

Les traducteurs agiles

[FAQ](#) [Traducteurs](#) [Traductions](#) [Traductions par catégories](#)

Utiliser les développements incrémental et itératif ensemble

Publié le 16 - 11 - 2009

Le développement incrémental est nettement différent du développement itératif dans ses objectifs ainsi que dans ses implications dans la gestion de projet. Les équipes rencontrent des problèmes en faisant l'un et pas l'autre, ou en essayant de les gérer de la même manière. Cet article explique leurs différences et comment les utiliser ensemble.

Les développements incrémental et itératif sont antérieurs au mouvement Agile. Je les ai d'abord étudiés lors de mes recherches pour le groupe IBM Consulting en 1991. A cette époque, j'ai appris comment ils différaient en termes d'objectif et de nature, et, finalement, comment il fallait les utiliser.

Ces différences semblent avoir été oubliées depuis. Je vois maintenant des équipes qui souhaiteraient faire de l'Agile et qui souffrent de ne faire que du développement incrémental, là où j'avais l'habitude de voir des projets, basés sur le modèle en cascade, souffrant de ne faire aucun des deux ou seulement du développement itératif.

Les deux sont nécessaires. Les gens ont besoin d'apprendre à les utiliser séparément et ensemble.

Définitions, s'il vous plaît !

Brièvement :

- Le développement incrémental est une stratégie par étape et planifiée dans lequel différentes parties du système sont élaborées à différents moments ou degrés et intégrées dès qu'elles sont terminées. La stratégie alternative au développement incrémental est de développer l'ensemble du système avec une intégration "big-bang" à la fin.
- Le développement itératif est une stratégie de remaniement (~rework) planifié dans lequel du temps est réservé pour réviser et améliorer certains aspects du système. La stratégie alternative au développement itératif est de prévoir d'avoir tout bon du premier coup.

Il est important de remarquer qu'aucune stratégie ne suppose, requiert, ou implique l'existence de l'autre. Il est possible de n'en utiliser qu'une seule, les deux, ou aucune.

Dans la pratique, il est conseillé d'utiliser les deux à doses raisonnables. Si vous faites seulement de l'incrémental, il pourra y avoir une surprise désagréable à la fin quand la qualité ne sera pas au rendez-vous. Si vous faites de l'itératif sur l'ensemble du système, les effets dominos dus aux nombreux changements pourront facilement vous faire perdre le contrôle du projet.

Ce n'est pas le modèle en cascade

Tout d'abord, nous avons besoin d'éviter de tomber dans le piège du "ça ressemble au modèle en cascade".

Dans tout développement, qu'il s'agisse de prototypage, Agile, en spiral, ou en cascade, nous décidons d'abord de ce qu'il faut construire; nous concevons et programmons ensuite quelque chose. Seulement, après avoir fait un peu de programmation (généralement plus que ce que nous avons décidé initialement), nous nous préparons à déboguer le système. C'est seulement une fois que le système fonctionne que l'on peut valider le fait que ce que nous avons construit est la bonne chose et que c'est correctement construit. Ce cycle est illustré à la figure 1.

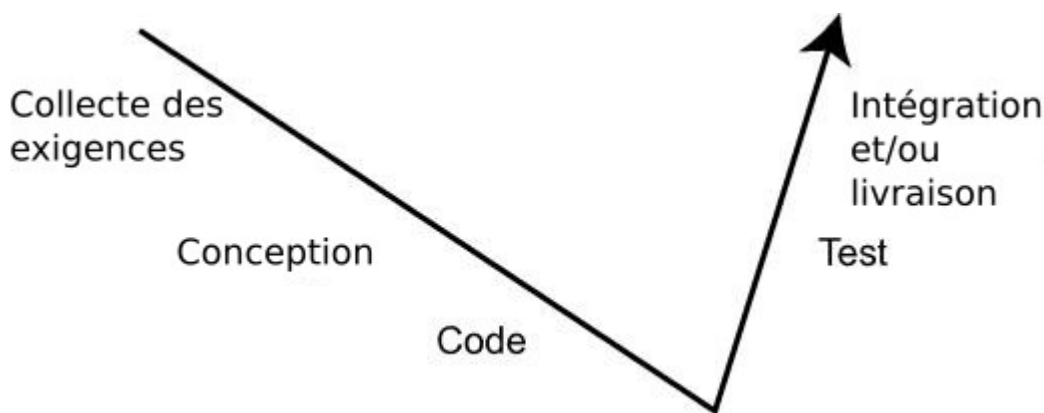


Figure 1: Le

mode de validation en V est un acte normal de la vie.

