Jean-Louis **Bénard** • Laurent **Bossavit**
Régis **Medina** • Dominic **Williams**

Gestion de projet eXtreme Programming

EYROLLES

Gestion de projet eXtreme Programming

Tests unitaires, Refactoring, Intégration, Gestion de projet, ROI, Aspects contractuels

Comme toute méthode de développement, l'Extreme Programming propose un cadre pour l'ensemble des aspects du projet logiciel, de l'analyse des besoins aux tests en passant par la conception. Mais à la différence de processus prédictifs, XP ne se fonde pas sur une définition exhaustive et précoce des besoins. En découle une redéfinition de la relation entre clients et fournisseurs, avec de surprenants résultats en termes de qualité de code, de délais... et de retour sur investissement !

Diplômé de l'École Centrale Paris, fondateur de la SSII F.R.A. puis directeur technique de Business Interactif, **Jean-Louis Bénard** est président de Brainsonic et enseignant à l'École Centrale. Il accompagne régulièrement de grandes entreprises françaises dans des projets d'architecture du système d'information et dans la mise en place de méthodes de développement pragmatiques.

Laurent Bossavit est consultant et formateur indépendant. Ses quinze années d'expérience en tant que développeur, chef de projet ou architecte, l'ont amené à accompagner des entreprises, PME éditrices de logiciels ou grands comptes industriels ou financiers, lors de leur adoption d'XP. Conférencier, écrivain et enseignant, il contribue à l'évolution et à l'adoption de l'Extreme Programming.

Régis Medina intervient en tant qu'ingénieur indépendant sur des projets de développement en technologies objet. Pionnier de l'Extreme Programming en France, il en applique les principes dès 2000 en tant que chef de projet XP et met en évidence ses avantages dans le cadre de grands projets de télécommunications.

Dominic Williams développe depuis plus de dix ans des logiciels dans des domaines allant de l'océanographie à la gestion de biens en passant par la signalisation ferroviaire. Il pratique la méthode XP depuis 1999 et lui attribue ses plus belles réussites : des logiciels de qualité livrés plus vite par des équipes motivées et efficaces. Il promeut XP auprès des entreprises et grâce au projet Open Source XP Dojo.

architecte logiciel

Quelles règles pour la création logicielle ?
Quelles méthodes, quels outils ?
L'enjeu est de taille : garantir la souplesse et l'interopérabilité des applications métier.

Au sommaire

Conception : Nord Compo

EYROLLES

Code éditeur : G11561
ISBN : 2-212-11561-X

Gestion de projet eXtreme Programming

CHEZ LE MÊME ÉDITEUR

Dans la collection Architecte logiciel

P. ROQUES, F. VALLÉE. – **UML 2 en action**. *De l'analyse des besoins à la conception J2EE.*
N°11462, 3e édition, 2004, 380 pages + poster.

Modélisation et méthodes logicielles, programmation objet

P. ROQUES. – **UML 2 par la pratique, 3e édition**. *Cours et exercices.*
N°11480, 2004, 340 pages.

P. ROQUES. – **UML : modéliser un site e-commerce**.
N°11070, 2002, 168 pages (coll. *Cahiers du programmeur*).

D. CARLSON. – **Modélisation d'applications XML avec UML**.
N°9297, 2002, 354 pages.

A. COCKBURN. – **Rédiger des cas d'utilisation efficaces**.
N°9288, 2001, 320 pages.

H. BERSINI, I. WELLESZ. – **L'orienté objet**. *Cours et exercices en UML 2 avec Java, Python, C# et C++.*
N°11538, 2004, 520 pages

B. MEYER. – **Conception et programmation orientées objet**.
N°9111, 2000, 1223 pages.

I. JACOBSON, G. BOOCH, J. RUMBAUGH. – **Le Processus unifié de développement logiciel**.
N°9142, 2000, 487 pages.

R. PAWLAK, J.-P. RETAILLÉ, L. SEINTURIER. – **Programmation orientée aspects pour Java/J2EE**.
N°11408, 2004, 460 pages.

Développement Java et .NET

L. MAESANO, C. BERNARD, X. LEGALLES. – **Services Web en J2EE et .Net**.
N°11067, 2003, 1088 pages.

T. PETILLON. – **ASP.NET**. (coll. *Cahiers du programmeur*)
N°11210, 2003, 200 pages.

E. PUYBARET. – **Java 1.4 et 5.0**. (coll. *Cahiers du programmeur*)
N°11478, 2004, 400 pages

J. MOLIÈRE. – **Conception et déploiement Java/J2EE**.
N°11194, octobre 2003, 192 pages.

P.-Y. SAUMONT. – **Le Guide du développeur Java 2**.
Meilleures pratiques de programmation avec Ant, JUnit et les design patterns.
N°11275, 2003, 800 pages + CD-Rom.

K. DJAFAAR. – **Développement J2EE avec Eclipse et WSAD**.
N°11285, 2004, 640 pages + 2 CD-Rom.

L. DERUELLE. – **Développement Java/J2EE avec Jbuilder**.
N°11346, 2004, 726 pages avec CD-Rom.

Jean-Louis **Bénard** • Laurent **Bossavit**
Régis **Medina** • Dominic **Williams**

Gestion de projet eXtreme Programming

EYROLLES

ÉDITIONS EYROLLES
61, bd Saint-Germain
75240 Paris CEDEX 05
www.editions-eyrolles.com

Cet ouvrage a fait l'objet d'un reconditionnement à l'occasion de son deuxième tirage (nouvelle couverture). Le texte de l'ouvrage reste inchangé par rapport au tirage précédent.

