Java Productivity Primer

Douze recommandations pour augmenter votre productivité avec une usine logicielle

Juillet 2009

Cabinet d'Architectes en Systèmes d'Information

Table des matières

Résumé

Les équipes de développement attendent de Java qu'il les aide à augmenter leur productivité. Le présent document fournit un ensemble de recommandations qui vous aideront à être plus productif avec Java lorsque vous développez des applications d'entreprise. Il comprend une brève présentation des principes du développement moderne, et vous guide à travers les bases de l'organisation de vos activités autour d'une usine logicielle productive, c'est-à-dire un ensemble d'outils pour automatiser les tâches répétitives et aider à mettre en œuvre un workflow de développement efficace. Les experts seront sans doute déjà familiers avec tout ou partie de ces recommandations. Néanmoins, le présent document est un bon moyen de lancer les discussions et de partager les connaissances.

Les auteurs, une équipe d'architectes d'OCTO Technology, conçoivent et fournissent l'assistance pour une telle usine logicielle au sein d'une grande institution financière. Ce document est le fruit de leur expérience sur le terrain, des leçons qu'ils ont retenues dans leur quête de productivité, et des avis de la communauté des architectes logiciels d'OCTO.

Introduction

La plate-forme Java est en perpétuelle évolution. Adopter les nouveaux modèles d'architecture (par exemple les applications Web) et développer des logiciels d'entreprise en utilisant cette plate-forme devient un véritable défi. L'écosystème Java dont la grande vivacité rime aussi trop souvent avec grande complexité, est souvent quelque peu déroutant pour ceux qui découvrent la plate-forme pour la première fois. Certains sont habitués à la productivité que l'on peut attendre d'environnements de développement clients/serveurs classiques comme PowerBuilder ou VB. Du coup, ils éprouvent des difficultés à trouver le moyen d'atteindre, pour ne pas dire dépasser, un tel niveau de productivité avec Java.

Etre productif avec Java ?

Les performances au niveau de la productivité, mises en avant par les utilisateurs et par les fournisseurs Java, est l'une des questions les plus discutées dans la communauté Java. Ce souci de productivité est au cœur des évolutions de Java Enterprise Edition (JEE 5 puis JEE 6 [1]) de SUN. Il a entraîné des changements notables, à commencer par le langage Java lui-même avec Java 5 [2], ainsi que l'adoption de modèles de programmation plus performants, issus de la mouvance des conteneurs légers (les lightweight containers). Toutefois, le développement Java en entreprise s'apparente encore à un parcours semé d'embuches et d'impasses.

Dans les entreprises, les systèmes existants sont souvent intégrés de multiples façons, avec de nombreux flux et référentiels de données de qualité très variables. Ils forment ensemble un empilement de couches logicielles avec lesquelles il faut interagir dans nos projets Java.

Face à ces défis, la communauté des développeurs Java adopte des pratiques améliorant la productivité. Elles combinent des outils de développement et des processus agiles et efficaces, regroupés dans la notion d'usines logicielles (ou software factories). Le présent document a pour ambition de vous présenter ces formes de développement. Il est structuré sous forme de douze recommandations pratiques, saupoudrées de conseils directement applicables.

C'est quoi, la productivité, finalement ?

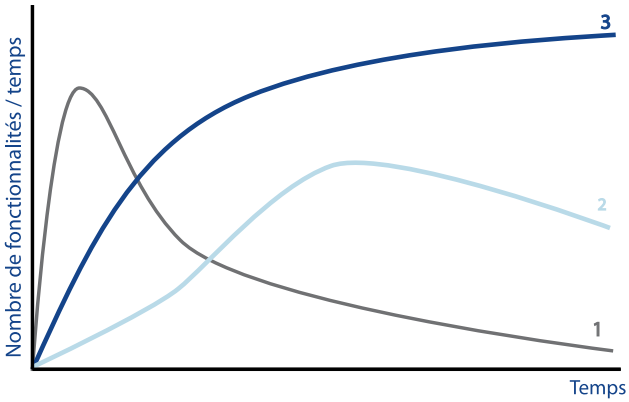
Même si la productivité est une notion délicate à définir, nous pensons qu'un bon indicateur pour la mesurer est le nombre de fonctionnalités sans erreur, ajoutées au logiciel au cours d'une période donnée.

Dans cette définition, arrêtons-nous sur quelques aspects :

- « **Nombre de fonctionnalités** » : c'est simple ; si vous n'êtes pas en mesure d'ajouter les fonctionnalités qui ont été demandées, votre productivité est nulle.
- « **Sans erreur** » : une fonctionnalité qui ne marche pas est sans valeur, vous ne la comptez pas. Vous pouvez être tenté de compter une fonctionnalité livrée même s'il y subsiste juste un petit bug, qui ne la bloque pas. Gardez à l'esprit que cela aura probablement un impact négatif sur votre productivité future, et ce pour au moins deux raisons. La première : ce bug peut causer des problèmes quand vous ajouterez la prochaine fonctionnalité. La seconde : quand vous avez des douzaines de bugs à corriger, vous ne travaillez pas sur les vraies fonctionnalités.
- « **Sur une période donnée** » : ici, nous voulons choisir une durée suffisamment réduite pour nous permettre de faire des mesures fréquentes. N'oublions pas qu'avec peu de mesures, cet indicateur ne signifie pas grand-chose, car il est très difficile de le comparer sur des projets avec des équipes de tailles différentes, des durées différentes, etc. De plus, vous voulez pouvoir détecter suffisamment tôt l'effet sur votre productivité, qu'il soit positif ou négatif ; privilégiez les courtes durées, entre une semaine et un mois.

La productivité s'observe au fil du temps.

Même si vous devez faire des mesures fréquentes, il faut s'intéresser aussi à la productivité à long terme, car vous souhaitez qu'elle reste la plus élevée possible, tout le temps. Vous serez même amenés à vivre des périodes de très faible productivité lorsque vous modifierez le code existant afin de la maximiser dans le futur.



Dans ce graphique, nous présentons trois profils extrêmes de productivité que l'on peut mesurer sur des projets.

(1) Le premier montre des projets où l'on plonge directement dans le code, en utilisant les outils disponibles les plus simples pour faire ce qu'on a à faire. Malheureusement, lorsque ces projets grossissent ils atteignent les limites de ces outils, et la productivité chute rapidement. Puisque ce n'était pas la tendance attendue, elle est difficile à inverser.

(2) Le deuxième montre les projets qui essayent de tout anticiper. Un temps important est gaspillé à choisir le meilleur outil possible, choisir l'architecture la plus logique, puis vérifier que tout fonctionne comme il devrait. Au début, la productivité est très faible, mais tout va bien : « ce sont des bons choix qui nous aideront à augmenter notre productivité ». C'est sûrement vrai dans certains cas. Mais pour beaucoup d'autres, on aboutit à des systèmes excessivement complexes, et ce retard initial de productivité n'est jamais rattrapé.

Ces deux profils, certes caricaturaux, présentent certains avantages : le premier connaît une productivité élevée au début, et le second une meilleure productivité au bout de quelque temps. On pourrait discuter de la meilleure de ces deux approches, mais nous passerions à côté de l'essentiel. En fait, les deux approches souffrent du même problème : leur productivité décroît ! Voilà qui est essentiel.

Ce que vous recherchez en réalité, c'est une productivité élevée au plus tôt, et qui continue à augmenter avec le temps. Cette caractéristique est représentée par la troisième ligne sur notre schéma. (3)

Les douze recommandations que nous présentons vont vous aider à mettre en place les conditions nécessaires pour conserver cette caractéristique dans vos projets.

Recommandations pour la productivité

Recommandation 1 – Choisissez une plate-forme Java productive taillée pour votre projet

Notre expérience nous rend plutôt optimistes sur les possibilités de développement productif en Java, et vous en lirez davantage dans les chapitres suivants. Nous allons néanmoins commencer par une mauvaise nouvelle -une prise de conscience pénible par laquelle nous devons tous passer : on ne peut pas acheter aveuglément une plate-forme Java quelconque et tout son attirail, et l'utiliser telle quelle, croyant que « c'est le métier du fournisseur de nous donner la plus aboutie, la plus productive des plate-formes Java ».

