Chapitre 12

Processus et méthode

La gestion d'une équipe informatique à distance demande bien évidemment plus de rigueur que celle d'une équipe locale. Pour que cette approche puisse être appliquée simplement, il faut l'accompagner d'un outillage approprié pour organiser, contraindre et suivre le travail réalisé à distance.

La mise en place d'un processus de développement structuré est souvent intimidante. On se retrouve en face d'une forêt de tâches, de rôles, de documents et de flux complexes, dont on ne perçoit pas immédiatement la portée. La seule exploration des processus n'est pas à la portée de tous. Les guides qui accompagnent ces méthodologies ne sont réellement efficaces que si l'on a l'expérience qui convient pour les appliquer intelligemment. L'application à la lettre des directives d'installation mènerait à un échec certain, car elles prévoient tous les cas de figure et sont d'une lourdeur excessive.

Les méthodologies formelles sont assez nombreuses. La plupart des grands éditeurs de logiciels ont créé la leur, et certains l'incluent dans leurs offres de produits ou la vendent en la mettant plus ou moins en avant. Les deux approches méthodologiques qui se détachent assez nettement sont le RUP (Rational Unified Process) d'IBM *(www-306.ibm.com/software/awdtools/rup/)* et XP (eXtreme Programming) *(www.extremeprogramming.org/).* Microsoft propose avec MSF (Microsoft Solutions Framework) son propre cadre de travail, qui complète sur de nombreux sujets les approches du RUP ou de XP en insistant davantage sur la gestion des hommes et les flux de travail que sur la structuration des processus et documents *(www.microsoft.com/msf/).*

Ce chapitre se penche sur les quelques éléments qui doivent impérativement être implémentés lorsqu'on travaille avec l'offshore. La plupart des méthodologies récentes appliquent ces recommandations, bien qu'en les présentant de façons assez différentes.

La méthodologie et les hommes

La méthodologie ne garantit pas un résultat indépendamment des équipes en place. Comme expliqué au chapitre 11, le succès d'un projet est avant tout conditionné par la qualité des hommes, leurs interactions, leur management et leur motivation.

Plus le projet est important, plus la méthodologie est indispensable, car il faut harmoniser le travail entre différentes équipes et gérer des masses importantes d'informations. Lorsque le projet est très petit, les efforts nécessaires pour déployer des procédures industrielles prennent un poids exagérément important par rapport aux tâches de production, et l'on privilégie des procédures légères. Les qualités humaines font toute la différence dans ces projets légers.

Les méthodes essaient de prendre en compte toutes les étapes d'un projet, depuis les premiers moments, quand le projet prend forme, jusqu'aux évolutions et corrections du produit final en exploitation et à la préparation du projet suivant.

La mise en place d'une méthodologie est souvent dépersonnalisante. Elle vise à structurer les échanges et la traçabilité et à centraliser les responsabilités. La plupart des outils méthodologiques utilisent la gestion de *work orders*, qui attribuent des tâches unitaires à des personnes d'une façon plus ou moins automatique. Les développeurs qui découvrent leurs tâches dans ces outils méthodologiques, assorties de dates de livraison, ont l'impression de n'être que des exécutants, qui ne décident de rien. Il est donc important de respecter les motivations, engagements, initiatives personnelles et responsabilités que chacun apporte aux tâches qu'il réalise.

Cela peut paraître évident, mais lorsqu'on met en place une méthodologie, on tend à s'imprégner de tous les concepts, et l'on a tendance à oublier le facteur humain. Il est important de ne demander aux collaborateurs d'engager leur pleine responsabilité que s'ils ont pu étudier et accepter les tâches à réaliser et s'ils ont effectivement la possibilité de mener à bien les réalisations. Un collaborateur que l'on tient pour responsable de tâches sur lesquelles il ne s'est pas engagé personnellement ou pour lesquelles il n'a pas de possibilité d'agir se désintéresse de ses objectifs.

Si certains outils méthodologiques peuvent mener à des effets négatifs sur la gestion des ressources humaines, ces mêmes outils utilisés correctement peuvent aider à structurer l'organisation des workflows et l'utilisation du référentiel tout en conservant les motivations et responsabilités des collaborateurs.

Procédures et outils méthodologiques n'ont de sens que pour accompagner les collaborateurs dans la structuration de leur travail et dans les échanges d'informations. On doit donc mettre en place ces méthodes et outillages avec la volonté de préserver l'initiative, la responsabilisation et l'engagement personnels des collaborateurs.

> **EN RÉSUMÉ**
> **Outils méthodologiques et motivation**
>
> Les outils méthodologiques peuvent mener à une gestion des ressources humaines réduisant les collaborateurs à des rôles d'exécutants. La motivation et l'engagement des collaborateurs sont essentiels pour garantir le succès et la qualité des réalisations. Il faut donc veiller à conserver ces valeurs dans le management des ressources humaines lors de la mise en place des procédures.

Un autre piège des méthodologies est la façon dont elles présentent les rôles. Un rôle assure un certain nombre de tâches ciblées. Le nom des rôles est souvent trompeur, car certains d'entre eux correspondent à des postes (par exemple, développeur), tandis que d'autres se rapprochent d'une activité (par exemple, *test designer*), qui doit être assurée. Les rôles peuvent être très précis, comme *test reviewer*, même si cela ne correspond pas à un poste à temps plein.

Les collaborateurs qui ont peu d'expérience des descriptions des rôles dans les méthodologies les associent à tort à des postes à temps plein. Certains rôles semblent en outre être plus valorisés que d'autres. Ainsi, l'analyste semble plus gradé que le développeur et le *test designer* que le testeur. C'est un piège qu'il faut éviter dès le commencement. Les rôles présentés dans les méthodes ne correspondent pas nécessairement à un poste. Il s'agit plutôt de fonctions consistant à effectuer certaines tâches et à prendre la responsabilité de certains livrables. Par ailleurs, les méthodologies n'expliquent pas comment déterminer l'attribution des rôles aux postes ni si certains rôles peuvent être cumulés par une même personne sans être dénaturés.

Le chapitre 11 montre comment construire des équipes efficaces. Sur les postes identifiés, on doit allouer des rôles à la méthodologie. Les rôles d'analyste et de développeur, par exemple, sont souvent alloués aux mêmes personnes afin de ne pas répéter les erreurs des débuts de l'informatique, quand l'analyste concevait le programme et le pupitreur entrait le code. Il est peu motivant pour un développeur de recevoir un modèle de design entièrement conçu par une personne qui ne codera pas et de se contenter de réaliser un code, bien souvent en critiquant le travail fait en amont. De même, l'analyste qui conçoit ses modèles d'analyse sans coder une seule fonction ne peut travailler ses modèles avec le même soin que s'il sait devoir les coder.

> **EN RÉSUMÉ**
> **Rôles, postes et méthodologies**
> Les méthodologies attribuent des rôles précis à des activités et à des livrables. Il se peut que des collaborateurs s'associent à des rôles choisis pour leur aspect valorisant et intéressant correspondant plus ou moins à leur rôle courant. Les rôles des méthodologies ne sont pas des postes, et un poste cumule souvent plusieurs rôles.

Choix de la méthodologie et des outils

Les outils permettent d'améliorer les communications avec l'offshore en rendant transparents les workflows, les livraisons et la progression du projet. Ce chapitre ne prétend pas aider à choisir la méthodologie ou les outils qui doivent être utilisés pour travailler avec des équipes en offshore. Il propose simplement un tour d'horizon des fonctionnalités les plus utiles pour les projets en offshore parmi celles offertes par la plupart de ces outils de développement.

En tout premier lieu, il importe de choisir une méthodologie qui utilise une approche itérative et qui est correctement accompagnée de l'outillage permettant d'en appliquer, voire d'en forcer les principes essentiels.

La méthode

La méthode peut se présenter sous la forme d'un produit packagé et versionné, pour les méthodologies qui sont gérées comme des produits, tel le RUP, ou sous forme d'archive compressée à télécharger sur Internet. Qu'elle soit livrée sur un CD ou disponible sur le Web, elle doit être adaptée aux besoins du projet.

La solution la plus simple pour installer la méthode consiste à créer un intranet à partir du site brut proposé par la méthodologie et de le personnaliser selon les activités, livrables et workflows que l'on souhaite mettre en œuvre.

Les méthodes couvrent le plus souvent un spectre beaucoup plus large de situations que celles que l'on rencontre dans un projet réel. Certains flux, comités ou documents peuvent n'avoir qu'une utilité réduite dans le cadre d'un projet. Il faut donc adapter le cadre méthodologique afin de ne déployer que les éléments utiles, tout en respectant les principes de base.