Remerciements particuliers à Pierre Tran et Thierry Cros pour avoir favorisé la naissance de cet ouvrage.

Remerciements

Je tiens à remercier Hervé Ségalen pour m'avoir communiqué la passion des méthodes ; Muriel Shan Sei Fan pour son soutien sans faille à ce projet ; Pierre Tran qui nous a permis de croiser nos expériences ; et bien sûr Sophie, Jean, Thérèse et Christine pour leur patience et leurs encouragements.

Jean-Louis Bénard

Ce livre ne serait pas ce qu'il est si nous n'avions été précédés par Ron, Kent, Ward, Martin et tous les extrémistes de la première heure. Aujourd'hui encore je ne sais pas si je dois les maudire de m'avoir fait perdre mes illusions ou les remercier de m'avoir ouvert les yeux – à défaut d'être sage, j'opte pour la seconde voie. D'autres programmeurs et gens du métier sont responsables, souvent sans le savoir, des quelquesbonnes idées que j'ai pu glisser dans ces pages – les âneries sont naturellement de mon fait (sauf quand c'est la faute de Chet). Du cerveau à la page imprimée, il y a un chemin qui passe par le travail et la patience d'une équipe éditoriale : Muriel, Patrice, Anne, Sophie ont su en faire un véritable plaisir. Last but not least – comme bien des auteurs, c'est à ma famille que je réserve mes premiers instants de repos après avoir écrit le mot «fin», pour les remercier comme il se doit – Sophie, Axel, Erwan ; sans oublier «Nom de code Zébulon», ainsi qu'Alain et Ginette pour leurs encouragements.

Laurent Bossavit

Je remercie en premier lieu les Three extremos – Kent Beck, Ward Cunningham, Ron Jeffries –, dont j'admire le génie et surtout les qualités humaines pour avoir inventé une démarche qui transforme le compte à rebours infernal de chaque projet en une aventure collective passionnante. Merci aussi à ceux avec qui j'ai eu le plaisir de faire ce livre – je pense bien sûr aux autres auteurs, Jean-Louis, Laurent, Dominic, mais aussi à Muriel Shan Sei Fan, Patrice Beray, Sophie Hincelin et Anne Garcia pour la partie éditoriale. Je n'oublie pas non plus Thierry Cros, qui m'a permis d'embarquer sur ce projet il y a quelques mois.

Je souhaite également remercier ceux avec qui j'explore XP depuis quelques années : Marc Guiot, Thomas Novelle et Sébastien Crego – j'espère que nos aventures ne font que commencer. Un grand merci enfin à ma petite famille, Séverine, Chloé et Roméo, pour avoir eu la patience de vivre ces quelques mois avec un papa toujours collé à son écran.

Régis Médina

Je tiens à remercier d'abord mes coéquipiers, avec qui j'ai eu la chance de pratiquer XP : Fabrice, Michel, Michaël, Florent, Jean-Baptiste, François A., Franck, François M., Nicolas, Jacques, Patrice B., Patrice C., Cédric, Virgile et Alexandre. Vous m'avez appris tout ce qu'il y a de plus important, et travailler avec vous a été un réel plaisir. Merci aussi à Édith, Jean-Michel, Serge et Jean de nous avoir fait confiance, et d'être restés exigeants mais constructifs.

Je remercie mes coauteurs, Jean-Louis, Laurent et Régis ainsi que notre éditeur, Muriel Shan Sei Fan – vous avez fait de l'écriture de ce livre une expérience passionnante et enrichissante à de nombreux points de vue.

Enfin, je remercie de tout cœur ma petite famille : c'est en m'inspirant de ton intégrité, Pascale, et de ton dynamisme, Chloé, que je trouve la force morale et physique pour tout ce que j'entreprends.

Dominic Williams

Préface

Il est un aspect de l'Extreme Programming qui plonge certains lecteurs dans un désespoir mêlé de rage, et suscite l'admiration chez d'autres : sa vulnérabilité face au contexte culturel.

S'il existe bien une culture mondiale des « fêlés du code» *(geeks)* – et XP rend, par certains côtés, hommage aux traits de caractère et aux faiblesses humaines nécessaires pour devenir un maître programmeur –, il reste que l'activité de programmeur ne se conçoit que dans un contexte culturel plus vaste – celui d'une culture d'entreprise, d'une culture nationale…

Or, les éléments d'XP qui tentent de jeter un pont entre la culture des «geeks» et celle des entreprises américaines ne seront pas universellement applicables…

D'où ce livre, qui tire son efficacité de ce que ses auteurs sont partis des ingrédients fondamentaux de l'Extreme Programming, y ont mêlé leurs propres expériences de mise en œuvre en France, ont fait mijoter le tout au feu de leur dialectique collective, non sans l'avoir assaisonné d'une présentation originale.

Kent Beck

Table des matières

1

Introduction

Le contraire d'une vérité profonde peut très bien être une autre vérité profonde.

– Niels Bohr

L'Extreme Programming (XP) est un ensemble de pratiques qui couvre une grande partie des activités de la réalisation d'un logiciel – de la programmation proprement dite à la planification du projet, en passant par l'organisation de l'équipe de développement et les échanges avec le client. Ces pratiques n'ont en soi rien de révolutionnaire : il s'agit simplement de pratiques de bon sens mises en œuvre par des développeurs ou des chefs de projet expérimentés, telles que les livraisons fréquentes, la relecture de code, la mise en place de tests automatiques… La nouveauté introduite par XP consiste à pousser ces pratiques à l'extrême – d'où le nom de la méthode – et à les organiser en un tout cohérent, parfaitement défini et susceptible d'être répété.