En pratique, ça ne marche pas comme ça. Il faut garder à l'esprit qu'en règle générale, l'objectif principal d'un éditeur logiciel est de vous en vendre un maximum, pas d'augmenter votre productivité. Vous allez devoir entrer dans le jeu et mener le sujet vous-même !

Un des piliers de cet effort consiste à choisir une plate-forme Java qui fonctionnera parfaitement dans votre contexte. C'est important car il n'existe pas seulement « une » plate-forme Java, mais un écosystème industriel incroyablement riche, constitué de milliers d'outils et de bibliothèques proposés par un grand nombre de fournisseurs et de sociétés.

J2EE 1.4 [3], un « standard » d'entreprise autrefois promu par SUN et divers fournisseurs, a beaucoup péché au niveau de la productivité, ce point a été abondamment discuté dans la littérature sur le développement . Cette situation a fait émerger une multitude de frameworks offrant de bien meilleurs profils de productivité. SUN a incorporé beaucoup de leurs caractéristiques et fonctionnalités dans les dernières versions de ses plates-formes de référence, comme JEE 5 et 6 (qui succèdent à J2EE 1.4).

Néanmoins, JEE n'a pas encore rattrapé son retard en termes de productivité de développement. Elle se contente de définir une plate-forme Java générique, mais ce n'est pas forcément la plate-forme qui convient à votre projet. Vous n'utiliserez vraisemblablement que certaines parties de JEE, en combinaison avec des frameworks et des outils tiers ou développés en interne.

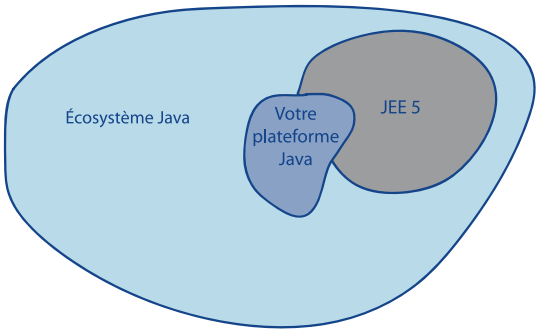


Figure 1 – Le choix des technologies appropriées

Les outils et frameworks Java ne sont pas tous à un niveau de maturité suffisant pour qu'on puisse les considérer comme équivalents. On en est même loin : les mauvais choix quant à la technologie et à l'architecture peuvent être extrêmement néfastes pour votre productivité et pour la qualité de votre logiciel. C'est pourquoi il est important de choisir et combiner les technologies Java appropriées.

Trucs et astuces

Quand les gens nous demandent «Quelle est la meilleure architecture Java ?» , « Quel est le meilleur framework ? » ou « Avec quelle plate-forme dois-je démarrer mon projet ? » nous sommes régulièrement coincés, car il n'y a pas de solution « taille unique ». Cela dépend de tellement de facteurs comme l'expérience de l'équipe, sa taille, ou les systèmes existants, que nous sommes incapables de répondre sans information précise sur le contexte. Cependant, nous avons quelques éléments de l'équation :

- Nous avons besoin d'être productifs tout de suite.
- Une solution efficace pour de grands projets avec des dizaines de développeurs sur plusieurs années est probablement une usine à gaz pour un petit projet de quelques semaines avec un ou deux développeurs.
- Nous savons qu'en lançant un projet aujourd'hui, nous allons être confrontés à des problèmes déjà rencontrés par le passé sur d'autres projets.
- Nous ne savons pas quand nous tomberons sur un problème en particulier, ni même si ce problème surviendra.

Notre réponse à cette question est donc de **démarrer avec une plate-forme simple** permettant d'être **aussi productif que possible**. Pas la peine pour autant d'aller vers la plus clinquante. Commencez simplement avec la plate-forme qui lève l'obstacle le plus important que vous ayez rencontré, et qui est assez évolutive pour vous permettre de résoudre les suivants. Vous l'améliorerez jour après jour.

Ne payez pas le prix fort en prévision de tout ce qui pourrait arriver ; **payez simplement le bon prix au bon moment.**

Vous trouvez ci-dessous un exemple de plate-forme Java que nous avons utilisée dans plusieurs projets Web. Il présente certains des composants qui nous ont été particulièrement utiles :

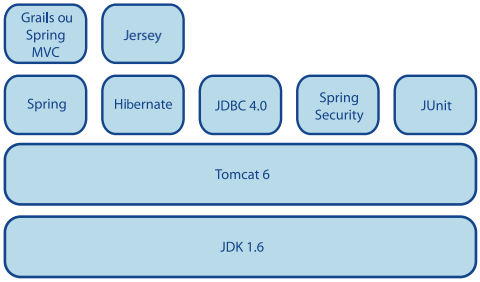


Figure 2 – Exemple de plate-forme Java évolutive et économique

À l'heure où nous écrivons ces lignes, le JDK (Java Development Kit) 1.7 est la dernière et la meilleure version disponible du JDK de SUN. Elle offre une plate-forme de développement et d'exécution Java standard et inclut des avancées récentes qui augmentent considérablement la productivité. Certaines de ces améliorations proviennent d'évolutions sur le langage lui-même (par exemple les annotations, introduites avec Java 5, qui réduisent le besoin en fichiers de configuration XML). D'autres résultent de frameworks Java nouveaux ou améliorés [21].

Tomcat 6 est un serveur d'application open-source de grande qualité. Il fournit un moteur de Servlet/JSP, et comprend un serveur Web.

Spring [4] est un framework open-source qui aide à mettre en œuvre des motifs récurrents typiques des applications d'entreprise (par exemple la configuration de composants, le cycle de vie des objets, l'accès aux données, la gestion des transactions, etc.). Ces dernières années, Spring s'est imposé comme une alternative populaire à J2EE, en proposant des modèles de programmation plus productifs, ainsi qu'un framework de développement plus simple et plus attractif. Spring a également fortement encouragé la testabilité des composants applicatifs. Plus tard dans ce cahier, nous évoquerons comment les tests peuvent devenir un facteur clé de productivité (même si ça peut sembler surprenant au premier abord).

Hibernate [5] peut doper la productivité des développements liés aux accès aux bases de données relationnelles. C'est un framework sophistiqué pour le mapping objet-relationnel. Il permet de créer et de synchroniser automatiquement un graphe d'objets en mémoire avec les données correspondantes de la base. Hibernate a d'ailleurs inspiré JPA [6] (Java Persistence API), le dernier standard de persistance objet de SUN, dont il est l'implémentation de référence.

JDBC est l'API de bas niveau qui permet d'interagir avec les bases de données en utilisant le langage SQL. Depuis quelques années, on considère son utilisation directe comme particulièrement fastidieuse ; cette situation a conduit à développer des frameworks de mapping objet-relationnel comme Hibernate. Cependant, dans certains cas, il reste tout de même préférable d'utiliser JDBC directement, plutôt qu'à travers ces outils. JDBC 4.0 [22], qui constitue une évolution majeure du modèle de programmation JDBC, est plus productif que ses versions précédentes.

Spring Security [23] (anciennement connu sous le nom d'Acegi) est un framework open-source efficace et modulaire pour gérer la sécurité des applications web.

JUnit [17], simple et stable depuis des années, est tout simplement l'inamovible framework pour construire ses tests unitaires. La version 4 tire parti des annotations pour accélérer vos développements.

Spring MVC [24] est un composant optionnel de Spring. C'est un framework qui permet de gérer la navigation des utilisateurs. Sans pour autant en révolutionner les concepts, c'est une évolution réussie du classique Struts.

Grails [25] est unique en son genre. Il repose sur le langage de programmation Groovy, un langage dynamique de plus haut niveau qui s'exécute dans la plate-forme Java, et peut interagir de façon transparente avec les objets Java. Grails offre des gains de productivité spectaculaires, à condition qu'il puisse s'intégrer dans la culture et les préférences de l'équipe de développement.

