsu*, ou propre (pensez à "standardiser" en français). L'équipe est d'accord sur la manière de conserver un espace de travail propre. Pensez-vous que ce livre traite du style de codage et de pratiques cohérentes au sein de l'équipe ? D'où proviennent ces standards ? Poursuivez votre lecture.
- *Shutsuke*, ou éducation (pensez à "suivi" en français). Autrement dit, il faut s'efforcer de suivre les pratiques et de réfléchir au travail des autres, tout en étant prêt à évoluer.

Si vous relevez le défi – oui, le défi – de lire et d'appliquer les conseils donnés dans cet ouvrage, vous finirez par comprendre et apprécier le dernier point. Les auteurs nous guident vers un professionnalisme responsable, dans un métier où le cycle de vie des produits compte. Lorsque la maintenance des automobiles et des autres machines se fait sous la TPM, la maintenance de niveau dépannage – l'attente de l'arrivée des bogues – constitue l'exception. Au lieu de dépanner, nous prenons les devants : les machines sont inspectées quotidiennement et les éléments usagés sont remplacés avant qu'ils ne cassent ; autrement dit, nous effectuons l'équivalent de la vidange des 10 000 km afin d'anticiper l'usure. Dans le code, le remaniement doit se faire de manière implacable. Il est toujours possible d'apporter une amélioration, comme a innové la TPM il y a une cinquantaine d'années : construire dès le départ des machines dont la maintenance est plus facile. Rendre le code lisible est aussi important que le rendre exécutable. La pratique ultime, ajoutée dans la TPM vers 1960, consiste à renouveler totalement le parc machine ou à remplacer les plus anciennes. Comme le conseille Fred Brooks, nous devons reprendre à zéro des parties importantes du logiciel tous les sept ans afin de retirer tous les éléments obsolètes qui traînent. Il faut sans doute revoir les constantes de temps de Brooks et remplacer les années par des semaines, des jours ou des heures. C'est là que résident les détails.

Les détails renferment une grande puissance, encore qu'il y ait quelque chose d'humble et de profond dans cette vision de la vie, comme nous pourrions, de manière stéréotypée, le penser d'une approche qui revendique des racines japonaises. Mais il ne s'agit pas seulement d'une vision orientale de la vie ; la sagesse occidentale est elle aussi pleine de réprimandes. La citation *Seiton* précédente a été reprise par un ministre de l'Ohio qui a littéralement vu la régularité "comme un remède à chaque degré du mal". *Quid* de *Seiso* ? *La propreté est proche de la sainteté*. Une maison a beau être belle, un bureau en désordre lui retire sa splendeur. Qu'en est-il de *Shutsuke* sur ces petites questions ? *Qui est fidèle en très peu de chose est fidèle aussi en beaucoup*. Pourquoi ne pas s'empresser de remanier le code au moment responsable, en renforçant sa position pour les prochaines "grandes" décisions, au lieu de remettre cette intervention à plus

tard ? *Un point à temps en vaut cent*. *Le monde appartient à celui qui se lève tôt*. *Ne pas remettre au lendemain ce qu'on peut faire le jour même*. (C'était là le sens original de la phrase "le dernier moment responsable" dans Lean, jusqu'à ce qu'elle tombe dans les mains des consultants logiciels.) Pourquoi ne pas calibrer la place des petits efforts individuels dans un grand tout ? *Petit poisson deviendra grand*. Ou pourquoi ne pas intégrer un petit travail préventif dans la vie de tous les jours ? *Mieux vaut prévenir que guérir*. *Une pomme chaque matin éloigne le médecin*. Un code propre honore les racines profondes de la sagesse sous notre culture plus vaste, ou notre culture telle qu'elle a pu être, ou devait être, et peut être en prenant soin des détails.

Dans la littérature sur l'architecture, nous trouvons également des adages qui reviennent sur ces détails supposés. Nous retrouvons le concept *Seiri* dans les poignées de portes de Mies van der Rohe. Il s'agit d'être attentif à chaque nom de variable. Vous devez nommer une variable avec la même attention que vous portez au choix du nom de votre premier-né.