La figure 1 devrait vraiment être lue à l'envers, comme un diagramme de dépendance : nous ne pouvons pas livrer tant que nous n'avons pas débogué et validé; nous ne pouvons pas déboguer tant que nous n'avons pas codé; nous ne pouvons pas coder tant que nous n'avons pas conçu; nous ne pouvons pas concevoir tant que nous n'avons pas décidé quoi concevoir.

En d'autres termes, la validation en V est un acte normal de la vie, et nous aurons à le traiter aussi bien dans le développement incrémental qu'itératif.

Développement incrémental

Dans le développement incrémental, nous découpons les tâches en petits morceaux et les planifions pour être développées au fil du temps et intégrées dès qu'elles sont terminées. Les figures 2 à 4 illustrent ce cycle.

Imaginez que les premiers blocs du dessus représentent divers composants de l'interface utilisateur, les blocs du milieu représentent le middleware, et les blocs du bas représentent les composants du backend ou de la base de données.

La figure 2 montre que, lors du premier incrément, un ensemble complet de fonctionnalités est construit de l'interface utilisateur (IHM) jusqu'au backend (et dans ce cas, des morceaux supplémentaires de l'interface utilisateur sont également construits). Dans le second incrément (figure 3), nous voyons que la fonctionnalité supplémentaire qui est ajoutée concerne toutes les couches du système. Cela peut constituer un stade d'avancement suffisant pour déployer le système tel qu'il est pour les utilisateurs finaux et commencer à en faire bénéficier le métier. Dans le troisième incrément (figure 4), le reste du système est complété.

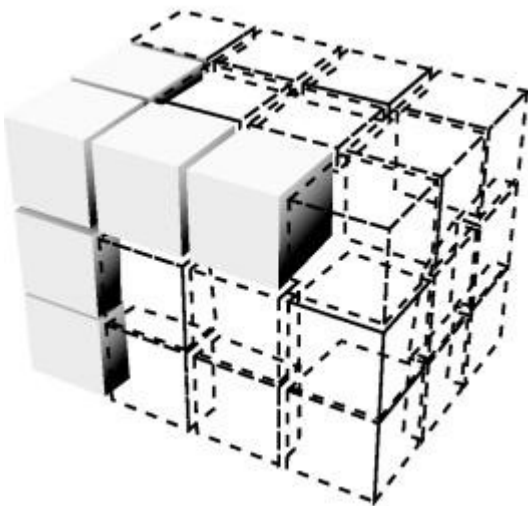


Figure 2 : Développement incrémental – Niveau 1

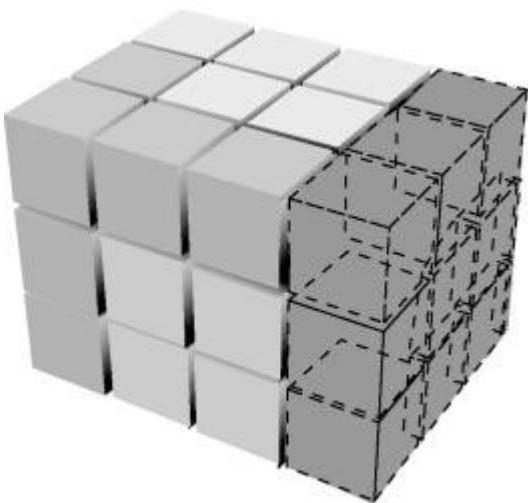


Figure 3 : Développement incrémental – Niveau 2



Figure 4 : Développement incrémental – Niveau 3

Le modèle qui vient d'être décrit est celui utilisé par la plupart des projets actuels, Agile ou non. C'est une stratégie par étape qui a connu beaucoup de succès.

L'erreur que font aujourd'hui les gens c'est d'oublier d'itérer. Ils ne prennent pas le temps d'apprendre à partir de ce qu'ils ont mal compris quand ils ont décidé de ce qu'il fallait construire au tout début et de réfléchir à ce qui devait être amélioré dans la phase de conception.

Cette erreur aboutit au classique échec de livrer des choses que les gens ne veulent pas. Je tiens à souligner que, même de nombreuses équipes de projet Agile commettent cette erreur.

La stratégie corrective est le développement itératif.