Enfin, pour mettre en œuvre des services Web, le framework **Jersey [26]** nous semble être un excellent choix. On notera que, dans le présent document, nous utilisons le terme Web services pour désigner des services auxquels d'autres applications accèdent via HTTP. Le framework Jersey est conçu selon le modèle d'architecture REST, qui est le modèle d'architecture natif de HTTP. Il vous permet de tirer avantage des qualités intrinsèques du Web (évolutivité, couplage lâche, etc.), et reste plus accessible que les frameworks de type RPC ou SOAP (par exemple JAX-RPC/JAX-WS de JEE). Pour plus de détails, voir [27].

Points clés

- Il n'existe pas *une seule vraie* plate-forme Java mais un écosystème complet qui vous permettra de construire, selon vos besoins, une plate-forme d'exécution et de développement sur mesure.
- Méfiez-vous des plates-formes préfabriquées et des modèles d'architectures des fournisseurs de logiciels. Ils sont souvent surdimensionnés et seront inefficaces si on les utilise sans tenir compte des besoins réels et de l'environnement.
- Les améliorations récentes dans le développement Java peuvent vous rendre bien plus productifs. Surveiller les évolutions dans le monde Java et mettre à jour votre plate-forme de manière appropriée vous aideront à tirer avantage de ces nouvelles démarches techniques et organisationnelles.
- Choisissez une plate-forme simple et productive dont vous savez qu'elle apportera les solutions à vos problèmes : payez le bon prix au bon moment.

Recommandation 02 – Mettez en place une usine logicielle adaptée à votre équipe de développement

Comme vous le savez, un projet requiert une infrastructure technique pour permettre aux développeurs d'effectuer les tâches suivantes :

- Concevoir et produire du code
- Coordonner les modifications effectuées sur le code partagé, avec le reste de l'équipe.
- Gérer le processus de production, ou build (ce qui inclut le traitement des dépendances entre les packages et autres joyeusetés)
- Suivre l'avancement du projet : ce qui est fait et ce qui fonctionne correctement, ce qui manque ou n'est pas terminé
- Effectuer un suivi d'anomalies et autres problèmes
- Dérouler les tests de l'application

Beaucoup d'équipes l'apprennent dans la douleur : installer un IDE (environnement de développement intégré) sur le poste de travail de chaque développeur pour commencer à coder n'est pas suffisant. Une usine logicielle est un ensemble d'outils de développement, de modèles et de processus dont le but est d'industrialiser vos développements en identifiant des actions élémentaires (comme celles listées ci-dessus), et de vous aider à les réaliser de manière productive, extrêmement structurée et automatisée.

Une usine logicielle ne va-t-elle pas transformer les développeurs en simples ouvriers, comme sur une ligne d'assemblage ?

Pas du tout. Bien au contraire, l'usine logicielle telle qu'on la décrit dans le présent document prend en charge la plupart des tâches fastidieuses, répétitives et sources d'erreurs, et de fait, libère les développeurs en automatisant ce travail. Les développeurs peuvent alors se concentrer sur des tâches plus intéressantes et plus exigeantes : innover, résoudre les vrais problèmes, programmer, prendre du plaisir dans leur travail et, nous l'espérons, créer du bon logiciel.

Votre usine logicielle doit couvrir les divers besoins de votre projet Java, depuis la programmation (en fournissant un IDE, une sélection de frameworks et d'exemples de code) jusqu'à l'intégration de votre code avec celui des autres membres de l'équipe, en passant par les tests, le suivi des anomalies, etc. De plus, nous recommandons d'adopter une approche modulaire afin de pouvoir ajuster votre usine à divers projets et environnements.

Installer une usine logicielle complète peut prendre du temps, et nous avons vu des projets (y compris certains des nôtres) où les développeurs ont pris plus de temps à installer et à fignoler leur usine logicielle qu'à réellement coder. Au final, aucun gain de temps. Ainsi, une fois de plus, nous vous

recommandons d'avancer progressivement, en installant un ou deux éléments à la fois : **commencez par la plus petite usine dont vous avez besoin pour être productif, et apportez-y régulièrement des améliorations afin d'être le plus productif possible pendant toute la durée du projet.**

Par exemple, vous pouvez commencer par mettre en place un référentiel de code source partagé, ensuite un serveur d'intégration continue, puis aller plus loin avec un outil de suivi d'anomalies, etc. (nous passerons en revue ces outils un peu plus loin). Cela devrait éviter d'assommer votre équipe avec trop de choses à la fois, et vous permettre d'adopter progressivement de nouveaux outils et pratiques. Avec le temps, plus vous maîtriserez votre usine, plus vous pourrez la peaufiner.

Points clés

- Le développement logiciel avec Java implique un grand nombre de tâches variées et complexes.
- L'automatisation est la clé d'une productivité durable.
- Industrialisez vos développements en mettant en place une usine logicielle.
- Installez la plus petite usine logicielle pour vos besoins, pour être le plus productif rapidement, et engagez-vous à l'améliorer continuellement.

Recommandation 3 – Pour les petits projets, mettez en place une petite usine logicielle

Votre usine va varier selon l'organisation de votre équipe. N'avoir qu'**un seul développeur** (ou deux, si vous pratiquez la programmation en binôme) offre souvent le meilleur niveau de productivité, car toutes les tâches liées à la synchronisation avec les autres développeurs disparaissent. Dans ce cas, vous installerez une usine logicielle minimale sur son poste de travail.

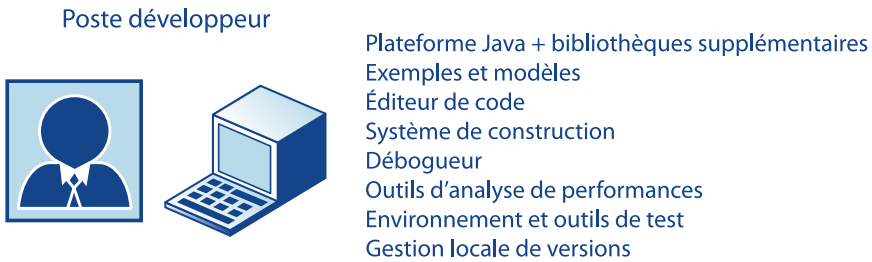


Figure 3 – Exemple d'usine logicielle minimale

Typiquement, un outil intégré fournit plusieurs composants de l'usine logicielle. Par exemple, nous utilisons fréquemment Eclipse [16], un environnement de développement intégré qui comprend un éditeur de code, un débogueur et un système local de construction. Eclipse est l'un des meilleurs IDE Java et, à ce titre, bénéficie d'un large soutien de l'industrie. Il peut être personnalisé et étendu par de nombreux plug-in [7] (il y en a des centaines) si vous avez des besoins spécifiques. (Voir la Recommandation 9 pour plus d'informations sur les IDE)

En fonction des préférences du développeur, un gestionnaire de versions peut être utilisé, ou bien versions et branches peuvent être gérées « manuellement » sans outil dédié (par exemple juste en copiant les fichiers du projet). De plus, Eclipse lui-même permettra de conserver automatiquement un historique de toutes les modifications effectuées sur chacun des fichiers sources sur le poste de développement, ce qui permet par exemple de retourner à un état antérieur de votre code, ou encore de visualiser l'historique des modifications apportées à un fichier.

Notez que si le projet va être développé activement sur une longue période (plusieurs mois) ou si le développeur est susceptible d'être remplacé en cours de développement, il est préférable d'utiliser un véritable système de gestion de versions, comme Subversion. Cela vous aidera à garantir une meilleure traçabilité de l'historique du projet, ce qui s'avèrera utile en cas de changement de développeur, ou pour comprendre un code créé il y a longtemps.