Comme le savent les propriétaires de maisons, l'entretien et les finitions n'ont pas de fin. L'architecte Christopher Alexander, le père des motifs et des *Pattern Languages*, voit chaque acte de conception lui-même comme un petit acte local de réparation. Il voit également l'art des structures fines comme le seul horizon de l'architecte ; les formes plus grandes peuvent être confiées aux plans et leur application aux habitants. La conception a non seulement lieu lors de l'ajout d'une nouvelle pièce à une maison, mais également lorsque nous changeons le papier peint, remplaçons la moquette usée ou modernisons l'évier de la cuisine. La plupart des arts reprennent des opinions analogues. Nous avons recherché d'autres personnes qui placent la maison de Dieu dans les détails. Nous nous somme retrouvés en bonne compagnie avec l'auteur français du XIX^e^ siècle Gustave Flaubert. Le poète Paul Valéry nous informe qu'un poème n'est jamais terminé et qu'il nécessite un travail perpétuel ; s'il n'est plus révisé, c'est qu'il est abandonné. Cette préoccupation du détail est commune à tous ceux qui recherchent l'excellence. Il n'y a donc sans doute rien de nouveau ici, mais en lisant ce livre vous serez mis au défi de reprendre une bonne discipline que vous aviez abandonnée au profit de la passivité ou d'un désir de spontanéité et pour juste "répondre au changement".

Malheureusement, de telles questions ne sont habituellement pas vues comme des fondements de l'art de la programmation. Nous abandonnons notre code très tôt, non pas qu'il soit terminé, mais parce que notre système de valeur se focalise plus sur les résultats apparents que sur la substance de ce que nous livrons. Ce manque d'assiduité finit par être coûteux : *un assassin revient toujours sur les lieux du crime*. La recherche, qu'elle soit industrielle ou académique, ne s'abaisse pas à garder un code propre. Lors de mes débuts chez Bell Labs Software Production Research (notez le mot Production dans le nom de la société !), nous avions des estimations qui suggéraient qu'une inden-

tation cohérente du code constituait l'un des indicateurs statistiquement significatifs d'une faible densité de bogues. Nous, nous voulions que l'architecture, le langage de programmation ou toute autre notion de haut niveau représentent la qualité. En tant que personnes dont le professionnalisme supposé était dû à la maîtrise d'outils et de méthodes de conception nobles, nous nous sommes sentis insultés par la valeur que ces machines industrielles, les codeurs, ajoutaient par la simple application cohérente d'un style d'indentation. Pour citer mon propre ouvrage écrit il y a dix-sept ans, c'est par un tel style que l'on différencie l'excellence de la simple compétence. Les Japonais comprennent la valeur essentielle de ce travail quotidien et, plus encore, des systèmes de développement qui sont redevables aux actions quotidiennes simples des travailleurs. La qualité est le résultat d'un million d'actes d'attention désintéressée, pas seulement d'une formidable méthode tombée du ciel. Que ces actes soient simples ne signifie pas qu'ils soient simplistes, et en aucun cas faciles. Ils n'en sont pas moins le tissu de la grandeur et, tout autant, de la beauté dans toute tentative humaine. Les ignorer, c'est alors ne pas encore être totalement humain.