La méthodologie elle-même doit être mise en place de façon itérative et permettre de revenir sur des éléments dont la définition ne serait pas pleinement efficace et que les collaborateurs voudraient améliorer.

Le référentiel

Le référentiel est l'outil le plus important de l'outillage méthodologique. Il reçoit tous les éléments qui sont créés, modifiés ou utilisés dans le cours du projet. On trouve aussi bien des textes, contenant des informations générales sur le projet, que des documents méthodologiques, des codes source, des scripts, des plans de test, etc. On peut y placer les différents jeux de données de test, ainsi que tous les outils qui permettent de reconstituer l'intégralité de l'environnement de développement et de déploiement.

Le référentiel permet de conserver le versionnement de tous les éléments que l'on y place. On peut ainsi retourner à des versions antérieures et savoir qui a effectué des modifications et quand.

C'est tout particulièrement sur la gestion du code source que l'on trouve les mécanismes les plus évolués. Le référentiel peut être lié aux autres outils qui sont mentionnés dans la suite du chapitre et qui permettent de gérer des informations sur le changement, maintenant ainsi non seulement les différentes versions des éléments mais les changements opérés sur ceux-ci. De cette façon, l'utilisateur extrait un code source du référentiel pour effectuer un ou des changements (correction d'anomalie, évolution fonctionnelle, nouvelle fonctionnalité, etc.) qui sont demandés dans l'outil de gestion du changement et le replace dans le référentiel une fois le changement effectué.

Le premier objectif de la gestion de référentiel est de gérer les extractions pour modifications et l'insertion de l'élément modifié. Le programmeur peut bloquer un élément pour éviter que plusieurs personnes ne travaillent dessus en même temps sans le savoir. L'outil sait aussi gérer la fusion de modifications simultanées si ces dernières sont situées dans des parties différentes du code. Il peut tout aussi bien demander au programmeur de résoudre les recouvrements de modifications.

Pour créer une version d'un produit, on identifie des *baselines*, qui sont un ensemble d'éléments versionnés qui doivent être extraits simultanément pour être compilés. Ces éléments doivent être compatibles entre eux et s'assembler correctement. Par exemple, si l'on fait évoluer une interface technique, les anciens programmes qui utilisent l'interface précédente ne fonctionnent plus. Il faut donc attendre que les programmes aient été retravaillés pour prendre en compte la nouvelle interface afin qu'ils fonctionnent à nouveau. La *baseline* identifie les versions des éléments qui sont compatibles avec la nouvelle interface afin de les compiler et créer un produit exécutable.

Une *baseline* peut être « promue » selon l'état de test et de stabilité du produit exécutable. La *baseline* peut ainsi correspondre à une version en cours de développement (très

instable), en cours de test unitaire (instable), à intégrer pour test (boguée), testée (stable) et recettée (très stable), etc., selon le cycle de promotion choisi par le client.

Un autre mécanisme de base du référentiel est la gestion des streams. Un stream est un chemin de développement sur lequel on peut agir indépendamment des autres chemins. À partir d'un développement, on peut souhaiter figer une version pour la stabiliser, par exemple. Les anomalies seront appliquées au stream de livraison jusqu'à obtenir une stabilité suffisante. Par ailleurs, sur le stream de développement, de nouvelles fonctionnalités sont ajoutées, qui n'apparaissent pas dans le stream de livraison. Les anomalies corrigées dans le stream de livraison doivent être également corrigées dans le stream de développement.

La figure 12.1 illustre un stream créé pour validation, sur lequel on doit corriger les anomalies 1 et 3. Les nouveaux développements et les autres corrections d'anomalies ne sont pas réalisés sur le stream de développement, lequel est conservé aussi stable que possible de façon à pouvoir en établir le niveau de qualité. S'il devait continuellement être en évolution, on ne pourrait estimer son niveau de qualité.

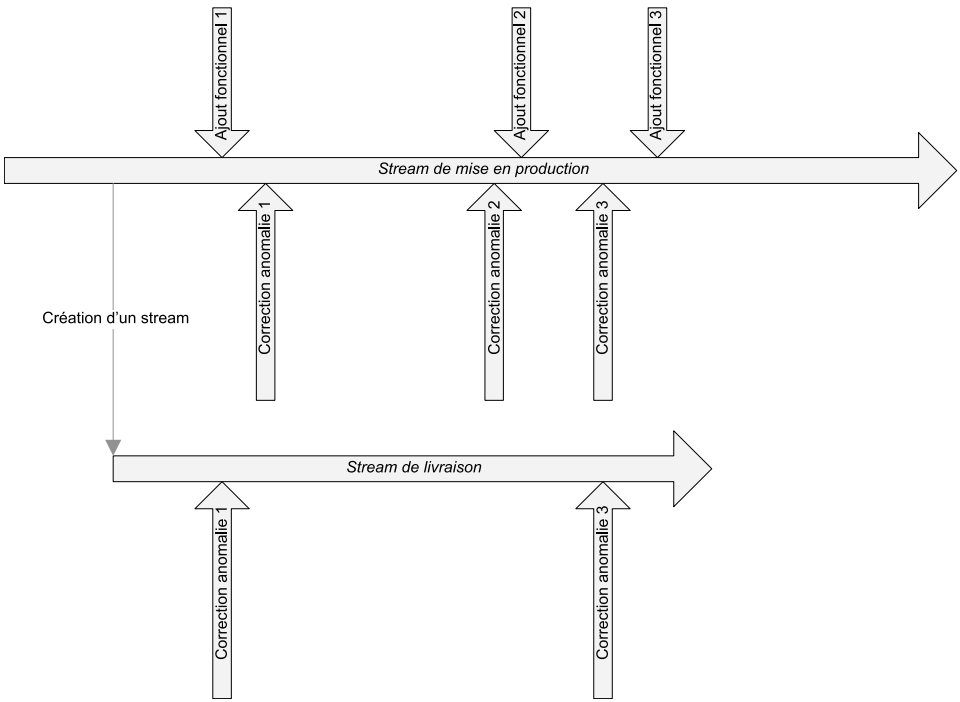

Figure 12.1. Gestion des streams

On peut créer plusieurs streams en parallèle selon les objectifs que l'on s'est fixés :

- Version majeure : création d'une version majeure suivante, qui ajoute un ensemble important de fonctionnalités, voire s'accompagne de couches techniques retravaillées.

- Version spécifique : ajouts fonctionnels pour un client spécifique donnant lieu à une dérivation client du produit.
- Service Pack : livraison sous forme de patch d'un ensemble de corrections d'anomalies.
- Hot Fix : correction urgente d'un bogue bloquant à livrer dès que possible sur la version en production.

Cet exemple montre qu'on peut entreprendre quatre séries de modifications en parallèle sur les mêmes codes source, avec des objectifs de livraison client qui varient entre quelques heures (Hot Fix) et plusieurs mois, voire plus d'un an, dans le cas du travail sur une nouvelle version majeure. Le référentiel sait gérer ces streams différents et, dans une certaine mesure, assister à la fusion des actions d'un stream sur l'autre, comme reporter la correction de l'anomalie urgente sur les autres streams , afin d'éviter les régressions.

Le référentiel assure la sécurité des informations qui lui sont confiées. Il est, par exemple, possible d'assurer une partition des informations entre des personnes ayant accès à certains éléments et d'autres qui n'y ont pas accès.

Certains référentiels sont conçus pour être utilisés sur plusieurs sites, avec réplication totale ou partielle des éléments. Lorsqu'on travaille avec un prestataire distant, on peut répliquer, en temps réel ou périodiquement, toutes les informations relatives à leurs développements. Tous les éléments placés dans le référentiel se retrouvent immédiatement chez le client et réciproquement, tandis que les éléments qui ne sont pas synchronisés restent locaux.

Le référentiel est au cœur de l'outillage méthodologique. C'est l'outil sur lequel tous les autres s'appuient. Il est donc à choisir avec beaucoup de soins.

Gestion du changement

L'outil de suivi du changement — on sépare parfois la gestion des anomalies et celle des évolutions ou nouvelles fonctionnalités — assure plusieurs rôles, notamment les suivants :

- Il permet de suivre précisément toutes les détections d'anomalies ou demandes de changement en garantissant qu'aucune information ne se perd.
- Il permet de mettre en place un flux de traitement des requêtes. On peut, par exemple, définir des phases de qualification des anomalies, au cours desquelles on détecte si le défaut signalé est bien une anomalie et si elle n'est pas déjà répertoriée. Une fois qualifiée, l'anomalie suit un chemin qui permet de lui associer les informations nécessaires pour décider de la corriger et quand (importance, temps estimé de correction, risques d'instabilité, etc.).