Avec deux développeurs, vous pouvez opter pour l'installation d'une usine minimale sur le poste de travail de chaque développeur et les laisser se coordonner manuellement. Cela peut être une option quand les développeurs travaillent dans la même pièce, que le projet doit se dérouler sur une courte période (quelques semaines), ou s'il peut être divisé en plusieurs parties relativement indépendantes. Toutefois, la configuration d'une usine logicielle plus sophistiquée (comme celle décrite dans la recommandation suivante) fonctionnera également très bien dans un tel contexte.

Pour les projets qui nécessitent trois développeurs ou plus, il faudra industrialiser davantage votre processus de développement afin d'être productif. Dans ce cas, vous utiliserez l'usine logicielle plus extensible décrite dans les recommandations suivantes. Notez que, dans tous les cas, avoir l'équipe de développement dans la même pièce est bien meilleur pour votre productivité.

Chez OCTO, nous aimons Subversion, alors nous l'utilisons aussi souvent que possible, même pour les petits projets avec un seul développeur. Par exemple, alors même que nous rédigeons ce document, celui-ci est dans notre référentiel Subversion.

Points clés

- Les projets avec un seul développeur (ou un binôme) sont potentiellement plus productifs car aucun processus de synchronisation n'est requis.
- Pour ces projets, une usine logicielle réduite permettra une productivité optimale.
- Des projets avec deux développeurs peuvent utiliser une usine minimale ou une plus avancée, en fonction des préférences des développeurs et des caractéristiques du projet (durée et rotation de l'équipe).

Recommandation 4 – Pour les projets plus grands, mettez en place une usine logicielle sur une plate-forme partagée

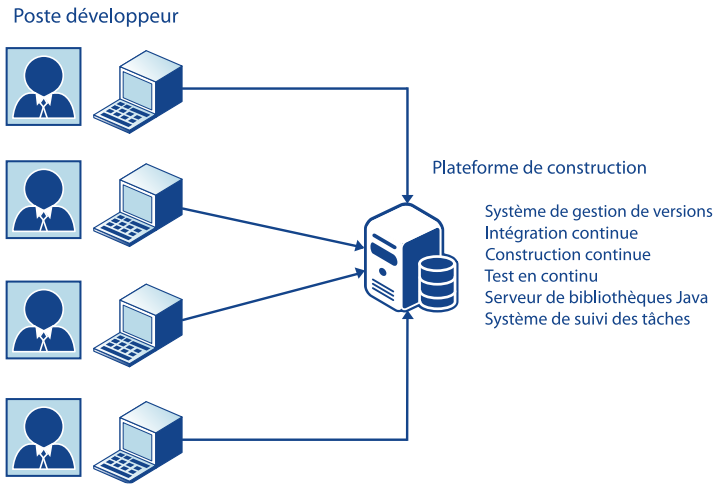


Figure 4 – Usine logicielle avec plate-forme partagée

Pendant le développement, le poste de travail de chaque développeur doit comporter :

- Un ensemble productif d'outils de développement (éditeur de code, débogueur, système de construction, etc.).
- Tout le code source et les autres ressources dont a besoin le développeur pour construire et tester ses développements.
- Toutes les bibliothèques nécessaires au développeur. Cela comprend l'environnement standard Java ainsi que tous les frameworks additionnels.
- L'infrastructure requise pour tester le code. Par exemple : si le code interagit avec une base de données, le poste de travail utilisé pour le développement doit soit comporter une base de données de test, soit être connecté à une base distante de test, si le code a besoin de s'exécuter dans un serveur d'application, il faut qu'il y en ait un de disponible sur le poste de travail, etc.
- Les outils nécessaires pour interagir **de façon transparente** avec la plate-forme de construction.

Dans cette configuration, chaque développeur peut travailler de façon indépendante. Quand un développeur considère qu'un morceau de code est prêt, il l'envoie sur le serveur d'intégration hébergé par la plate-forme de construction. De même, quand il veut mettre à jour sa base locale de code pour profiter des dernières contributions de ses collègues, il synchronise son référentiel local de code avec celui du serveur d'intégration.

Trucs et astuces

Une fois que chaque développeur a synchronisé son code, commence une tâche qui peut prendre beaucoup de temps : réparer les erreurs des autres. Il n'y a rien de plus frustrant que de synchroniser son code, découvrir qu'un test ne passe plus, passer du temps dessus… et finalement réaliser qu'on n'est pour rien dans cette régression. Rien de plus frustrant, sauf peut-être de récupérer du code qui ne compile pas : **on ne peut pas travailler avec du code qui ne compile pas.**

Dans certaines équipes, nous utilisons deux règles simples pour éviter de tels problèmes :

- Quiconque aura validé un code qui n'aura pas réussi les tests devra payer le café à toute l'équipe.
- Quiconque aura validé un code qui ne compile plus devra payer les croissants le lendemain.

Si ces mesures n'ont pas résolu tous les problèmes, elles ont néanmoins permis d'en solutionner la plupart. Quant aux problèmes persistants, elles ont rendu leur traitement beaucoup plus stimulant.

À l'instar des postes de travail des développeurs, la plate-forme de construction est un environnement complet où le projet peut être construit et testé. La principale différence est que le serveur d'intégration contient le résultat de toutes les contributions des développeurs. Les diverses versions et branches du code source et des ressources du projet sont stockées et gérées par le gestionnaire de versions.

Pour les tests, la plate-forme de construction contient des environnements de test contrôlés, isolés proprement, et plus représentatifs des futurs contextes d'exécution que les environnements des postes de travail.

Hum… Tout cela a l'air bien gentil, mais ça doit être pénible à installer !

Mettre en place une infrastructure complète de développement conforme aux meilleures pratiques de développement logiciel peut être une tâche décourageante. Ainsi pour éviter de passer des jours à installer chaque poste de développement pour plusieurs développeurs, vous allez vouloir packager tous les composants de votre usine et en automatiser l'installation. D'abord vous devriez le faire pour les postes de travail : il est moins prioritaire de le faire pour la plate-forme de construction puisqu'elle sera généralement partagée par plusieurs projets et donc installée une seule fois.

Diverses technologies sont disponibles pour automatiser vos installations. Par exemple, sur Windows, nous utilisons NSIS (Nullsoft Scriptable Install System [19]).

Les environnements de tests dans la plate-forme de construction doivent être aussi proches que possible de l'environnement de déploiement cible. Vous inclurez probablement des ressources spécifiques dans cette plate-forme, par exemple une base de données ou un système ERP ; en dernier recours, vous la connecterez à des ressources externes existantes.

En outre, la plate-forme de construction est le lieu idéal pour effectuer les sauvegardes périodiques du code source.

Points clés

- Les développeurs doivent être capables de développer simultanément.
- Les développeurs doivent à chaque instant être capables d'intégrer et de tester facilement.
- Pour les projets avec trois développeurs ou plus, fractionnez l'usine entre des postes de développeurs et un serveur partagé, pour une intégration et des tests en continu.
- Automatisez l'installation des postes de développement quand les installations répétées commencent à devenir pénibles.

Recommandation 5 – Ne perdez pas votre temps : automatisez votre processus de construction

Du point de vue du développeur, la construction (c'est-à-dire la transformation du code source et des ressources associées en un système exécutable) peut être réalisée localement sur son poste de travail ou bien à distance sur le serveur d'intégration hébergé par la plate-forme de construction. Dans tous les cas, la construction est souvent un processus complexe, qui se déroule en plusieurs étapes telles que :

- Localiser et récupérer le code source modifié depuis la dernière construction, ainsi que les éléments qui en dépendent
- Traiter ces éléments à l'aide de divers outils (préprocesseurs, compilateurs, etc.)
- Produire des artéfacts exécutables, prêts au déploiement (par exemple les applications exécutables, les fichiers .jar ou .war)

Si gérer toutes ces étapes à la main peut porter un coup fatal à votre productivité, les automatiser va vous faire gagner en simplicité. Les outils de construction intelligents permettent cette automatisation : par exemple, ils calculent les dépendances entre bibliothèques et les assemblent, ils ne demandent que les paramètres de configuration spécifiques et déduisent tous les autres, ils permettent de faire échouer la construction quand certaines conditions ne sont pas respectées (échec de tests unitaires par exemple), etc.