Développement itératif

Avec le développement itératif, nous réservons du temps pour améliorer ce que nous avons.

Les spécifications et les interfaces utilisateurs sont historiquement connues pour être génératrices de tâches d'adaptation, mais ce ne sont pas les seules. La technologie, l'architecture et les algorithmes sont également susceptibles d'avoir besoin d'inspection et d'adaptation. La sous-performance est souvent mal anticipée dans les premières phases de conception, et nécessite une grosse adaptation de l'architecture.

Si l'on regarde le modèle de validation en V, la différence est qu'au lieu d'intégrer et peut-être même de livrer le logiciel à la fin du cycle, on "l'examine" sous divers angles : était-ce la bonne chose à développer ? les utilisateurs apprécient-ils la façon dont cela fonctionne ? est-ce que cela fonctionne assez rapidement ?

La figure 5 montre le modèle de validation en V pour un cycle de développement itératif.

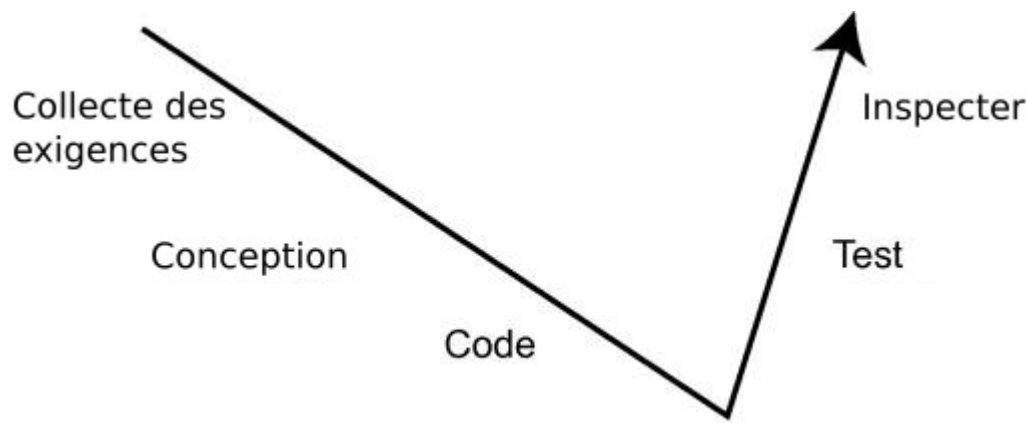


Figure 5: Modèle de validation en V pour un cycle itératif

Il y a deux stratégies de remaniement spécialisées et particulières :

- Développez le système aussi bien que possible en ayant à l'esprit que, si c'est suffisamment bien fait, les modifications seront minimales et pourront être rapidement intégrées.
- Développez le moins possible avant que cela soit soumis à évaluation, en ayant à l'esprit que vous aurez limité le gaspillage lorsque vous aurez la bonne information.

Il y a des aficionados des deux approches. En effet, les deux fonctionnent bien dans certaines circonstances. Un chef de projet doit apprendre à utiliser les deux.

Ce qui suit est une utilisation efficace de la première stratégie : un musicien et un photographe ont réalisé un DVD ensemble. Le musicien a fait un enregistrement de quatre minutes. Le photographe a remarqué qu'un petit ensemble de transitions entre les diapositives ne correspondait pas à la musique. Le musicien a uniquement ré-enregistré ces parties et les a insérées dans l'enregistrement.

Pour démontrer une utilisation efficace de la deuxième stratégie, j'adapte l'exemple de Jeff Patton concernant la peinture de la Joconde et en imaginant un débat entre Léonard de Vinci et son client (figures 6 à 8).

Léonard dessine une esquisse de ce qu'il compte faire (figure 6) et se dirige vers son client en demandant : "Est-ce que ça va vous aller ?"



Figure 6 : Développement itératif de la Joconde, étape 1

Le client répond : “Non, non, non. Elle ne peut pas regarder à droite, elle doit regarder vers la gauche !”. Heureusement, Léonard n’a pas trop travaillé pour le moment, donc c’est facile à changer.

Léonard s’en va, renverse l’image, rajoute un peu de couleur et des détails (figure 7). Il revient voir le client : “J’en ai à peu près réalisé le tiers. Qu’en pensez-vous maintenant ?”



Le client répond : “Non, vous ne pouvez pas lui faire une tête de cette taille ! Proportionnez sa tête avec le reste de son corps.” (Oui, ils avaient l’équivalent de Photoshop et des options de retouche d’image à l’époque – cela s’appelait “Léonard”).