Bien entendu, je suis toujours partisan d'une réflexion de plus grande ampleur, et particulièrement de la valeur des approches architecturales ancrées dans une profonde connaissance du domaine et des possibilités du logiciel. Il ne s'agit pas du propos de ce livre, ou, tout au moins, pas de celui affiché. Le message de cet ouvrage est plus subtil et sa profondeur ne doit pas être sous-estimée. Il s'accorde parfaitement à l'adage actuel des personnes orientées code, comme Peter Sommerlad, Kevlin Henney et Giovanni Asproni. "Le code représente la conception" et "du code simple" sont leurs mantras. S'il ne faut pas oublier que l'interface est le programme et que ses structures ont un rapport étroit avec notre structure de programme, il est essentiel de maintenir que la conception vit dans le code. Par ailleurs, alors que, dans la métaphore de la fabrication, le remaniement conduit à des coûts, le remaniement de la conception produit une valeur. Nous devons voir notre code comme la belle articulation de nobles efforts de conception – la conception en tant que processus, non comme une fin statique. C'est dans le code que les métriques architecturales de couplage et de cohésion s'évaluent. Si vous écoutez Larry Constantine décrire le couplage et la cohésion, vous l'entendrez parler de code, non de concepts très abstraits que l'on peut rencontrer en UML. Dans son article "Abstraction Descant", Richard Gabriel nous apprend que l'abstraction est le mal. Le code lutte contre le mal, et un code propre est sans doute divin.

Pour revenir à mon petit paquet de Ga-Jol, je pense qu'il est important de noter que la sagesse danoise conseille non seulement de faire attention aux petites choses, mais également d'être sincère dans les petites choses. Cela signifie être sincère envers le code, envers nos collègues quant à l'état de notre code et, plus important, envers nous-mêmes quant à notre code. Avons-nous fait de notre mieux pour "laisser l'endroit plus propre que nous l'avons trouvé" ? Avons-nous remanié notre code avant de l'archiver ?

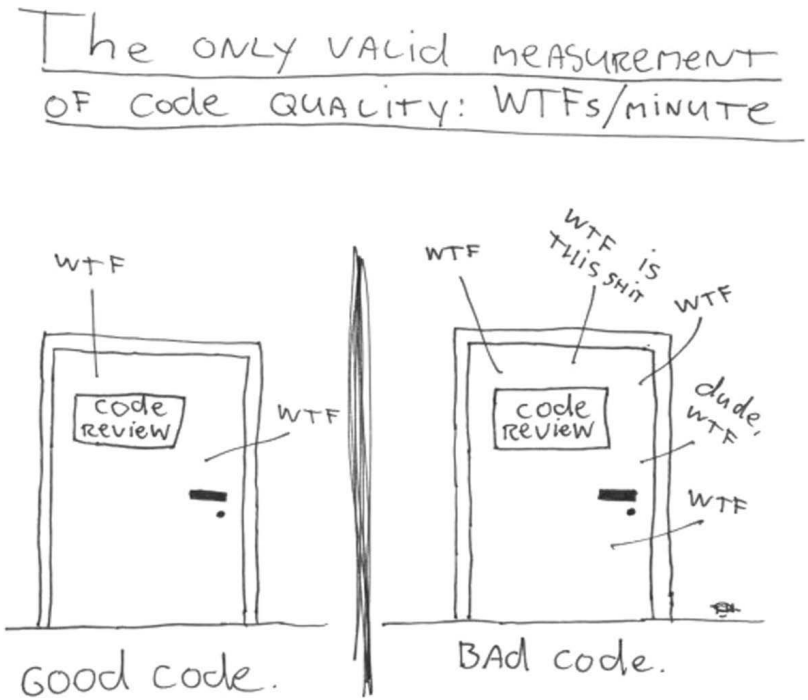
Il s'agit non pas de points secondaires, mais de questions au centre des valeurs agiles. Scrum recommande de placer le remaniement au cœur du concept "Terminé". Ni l'architecture ni un code propre n'insistent sur la perfection, uniquement sur l'honnêteté et faire de son mieux. *L'erreur est humaine, le pardon, divin*. Dans Scrum, tout est visible. Nous affichons notre linge sale. Nous sommes sincères quant à l'état de notre code, car le code n'est jamais parfait. Nous devenons plus humains, plus dignes du divin et plus proches de cette grandeur dans les détails.