Par opposition aux méthodes de développement dites «lourdes», qui imposent des activités et des produits indirects consommateurs de temps et de ressources, XP se définit comme un processus léger dans lequel l'équipe se focalise sur la réalisation elle-même. Toutes les autres activités sont réduites à leur strict minimum, sachant que la priorité y est donnée à l'agilité : XP s'adresse principalement à de petites équipes (moins de dix personnes) qui souhaitent réaliser rapidement des logiciels et réagir aisément au changement, sans faire pour autant de concessions sur la qualité du logiciel produit.

Dans la pratique, ce critère d'agilité proposé par XP se retrouve à trois niveaux :

- au niveau du code lui-même, qui est maintenu toujours aussi clair et simple que possible par une activité de remaniement permanente, et qui est soutenu par une batterie de tests automatiques permettant d'y introduire de nouvelles fonctionnalités sans craindre des régressions cachées ;

- au niveau de l'équipe de développement, dont les membres travaillent toujours en commun et sont capables d'intervenir sur toutes les parties de l'application ;
- au niveau de la gestion du projet, à travers une démarche itérative qui permet à tous les intervenants du projet – aussi bien l'équipe de développement que le client – d'améliorer continuellement son pilotage en s'appuyant sur l'expérience acquise en cours de route.

Mais à elle seule, cette agilité ne suffit pas à expliquer l'intérêt que suscite XP auprès des équipes qui le pratiquent. En effet, au-delà d'une recherche purement cartésienne de l'efficacité, XP est guidé par un ensemble de valeurs humanistes, une vision positive du développement logiciel fondée sur la conviction que l'efficacité s'obtient en prenant le temps de faire un travail de qualité, et en donnant une place de premier plan aux contacts humains – au sein de l'équipe elle-même, mais aussi entre cette dernière et son client.

Ainsi, même si la plupart des pratiques XP prises une à une sont déjà connues, elles ont ensemble le potentiel nécessaire pour créer un environnement tout à fait particulier, à la fois efficace et vivant – et l'expérience prouve que ceux qui ont connu l'environnement d'un projet XP réussi ont souvent bien du mal à revenir à des démarches plus traditionnelles par la suite.

Les limites des démarches « par phases »

L'Extreme Programming marque une rupture vis-à-vis des processus séquentiels, dérivés pour la plupart du classique « cycle en V » (figure 1-1) dont les différentes phases visent à produire successivement le cahier des charges de l'application, le document de spécifications, le document de conception générale, le document de conception détaillée, une collection de plans de tests… et éventuellement l'application elle-même !

De nombreuses variantes de cette approche ont vu le jour, parfois plus légères, parfois assouplies par l'introduction de cycles itératifs. Mais quoi qu'il en soit, le canevas d'un projet de développement reste toujours *grosso modo* le même : on cherche à définir très précisément ce que l'on souhaite construire avant de se lancer dans la réalisation proprement dite.

Or, si cette démarche a largement fait ses preuves lorsqu'il s'agissait de construire des ponts ou des immeubles, on ne peut lui attribuer le même succès dans le domaine du développement logiciel. Par exemple, si l'on en croit les résultats d'une étude menée en 1994 par une société de conseil américaine sur un échantillon de plus de 8000 projets, il ressort que seuls 16 % d'entre eux ont été finalisés dans le temps et le budget initialement prévus[1]. Pire, 32 % de ces projets ont été interrompus en cours de route.

Le diagnostic de ces dérapages est souvent le même : après des spécifications ambitieuses et une conception poussée, la construction s'enlise, souvent mise à mal par une remise en ques-

1. *The CHAOS Report*, réalisé par *The Standish Group* : http://www.pm2go.com/sample_research/chaos_1994_1.asp

tion des choix initiaux du projet qui oblige à revoir des pans entiers du modèle de conception proposé.

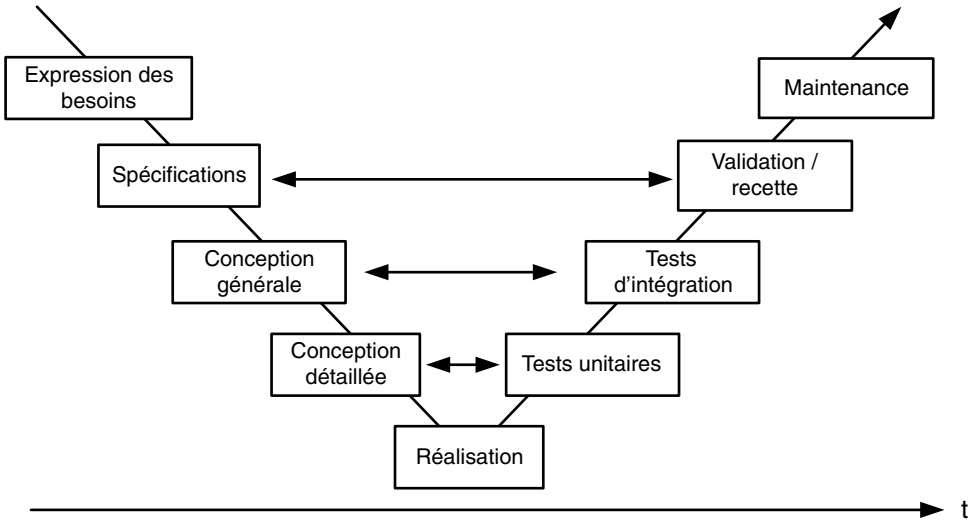

Figure 1-1. Le cycle en V

Mais si ces problèmes sont réellement chroniques, faut-il forcément critiquer la façon dont le processus est mis en œuvre par les équipes ? Ne faudrait-il pas plutôt se demander si le processus en question correspond réellement aux exigences du projet concerné ? L'approche linéaire se montre efficace dans certains contextes, mais il existe des cas dans lesquels les limites inhérentes à cette approche en font un piètre choix. Nous décrivons brièvement ces limites dans les sections qui suivent.