Le flux de gestion de l'anomalie ou de la demande est configurable. On peut trouver des flux très différents selon la nature de la requête. Par exemple, une anomalie technique suit un chemin différent d'une demande d'évolution fonctionnelle. Ces flux doivent être configurés par le client pour s'adapter au mieux à son organisation et à la répartition des rôles dans la société.

Cet outil est généralement fortement couplé au référentiel, car toutes les demandes de changement donneront probablement lieu à des interventions sur les codes source. On souhaitera suivre ces modifications avec les versionnements des éléments dans le référentiel.

L'outil de gestion du changement est indispensable lorsqu'on travaille en offshore. Il permet notamment d'éviter que le client et le prestataire s'attendent mutuellement et que des requêtes de changement se perdent. Il évite aussi les erreurs de communication sur les niveaux de priorité des requêtes.

Les petits projets ne nécessitent pas toujours de tels outils, parfois lourds et coûteux. Il est en ce cas nécessaire de les remplacer par des procédures strictement appliquées afin de communiquer efficacement.

> **EN RÉSUMÉ**
> **Gestion du référentiel et du changement**
> Les outils indispensables à la gestion de projet en offshore sont le référentiel et les outils de gestion du changement. Ils garantissent un suivi rigoureux de la production et de toutes les requêtes échangées entre les équipes ainsi qu'une traçabilité de l'état de santé du développement.

Comme indiqué au tableau 12.1, le référentiel et les outils de gestion du changement fournissent un grand nombre de mesures des développements en cours.

Tableau 12.1. Mesures issues des outils de gestion du référentiel et du changement

Mesure	Utilisation
Nombre d'anomalies	Mesure de la santé du produit, avec des informations sur la sévérité des anomalies connues. On étudiera particulièrement l'évolution des anomalies connues, le temps de résolution, la distribution par fonction, etc.
Nombre de requêtes de changement ou d'évolution	Mesure de la stabilité du produit. On étudiera particulièrement l'impact de ces requêtes sur les livraisons finales et la distribution des requêtes par fonction.
Temps de transition des requêtes dans un flux de travail	Cette mesure permet de comprendre le temps nécessaire entre la détection d'une anomalie et sa résolution et de repérer les décisionnaires qui prendraient un temps exagéré pour les traiter.
Stabilité des spécifications	On peut mesurer la stabilité des documents de spécifications en observant les mises à jour réalisées sur celles-ci, montrant ainsi la stabilité des demandes faites aux équipes de développement.
Taille des fonctions en lignes de code	Comptabilise les lignes de code de chaque fonction du produit.
Changements des lignes de code	Mesure du nombre de lignes de code modifiées, ajoutées et supprimées par fonction sur une période, montrant les domaines d'activité et la nature des interventions

Gestion des exigences

Les outils de gestion des exigences permettent de suivre toutes les exigences fonctionnelles et techniques qui ont été exprimées sur un projet. Chaque exigence est associée à une série d'attributs, comme l'importance de la fonctionnalité, une estimation de la durée de travail pour la réaliser, le risque lié à cette exigence ou tout autre élément que l'on souhaite utiliser pour les qualifier.

Ces attributs sont utilisés pour déterminer le contenu de chaque itération. Les décisions sont prises non plus en lisant le contenu de chaque exigence mais en travaillant directement sur ses attributs. Lors des premières itérations, on choisit le plus souvent d'éliminer toutes les incertitudes techniques et les risques les plus forts de façon à rendre les itérations suivantes de plus en plus fiables. À l'inverse, avant une livraison importante, on cherche à favoriser la stabilité, et les derniers développements concernent des tâches peu complexes et à faible risque. Les fonctionnalités portant une part de risques sont reportées à une version ultérieure, si c'est possible.

Les outils proposant une gestion des exigences maintiennent le plus souvent un lien entre l'exigence et tous les autres éléments qui en découlent, permettant de garder une cartographie des réalisations. Certains outils permettent aussi de gérer les workflows de modification des exigences.

Les exigences peuvent aussi être gérées sous forme de tableau de type Excel. On perd alors les mécanismes de liens entre une exigence et les éléments dérivés, mais ce n'est pas essentiel dans tous les projets.

> **EN RÉSUMÉ**
> **Gestion des exigences et itérations**
>
> La gestion des exigences permet d'associer à chaque exigence des attributs afin de déterminer les priorités selon lesquelles on construit un produit. Les attributs permettent, par exemple, d'éliminer les risques et de réaliser les tâches complexes le plus tôt possible dans le projet de façon à pouvoir envisager d'autres approches tant qu'il en est temps. Les engagements sur les itérations deviennent de plus en plus fiables au fur et à mesure de la progression du projet puisque les sources d'incertitude sont progressivement éliminées.

La gestion des exigences est une tâche continue. Les attributs que l'on pose au début d'un projet changent au fur et à mesure des réalisations. Il est possible, par exemple, que toute une série de fonctionnalités présentent un fort risque dû à une incertitude technique commune. Une fois cette incertitude levée, le risque doit être mis à jour pour toutes les fonctionnalités qui en dépendent. À l'inverse, de nouvelles difficultés ou de nouveaux risques peuvent apparaître.

L'importance des fonctionnalités change également. Il se peut qu'une fonctionnalité considérée comme secondaire devienne soudainement primordiale parce que l'utilisateur la réclame explicitement. La gestion des exigences est ainsi un document vivant, qui doit être mis à jour régulièrement, au moins à chaque itération, puisque c'est une des sources essentielles de détermination du contenu de l'itération future.

Gestion des flux (workflow)

Les workflows permettent de définir les états possibles d'un élément (information, document, fichier, etc.) et sa circulation entre les personnes qui le traitent et décident de le faire changer d'état. La plupart des outils de gestion du changement et des exigences incluent un outil de gestion de workflow des informations qu'ils gèrent.

La mise en place d'une gestion stricte des workflows relatifs au changement (anomalies et évolutions) est primordiale. Ces informations sont échangées en permanence entre les sites, et certaines informations peuvent être critiques. Si un seul workflow doit être défini sur un projet, ce doit être celui-ci.

Les workflows formalisés permettent de gérer des indicateurs de suivi et de mieux comprendre les dysfonctionnements éventuels. Ils prennent une grande importance lors des développements en offshore en permettant de contrôler finement certains flux chez le prestataire.

On trouve des workflows dans certains outils de gestion du changement et de référentiel. Il existe aussi des outils dont l'unique objet est de définir des workflows multiusages dans les sociétés. Dans tous les cas, on définit pour chaque type de livrable un cheminement à travers des états. Pour passer d'un état à un autre, des acteurs doivent réaliser certaines actions pour attribuer un nouvel état au livrable.

Gestion des tests

La gestion des tests est aussi un élément important de l'outillage méthodologique. Le tableau 12.2 recense les fonctions d'un certain nombre d'outils de test.

Tableau 12.2. Fonctions des outils de test

Outil	Fonction
Gestion des tests	Gère les scénarios de test couvrant les spécifications (cas d'utilisation), ainsi que les données des tests et les résultats.
Robot de test	Automatise les séquences d'actions permettant de répéter automatiquement des tests et de s'assurer de l'absence de régression sur des fonctionnalités. À tout moment, on peut comparer le comportement du système avec un comportement attendu.
Test de charge utilisateur	Injecte des actions que pourraient engager des utilisateurs soit sous forme de séquence d'actions enregistrées, soit directement sous forme de requêtes. L'injecteur d'activité permet de moduler comme on le souhaite une charge utilisateur afin d'étudier le comportement du système en charge.
Analyse de comportement technique *(profiling)*	Permet d'analyser le comportement du système lorsqu'il est sollicité par une série de requêtes ou transactions. On peut connaître, par exemple, le temps CPU consommé par chaque méthode ou fonction, l'évolution de la gestion de la mémoire, les échanges entre les composants, etc.
Couverture des tests	Permet de tracer les lignes de code effectivement exécutées lors d'un traitement. Cela peut se révéler utile pour visualiser les parties du code qui ne sont pas exécutées lors des tests et augmenter les scénarios de test en conséquence.

Pour le travail avec l'offshore, il est particulièrement important de s'accorder sur une représentation de l'état de qualité du produit. L'outil de gestion des tests permet de conserver l'état courant de tous les résultats de test et de les partager entre le client et le prestataire.

Il faut également être certain de ce que l'on teste. Il est souvent préférable de ne pas utiliser une version du produit exécutable copiée de machine en machine et de prévoir à la place un déploiement à partir de procédures automatisées qui extraient le code source du référentiel. On est alors certain de disposer de l'exacte représentation du code source placé dans le référentiel et qu'aucun autre élément n'est nécessaire à la création de l'exécutable. On teste de fait aussi les procédures d'extraction des sources, de compilation et de déploiement.