Dans notre métier, nous avons désespérément besoin de toute l'aide que nous pouvons obtenir. Si le sol propre d'un magasin diminue le nombre d'accidents et si une bonne organisation des outils augmente la productivité, alors, j'y adhère. Ce livre est la meilleure application pragmatique des principes de Lean aux logiciels que j'ai jamais lue. Je n'attendais pas moins de ce petit groupe d'individus réfléchis qui s'efforcent ensemble depuis des années non seulement de s'améliorer, mais également de faire cadeau de leurs connaissances à l'industrie dans des travaux tels que l'ouvrage qui se trouve entre vos mains. Le monde est laissé dans un état un peu meilleur qu'avant que je ne reçoive le manuscrit de l'Oncle Bob.

Ces nobles idées étant étalées, je peux à présent aller ranger mon bureau.

James O. Coplien
Mørdrup, Danemark

Introduction



CQCB : C'est quoi ce bordel

Reproduction et adaptation avec l'aimable autorisation de Thom Holwerda (**http://www.osnews.com/story/19266/WTFs_m**). © 2008 Focus Shift

Quelle porte ouvre sur votre code ? Quelle porte ouvre sur votre équipe ou votre entreprise ? Pourquoi êtes-vous dans cette pièce ? S'agit-il simplement d'une révision normale du code ou avez-vous découvert tout un ensemble de problèmes désastreux peu après le lancement ? Procédez-vous à un débogage en urgence, en plongeant dans du code que vous pensiez opérationnel ? Les clients partent-ils en masse et les managers vous surveillent-ils ? Comment pouvez-vous être sûr que vous serez derrière la *bonne* porte lorsque les choses iront mal ? Les réponses tiennent en une seule : l'*art du métier*.

La maîtrise de l'art du métier englobe deux parties : connaissances et travail. Vous devez acquérir les connaissances concernant les principes, les motifs, les pratiques et les heuristiques connus de l'artisan, et vous devez également polir ces connaissances avec vos doigts, vos yeux et vos tripes, en travaillant dur et en pratiquant.

Je peux vous enseigner la physique qui permet de rouler à vélo. Il est vrai que les mathématiques classiques sont relativement simples. Gravité, frottements, moment angulaire, centre d'inertie, etc. peuvent être expliqués en moins d'une page d'équations. Grâce à cette formule, je peux prouver qu'il vous est possible de rouler à vélo et vous apporter toutes les connaissances dont vous avez besoin pour ce faire. Néanmoins, vous tomberez immanquablement la première fois que vous grimperez sur la bicyclette.

Écrire du code n'est pas si différent. Nous pourrions rédiger tous les bons principes d'écriture d'un code propre et vous faire confiance pour réaliser le travail (autrement dit, vous laisser tomber lorsque vous monterez sur le vélo), mais quelle sorte de professeurs serions-nous alors et quelle sorte d'étudiant seriez-vous ?

Ce n'est pas l'orientation que nous donnons à ce livre.

Apprendre à écrire du code propre est un *travail difficile*. Cela ne se limite pas à connaître des principes et des motifs. Vous devez *transpirer*. Vous devez pratiquer et constater vos échecs. Vous devez regarder d'autres personnes pratiquer et échouer. Vous devez les voir hésiter et revenir sur leurs pas. Vous devez les voir se tourmenter sur des décisions et payer le prix de leurs mauvais choix.

Vous devez être prêt à travailler dur au cours de la lecture de cet ouvrage. Il ne s'agit pas d'un livre que vous pourrez lire dans un avion et terminer avant d'atterrir. Il vous imposera de *travailler, dur*. Qu'est-ce qui vous attend ? Vous allez lire du code, beaucoup de code. Vous devrez réfléchir aux points positifs et aux points négatifs de ce code. Il vous sera demandé de nous suivre pendant que nous découpons des modules, pour ensuite les réunir à nouveau. Cela demandera du temps et des efforts, mais nous pensons que cela en vaut la peine.

Nous avons décomposé ce livre en trois parties. Les premiers chapitres décrivent les principes, les motifs et les pratiques employés dans l'écriture d'un code propre. Ils contiennent beaucoup de code et ne seront pas faciles à lire. Ils vous prépareront à la deuxième partie. Si vous refermez le livre après avoir lu la première partie, nous vous souhaitons bonne chance !