L'utopie des spécifications complètes et immuables

Les oublis, erreurs ou changements dans les spécifications constituent la source principale de problèmes pour la plupart des projets. Mais faut-il vraiment s'en étonner ? Mis à part certains cas d'applications très simples, ou encore dans de rares contextes parfaitement connus et maîtrisés, la définition *a priori* d'un logiciel un tant soit peu complexe est un exercice extrêmement difficile. Qui plus est, cet exercice déterminant prend place au début du projet, au moment où les différents intervenants en savent le moins sur le contexte précis de l'application – en particulier, le client a souvent bien du mal à se faire une idée précise de ce que pourrait être l'application quelques mois plus tard, surtout lorsqu'il n'existe pas d'applications équivalentes qui puissent servir de référence.

Remarque

Nous utilisons ici le terme de «client» d'une manière volontairement assez vague, désignant à la fois le donneur d'ordres, l'utilisateur final et celui qui finance le projet. Nous reviendrons sur ce terme précis au chapitre 2 consacré aux rôles des différents intervenants d'un projet XP.

Il n'est donc pas surprenant que ces spécifications soient remises en question à plusieurs reprises en cours de projet, notamment lorsque le client est mis en présence des premières versions du logiciel. Ces remises en question représentent une perte de temps, puisqu'une partie des efforts, fondée sur les spécifications initiales, est perdue, mais elles représentent aussi un risque pour la conception logicielle mise en place, qui absorbera plus ou moins bien les adaptations nécessaires.

Certes, les méthodes et les outils de modélisation ne cessent d'évoluer, le dernier élément majeur de cette évolution étant l'*Unified Modeling Language* initialement créé par Grady Booch, Ivar Jacobson et James Rumbaugh[1]. Mais ces outils n'apportent pas de changements réellement significatifs sur le déroulement des projets de développement, et il convient de se demander si la solution à ces problèmes de spécification et de conception passe nécessairement par un effort supplémentaire de modélisation… Ne serait-il pas envisageable au contraire d'adopter une approche radicalement différente qui accorde une importance moindre à cette phase initiale ?

L'effet tunnel

Autre travers chronique de l'approche linéaire, la visibilité réduite qui est accordée au client pendant la phase de construction. L'équipe peut rester isolée plusieurs mois et présenter un beau jour au client un produit qui correspond peut-être à ce qu'il avait spécifié, mais pas nécessairement à ce dont il a réellement besoin. Et la situation est souvent d'autant plus difficile à rattraper qu'elle est détectée tardivement dans le projet.

La solution à ce problème consiste naturellement à faire intervenir plus fréquemment le client au cours du projet, mais encore faut-il que cette intervention ne soit pas réduite à un simple constat : il faut lui donner réellement le pouvoir d'influer sur le cours du projet, quitte à revenir sur des décisions précédentes, et les changements de cap ainsi imposés doivent pouvoir être absorbés sans traumatisme par l'équipe de développement.

Des équipes compartimentées

L'un des objectifs de la phase de conception initiale est d'identifier un découpage de l'application qui permette de paralléliser sa réalisation. Cette approche offre des avantages indé-

1. Voir à ce propos l'ouvrage *UML en action* de Pascal Roques et Franck Vallée, Eyrolles, 2001.

niables lorsqu'il s'agit de mener de gros projets faisant intervenir des équipes aux spécialités réellement différentes, mais on peut légitimement douter de son adéquation au sein même d'équipes de taille raisonnable.

En effet, le modèle d'organisation d'équipe qui semble le plus répandu aujourd'hui – des développeurs travaillant seuls sur leurs machines, encadrés par un chef de projet qui se charge d'une communication «en étoile» – souffre de deux défauts principaux. Le premier tient à une mauvaise distribution de l'information dans l'équipe, qui contraint la répartition de la charge et limite ainsi la productivité globale de l'équipe. Le second défaut est dû à une spécialisation abusive des développeurs : ceux-ci sont cantonnés dans des parties réduites de l'application, jusqu'à ce qu'ils se lassent de faire toujours la même chose et quittent le projet.

Et si l'on amenait ces développeurs à travailler réellement ensemble, plutôt que de chercher une performance hypothétique dans la parallélisation et la spécialisation ?

Un code soumis à une paralysie progressive

Le dernier problème fréquemment associé aux processus lourds est une sorte de paradoxe : à force de chercher à prouver la qualité du logiciel par ses propriétés extérieures – la documentation, les tests –, il arrive que sa qualité interne, celle du code lui-même, finisse par laisser à désirer. Ce phénomène est souvent lié au modèle «éclaté» d'organisation de l'équipe : la qualité du code d'une partie de l'application dépendra uniquement de la compétence et de la rigueur du développeur qui y travaille.