Léonard s’en va, termine le tableau (Figure 8) et envoie sa facture au client.



Figure 8 : Développement itératif de la Joconde, étape 3

Le client dit : “Vraiment, j’aurai préféré qu’elle ait les yeux plus gros, mais d’accord, je vous paye, disons-que c’est terminé.”

Ce que je voudrais souligner c’est que les deux stratégies sont valides et les deux portent l’étiquette “itératif”. Dans les deux cas, des remaniements ont été effectués sur une partie existante du système.

Fusionnez les deux

Les développements incrémental et itératif s’associent bien ensemble. En se basant sur le modèle de validation en V, nous pouvons le réorganiser en alternant des Vs “incrémental” avec des Vs “itératif” de diverses manières pour obtenir un nombre varié de stratégies mixant de l’itératif et de l’incrémental comme l’illustre la figure 9.

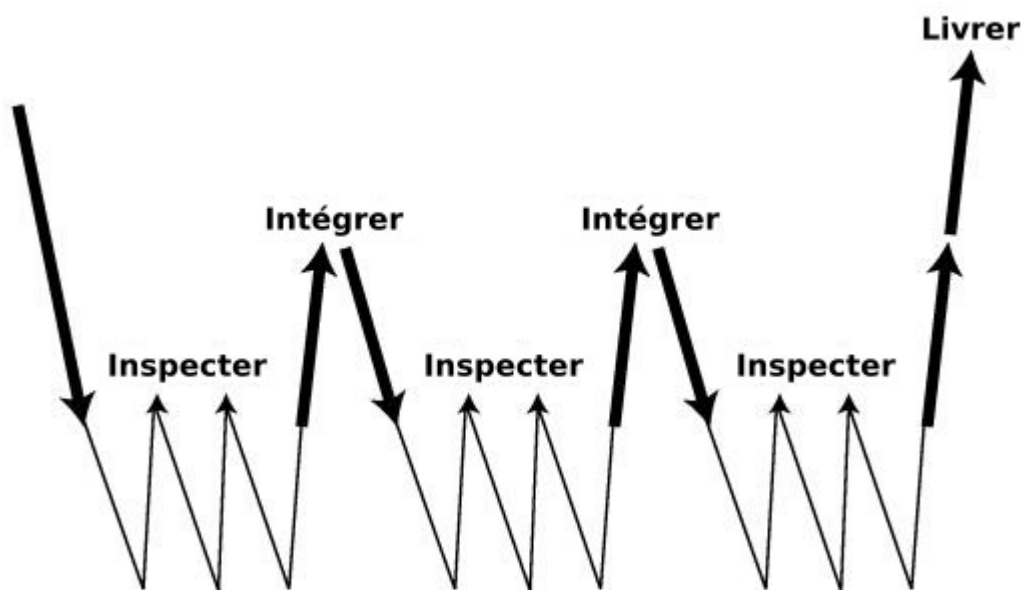


Figure 9 : Associer les développements itératif et incrémental

La figure 9 montre une stratégie dans laquelle chaque cycle incrémental comprend deux étapes d’examen/adaptation du système avant que le résultat soit intégré et prêt à livrer. La figure montre trois phases de développement incrémental, chaque incrément est intégré une fois terminé, le tout est ensuite prêt à être déployé.

Ceci est un, mais seulement un, des moyens d’associer les deux. Tant qu’ils sont clairement dans l’état “Examen” ou “Prêt à être livré” (ou “Prêt à être déployé”, ce qui est encore mieux), les Vs peuvent être combinés dans tous les sens.

La figure 9 offre un avantage supplémentaire en décrivant les incréments et itérations sous forme de Vs : le diagramme obtenu peut facilement s’insérer dans un calendrier. Chaque marque “Examinez”, “Intégrez” ou “Livrez” est un jalon dans le planning du chef de projet. Cela permet au chef de projet de visualiser à l’avance et de maîtriser le temps consacré aux adaptations. De cette façon, nous avons mis à profit le modèle de validation en V avec notre stratégie de développement incrémental-itératif.

Gérez les deux

A première vue, les deux se ressemblent. Pourtant, ils doivent être gérés différemment. Les incréments sont faciles à repérer, facile à séparer, relativement facile à estimer, et facile à planifier. Toute la stratégie peut être résumée en deux étapes :

- Divisez le système en fonctionnalités complètes et utiles (ou toute autre façon de découper le système, cela reste votre choix).
- Réalisez-les les unes après les autres.