D'accord, c'est facile d'énoncer « faites ci » ou « faites ça » mais moins facile de le faire. Pourriez-vous nous expliquer comment nous sommes supposés automatiser cette *construction* ?

La clé, c'est de mettre en place un système qui connait le mieux possible les différents aspects de votre projet pour ensuite orchestrer automatiquement vos différents outils. Par exemple, nous utilisons fréquemment Maven 2 [8] qui utilise une description des éléments du projet sous forme de fichier XML : le Project Object Model (ou POM). Vous avez juste à fournir votre POM à Maven (et à suivre les conventions Maven), et ce dernier coordonnera la récupération des sources avec Subversion, la compilation avec Javac, les tests unitaires avec Junit, pour finalement assembler votre livrable.

Trucs et astuces

Maven doit être transparent pour les développeurs.

Différentes solutions existent pour intégrer vos POM à Eclipse. Aujourd'hui, le plugin Eclipse ***m2eclipse*** est la meilleure solution dans une majorité de projets (utilisez une version à jour). Si toutefois le votre s'éloigne trop des conventions Maven, le plugin Maven ***eclipse:eclipse*** offre une alternative efficace (cf. documentation technique de ces deux solutions).

En synthèse, **vous devez travailler dans Eclipse.**

Si Eclipse n'arrive pas à compiler, tester ou lancer votre application, cela vient probablement d'un problème de configuration. Dans ce cas, n'essayez pas de court-circuiter l'IDE (par exemple, en utilisant Maven directement) : réparez votre configuration et faites fonctionner Eclipse avec ce POM.

Dans les chapitres suivants, nous vous conseillerons de construire et de tester vos composants, et votre projet entier, le plus souvent possible (généralement plusieurs fois par heure).

Cependant, ceci ne doit pas vous distraire de votre activité de programmation : ces processus doivent être les plus rapides possible. A titre de référence, nous préférons que les tests unitaires sur le composant en développement prennent quelques secondes, et quelques minutes pour l'application complète. De cette manière, vous pourrez le faire aussi souvent que vous le souhaitez en restant concentré.

Points clés

- Industrialisez votre processus de construction en utilisant un système de construction intelligent comme Maven 2.
- Assurez-vous que votre processus local de construction soit vraiment rapide.

Recommandation 6 – Gérez élégamment les développements de l'équipe : utilisez un système de gestion de versions

Pour être productif, vous devez développer libre de toutes contraintes. Par exemple, vous devez pouvoir développer à plusieurs sur le même code source. Ou encore, pouvoir essayer différentes implémentations et revenir en arrière si nécessaire (comme repartir d'une sauvegarde après une erreur honteuse dans un jeu vidéo).

Vous devez aussi pouvoir corriger des problèmes en production alors que vous travaillez sur la prochaine version : il vous faudra travailler sur plusieurs versions de vos projets.

Cela soulève un certain nombre de questions, comme :

- Comment stocker et conserver une trace des différentes versions de votre code source et des ressources associées ?
- Comment plusieurs développeurs peuvent-ils travailler simultanément sur le même code source ?
- Comment et à quel moment fusionner les modifications ?
- Comment savoir si des modifications concurrentes ont été réalisées ?
- Que faire si certaines modifications cassent temporairement une autre partie du projet ?
- Comment expérimenter sereinement diverses implémentations en étant assuré de pouvoir revenir en arrière ?
- Comment réparer un bug sur une version précédente alors que vous êtes en train de travailler sur la future version ?

Ce sont des problèmes récurrents et plutôt que de réinventer la roue, vous apprécierez un système qui les résout pour vous.

Le système de gestion de versions est avant tout le référentiel partagé pour le code et les autres artéfacts du projet. Il garantit l'intégrité de son contenu quelles que soient les interactions des développeurs (mise à jour et validation de code). Il peut gérer plusieurs versions et branches d'un même projet et permet même de comparer leurs différences. Il va également détecter les conflits potentiels lorsque plusieurs développeurs modifient le même morceau de code en même temps.

Ces possibilités sont fondamentales pour analyser la base du code et pour fusionner les modifications concurrentes. Les solutions courantes sont très perfectionnées, elles minimisent vos interventions dans cette étape délicate.

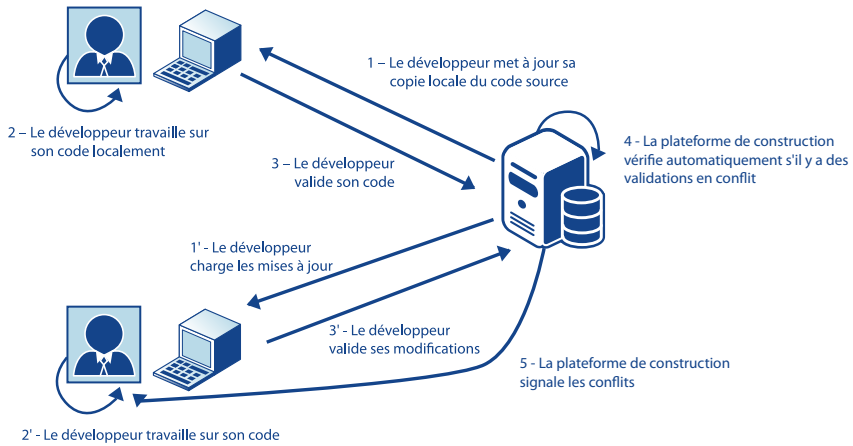


Figure 5 – Développement en équipe avec un système de contrôle de version

Pour le développeur, les principales étapes, quand il travaille avec un système de contrôle de version, sont :

- **Mise à jour :** ramener la dernière version des fichiers sur le poste de travail à partir du référentiel
- **Modification :** travailler sur le code
- **Validation :** remettre le code modifié dans le référentiel

Le système de gestion de versions vérifie qu'il n'y a pas de conflits potentiels, au moment où le code est validé. Il y a conflit quand deux développeurs modifient le même morceau de code, dans la même version du projet et en même temps. Comme chaque développeur travaille sur une copie locale du code, le conflit est détecté lors de la validation, quand le code est envoyé au référentiel. Le premier développeur qui valide verra sa soumission automatiquement acceptée. Le second développeur sera informé du conflit au moment de valider son code, ou quand il met à jour son code. Il pourra alors fusionner ses modifications avec celles du premier développeur.

Il est possible d'anticiper ces conflits en mettant à jour le code avant de le valider, ce qui permet de détecter et de résoudre les conflits.

Trucs et astuces

Il est beaucoup plus facile de fusionner du code qui a peu divergé. Préférez des mises à jour et des validations fréquentes à des fusions massives de code.

Dans le même esprit, validez votre code dans le référentiel central dès que vous considérez qu'il est suffisamment solide pour être partagé avec les autres (assurez-vous en particulier qu'il passe tous les tests). Au plus tôt les autres développeurs récupèreront vos modifications, au plus les conflits seront simples à résoudre : **synchronisez votre code tôt et souvent.**

Dans certains projets, nous encourageons même les membres d'une équipe à agir de façon égoïste : « Vous n'aimez pas résoudre les conflits ? Validez vos modifications avant les autres ! ».

Mettre à jour régulièrement semble être une bonne idée, mais que faire si un autre développeur a validé des modifications sur un code sur lequel je suis en train de travailler ? Cela ne va-t-il pas effacer mes propres modifications ?

Non, rassurez-vous. Dans ce genre de situation, le système de gestion de versions réagira en vous informant du conflit et vous aurez la possibilité de les fusionner.

Les développeurs interagissent très souvent avec le système de gestion de versions, il est donc impératif que son utilisation soit intuitive. Aujourd'hui, l'outil « standard » est Subversion [9], solution open-source très bien intégrée avec les autres outils de développement.