L'importance de ce point particulier est souvent négligée par les décideurs du projet, qui considèrent d'une part que le code est exclusivement une affaire de développeurs, et d'autre part que les phases de validation ont justement pour objet de vérifier cette qualité. Seulement, un paramètre essentiel reste absent de ce raisonnement : la vitesse de développement. Plus le code dégénère, moins les extensions et les évolutions y sont aisées, et donc plus les développements coûtent cher. Ce problème est très difficile à déceler de l'extérieur ; soit il n'est jamais relevé, et l'application coûtera simplement plus cher que ce qu'il aurait dû en être, soit il apparaît un jour que l'application est devenue trop rigide, mais il est alors souvent trop tard pour corriger le tir et la récriture s'impose.

> **Remarque**
>
> Pour parer aux problèmes de qualité du code, il est par exemple possible d'organiser des séances de relecture de code croisée. Malheureusement, lorsqu'elles sont réalisées après l'écriture du code proprement dit, ces activités sont très coûteuses en temps, et peu d'équipes peuvent se permettre d'y consacrer l'investissement nécessaire.

Or, si le code – qui reste le livrable principal d'un projet de développement logiciel – fait l'objet d'attentions insuffisantes, c'est avant tout parce qu'il est généralement associé à une phase de *fabrication* du logiciel, qui devrait pouvoir découler des phases de conception

détaillées menées en amont. Mais l'expérience prouve que l'implémentation ne se réduit pas à cela : c'est au cours de l'implémentation que la validité des spécifications et de la conception est mise à l'épreuve. Pour que l'application puisse absorber les inévitables changements de spécifications et de conception qui interviennent dans un projet, il faut intégrer l'activité d'implémentation dans celles de spécification et de conception – la fabrication proprement dite étant l'affaire des compilateurs.

Un changement de référentiel

Si l'approche linéaire du développement présente certaines limites, il n'empêche qu'elle reste aujourd'hui à la base de la quasi-totalité des projets de développement – à tel point qu'il paraît bien difficile d'imaginer une autre façon de procéder. Mais pourquoi en est-il ainsi ?

L'une des idées principales qui soutiennent cette approche est que, plus on avance dans le projet, plus le coût des modifications dans le logiciel est élevé. Le coût du changement augmenterait même de manière exponentielle avec le temps, comme l'illustre la figure 1-2.

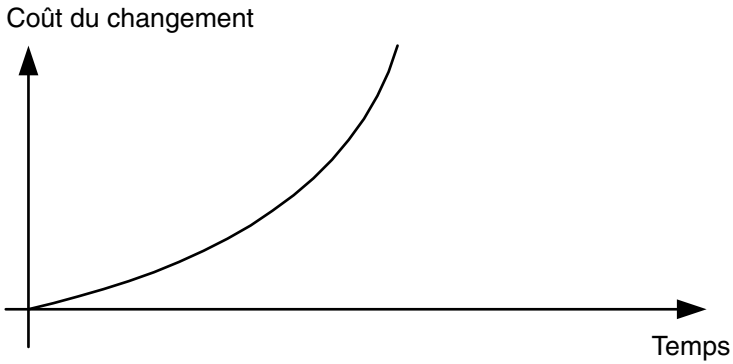

Figure 1-2. Évolution supposée du coût du changement dans un logiciel

Dans un tel contexte, il est tout à fait justifiable – et même recommandé – de chercher à définir complètement le logiciel avant d'entamer sa construction. Partant de cela, une approche linéaire plus ou moins inspirée du cycle en V finit d'ailleurs par s'imposer.

Remarque

Concernant l'hypothèse d'une augmentation exponentielle du coût du changement avec le temps, on peut cependant se demander dans quelle mesure une approche fondée sur cette supposition ne tend pas à générer des logiciels qui la respectent, créant ainsi un phénomène de renforcement réciproque.

Or, ce qu'ont découvert les créateurs d'XP – qui étaient avant tout des experts de la conception et de la programmation objet –, c'est qu'en appliquant certains principes de conception et de programmation, il devenait possible de garder l'application suffisamment «souple» pour que les changements restent aisés tout au long du projet. La courbe d'évolution du coût du changement présentait alors un autre profil, pour adopter celui présenté en figure 1-3.

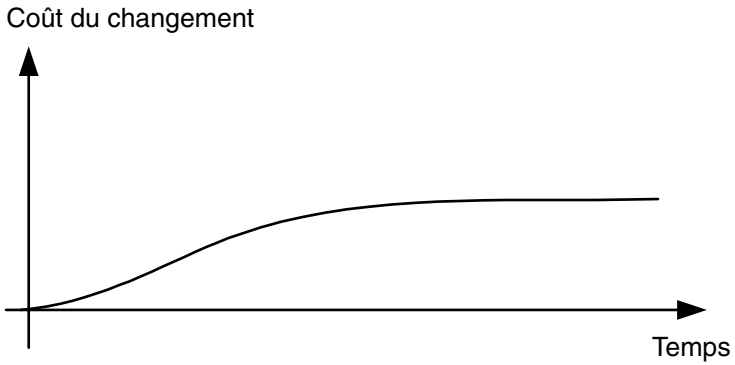

Figure 1-3. L'évolution du coût du changement dans le logiciel revue par les auteurs d'XP

Dans un tel contexte, la stratégie gagnante ne consiste plus à tout figer dès le début, mais au contraire à diffuser la prise de décisions tout au long du projet pour tirer parti de l'expérience acquise en cours de route. Cela est valable pour la conception du logiciel – les choix de conception sont faits au moment où l'implémentation le réclame – mais aussi pour les spécifications elles-mêmes – le client précise les détails de ses besoins au moment où les développeurs implémentent les fonctionnalités correspondantes. De la sorte, les décisions ne sont plus prises sur la base d'analyses et de suppositions abstraites, mais à partir des informations concrètes, disponibles au moment même où elles doivent être appliquées.