Les itérations sont considérablement plus difficiles. Elles sont difficiles à séparer, difficiles à estimer, et difficiles à planifier. Bien sûr, être difficile ne signifie pas que vous n'avez pas à faire toutes ces choses. Vous devez les faire, et répondre aux trois questions suivantes :

- Quels éléments doivent bénéficier de périodes d'adaptations planifiées ?
- Combien de périodes d'adaptations chaque élément nécessite-t-il ?
- Combien de temps doit durer chaque période d'adaptation ?
- Bien qu'il n'y ait pas de réponse simple et fiable à ces questions, il existe une stratégie simple que vous pouvez librement adapter pour votre projet.
- Planifiez de façon certaine l'adaptation des interfaces utilisateurs, planifiez l'adaptation des exigences au moment où les utilisateurs finaux commencent à utiliser le système, et attendez-vous à ce que l'architecture soit remise en question pour raison de sous-performance et adaptée.
- Allouez deux périodes d'adaptation pour la conception de l'interface utilisateur, et une de plus pour l'évolution des exigences et l'adaptation de l'architecture.
- Allouez pour la première période d'adaptation le tiers du temps de développement initial, et pour la deuxième période la moitié de cela.

Vous aurez besoin d'estimer et tester vos propres ratios, mais ceux que j'ai proposé ne doivent pas être si mauvais que ça pour une première estimation.

Si vous faites des développements Agile avec Scrum ou eXtreme Programming, assurez-vous que toutes les user stories ou cartes du backlog passent à travers le sprint avec un coefficient multiplicateur de trois sur leurs estimations de taille.

Trois histoires

Pour finir, je vous raconte trois histoires de développement incrémental / itératif qui se sont bien ou mal passées. Le premier, le projet "Baker", met en jeu de l'itératif mélangé avec de l'incrémental, sachant qu'ils auraient dû faire de l'incrémental lorsqu'ils faisaient de l'itératif. Le second, le projet "Laddie", témoigne du problème des projets modernes Agile qui font de l'incrémental sans itératif. Le dernier projet, "Winifred", s'est bien déroulé.

Le projet Baker était à coût et périmètre fixés et comptait 200 personnes. Elles travaillaient selon un cycle mensuel (une stratégie de développement incrémental).

Les équipes étaient séparées et travaillaient selon un mode pipeline de telle façon que les spécificateurs rédigeaient pendant 1 mois avant de passer leurs livrables aux concepteurs le début du mois suivant. Un mois plus tard, les concepteurs transmettaient leurs livrables aux programmeurs qui développaient pendant un mois. À la fin, les testeurs recevaient des morceaux de code à tester et intégrer.

En se méprenant sur le terme “développement itératif”, ils ont ensuite donné des instructions à tous comme quoi les spécifications et la conception pouvaient changer à n’importe quel moment (ceci constituait leur stratégie de développement itératif).

Le tohu-bohu qui a suivi est exactement ce que à quoi vous pensez. Chaque mois, les spécificateurs révisaient des parties des spécifications, qui changeaient des parties de la conception et des développements. Dès le troisième mois, il était évident pour les développeurs qu’ils étaient en train de programmer un système qui était déjà en cours de modification chez les concepteurs qui eux-mêmes étaient simultanément conscients qu’ils étaient en train de concevoir un système qui était déjà en cours de modification chez les spécificateurs. Les testeurs n’ont jamais rien reçu qui corresponde.

Le projet Baker était en difficulté dès le départ, en partie à cause à cette stratégie de pipeline, mais plus encore à cause de la non maîtrise de chaque itération.

Penchons-nous maintenant sur un cas d’échec Agile.

Le projet Laddie utilisait une approche Agile avec des itérations de deux semaines. Toutes les user stories étaient répertoriées dans une longue liste et réalisées à la fin de chaque itération (c’était leur stratégie de développement incrémental). À la fin de chaque itération, on montrait au client ce qui avait été construit durant ces deux semaines. Bien entendu, deux semaines étant un délai très court, il n’y avait jamais assez de temps pour montrer au client ce qui avait été conçu et il y avait donc généralement des corrections à apporter.

Le client avait le choix de retarder la réalisation de nouvelles user stories afin de corriger les erreurs commises, ou bien alors de repousser ces corrections en fin de backlog (c’était leur stratégie de développement itératif – pas très agréable du point de vue du client).