Trucs et astuces

Des outils graphiques puissants sont disponibles pour visualiser les modifications et résoudre facilement les conflits. Ils présentent côte à côte les versions de code, montrent les différences, et vous permettent de spécifier le résultat final souhaité. Un tel outil est par exemple inclus dans l'IDE Eclipse. Grâce au plugin Subclipse [20], il s'intègre de façon transparente avec *Subversion.*

Quand vous validez du code, vous avez la possibilité d'y associer un commentaire. Ce commentaire apparaîtra dès que vous, ou les autres développeurs, regarderez l'historique de validation (par exemple depuis l'outil de visualisation de l'historique dans Eclipse). **C'est pourquoi il est important de toujours laisser un commentaire explicite lors d'une validation.** Le commentaire doit être suffisamment informatif pour que vos collègues développeurs puissent comprendre le but des ajouts ou des modifications réalisés.

Tous les fichiers nécessaires à la production des livrables de votre projet doivent être stockés dans votre système de gestion de versions. Cette règle ne s'applique pas aux bibliothèques de provenance tierce. Vous les mettrez dans un entrepôt de bibliothèques Java. Notre équipe utilise fréquemment le traditionnel Archiva [10.a] mais d'autres solutions, notamment Nexus [10.b], arrivent sur le marché.

Points clés

- Utilisez un système de gestion de versions
- Synchronisez régulièrement votre code avec le reste de l'équipe
- Valider vos modifications le plus souvent possible
- Commentez explicitement chaque modification
- Utilisez les outils graphiques de fusion de code

Recommandation 7 – Evitez l'effet tunnel : mettez en place un processus d'intégration et de test en continu

Prenons un exemple classique : en charge d'un nouveau projet, une équipe de développement commence par décomposer l'application cible en différents modules, et à en attribuer un certain nombre à chaque développeur. Elle prévoit d'intégrer, tous les modules ensemble une fois fini, puis de livrer le projet. Mais après quelques semaines de développement, l'équipe réalise que le projet est en train de partir dans le mur. Chaque développeur était occupé à travailler sur sa partie de manière isolée et il n'y avait donc aucune visibilité possible sur l'état du projet dans son ensemble.

Cette situation est facile à expliquer. Quand un développeur considère qu'un module développé par ses soins est prêt, comment peut-il être sûr qu'il l'est vraiment dans le projet global ? Comment peut-il déterminer si son module fonctionnera, une fois confronté aux autres parties du projet ? Comment éviter que l'équipe reste dans le noir jusqu'à la fin du projet, en priant que tout fonctionne comme prévu lors de l'intégration des différentes parties (mais en sachant, par expérience, qu'il y aura sûrement des incompatibilités et des bugs) ?

Pour faire face à ce genre de problèmes, l'équipe décide de faire les choses petit à petit, en prévoyant plusieurs étapes intermédiaires d'intégration pour s'assurer du fonctionnement du système global. Ils réalisent, avec consternation, à quel point intégrer et tester le projet manuellement à différentes étapes du développement leur fait perdre du temps. Les dates de livraison dérapent, le moral tombe, tout comme le projet peu de temps après…

Heureusement, de telles situations peuvent être évitées, et la productivité rétablie, en mettant en place un processus d'intégration et de tests en continu. Une intégration et des tests en continu signifient que la plate-forme de construction est configurée pour automatiquement intégrer, construire, exécuter et tester la dernière version du projet à intervalles réguliers : par exemple une ou deux fois par jour, ou à chaque fois qu'un développeur valide une modification de code.

Les modules sont compilés, et des séries de tests automatisés sont réalisées. À la fin, un rapport, automatiquement généré, montre ce qui fonctionne et ce qui ne fonctionne pas, ainsi que les erreurs rencontrées pendant les phases de construction et de test.

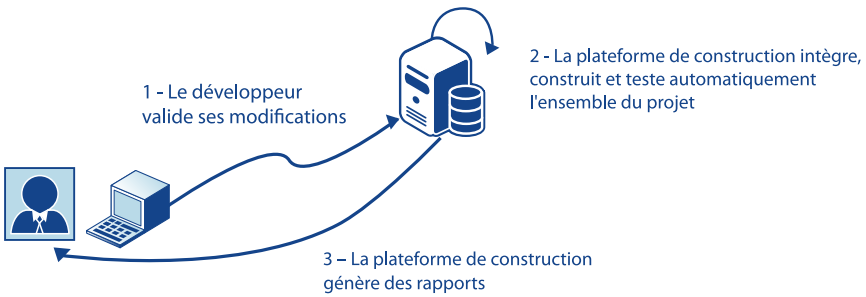


Figure 6 – Intégration et tests en continu

Ces rapports peuvent inclure toutes les mesures que vous souhaiteriez extraire des processus de construction et de tests. Parmi les exemples les plus répandus :

- Nombre de lignes de code par classe
- Erreur de codage potentielle à rechercher
- Niveau de duplication de code
- Respect des directives de codage
- Couverture des tests
- Mesures de performance
- Etc.

Trucs et astuces

Quand on commence par relever des mesures comme les règles de codage, une des erreurs classiques est d'ajouter toutes les règles en même temps. Cela génère généralement des centaines ou des milliers d'erreurs. Ceci étant forcément très décourageant, vous avez l'impression d'être devant une montagne infranchissable. Un autre problème avec cette démarche est qu'elle mélange des règles vraiment importantes avec d'autres qui le sont beaucoup moins.

Nous préférons une méthode plus en douceur. Nous commençons avec une seule règle, la plus importante, et on explique à chacun pourquoi elle est importante. Ensuite, des rapports montrent où la règle n'est pas appliquée pour que tout le monde puisse s'y conformer. Comme il n'y a qu'une seule règle, le nombre d'alertes est limité. Quand il n'y a plus de violations, nous ajoutons une autre règle, et attendons que ces deux règles soient respectées pour ajouter la suivante, et ainsi de suite.

L'intégration continue est un outil très pratique qui permet aux membres de l'équipe de suivre les avancées du projet dans son ensemble, après intégration automatique des différentes parties. Elle vous permet d'identifier au plus tôt les problèmes ou les régressions, et de prévoir les actions suivantes. Elle révèle les problèmes d'interaction entre les sous-parties de votre projet, alors que vous les croyiez indépendantes. Elle permet également de montrer facilement aux commanditaires du projet les parties de vos applications qui fonctionnent. Vous récolterez souvent leur avis et vous pourrez ainsi développer de façon itérative et incrémentale, en collaborant pleinement avec les futurs utilisateurs de votre logiciel.

Trucs et astuces

Vous disposez donc maintenant d'un outil d'intégration continue. C'est bien, mais l'utilisez-vous vraiment ? Les outils ne résolvent pas tous les problèmes. Christian Blavier, architecte chez OCTO Technology, nous aide à répondre à cette question dans notre blog :

http://blog.octo.com/index.php/2008/04/30/111-faites-vous-vraiment-de-l-integration-continue

L'intégration en continu aide en outre à réduire considérablement les risques dans votre projet. Martin Fowler, historiquement un des grands partisans de cette démarche, en donne un aperçu :