L'Extreme Programming prend place dans ce nouveau référentiel. Il propose un ensemble de pratiques concrètes qui permettent, d'une part, d'obtenir la souplesse requise et, d'autre part, d'exploiter cette souplesse pour servir au mieux les intérêts du client et de l'équipe de développement.

Les pratiques d'XP

XP définit treize pratiques «canoniques», que nous classons ici en trois grandes catégories : celles relatives à la programmation, celles relatives au fonctionnement interne de l'équipe et celles qui sont liées à la planification et aux relations avec le client.

Les pratiques de programmation

Au cœur des pratiques XP, les pratiques de programmation (voir chapitres 3 et 4) que nous présentons ci-après permettent d'améliorer continuellement la conception et le code de l'application pour qu'elle reste toujours aussi claire et simple que possible :

- **Conception simple** (*simple design*) : les développeurs implémentent toujours la solution la plus simple qui puisse fonctionner. En particulier, ils n'inventent pas de mécanismes génériques si le besoin immédiat ne l'exige pas.
- **Remaniement** (*refactoring*) : les développeurs n'hésitent pas à revenir sur le code écrit pour le rendre plus «propre», le débarrasser d'éventuelles parties inutilisées, et le préparer à l'ajout de la fonctionnalité suivante. D'une manière plus générale, cette pratique propose une démarche de conception continue qui fait émerger la structure de l'application au fur et à mesure du développement.
- **Développement piloté par les tests unitaires** (*test-first programming, unit tests, developer tests*) : les développeurs écrivent des tests automatiques pour le code qu'ils produisent, et ce au moment même d'écrire le code en question. Cela leur permet d'une part de mieux cerner le problème avant d'écrire le code, et d'autre part de constituer progressivement une batterie de tests qui les autorise ensuite à apporter rapidement des changements dans l'application, tout en conservant une certaine sérénité.
- **Tests de recette** (*acceptance tests, customer tests*) : le client précise très explicitement ses besoins et les objectifs des programmeurs – en participant à la rédaction de tests de recette. Comme les tests unitaires, les tests de recette doivent être automatiques afin de pouvoir vérifier tous les jours la non-régression du produit.

Les pratiques de collaboration

Dans une équipe XP, tous les développeurs travaillent ensemble (voir chapitre 5) et interviennent sur la totalité de l'application. Cela garantit la qualité de cette dernière à travers les relectures croisées que cela engendre, mais cela rend également le travail plus motivant et offre une plus grande souplesse dans l'affectation des tâches. Les pratiques qui régissent cette organisation sont les suivantes :

- **Programmation en binôme** (*pair programming*) : lorsqu'ils écrivent le code de l'application, les développeurs travaillent systématiquement à deux sur la même machine – il s'agit là d'une forme «extrême» de relecture de code, dans laquelle les deux développeurs collaborent activement pour résoudre les problèmes qu'ils rencontrent. Les binômes changent fréquemment, ainsi chacun est amené à travailler tôt ou tard avec tous les autres membres de l'équipe.
- **Responsabilité collective du code** (*collective code ownership*) : tous les développeurs de l'équipe peuvent être amenés à travailler sur toutes les parties de l'application. De plus, ils

ont le devoir d'améliorer le code sur lequel ils interviennent, même s'ils n'en sont pas les auteurs initiaux.

- **Règles de codage** (*coding standards*) : les développeurs se plient à des règles de codage définies par l'équipe elle-même, de manière à garantir l'homogénéité de leur code avec le reste de l'application, et ainsi à faciliter l'intervention d'autres développeurs.
- **Métaphore** (*metaphor*) : les développeurs n'hésitent pas à recourir aux métaphores pour décrire la structure interne du logiciel ou ses enjeux fonctionnels, de façon à faciliter la communication et à assurer une certaine homogénéité de style dans l'ensemble de la conception, l'idéal étant de décrire le système dans son ensemble par une métaphore unique.
- **Intégration continue** (*continuous integration*) : les développeurs synchronisent leurs développements aussi souvent que possible – au moins une fois par jour. Cela réduit la fréquence et la gravité des problèmes d'intégration, et permet de disposer à tout moment d'une version du logiciel qui intègre tous les développements en cours.

Les pratiques de gestion de projet

Les pratiques de programmation et de collaboration (voir chapitre 6) permettent de créer un contexte dans lequel une démarche de spécification et de conception purement itérative devient viable. Les pratiques suivantes montrent comment XP exploite cet avantage afin de s'assurer que l'équipe et son client restent en phase tout au long du projet, de façon à converger au plus tôt vers un produit adapté aux besoins du client :

- **Livraisons fréquentes** (*frequent releases*) : l'équipe livre des versions du logiciel à un rythme régulier, aussi élevé que possible – la fréquence précise étant fixée par le client. Cela permet à l'équipe comme au client de s'assurer que le produit correspond bien aux attentes de ce dernier et que le projet est sur la bonne voie.
- **Planification itérative** (*planning game*) : la planification du projet est réalisée conjointement par le client et l'équipe de développement, au cours de séances dédiées, organisées régulièrement tout au long du projet.
- **Client sur site** (*on-site customer, whole team*) : le client est littéralement intégré à l'équipe de développement pour arbitrer les priorités, et définir précisément ses besoins, notamment en répondant en direct aux questions des programmeurs et en bénéficiant du feedback immédiat d'une application aux livraisons fréquentes.
- **Rythme durable** (*sustainable pace*) : l'équipe adopte des horaires qui lui permettent de conserver tout au long du projet l'énergie nécessaire pour produire un travail de qualité et mettre en œuvre efficacement les autres pratiques.