Gestion de la documentation

Certaines suites méthodologiques disposent d'outils de construction des dossiers de documentation du projet à partir d'une grande quantité d'informations.

Ces outils sont peu utiles dans le cours des réalisations des projets informatiques. Ils sont surtout utilisés pour créer un corpus de documentation une fois le projet terminé.

Mise en place de la méthodologie

Les méthodologies sont souvent très intimidantes, et de nombreuses sociétés ne parviennent pas à les mettre en place efficacement. Elles apparaissent comme un mur infranchissable et sont de ce fait reléguées au rang des créations intellectuelles qui ne sont pas réellement applicables. Parfois, on estime qu'elles sont certainement utiles à d'autres, mais que le cas que l'on rencontre les rend inefficaces. Il va de soi que les méthodologies sont applicables à tous les projets et sont partout aussi difficiles à mettre en œuvre.

Les sections qui suivent récapitulent les principales difficultés de mise en place d'une méthodologie industrielle et les façons de les résoudre.

Méthodologie et offshore

Lorsqu'on démarre un projet avec une équipe distante et que l'on souhaite mettre en œuvre une méthodologie, on pense souvent étendre l'application de cette méthodologie à toute la production de la société et donc à ses équipes de développement locales. L'intention est louable, mais elle complexifie énormément la tâche. Il n'est guère facile de mener de front deux déploiements de la méthodologie tout à fait différents.

Mettre en place une méthodologie industrielle pour une équipe en place depuis de nombreuses années chez le client est avant tout un travail de communication consistant à choisir le chemin de migration vers la méthodologie le plus efficace. Lorsqu'on l'installe pour une nouvelle équipe, comme une équipe que l'on crée en offshore, on peut choisir de l'imposer, puisque les participants adoptent à la fois le client, le projet et la méthodologie .

Le déploiement de la méthodologie dans l'équipe remet en question les rôles de chacun. Les responsabilités des informaticiens et la façon de travailler sont redéfinies, souvent en spécialisant les rôles et en leur retirant certaines responsabilités. Les collaborateurs se focalisent tout naturellement sur ce qu'ils vont perdre plutôt que sur les avantages de la nouvelle organisation. Comme l'organisation existante est le résultat d'années d'ajustements successifs, la nouvelle méthodologie, avec ses fonctionnements encore bruts, apparaît en outre comme inférieure et fait l'objet de violentes critiques.

Pour éviter ces écueils, il suffit d'isoler le déploiement de la méthodologie industrielle en offshore de celui sur les équipes locales. On peut traiter les deux projets en parallèle et de façon indépendante, si possible décalée dans le temps. Il est beaucoup plus facile de mettre en place une méthodologie industrielle sur un projet distant, où l'on conçoit immédiatement l'utilité des procédures à mettre en œuvre, que sur une équipe locale qui fonctionne raisonnablement bien et qui sera certainement réticente à changer de mode de travail.

Se concentrer sur l'essentiel

Un autre piège classique lorsqu'on déploie une méthodologie est de vouloir la mettre intégralement en œuvre. Les méthodologies industrielles cherchent à couvrir toutes les situations que l'on peut trouver sur un projet, nombre d'entre elles n'ayant aucune utilité pour un projet donné.

Il convient de prendre suffisamment d'altitude pour comprendre pourquoi ces recommandations sont prévues dans la méthodologie et décider si l'on doit couvrir tels et tels points ou s'ils sont sans objet dans le cadre du projet. De plus, les méthodologies souhaitant être complètes, elles prévoient des documents ou des procédures qui ne sont pas toujours utiles et qui peuvent être difficiles à rédiger.

Par exemple, un des documents initiaux prévoit de lister les ressources allouées au projet alors que le budget peut ne pas être si clair. Les managers ont peut-être prévu certaines réserves pour couvrir des retards et des dysfonctionnements, par exemple, et ne souhaitent pas en faire état. La personne qui chercherait à tout prix à obtenir ces chiffres se heurterait à un refus du management. Les responsables financiers pourraient alors s'élever contre la méthode, la jugeant trop intrusive.

Il convient donc de comprendre les éléments essentiels de la méthodologie, d'en respecter les principes fondateurs et d'adapter le reste à la réalité du projet que l'on gère. Un grand nombre de documents peuvent ne pas avoir une grande importance dans le cadre du projet. Il peut aussi y avoir des informations impossibles à obtenir, ne serait-ce que le budget réel du projet.

Ce sont les objectifs forts, centrés sur les éléments structurants de la méthodologie et auxquels on ne doit pas déroger, que l'on mettra en place au plus tôt.

> **EN RÉSUMÉ**
> **Se concentrer sur les éléments essentiels**
> Seuls les quelques éléments essentiels des méthodologies doivent être mis en place sans délai. Les autres éléments doivent être considérés comme optionnels et d'une priorité faible. Ils ne seront mis en place que si le problème qu'ils résolvent a besoin d'être couvert dans le projet.

Dans les méthodologies RUP et XP, les principes essentiels que l'on doit absolument respecter dans un projet confié à une équipe distante sont, par ordre d'importance :

- le développement par itérations avec une gestion intelligente de celles-ci ;
- une documentation exhaustive et structurée des fonctionnalités à développer, si possible sous forme de scénarios d'utilisation ;
- une architecture logicielle facilitant la distribution du projet en composants et son évolution ;
- la construction aussi fréquente que possible, au moins une fois par itération, d'un build permettant de valider la réalité de la progression du projet.

Favoriser la motivation

La gestion des ressources humaines et ses principes motivants visant productivité et qualité ne doivent pas être oubliés lors de la mise en place des procédures. Il est commun de se concentrer sur les procédures et d'oublier la gestion des ressources humaines.

Comme nous l'avons vu, l'application formelle et automatique des procédures d'une méthodologie réduit les responsabilités et dépersonnalise les rôles. Plus les hommes

reçoivent de directives contraignantes, moins ils se sentent responsables de leur production, et plus la productivité baisse.

Il faut veiller à ce que la motivation de chacun ne soit pas atteinte par la mise en place d'une méthodologie industrielle. Il faut certes faire entrer chaque collaborateur dans un rôle structuré avec les responsabilités qui y sont attachées, mais les procédures doivent avant tout permettre aux hommes de travailler plus efficacement et d'éviter les malentendus et les situations floues où l'on ne sait comment faire ni qui doit décider.

On veillera surtout à ne jamais mettre en place des procédures qui dévalorisent les hommes. Gérer les équipes de développement comme des exécutants qui reçoivent des ordres de développement selon un outil de workflow, sans engagement personnel sur les réalisations et sans créativité, est dévalorisant et ne peut qu'entraîner une réduction de la motivation et donc de la qualité du travail.

> **EN RÉSUMÉ**
>
> **Des procédures au service des hommes**
>
> Les procédures doivent aider les hommes à travailler plus efficacement en réglant de façon univoque l'état des documents, leur circulation et leur validation. L'organisation des rôles et des équipes doit être telle qu'elle favorise la motivation et la responsabilisation. Il est utile de garder les hommes au plus près de chacun des livrables dont ils sont responsables afin que le succès ou l'échec de chaque livraison puisse clairement s'appliquer à ceux qui y ont participé.

Les outils d'aide au travail des collaborateurs

Les outils qui accompagnent la méthodologie sont hautement configurables pour s'adapter à la façon de travailler que l'on souhaite mettre en place et aux procédures que l'on veut utiliser. Il ne s'agit pas de considérer que ces outils doivent imposer aux collaborateurs leur façon de travailler.

Par exemple, le principe itératif est rarement directement intégré dans les outils. De nombreux workflows sont organisés autour des itérations dans la méthode, comme la construction du contenu des développements de chaque itération qui découle d'une analyse globale du réalisé, ainsi que de l'expérience et des tâches restant à réaliser. Si l'on trouve effectivement des outils pour gérer ces informations, l'organisation en itération doit être mise en place par ceux qui s'occupent de la méthodologie et qui définiront les guidelines du processus itératif.

Mise en place progressive des procédures

Le très grand nombre de rôles, documents, activités, workflows et guides que proposent les méthodologies est déroutant de prime abord. Il est recommandé de ne mettre en place que ce qui est réellement utile au projet au fur et à mesure que les besoins apparaissent. La méthodologie accompagne ainsi le développement selon les principes forts que l'on veut faire respecter.

Adopter une méthodologie ne signifie pas en accepter tous les aspects. La plupart des fonctionnalités qu'elle propose sont modifiables sans pour autant distordre la méthodologie.

Par exemple, RUP recommande de mettre en place un comité pour gérer les demandes de changement. Ce comité (*change control board*) est censé réunir toutes les personnes concernées par les changements. Cette recommandation est généralement très utile puisqu'elle

permet que toutes les parties soient informées et s'expriment. Elle suppose cependant une organisation lourde, puisque les personnes concernées sont nombreuses et que les réunions ont toute chance de s'éterniser.