> *Le problème avec l'intégration différée est qu'il est très difficile de prévoir combien de temps sera nécessaire pour la réaliser et, encore pire, de savoir où on en est dans le processus. En conséquence, vous vous retrouvez sans la moindre visibilité alors que vous êtes dans une des parties les plus délicates du projet.*
>
> *L'Intégration Continue résout parfaitement ce problème. Il n'y a plus d'intégration longue. Vous éliminez entièrement « l'angle mort ». À tout moment, vous savez où vous en êtes, ce qui fonctionne ou pas, quels sont les bugs non résolus dans votre système.*
>
> *L'Intégration Continue ne fait pas disparaître les bugs, mais elle permet de les repérer et de les éliminer beaucoup plus facilement. À cet égard, cela ressemble à du code qui s'auto-testerait. Si vous introduisez un bug, plus vous le détecterez rapidement, plus il sera facile de s'en débarrasser. Comme vous n'avez changé qu'un petit bout du système, vous n'aurez pas à aller chercher très loin. Et comme ce bout du système est celui sur lequel vous venez juste de travailler, il est encore frais dans votre mémoire, ce qui vous permet de trouver encore plus facilement le bug. Vous pouvez également déboguer à coup de diff, en comparant la version actuelle du système à une précédente qui n'avait pas le bug.*
>
> *Les bugs peuvent s'accumuler. Plus vous avez de bugs, plus il est difficile d'enlever chacun d'eux. C'est en partie parce que vous pouvez avoir des interactions entre les bugs, les défaillances provenant de multiples défauts, ce qui rend chacun d'eux d'autant plus difficiles à trouver. C'est aussi un problème d'ordre psychologique : les gens ont moins d'énergie à trouver les bugs et à s'en débarrasser quand il y en a beaucoup.*
>
> *En conséquence, les projets réalisés avec une Intégration Continue ont tendance à être nettement moins bogués, à la fois en production et dans le processus. Néanmoins, je dois souligner que l'importance de ces avantages est directement liée à la qualité de votre suite de tests. Vous verrez qu'il n'est pas trop difficile de réaliser une suite de tests qui fait la différence. Mais en général, il faut du temps avant que l'équipe n'arrive véritablement au faible niveau de bugs dont elle a le potentiel. Pour y arriver, il faut y travailler et améliorer vos tests en permanence.*
>
> **Martin Fowler, Continuous Integration [11]**

L'intégration et les tests en continu sont la pierre angulaire d'un processus productif de développement de logiciels. Par conséquent, une usine logicielle doit comprendre l'architecture et les outils nécessaires pour mettre en œuvre l'intégration et les tests en continu. Il existe un certain nombre d'outils pour configurer et exécuter automatiquement ce type de processus. Hudson [12] est notre outil préféré, mais certains autres outils, comme Luntbuild [13] ou Bamboo [14], sont également de bonnes alternatives.

Ci-dessous, nous donnons un aperçu détaillé d'un processus classique d'intégration continue :

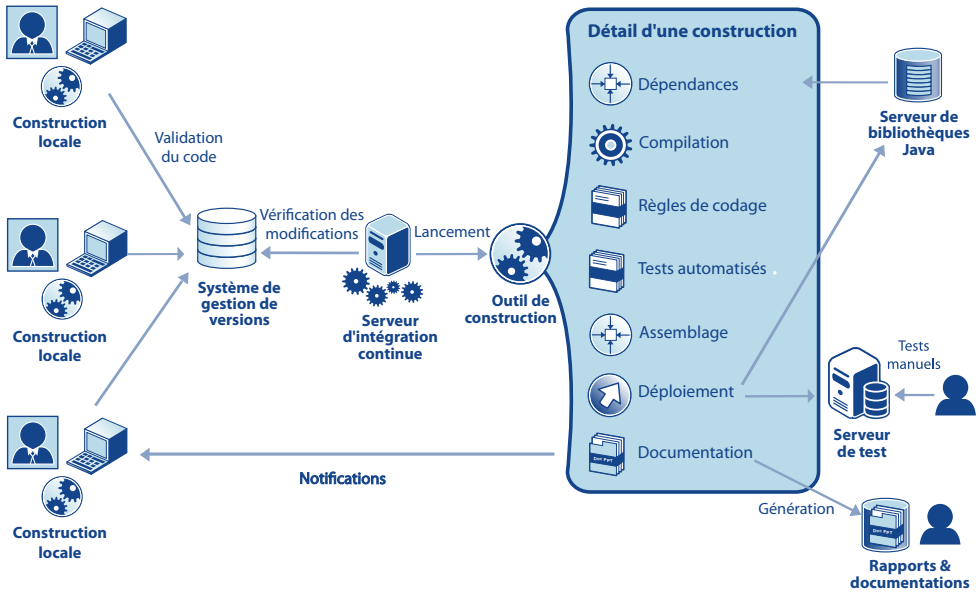


Figure 7 – Exemple de système d'intégration continue

Pourquoi faire les tests d'intégration sur la plate-forme de construction et non sur les postes des développeurs (avant de valider le code) ?

Cela n'a rien d'obligatoire. Evidemment, si vous avez une copie complète du projet sur votre poste, un accès à toutes les ressources nécessaires pour les tests d'intégration (bases de données, fichiers, jetons de sécurité, etc.) et une puissance de traitement suffisante, il est préférable d'effectuer les tests d'intégration sur votre poste. De cette manière, vous ne validerez le code que lorsque les tests sont OK, et vous éviterez ainsi d'affecter les autres développeurs avec du code bogué. En pratique, il est cependant souvent difficile de réaliser les tests d'intégration sur les postes des développeurs et cela à cause des contraintes d'environnement. De plus, même si vous pouvez effectuer ces tests sur les postes des développeurs, ils seront lancés automatiquement sur la plate-forme de construction. Grâce aux rapports de test, vous aurez ainsi une certaine visibilité sur la dernière version validée du projet, qui tient compte de toute fusion produite lors de la validation.

Il existe actuellement un certain nombre d'options disponibles sur le marché, pour créer votre plate-forme. Nous avons testé pas mal de trucs et voici les outils que nous avons trouvé efficaces pour construire une plate-forme de construction pratique et peu couteuse [2] :

Name	Type	Main functions
Subversion	Système de gestion de versions	Stocke le code source, gère les versions et les fusions
Archiva	Serveur de bibliothèques Java	Sauvegarde et gère les bibliothèques Java et les artefacts de construction
Maven 2	Outil de construction	Construit le projet : compile, met en forme, lance les tests, déploie, etc.
Hudson	Serveur d'intégration continue	Orchestre et programme l'exécution des tâches dans l'usine, pour prendre en charge les processus d'intégration continue et de tests automatiques
JIRA	Système de suivi des tâches	Gère les informations liées aux évolutions, corrections, améliorations

[2] Dans un souci de transparence, nous précisons qu'un des auteurs est membre des comités de direction pour les projets Maven et Archiva au sein de l'Apache Software Foundation.

Points clés

- L'intégration et les tests en continu ont lieu pendant le développement et impliquent de compiler régulièrement les composants du projet, de les intégrer, et de tester le système obtenu.
- L'intégration et les tests en continue aident à la gestion du projet : ils améliorent la visibilité sur l'avancement du projet, et aident à trouver les bugs au plus tôt.
- L'intégration et les tests en continu ne sont praticables que s'ils sont automatisés.

Recommandation 8 – Utilisez un système de suivi des tâches

Ce type d'outil aide à coordonner les membres de l'équipe en signalant des problèmes (bugs, demandes de fonctionnalité, suggestions, etc.), en allouant les tâches, en les transférant entre collègues, et en supervisant leur progression. Il deviendra rapidement un outil central pour la collaboration et l'organisation de votre équipe. Une fois votre projet déployé, le système de suivi des tâches peut également être utilisé pour le suivi et la gestion des problèmes signalés par les utilisateurs finaux de vos applications.

Comme nous l'avons vu dans le chapitre précédent, nous avons l'habitude d'utiliser JIRA [15], un système de suivi des tâches puissant et prêt à l'emploi. JIRA est déployé sur la plate-forme de construction, et les membres de l'équipe projet (développeurs, analystes métier, etc.) l'utilisent via une interface Web.

Trucs et astuces

Quand vous créez une nouvelle entrée dans votre outil de suivi des tâches, utilisez un titre qui résume clairement ce dont il s'agit. Supposons par exemple que vous avez trouvé un bug quelque part dans l'application, et que vous voulez en informer les autres membres de l'équipe. Voici divers titres que vous pourriez utiliser pour cette entrée.

`Ma première entrée !`	**Mauvais**
`Bug`	**Encore mauvais**
`Bug dans fenêtre rapport`	**Pas top**
`Bug : Total inexact dans la fenêtre de rapport`	**Bien**

Le dernier titre est clairement plus informatif et permettra à vos collègues de savoir en un clin d'œil quel est le problème. Quand un projet de développement bat son plein, il peut y avoir beaucoup d'entrées créées et traitées quotidiennement. Fournir des titres informatifs aidera votre équipe à utiliser cet outil de façon productive.