Le client s’était plaint après un moment qu’il sentait qu’il fallait bien faire les choses dès la première fois puisque les choix qu’on lui laissaient sur le comment et quand il faudrait corriger les erreurs n’étaient pas très agréables. Ceci, avait-il correctement estimé, violait l’esprit même du développement Agile.

Terminons avec une histoire heureuse.

Le projet Winifred était à prix, périmètre et délai fixé, 18 mois avec une pointe à environ 45 personnes. Le cycle de développement était de trois mois, chaque fin de cycle entraînant un déploiement (c'était la stratégie des petits pas).

Il n'y avait pas de stratégie incrémentale particulière requise au sein de chaque cycle de développement. Les équipes pouvaient développer les fonctionnalités dans l'ordre qu'elles souhaitaient. Par contre, chaque équipe devait montrer ses travaux en cours aux utilisateurs finaux au minimum deux fois par cycle, de sorte que les utilisateurs pouvaient modifier ou corriger ce qui était en cours de construction (ceci était leur stratégie de développement itératif). Cela devait être le logiciel réel, de l'interface utilisateur jusqu'à la base de données, et non de simples maquettes d'écran bouchonnées.

Typiquement, chaque équipe montrait aux utilisateurs ce qu'ils avaient construit après six semaines de travail et de nouveau après huit semaines de travail. Lors de la première rencontre utilisateur, peut-être 60 à 80% des fonctionnalités étaient complètes. Les utilisateurs avaient le droit de changer quoi que ce soit qu'ils n'aimaient pas, y compris, "je sais que c'est ce que j'ai dit que je voulais, mais maintenant que je le vois, ce n'est pas du tout ce que je veux."

À la deuxième rencontre, peut-être 90 à 95% des fonctionnalités étaient complètes, et les utilisateurs avaient seulement le droit de faire corriger des erreurs flagrantes et de demander des petits ajustements. Cela a permis de corriger à la fois les spécifications et les interfaces utilisateurs tout en conservant encore un sens pour un contrat au forfait.

Le projet Winifred a été déployé avec succès et les utilisateurs ont obtenus plus ou moins ce qu'ils voulaient. Le système est encore utilisé et est encore maintenu dix ans plus tard, ce qui est un bon indicateur de réussite.

Notez bien que la stratégie combinant itératif et incrémental du projet Winifred respecte le style de l'histoire de la Joconde citée précédemment.

Résumé

Le terme "incrément" signifie à la base "ajouter".

Le mot "itérer" signifie à la base "refaire".

Malheureusement, le "développement itératif" renvoie aujourd'hui aussi bien à incrémental qu'itératif, sans ne faire aucune différence. C'est malheureusement regrettable pour notre industrie logicielle puisque chacun a un objectif différent et doit être géré différemment.

Le développement incrémental vous donne la possibilité d'améliorer votre processus de développement, ainsi que d'ajuster les exigences à l'évolution de l'environnement.

Le développement itératif vous aide à améliorer la qualité de votre produit. Oui, cela veut dire remanier (~rework), et oui, vous aurez probablement besoin d'en faire un peu pour obtenir un produit propre.

Le processus de développement, les fonctionnalités et la qualité du produit ont besoin d'être constamment améliorés. Utilisez une stratégie incrémentale, après réflexion, afin d'améliorer les deux premiers axes. Utilisez une stratégie itérative, après réflexion, afin d'améliorer le troisième axe.

Auteur : [Alistair Cockburn](#)

Source : [Using Both Incremental and Iterative Development](#)

Date de parution originale : Mai 2008

Traducteur : [Fabrice Aimetti](#)

Date de traduction : 16/11/2008




Ce(tte) oeuvre est mise à disposition selon les termes de la [Licence Creative Commons Attribution - Pas d'Utilisation Commerciale - Partage dans les Mêmes Conditions 4.0 International](#).

Partager cette traduction sur vos réseaux sociaux favoris

[Twitter](#) [Facebook](#) [Google+](#) [Pinterest](#) [LinkedIn](#) [Tumblr](#) [Reddit](#) [Viadeo](#)

Les traducteurs agiles

Les traducteurs agiles
contact@les-traducteurs-agiles.org

 [les-traducteurs-agiles](#)

 [traducteuragile](#)

Partager avec chacun la connaissance
acquise auprès de chacun