Il est souvent préférable de nommer une personne pour gérer les modifications fonctionnelles et une autre pour les modifications techniques. Selon la nature des changements, ces responsables s'organisent et lancent les consultations qui conviennent auprès des personnes réellement concernées afin de les associer aux évolutions. Seules les décisions majeures, comme le remplacement d'un outil technique important du développement, font l'objet de réunions plus générales en vue d'aboutir à une décision concertée.

Le planning

La plupart des méthodologies abordent assez peu la problématique de la planification, alors même qu'il s'agit d'un des aspects fondamentaux de la gestion de projet. On trouve toutefois souvent la possibilité d'exporter des tâches ou même des parties de planning à partir de certains outils méthodologiques. Il est parfois possible de les importer, mais la gestion du planning n'est pas réellement intégrée dans les suites méthodologiques.

Les sections qui suivent se penchent sur plusieurs aspects de la gestion de planning mis en œuvre efficacement dans des projets et qui s'accordent bien avec les principes itératifs.

De même que la mise en place des processus peut monopoliser toute l'attention, un chef de projet peut faire l'erreur de se concentrer sur la gestion du planning, au détriment de tous les autres aspects. Il croit pouvoir piloter toutes les opérations en offshore à travers celui-ci et en oublie les principes d'itération et la gestion des hommes. Pour éviter cela, nous proposons une approche permettant de lier l'efficacité d'un planning avec les principes méthodologiques essentiels.

Le tableau 12.3 indique plusieurs niveaux de planification permettant de gérer le développement du projet.

Tableau 12.3. Niveaux de planification

Planification	Fonction	Utilisation
Release plan	Distribue les fonctions sur les différentes livraisons du produit en faisant apparaître les livraisons de validation.	Permet de gérer la distribution des livrables dans le temps afin de les communiquer au client utilisateur du produit. Les différentes livraisons peuvent concerner des livraisons préliminaires pour validation ou des livraisons successives en production. C'est le planning rendu public qui correspond à un engagement envers les utilisateurs.
Plan de développement	Distribue le contenu des itérations pour atteindre les objectifs.	Permet de distribuer les objectifs des livraisons par itération jusqu'à la fin du projet en utilisant des charges estimées grossièrement. Ce planning va de pair avec le plan de charge des itérations.

Tableau 12.3. Niveaux de planification *(suite)*

Planification	Fonction	Utilisation
Plan de charge des itérations	Renseigne sur la charge de réalisation de chaque itération pour réaliser le release plan.	C'est la première confrontation du release plan avec la réalité. On voit si les ressources disponibles permettent de réaliser le planning attendu et l'on peut le modifier au besoin pour le rendre réaliste. On ne se préoccupe pas encore de la séquence des tâches ni des tâches périphériques aux développements.
Plan détaillé de l'itération	Inclut le détail de l'itération qui est sur le point de démarrer ainsi que tous les détails de toutes les tâches planifiables.	Ce planning permet de suivre l'itération dans le détail, ainsi que les affectations à chaque personne. On peut ainsi suivre la progression dans l'itération et comprendre les causes des retards.

Tous ces plannings sont revus et corrigés après chaque itération pour tenir compte de l'expérience acquise. L'itération dirige fortement le plan de développement. C'est l'unité de temps selon laquelle on prévoit la progression et les ressources. Seule l'itération en cours est planifiée avec tous les détails. Comme elle est de courte durée, elle ne doit pas être perturbée par des éléments extérieurs.

Le plan des itérations définit le contenu du plan détaillé de l'itération. On y découvre le détail des dépendances entre les tâches et les attributions à chaque informaticien, notamment les disponibilités, compétences, etc. Si nécessaire, les résultats du planning détaillé sont pris en compte dans le plan des itérations afin de parvenir à plus de précision dans les plannings suivants.

Gestion des corrections et des nouveaux développements

Les équipes de développement doivent réaliser de nouveaux développements (nouvelles fonctionnalités, corrections majeures planifiées ou évolutions majeures). Elles sont également sollicitées pour des corrections d'anomalies, dont certaines doivent être prises en compte au plus tôt. Ces demandes de correction doivent être gérées convenablement pour ne pas perturber les engagements en cours des équipes.

De nombreux ouvrages recommandent de créer une équipe dédiée aux corrections d'anomalies afin de séparer les développeurs qui travaillent sur celles-ci et ceux qui travaillent sur les nouvelles fonctionnalités. L'avantage majeur de cette approche est d'isoler l'équipe travaillant sur les nouvelles réalisations et de garantir ainsi la progression stable et planifiée du projet, avec des équipes qui ne seront pas perturbées par les demandes urgentes de correction, qui interrompent les tâches en cours et retardent les livraisons. Son autre effet bénéfique est de limiter et contrôler mécaniquement la quantité de travail que l'on peut investir dans les corrections d'anomalies en fonction du nombre de personnes qui y sont dédiées. On évite ainsi que les développeurs décident des priorités entre les tâches de développement planifiées et celles relatives à des corrections urgentes.

On peut opposer trois inconvénients majeurs à cette approche :

- Difficulté à garder motivés les développeurs qui passent leur temps à corriger des anomalies.
- Difficulté à convaincre les développeurs qui travaillent sur les nouveaux développements à viser la meilleure qualité, alors qu'ils savent qu'ils ne corrigeront pas les anomalies qu'ils ont créées.
- Difficultés engendrées par les multiples fusions de codes source écrits par des développeurs différents pour gérer les corrections d'anomalies et les nouveaux développements.

À l'opposé, l'avantage majeur d'une équipe dédiée aux corrections d'anomalies est l'assurance de la bonne maintenabilité des codes source puisqu'ils sont pris en main par d'autres développeurs que ceux qui les ont initialement écrits.

Une autre façon efficace de séparer les activités de correction des nouveaux développements est d'allouer une proportion du temps d'une équipe aux corrections en interdisant que les développeurs ne dépassent cette allocation. Le pourcentage de temps alloué aux corrections est déterminé pour chaque équipe et pour chaque itération selon le niveau d'effort que l'on souhaite fournir sur ces tâches.

En mode de maintenance d'une application qui continue à évoluer normalement, on peut allouer 20 % du temps d'une équipe à corriger les anomalies et les 80 % restants aux nouvelles réalisations. Cette organisation présente plusieurs avantages :

- On isole effectivement les deux types de traitement du code.
- On assure que la personne qui a codé la fonction intervient elle-même sur ses corrections. Elle est alors la plus efficace pour déterminer la cause de l'anomalie et ne pas la reproduire sur les nouveaux développements.
- On peut choisir le moment qui convient pour réaliser ces corrections de façon à ne pas avoir à gérer des fusions de corrections sur deux streams qui évoluent en parallèle.

Si l'on ne prend pas soin d'isoler parfaitement les corrections d'anomalies des nouveaux développements, les équipes techniques annoncent trop souvent qu'elles ont favorisé l'une ou l'autre activité de leur propre initiative. Seul le temps prévu doit être consommé sur chacune de ces activités si l'on veut être capable de planifier les réalisations ou de construire certains indicateurs.

EN RÉSUMÉ

Corrections et nouveaux développements

Pour pouvoir planifier les nouveaux développements et la quantité d'anomalies qui peuvent être corrigées, il est important d'isoler le temps réservé aux corrections d'anomalies de celui dédié aux nouveaux développements et aux évolutions majeures.

Pour résoudre cette question, deux approches sont communément retenues. La première consiste à constituer une équipe de correction d'anomalies dédiée à cette tâche, une autre équipe ne s'occupant que des nouvelles réalisations. La seconde solution consiste à allouer à chaque équipe de développement une répartition du temps à passer aux corrections des anomalies et aux nouveaux développements et à s'assurer que ces proportions sont respectées.

Release, Service Pack, patch et Hot Fix

Il existe plusieurs façons de livrer une nouvelle version d'un produit selon le degré d'urgence des ajouts et corrections à apporter.

En règle générale, plus le contenu est important, plus il est long et difficile de le recetter et de le stabiliser, et plus il faut de temps pour le livrer. En revanche, une correction simple, bien circonscrite et apparemment sans danger, peut faire l'objet d'une correction rapide, de tests limités et d'un déploiement prompt avec un risque limité.

Le tableau 12.4 présente un découpage de différents types de livrables selon l'urgence du besoin rencontré.