C'est dans la deuxième partie que se trouve le travail le plus difficile. Elle est constituée de plusieurs études de cas dont la complexité va croissant. Chaque étude de cas est un exercice de nettoyage : une base de code qui présente certains problèmes doit être transformée en une version soulagée de quelques problèmes. Dans cette section, le niveau de

détail est élevé. Vous devrez aller et venir entre le texte et les listings de code. Vous devrez analyser et comprendre le code sur lequel nous travaillons et suivre notre raisonnement lors de chaque modification effectuée. Trouvez du temps, car *cela demandera plusieurs jours*.

La troisième partie sera votre récompense. Son unique chapitre contient une liste d'heuristiques et d'indicateurs collectés lors de la création des études de cas. Pendant l'examen et le nettoyage du code dans les études de cas, nous avons documenté chaque raison de nos actions en tant qu'heuristique ou indicateurs. Nous avons essayé de comprendre nos propres réactions envers le code que nous lisions et modifiions. Nous avons fait tout notre possible pour consigner l'origine de nos sensations et de nos actes. Le résultat est une base de connaissances qui décrit notre manière de penser pendant que nous écrivons, lisons et nettoyons du code.

Cette base de connaissance restera d'un intérêt limité si vous ne faites pas l'effort d'examiner attentivement les études de cas de la deuxième partie de cet ouvrage. Dans ces études de cas, nous avons annoté consciencieusement chaque modification apportée, en ajoutant également des références vers les heuristiques. Ces références apparaissent entre crochets, par exemple [H22]. Cela vous permet de savoir dans quel *contexte* ces heuristiques ont été appliquées et écrites ! C'est non pas tant les heuristiques en soi qui ont de la valeur, mais *le lien entre ces heuristiques et chaque décision que nous avons prise pendant le nettoyage du code*.

Pour faciliter l'emploi de ces liens, nous avons ajouté à la fin du livre des références croisées qui indiquent le numéro de page de chaque référence d'heuristique. Vous pouvez les utiliser pour rechercher chaque contexte d'application d'une heuristique.

Si vous lisez la première et la troisième partie, en sautant les études de cas, vous n'aurez parcouru qu'un livre de plus sur la bonne écriture des logiciels. En revanche, si vous prenez le temps de travailler sur les études de cas, en suivant chaque petite étape, chaque décision, autrement dit en vous mettant à notre place et en vous forçant à réfléchir à notre manière, alors, vous comprendrez beaucoup mieux ces principes, motifs, pratiques et heuristiques. Ils ne seront plus alors des connaissances de "confort". Ils seront ancrés dans vos tripes, vos doigts et votre cœur. Ils feront partie de vous, de la même manière qu'un vélo devient une extension de votre volonté lorsque vous savez comment le conduire.

Remerciements

Illustrations

Je voudrais remercier mes deux artistes, Jeniffer Kohnke et Angela Brooks. Jeniffer est l'auteur des illustrations créatives et sensationnelles qui débutent chaque chapitre, ainsi que des portraits de Kent Beck, Ward Cunningham, Bjarne Stroustrup, Ron Jeffries, Grady Booch, Dave Thomas, Michael Feathers et moi-même.

Angela s'est chargée des illustrations qui agrémentent le contenu de chaque chapitre. Elle a déjà réalisé de nombreuses figures pour moi, notamment pour l'ouvrage *Agile Software Development: Principles, Patterns, and Practices*. Elle est également ma fille aînée, qui me comble de bonheur.

Nous espérons que cet extrait vous a plu !

Pour acheter ce livre, choisissez sur la liste
de nos libraires le plus proche de chez vous.
Chez certains libraires, vous pouvez commander
en ligne et vous faire livrer à domicile.

Les livres Nouveaux Horizons coûtent
trois fois moins cher

Merci de votre confiance, à bientôt !

nouveaux horizons

PRIX SUBVENTIONNÉ