Les quatre valeurs d'XP

Si elles sont indispensables aujourd'hui pour mettre l'Extreme Programming en œuvre, les pratiques que nous venons de présenter ne suffisent pas pour autant à le définir. Il ne s'agit en définitive que de *techniques*, destinées à faire émerger un environnement de travail marqué par les quatre qualités érigées en valeurs par XP et qui en font l'essence : la communication, la simplicité, le *feedback* et le courage.

La communication pour une meilleure visibilité

Du point de vue d'XP, un projet de développement est avant tout un effort *collectif* de création, dont le succès dépend d'une part de la capacité de ses différents intervenants à s'accorder sur une vision commune de ce qui doit être produit, et d'autre part de leur capacité à synchroniser leurs actions individuelles pour atteindre l'objectif commun. Or, ces deux conditions dépendent en majeure partie de la qualité de la communication qui lie ces intervenants entre eux. Mais sur le fond, il n'y a finalement pas grand-chose de spécifique à XP sur ce point.

Ce qui démarque l'approche XP dans ce domaine, c'est l'accent mis sur la communication directe, sur le contact humain. Certes, la communication orale présente des faiblesses en termes de structuration de l'information et de traçabilité, mais elle tire sa force de sa simplicité et des interactions rapides entre les interlocuteurs qui permettent de converger rapidement sur les informations essentielles. En outre, le contact direct permet de véhiculer des informations beaucoup plus personnelles – cela donne ainsi l'occasion d'identifier et de résoudre des blocages qui relèvent de positions individuelles, qui sans cela pourraient compromettre les chances de succès du projet.

Au quotidien, la communication directe permet donc d'obtenir une « bande passante » nettement supérieure à l'écrit, et XP exploite largement cet avantage dans la plupart de ses pratiques. Ainsi, qu'il s'agisse de l'identification des besoins, de la planification, de la répartition des tâches ou encore de la programmation proprement dite, la plupart des pratiques d'XP sont conçues pour amener les intervenants du projet à communiquer directement et résoudre les problèmes ensemble.

Remarque

Cette approche constitue l'une des forces majeures d'XP, mais c'est aussi d'un certain point de vue sa principale faiblesse. D'une part, ce besoin de communication est le principal facteur limitatif au regard de la taille de l'équipe et, d'autre part, cela rend XP très sensible au type de relations humaines qui ont cours dans le projet : appliquer XP dans un projet marqué par des antagonismes internes forts, c'est courir à la catastrophe – mais on peut légitimement se demander comment un tel projet peut bien réussir, indépendamment du processus adopté.

Cependant, si l'accent est placé sur la communication directe, la communication écrite n'est pas laissée de côté pour autant. Mais elle apparaît sous une forme différente : toutes les informations ayant trait à l'implémentation et la conception se retrouvent dans le code lui-même – dont la clarté fait l'objet de nombreux efforts – et dans la batterie de tests qui l'accompagne, et celles relatives aux besoins du client sont consignées d'une manière on ne peut plus formelle, sous la forme de tests de recette automatiques. Si le client souhaite obtenir des documents supplémentaires, il lui suffit de définir et de planifier les tâches correspondantes comme pour tout autre besoin.

La simplicité comme garantie de productivité

Une personne qui arrive sur un projet XP ne doit pas s'étonner de s'entendre répéter les quelques « mantras » suivants : « La chose la plus simple qui puisse marcher » (*The simplest thing that could possibly work*), « Tu n'en auras pas besoin » (*You ain't gonna need it* – et son acronyme YAGNI), ou encore « Une fois et une seule » (*Once and only once*). Tous trois sont l'expression d'une même valeur : la simplicité.

Cette simplicité touche d'abord à la réalisation même du logiciel, dans laquelle un soin particulier est apporté au code pour le débarrasser de toute complexité superflue. En particulier, les mécanismes de généricité – le « péché mignon » des développeurs – ne sont encouragés que lorsqu'ils servent un besoin concret et immédiat, et en aucun cas un besoin futur plus ou moins imaginaire. Les développeurs sont donc invités à implémenter *la chose la plus simple qui puisse marcher* et, en ce qui concerne ces fameux mécanismes génériques, ou encore ces outils « magiques » qui en font plus que ce qu'on leur demande, *ils n'auront pas besoin* ! En revanche, simple ne veut pas dire simpliste : les solutions adoptées doivent être aussi simples que possible, mais toutes les duplications doivent être éliminées de sorte que chaque information, chaque mécanisme ne soit exprimé qu'*une fois et une seule* – ce principe est garant de la facilité de modification du code sur le long terme.

Cet effort de simplicité s'applique également au client, à qui l'équipe demandera de définir ses besoins et ses priorités avec une grande précision pour éviter d'implémenter des choses inutiles et pour pouvoir se focaliser sur les besoins réellement importants. La simplicité est également recherchée dans le choix des outils (de programmation ou de gestion de projet) et dans la méthode de travail elle-même.

En définitive, XP est fondé sur un pari : « faire simple un jour, avec la possibilité de revenir en arrière le lendemain si un besoin différent apparaît. L'éventuel coût supplémentaire induit sera, d'une part réduit parce que l'application est restée assez simple pour évoluer facilement, et d'autre part sera bien peu de choses au regard des économies réalisées à ne pas faire ce qui n'aurait servi à rien. » C'est au succès de ce pari qu'une équipe XP doit sa vitesse de développement et son ouverture au changement.