Tableau 12.4. Livraisons et risques

Livrable	Besoin	Préparation	Description	Risque
Nouvelle version	Livrer de nouvelles fonctionnalités importantes et des évolutions majeures	Planifiée plusieurs mois à un an à l'avance, avec environ une version par an	Des modifications importantes sont réalisées sur le produit. Le modèle de la base de données et la plupart des composants changent. Tous les tests sont réalisés entièrement avant livraison.	Très faible
Version mineure	Livrer de nouvelles fonctionnalités peu volumineuses et des évolutions majeures	Planifiée plusieurs mois à l'avance	Les versions mineures sont technologiquement proches des versions majeures. Les différences résident dans le volume des nouvelles fonctionnalités livrées et, chez les éditeurs de logiciels, la présentation commerciale de la version.	Très faible
Service Pack	Livrer une série de corrections d'anomalies et quelques évolutions fonctionnelles	Régulière dans le temps et planifiée à l'avance	Les corrections des anomalies sont livrées en bloc dans un Service Pack entièrement testé. On reste dans un fonctionnement planifié à l'avance.	Très faible
Patch	Livrer une ou quelques corrections d'anomalies qui doivent être déployées rapidement et ne peuvent attendre une livraison planifiée.	Procédure d'urgence faible ou moyenne avec livraison non planifiée	Une ou plusieurs anomalies gênantes ne peuvent attendre le prochain Service Pack ou une livraison mineure ou majeure. Une série limitée de corrections sont effectuées et testées rapidement. On réalise des tests automatisés pour vérifier les régressions. La partie modifiée est testée en profondeur. Avec des tests réduits, le produit suit un cycle de validation rapide mais normal.	Moyen

Tableau 12.4. Livraisons et risques *(suite)*

Livrable	Besoin	Préparation	Description	Risque
Hot Fix	Livrer une correction bien isolée pour corriger une anomalie critique dès que possible. Le produit n'est peut-être pas utilisable sans cette correction.	Urgence très élevée. La correction des anomalies est pleinement préemptive.	La correction est portée sur le code source en production. Ces tâches de correction sont préemptives. L'urgence fait que l'on réduit les tests uniquement aux parties modifiées. Parfois, la correction est directement effectuée sur le site de production.	Élevé

La méthodologie doit prévoir les différents types de release et avertir les participants au projet des risques pris sur les urgences fortes et les coûts relatifs souvent élevés, qui peuvent affecter le reste de la production. Un patch interrompt certaines tâches en cours. On travaille sur un stream qui devra être reporté sur tous les autres streams, multipliant d'autant les efforts. Les tests pour valider cette livraison n'ont d'utilité que pour ce patch, et de nombreux tests manuels doivent être réalisés. Si l'on multiplie fortement le recours aux tâches urgentes, on ralentit significativement les autres réalisations.

Il est donc important de porter une attention particulière à ces procédures si l'on veut que l'urgence des requêtes soit respectée comme il convient, en tenant compte des risques et des coûts de chaque action. Les différents streams doivent pouvoir être créés et gérés par le référentiel pour limiter les risques et apporter la flexibilité et la réactivité souhaitées.

Le release plan

Le release plan est assez proche de la gestion des exigences. On peut le représenter sous forme de tableau listant toutes les *features* que l'on souhaite réaliser, avec les dates prévues de livraison aux utilisateurs, ou sous forme de planning, afin de mieux juger de l'étalement dans le temps des livraisons.

Le tableau de release plan donné en annexe permet de suivre toutes les exigences et de les attribuer à chaque release. On y trouve également une estimation de la charge de chaque fonctionnalité permettant d'estimer globalement le nombre de jours de codage alloué à chaque release, qui sera à comparer au nombre de jours disponibles que l'on anticipe de façon à estimer grossièrement la faisabilité du plan. On s'entendra sur ce que représente cette charge. Le plus simple est de mesurer la charge de codage et de définir des règles permettant d'y ajouter une charge d'analyse (architecture), de test ou d'autres interventions de mentors, ces charges dépendant de la nature de l'intervention (évolution ou nouvelle fonctionnalité).

Bien souvent, l'état des cas d'utilisation, ou UC (Use Case), est renseigné dans le tableau. Les cas d'utilisation ne sont pas toujours disponibles dès le commencement du projet mais sont parfois créés au moment où l'on en a besoin. Il est important de suivre la disponibilité des spécifications détaillées pour s'assurer que les équipes sont capables de commencer à travailler au moment prévu.

Le release plan est mis à jour régulièrement. On vérifie que les priorités qui ont été posées sont toujours valides, en tenant compte de l'expérience acquise au cours des itérations et des changements de priorité éventuels des clients.

Plan de développement et charge des itérations

Le plan de développement consiste à distribuer le *release plan* par itération. On ne cherche toujours pas à donner le détail des tâches ni à définir les personnes qui travailleront sur chacune d'elles, mais on se concentre sur la charge de travail.

On introduit pour cela certaines dépendances des grands blocs de développement. Par exemple, s'il faut que certaines fonctionnalités techniques soient disponibles pour que les développeurs fonctionnels commencent à les utiliser, on s'arrange pour s'assurer de leurs réalisations dans l'itération qui précède les développements fonctionnels qui les utilisent.

Pour chaque itération, on peut raisonnablement anticiper la charge de travail disponible pour de nouveaux développements. Les autres charges de travail découlent de ces hypothèses. On prévoit le travail de correction des anomalies, selon l'avancement des développements dans le projet æ les premières itérations n'auront probablement pas de charge affectée aux corrections d'anomalies æ, ainsi que le travail de test, d'architecture, etc.

On peut essayer d'anticiper les besoins d'embauche et les congés selon les dates des itérations afin d'affiner au mieux les plans d'itération. Le client peut aussi utiliser des règles pour ajouter des quantités de travail et représenter les risques. Par exemple, on attribue 30 % de charge supplémentaire à toutes les tâches de risque technique élevé.

Toutes ces hypothèses sont posées par écrit et accompagnent chaque itération.

La création du plan de charge des itérations est un travail assez théorique, qui permet de vérifier plus finement que dans le release plan le réalisme du planning. Ce document fournit la base de ce qui deviendra le planning détaillé de l'itération.

En fin d'itération, lorsqu'on dispose du retour d'expérience, on retourne sur tous les plannings, tout particulièrement sur le plan de développement et le planning de charge.

Les trois plannings illustrés à la figure 12.2 montrent la façon de suivre l'impact des décisions et événements sur les livraisons. On peut prendre des décisions en collaboration avec les équipes distantes en restant concentré sur l'itération en cours. On ne perd donc pas la vision globale du plan des livraisons, qui représente l'engagement envers les utilisateurs. Le processus itératif est bien appliqué et se gère assez naturellement par la structuration des plannings.

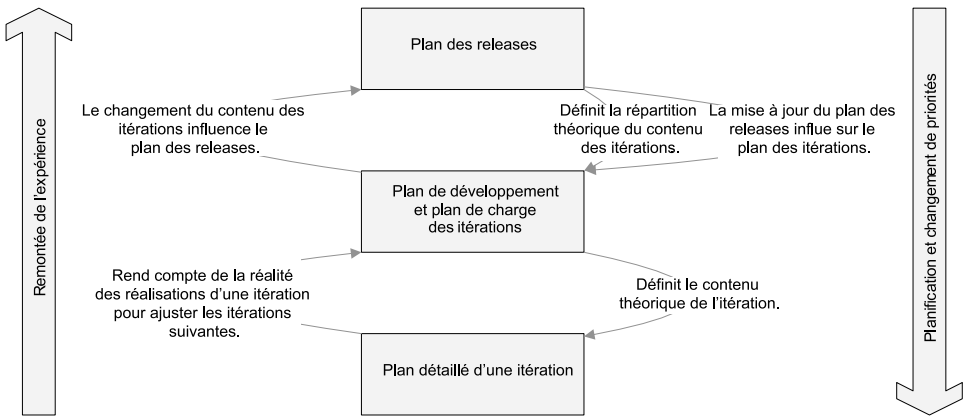

Figure 12.2. Les planifications

Le planning détaillé de l'itération

Le planning détaillé de l'itération présente toutes les tâches de tous les membres des équipes avec un détail précis de celles-ci. Elles durent de deux à cinq jours et détaillent les attributions de chaque personne. Ce planning est dans la directe continuation du planning de l'itération précédente.

La gestion du planning d'une équipe en offshore est particulièrement importante. Moyen essentiel de communication entre le prestataire et le client, elle consiste à mettre en œuvre une planification réellement utile aux deux parties.

Généralement, on construit le planning à l'aide d'outils tels que Microsoft Project.

La plupart des méthodologies proposent des modèles qui s'appuient sur les phases du projet, chaque fonctionnalité étant regroupée dans la phase. Dans la réalité, mieux vaut organiser le planning autour des domaines fonctionnels, et ce pour deux raisons :

- Les phases du projet sont rarement séquentielles dans la réalité. Par exemple, la phase d'*inception* (création) du projet, où l'on travaille sur les spécifications et la préparation, s'étend en réalité jusque très tard dans le projet, puisque des spécifications finalisées sont bien souvent fournies juste avant le développement de certaines tâches prévues tard dans le cours du développement.
- Il est préférable de confier aux responsables d'équipe la définition du détail du planning des tâches dont ils sont responsables. Chacun d'eux maîtrise le planning complet de son équipe en tenant éventuellement compte de points de synchronisation avec les autres équipes qui fournissent des livrables sur lesquels il doit se synchroniser.

On peut définir une série de règles pour construire le planning détaillé. Récapitulées au tableau 12.5, ces règles doivent être adaptées au projet que l'on rencontre.

Tableau 12.5. Règles de construction du planning

Règle	Description
Définir le guide du planning	Le guide du planning décrit précisément comment il doit être construit, avec le plan de structuration des tâches, les règles de nommage, etc. Par exemple, il est pratique de nommer les tâches relatives à une requête de changement par un libellé qui commence par l'identifiant de celui-ci.
Définir les chefs d'équipe responsables de leurs plannings	Chaque responsable d'équipe construit son planning détaillé, qui correspond à ses engagements sur l'itération. Ce planning est construit conjointement avec son équipe, laquelle doit accepter les engagements pris dans celui-ci. Le planning est donc essentiellement regroupé par domaine fonctionnel.
Lister toutes les dépendances entrantes	Le planning commence par lister toutes les dépendances des livraisons provenant d'autres équipes ou du client. Il arrive souvent que le projet prenne du retard du fait de retards de livraison d'éléments provenant du client. Comme il est toujours difficile pour le prestataire de faire des reproches à son client, le planning permet de mettre en évidence les responsabilités des retards, y compris celles du client.

Tableau 12.5. Règles de construction du planning *(suite)*

Règle	Description
Verrouiller le temps alloué aux corrections d'anomalies	Quelle que soit la méthode retenue pour allouer du temps aux corrections d'anomalies, le planning liste les temps alloués par personne pour les corrections d'anomalies. Ce temps peut être alloué avec précision à chaque ressource.
Lister toutes les tâches de développement	Ces tâches sont listées par domaine fonctionnel et identifient précisément les requêtes de changement ou les nouveaux développements.
Lister les tâches de fond qui ne correspondent pas à des livrables explicites	Ces tâches de fond concernent l'application de procédures, comme la remise à niveau des règles de nommage ou l'amélioration de l'utilisation des couches techniques. Par exemple, on réserve certains jours pour harmoniser les méthodes d'accès aux données après des recommandations de l'équipe technique. On trouve aussi les tâches d'optimisation de fonctions.
Lister les tâches de management	Les managers réservent le temps nécessaire au management (suivi, réunion, planification, etc.).
Lister toutes les dépendances sortantes	Le planning liste précisément tous les livrables qui sont attendus par une autre équipe.
Assembler les plannings	Les sous-plannings de chaque équipe sont assemblés par un responsable du planning, qui vérifie la cohérence de chacun d'eux et les assemble en un planning général.

Le modèle de planning fourni en annexe reprend ces recommandations. Tous les aspects du modèle ne sont pas présentés ici, notamment la gestion des priorités des tâches, les synchronisations entre itérations, etc.

Comme expliqué précédemment, les engagements des équipes et des hommes concernent avant tout des réalisations livrables, recettables et utilisables, plutôt que des livrables personnels représentant des états intermédiaires. Le planning doit de même s'organiser autour des livrables d'équipe, dont on suit la date de livraison. On laisse ainsi une grande latitude aux équipes pour organiser leurs livrables intermédiaires.

Il est recommandé de mettre à jour les pourcentages d'avancement du projet régulièrement au cours de l'itération, par exemple toutes les semaines. Cela permet de mieux comprendre le détail des causes de retard dans une itération. Le chef de projet estime les pourcentages d'avancement de chaque fonctionnalité.

Le planning doit être perçu non comme un outil permettant d'identifier le coupable d'un retard mais comme une aide pour comprendre les problèmes qui se posent afin d'éviter qu'ils se reproduisent.

Les causes les plus classiques de retard sont les erreurs d'estimation de charge, les retards de livraison d'autres équipes dont on dépend, des directives du client demandant de se concentrer sur une tâche urgente non planifiée ou encore l'absence de réponse à une question importante. Tous ces problèmes peuvent être améliorés à condition d'une attention soutenue du chef de projet du client.

Analyse de l'itération terminée

Le processus itératif révèle toute sa puissance si l'on prend soin de réaliser des *iteration assessments* (analyses d'itération). Ces dernières visent à analyser les objectifs qui n'ont pas été atteints et à comprendre les raisons qui ont contribué à l'échec.

Le modèle d'*iteration assessment* donné en annexe en présente la structuration. Il est recommandé d'utiliser un document par équipe. Le chef de projet du client assemble les *iteration assessments* provenant des équipes et en contrôle la cohérence. Par exemple, si une équipe est en retard du fait d'une livraison tardive d'une autre équipe, cet événement doit se retrouver dans l'*assessment* de l'équipe en faute.

Certaines raisons peuvent expliquer des retards, sur lesquelles on peut intervenir efficacement. Par exemple, si le client livre les spécifications tardivement, il est certainement possible d'éviter ce type de cause de retard. D'autres permettent de mieux connaître les équipes, comme la sous-estimation systématique du travail à réaliser.

L'analyse de l'itération doit aller beaucoup plus loin qu'une simple revue des dysfonctionnements de l'itération. Selon les résultats constatés, le chef de projet peut reconsidérer tous les éléments du projet :

- Les procédures ont-elles été mises en défaut ? Certaines procédures sont-elles manquantes ou leur application pose-t-elle des problèmes ?
- Les équipes en place sont-elles adaptées à ce que l'on souhaite faire ? Certaines équipes ont-elles besoin d'assistance ? Faut-il changer les membres de ces équipes ? Faut-il revoir l'organigramme ? Les mentors ont-ils joué leur rôle ?
- Les équipes chez le client ont-elles été la cause de retard du fait de réponses tardives aux questions ? Y a-t-il eu des changements de priorité au cours de l'itération ? Cela aurait-il pu être évité ?
- Le plan de développement doit-il être revu ? Le plan de charge doit-il être ajusté ? Les performances constatées doivent-elles influencer les plannings ? Quelle est la profondeur des ajustements ? Le release plan peut-il être maintenu ? Y a-t-il des impacts sur les coûts ?
- Le matériel et l'environnement de travail sont-ils adaptés ? Les moyens matériels sont-ils la cause de certains problèmes ?
- A-t-on bien pris en compte les enseignements de cette itération pour définir l'itération suivante (assistance aux équipes en difficulté, assistance à la planification, etc.) ?

Valider la réalité des réalisations

La seule validation vraiment utile de la progression d'un projet est la vérification régulière de la réalité et de la qualité des livrables recettables. Les estimations d'un chef de projet sur l'avancement de son travail sont hautement douteuses.

Par exemple, un chef de projet peu expérimenté aura beaucoup de mal à savoir s'il a atteint 80 % de progression. Dans de nombreux cas, on s'aperçoit qu'il a à peine atteint la moitié de sa tâche, malgré son estimation. Cela peut être dû à une sous-estimation de

l'effort nécessaire pour finaliser une tâche. Seuls les livrables recettables, c'est-à-dire que l'on peut réellement vérifier, permettent de quantifier une progression.

La distance rend la communication moins efficace. Cette dernière est limitée aux échanges formels, et le client a peu de moyens de se rendre compte par lui-même des problèmes. Si les collaborateurs distants comprennent que le client ne peut vérifier la réalité du travail accompli, ils peuvent être tentés de cacher leurs retards et leurs problèmes en espérant les résoudre avant que le problème apparaisse au grand jour et cause des difficultés. C'est aussi l'une des raisons pour lesquelles le client doit utiliser les informations sur les problèmes pour aider les équipes à les résoudre et non pour rechercher et punir les coupables.

ÉTUDE DE CAS

Une dure journée

Une anecdote exprime très bien la situation que rencontrent la plupart des managers de développements informatiques, tout particulièrement en offshore.
Une livraison importante est prévue pour le 30 du mois. Le manager explique aux chefs de projet l'importance de la livraison pour cette date, et tous se mettent au travail. Le 5 du mois, le manager fait le tour des chefs de projet, qui répondent l'un après l'autre que ce sera prêt dans les temps. Le 15 et le 20, le message est identique. Le 25, ils annoncent que ce sera dur. Le 29, ils font savoir que ce sera très dur mais qu'on y arrivera quand même.
Arrive le 30. Les chefs de projet expliquent qu'ils étaient presque prêts mais que, lorsqu'on a intégré leur partie avec celle d'un autre groupe, on s'est rendu compte d'incompatibilités majeures et qu'il fallait retravailler le code en profondeur. Le produit sera prêt au mieux trois semaines plus tard. Dure journée ! Le 29, tout allait bien, et on a pris trois semaines de retard en une journée...