Le feedback comme outil de réduction du risque

Malgré ce que peuvent laisser supposer son nom et ses apparences immédiates, l'Extreme Programming est avant tout un processus de réduction du risque dans le projet. Le risque est en effet soigneusement contrôlé à tous les niveaux, par la mise en place de boucles de *feedback* qui permettent à l'équipe de développement, comme à son client, de savoir à tout moment dans quel état se trouve réellement le projet, et de pouvoir rectifier le tir au fur et à mesure pour mener le projet à son terme avec succès.

L'exemple le plus saillant de ce principe concerne la pratique des livraisons fréquentes, qui donne à tous les intervenants du projet un feedback régulier sur l'état d'avancement du projet et l'état réel du produit. Mais le feedback ne se borne pas à l'observation : la pratique de la planification itérative permet de tirer parti des informations recueillies pour, à la fois, améliorer la planification elle-même et faire converger le produit vers une solution mieux adaptée aux besoins réels du client.

L'activité de programmation fait également l'objet de divers mécanismes de feedback, tout d'abord à travers les tests unitaires mis en place, qui donnent aux développeurs des indications immédiates sur le fonctionnement du code qu'ils écrivent. Le feedback est également intégré à l'activité de conception, qui est menée tout au long de la réalisation pour s'appuyer sur l'état réel de l'application plutôt que sur l'idée que l'on pourrait en avoir *a priori*. Enfin, les développeurs s'appuient en permanence sur le feedback de leur binôme pour s'assurer de la validité et de la qualité du code qu'ils produisent.

Ce feedback permanent est un facteur de qualité, puisque les intervenants du projet améliorent sans cesse leur travail à partir de l'expérience qu'ils accumulent. Mais c'est également un facteur de vitesse : une équipe XP peut programmer vite, tout en restant sereine parce qu'elle sait qu'elle dispose de nombreux mécanismes pour s'assurer que le projet reste sur la bonne voie.

Le courage de prendre les bonnes décisions

L'expérience fait apparaître que la mise en œuvre des pratiques XP requiert une certaine dose de cran. En effet, il faut du courage pour se lancer dans un projet sans avoir au préalable tout spécifié et conçu dans le détail, même si l'on sait que le processus suivi comporte de nombreux mécanismes de feedback.

Il faut également du courage pour se borner à réaliser des choses simples, se focaliser uniquement sur les besoins du moment en se disant qu'on pourra adapter l'application à de nouveaux besoins, le moment venu. Il en faut aussi pour accepter de jeter une partie du code qui est devenue inutile, ou récrire une partie de code jugée trop complexe.

Enfin, il faut du courage pour appliquer les principes de communication et de feedback, en particulier lorsqu'il s'agit de maintenir une transparence complète, même lorsque les nouvelles ne sont pas bonnes, ou encore lorsqu'il s'agit de travailler ouvertement avec son binôme en acceptant de lui montrer nos propres limites ou lacunes.

Plus généralement, nous avons pu constater qu'il faut du courage pour mettre en place des méthodes nouvelles sur des projets dont les enjeux sont toujours stratégiques.

Racines historiques et acteurs d'XP

XP est principalement l'œuvre de Kent Beck et Ward Cunningham, deux experts du développement logiciel, et plus particulièrement de la conception objet et des *patterns*. Avant le développement d'XP, Kent Beck devait surtout sa notoriété dans le monde Smalltalk à son livre, *Smalltalk Best Practice Patterns*.

La naissance d'XP

La naissance d'XP a eu lieu sur un projet Chrysler – le «C3» (*Chrysler Comprehensive Compensation System*) – où Kent Beck intervenait initialement pour travailler sur l'amélioration des performances du produit. Après avoir convaincu la direction que le produit souffrait de maux plus profonds, Kent Beck s'est vu confier la responsabilité d'encadrer l'équipe pour le récrire. C'est là qu'il a mis au point les pratiques d'XP, avec l'aide d'un autre acteur important d'XP : Ron Jeffries.

Le projet C3 fut réalisé en Smalltalk ; si, pour de nombreux observateurs, la démarche XP s'est largement affranchie du langage et s'applique aussi bien à Java ou C++, XP conserve de fortes affinités, en particulier, avec Smalltalk[1] et, d'une façon plus générale, avec la famille des langages «dynamiques».

Parmi les autres personnages influents d'XP, on trouve notamment Martin Fowler, auteur de livres sur l'UML et plus récemment du livre de référence sur le Refactoring (l'une des pratiques de base d'XP). On y rencontre également Robert C. Martin, président d'Object Mentor, une société de consultants de haut niveau spécialisés dans la conception objet.

XP est également le fruit d'un effort collectif supporté par le Wiki Wiki Web[2], un site collaboratif mis au point par Ward Cunningham. Les échanges menés sur ce site ont permis aux auteurs d'XP d'affiner la définition et la présentation de leur démarche, avant de pouvoir la rendre accessible au reste de l'industrie à travers des ouvrages dédiés.

D'une certaine manière, on a pris acte de la naissance d'XP en octobre 2000 avec la parution du livre *Extreme Programming Explained* de Kent Beck, dans lequel il expliquait les fondements de sa démarche. Les livres *Extreme Programming Installed* et *Planning Extreme Programming* ont suivi peu après, donnant un éclairage résolument plus pratique et concret de la démarche.

1. Voir à ce propos l'ouvrage *Squeak* de Xavier Briffault et Stéphane Ducasse, Eyrolles, 2002.
2. http://www.c2.com/cgi/wiki?ExtremeProgrammingRoadmap