Le seul moyen pour les participants au projet de se rendre compte de l'avancement réel du projet est de le prouver par la réalité de la progression. L'intégration des réalisations en continu permet de vérifier que les développements s'intègrent correctement, et les tests vérifient la qualité des livrables.

Intégration continue et périodique

Comme expliqué à la section précédente, l'intégration est le seul moyen de s'assurer de la progression des tâches de développement. Le code réalisé est compilé et intégré à une version en cours de construction afin de donner lieu à des tests unitaires ou fonctionnels et techniques plus complets. On peut décider de réaliser des intégrations périodiques, avec une période aussi courte que possible, ou l'intégration continue ou quotidienne.

Pour être utile, il faut que l'intégration soit recettée, au moins sommairement, afin de détecter les régressions majeures. Il est évident que si l'on a une intégration quotidienne, seuls les tests automatisés sont réellement efficaces puisqu'ils permettent de ne pas consommer une main-d'œuvre de test exagérée. Ces tests automatisés peuvent être exécutés de nuit, après une intégration et un déploiement eux-mêmes automatisés. Au matin, on analyse les sujets qui n'ont pas bien fonctionné pour éventuellement les

corriger. Sans une telle automatisation, mieux vaut viser une intégration hebdomadaire ou bihebdomadaire, avec un test minimal de non-régression.

L'intégration continue est très difficile à obtenir mais offre de très nombreux avantages, notamment les suivants :

- On dispose d'éléments tangibles pour vérifier la réalité des livraisons et leur qualité et les comparer aux engagements de livraison.
- La pression est maintenue de façon continue sur les équipes de développement, qui ne peuvent masquer leurs problèmes de production en espérant se rattraper par la suite.
- Les problèmes d'intégration entre les productions des différentes équipes sont détectés immédiatement si les modules livrés ne s'intègrent pas bien avec le reste de l'application.
- Si des tests automatisés sont mis en place, chaque intégration est testée, et les régressions sont immédiatement détectées, rendant ainsi leurs corrections plus aisées puisqu'on sait qu'elles proviennent obligatoirement des derniers éléments livrés.
- Le bon déroulement des tests automatisés de l'intégration quotidienne permet également de vérifier le bon fonctionnement des procédures en place, notamment sur la gestion du référentiel.

L'avantage le plus important est bien sûr de confronter la livraison avec la réalité et la validation de la qualité du livrable. On n'a même plus à demander aux chefs de projet si la fonctionnalité est livrée pour la considérer comme terminée. On s'adresse directement à l'équipe de test pour recueillir son avis sur le livrable. Le chef de projet livre tout naturellement sa production pour être testée dès que le code source est prêt.

L'obtention d'une intégration continue est un objectif difficile, qui ne peut être atteint que si l'on a réussi à mettre en œuvre un ensemble de procédures. Pour parvenir à mettre en place une intégration continue, il faut avoir su gérer correctement les points suivants :

- Un processus efficace de livraison dans le référentiel afin que les différents streams soient correctement gérés. Cela nécessite généralement une gestion efficace de tout le cycle de développement et de test.
- Un mode de promotion des éléments afin de pouvoir qualifier les builds.
- Un planning efficace permettant d'attendre harmonieusement les livrables dans un ordre cohérent.
- Une automatisation des déploiements et des tests sans laquelle l'intégration continue n'a qu'un intérêt réduit.

La recherche de l'automatisation des tests est particulièrement importante lorsqu'on gère des développements en offshore. Il s'agit d'abord d'automatiser la construction du build, donc l'extraction du référentiel des codes source, la compilation de ces codes et le déploiement sur la plate-forme cible. Une fois le déploiement réalisé, on peut engager les tests automatisés qui signalent les régressions sur le build, tant fonctionnelles qu'à l'égard des performances.

La production de builds continus testés automatiquement chaque nuit garantit l'application de procédures strictes et permet de suivre exactement la progression du projet quant à la réalité des livraisons.

Les tests unitaires

Les tests unitaires sont considérés comme une étape essentielle pour parvenir à un bon niveau de qualité de la production. Le développeur teste les fonctionnalités élémentaires d'un programme indépendamment de la façon dont il a été codé afin d'en valider le fonctionnement. C'est la première étape des tests visant à vérifier que le programme est intégrable et testable.

Il est important de limiter les allers-retours entre les équipes de développement et de test. Pour les réduire dès les premières livraisons des exécutables, il convient de s'assurer que les exécutables livrés aux équipes de test sont d'un niveau de qualité suffisant pour ne pas recéler de dysfonctionnements. Pour cela, le développeur prend en charge le premier niveau de tests unitaires afin de vérifier que les exécutables implémentent les fonctionnalités décrites dans les cas d'utilisation.

La plupart des développeurs n'aiment guère tester leurs réalisations et transmettent trop rapidement leur code aux équipes de test, estimant qu'ils trouveront bien les anomalies. Ces échanges sont consommateurs de temps. Les équipes de test s'arrêtent en rencontrant des anomalies bloquantes, qui doivent être documentées et retournées aux développeurs. Si les anomalies sont trop nombreuses, il est rapidement impossible d'anticiper la date à laquelle l'exécutable sera suffisamment stable. Les équipes de test s'impatientent, perdent du temps à rédiger des change requests, et une tension en résulte souvent entre les testeurs et les développeurs.

Si, au contraire, la livraison est tout de suite d'un niveau de qualité suffisant, l'équipe de test ne trouve que des anomalies mineures, quelques anomalies majeures et dans de rares cas des anomalies bloquantes.

Les scénarios des tests unitaires peuvent être définis par les équipes de test, qui demanderont que ceux-ci soient en premier lieu réalisés par les équipes de développement ou par les chefs de produit qui les associent directement aux cas d'utilisation. On peut décider de formaliser le jeu de données utilisé lors de ces tests afin qu'il soit représentatif de tous les cas fonctionnels possibles. Le résultat des tests unitaires est livré aux équipes de test avec l'exécutable de façon à garantir un certain niveau de qualité.

Automatisation des rapports de suivi

De la même façon que l'on peut automatiser les tests, il est souhaitable d'automatiser les rapports de suivi des développements afin d'éliminer toute interprétation humaine dans l'appréciation de l'avancement du projet.

On cherche alors à s'appuyer autant que possible sur les sources d'information dont on dispose. Si l'on a pu mettre en place le référentiel, la gestion du changement, un processus avec des procédures strictes, un planning représentant réellement la progression du projet et des résultats de test fiables, on dispose de l'essentiel des informations dont on a besoin pour construire des rapports de suivi.

Les procédures doivent être organisées de sorte à obtenir ces informations régulièrement et à donner une vision fiable et objective de l'avancement du projet. Si nécessaire, un détail précis de certains sujets qui ne fonctionnent pas correctement permet de choisir les meilleures actions correctives.

Conclusion

La mise en place d'une méthodologie pour l'offshore se résume à certains principes essentiels, sur lesquels il importe de se concentrer. À l'inverse, il ne faut pas se laisser dépasser par le grand nombre de procédures et documents qui sont d'une importance réduite mais consomment beaucoup de temps.

Plusieurs principes clés ne tolèrent aucune dérogation, notamment les suivants :

- On respecte l'organisation des hommes, conçue pour garantir un fort niveau de motivation et de productivité. Mieux vaut disposer d'une équipe motivée et organisée, avec des rôles bien définis, plutôt que de procédures censées organiser tous les aspects du projet. Les procédures doivent venir en support de l'organisation humaine pour la rendre plus efficace et régler les échanges entre les équipes.
- On applique un mode de travail itératif, de façon à maintenir des objectifs stables à court terme. Les managers planifient ainsi le travail plus aisément, et chacun dispose en permanence d'objectifs à court terme permettant de vérifier la progression du projet sur des livrables tangibles.
- On utilise des outils pour organiser et appliquer strictement les workflows autour de la gestion du référentiel et du changement. On dispose ainsi de mesures de la qualité des livrables et de moyens de travailler en équipe efficacement.
- On cherche à mobiliser toutes les équipes sur la qualité des livrables individuels et d'équipe. La qualité des livrables à tous les niveaux est le garant d'un meilleur travail d'équipe et d'une plus grande fiabilité des plannings.

Ces règles universelles sont d'autant plus importantes lorsqu'on travaille en offshore, où tous les défauts de qualité et de communication sont amplifiés.