Points clés

- Un outil de suivi des tâches peut devenir la pièce centrale d'un processus de collaboration exemplaire pour toute l'équipe. Il peut aussi être utilisé pour collecter les avis des futurs utilisateurs.
- Utilisez des titres explicites.

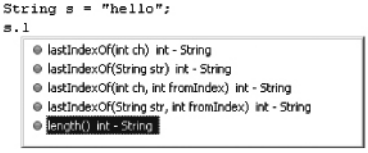
Recommandation 9 – Utilisez un environnement de développement intégré productif

Un IDE (environnement de développement intégré) rend votre travail intuitif et fluide, car il regroupe au mieux la prise en charge de vos diverses activités de programmation dans un seul outil. Notre équipe considère Eclipse [16] comme étant un bon IDE. Il offre un grand nombre de fonctionnalités pour améliorer la productivité, comme par exemple un très bon système de complétion de code, des outils pour naviguer dans le code et les ressources du projet, des fonctionnalités de remaniement de code, et bien plus encore.

À la première utilisation, Eclipse et les autres IDE Java actuels comme NetBeans et IntelliJ IDEA peuvent intimider. Souvent, certains développeurs n'ont pas conscience des excellentes sources de productivité cachées dans ces outils. C'est pourquoi nous vous conseillons vivement de passer un peu de temps à explorer ces fonctionnalités.

Trucs et astuces

Le système de *complétion automatique* est l'une des fonctionnalités de productivité les plus importantes d'Eclipse. Il offre une aide au codage, en proposant des suggestions de complétion de code, ainsi que d'autres gadgets sympathiques comme des conseils sur les types escomptés au moment où l'on saisit les paramètres d'une méthode.



Il peut être configuré pour réagir automatiquement lorsque que vous tapez. Vous pouvez également l'invoquer en tapant **Ctrl + Espace**, puis utiliser les flèches du clavier pour naviguer dans la liste des suggestions. Dans la plupart des cas, il est suffisamment intelligent pour déterminer le type d'objet que vous êtes en train de manipuler dans votre code, et vous propose ainsi des suggestions pertinentes. Par ailleurs, pour vous permettre de coder beaucoup plus vite, il peut aussi être utilisé comme outil d'apprentissage. En effet, il vous permet d'explorer facilement l'API fournie par les objets Java que vous utilisez.

Content Assist peut être configuré finement pour s'adapter au mieux à vos habitudes de codage. Sélectionnez **Windows> Préférences...** dans le menu, puis ouvrez le nœud **Java > Editor > Content Assist** pour modifier la configuration par défaut.

Notez que Content Assist n'est pas uniquement destiné au code Java. Il est également disponible pour divers types de contenus : Javascript, XML, JSP, HTML, etc.

« Vous dites qu'Eclipse est un bon outil, et IntelliJ alors ? Il est aussi bon, et peut-être même meilleur qu'Eclipse. »

Tout à fait ; IntelliJ est un bon outil, de même que NetBeans. En fait, nous voyons régulièrement des guerres d'IDE sur nos listes de diffusion à OCTO. Certains aiment IntelliJ, d'autres adorent Eclipse, et d'autres encore préfèrent TextMate quand ils codent sur leur Mac.

A tout prendre, ce n'est pas vraiment important. Ce qui compte, c'est le code et l'application finale. Vous pouvez donc utiliser un autre IDE. En fait, vous pouvez avoir des développeurs sur les mêmes projets qui utilisent des IDE différents. Assurez-vous seulement que tous les tests réussissent, et que votre logiciel est construit correctement. Les outils dont nous parlons vous aideront dans cette quête.

Points clés

- Eclipse propose un grand nombre d'outils et de raccourcis utiles.
- Ça vaut vraiment la peine de prendre du temps pour les découvrir.

Recommandation 10 – Mettez les tests au centre de vos développements

Une minute ! Les tests servent à garantir la qualité, n'est-ce pas ? Mais cela prend beaucoup de temps de concevoir et mettre en œuvre des suites de tests, même quand le processus effectif de test peut être réalisé automatiquement. Alors, comment cela est-il supposé me faire gagner en productivité ?

C'est vrai qu'on associe généralement les tests à la qualité du logiciel (c'est-à-dire garantir que le logiciel fonctionne correctement). Cependant, dans le présent document, nous l'aborderons principalement sous l'angle de la productivité. Nous espérons qu'après avoir lu ce chapitre, vous partagez notre point de vue : les tests font gagner en productivité.

Mettre les tests au centre du développement signifie développer les composants de votre application mais aussi les tests de ces composants. En fait, vous pouvez dire que sans les tests automatisés associés, les composants ne sont pas complets ni réellement fonctionnels. Prenez le temps d'écrire les tests unitaires.

Vous pouvez même choisir d'écrire vos tests avant de commencer à programmer le code correspondant : c'est ce qu'on appelle la démarche « Tests d'abord ».

Votre usine logicielle doit comporter des frameworks (comme JUnit [17]) qui permettent de développer vos tests en même temps que votre code applicatif. Cette méthodologie de test doit être intégrée tout au long de votre cycle de développement. Vous devez être capable de lancer facilement vos tests sur depuis votre IDE, et de visualiser immédiatement les résultats.

De plus, les tests doivent être effectués automatiquement sur la plate-forme d'intégration par le processus d'intégration et de test en continu, et vous fournir de précieux rapports.

Trucs et astuces

Votre cerveau n'arrête jamais de travailler, et si vous vous laissez distraire de la tâche en cours, il sera toujours difficile de se reconcentrer. Il se peut que votre cerveau se disperse quand la tâche en cours ne requiert pas une grande concentration. Par exemple, après avoir démarré un test unitaire sur un composant que vous venez de modifier, vous devez attendre que le test se termine pour savoir si votre code passe les tests.

Pendant cette période, vous êtes en danger de déconcentration, ensuite il vous faudra encore plus de temps pour vous reconcentrer : « Je testais quoi déjà ? Ha oui… donc le test n'a pas abouti. Mais qu'est-ce que j'ai modifié au fait ?... ».

Pour éviter cela, essayez de limiter le temps de test du code à moins de 3 secondes afin de rester focalisé sur la tâche courante.

Votre suite de tests deviendra l'un des meilleurs atouts de votre projet, parce qu'elle peut être exécutée automatiquement quand vous voulez, vous permettant de savoir si votre code applicatif fonctionne. Cela signifie que vous pouvez coder l'esprit tranquille. Si vous faites quelque chose qui introduit une régression dans votre application, votre suite de tests automatisée vous en avertira immédiatement. Ainsi, vous pouvez modifier votre code rapidement et librement, le remanier, l'améliorer, itérer, en d'autres mots programmer, car vous avez désormais un filet de sécurité. Et plus vous avez de développeurs ou de code complexe, plus cela se vérifie.

Kent Beck, qui a formalisé les principes de l'Extreme Programming (XP), explique brillamment cette approche des tests :

> *Programmer quand vous avez les tests est bien plus ludique que quand vous n'en n'avez pas. Vous codez avec beaucoup plus d'assurance. Jamais ces pensées lancinantes ne vous traversent l'esprit : « Bien, c'est la bonne chose à faire maintenant, mais je me demande ce que j'ai cassé ». Appuyez sur le bouton. Lancez tous les tests. Si le feu est vert, vous êtes prêt pour la suite, avec une confiance en vous renouvelée.*
>
> *Programmer et tester en même temps s'avère plus rapide que de seulement programmer. Je ne m'attendais pas à cela quand j'ai commencé, mais je l'ai évidemment remarqué et j'en ai entendu parler par de nombreuses autres personnes. En ne testant pas, vous gagnerez peut-être en productivité pendant une demi-heure. Pourtant, une fois que vous vous êtes habitué à effectuer des tests, vous remarquerez rapidement la différence en termes de productivité. Le gain de productivité vient de la diminution du temps passer à débugger : vous ne passez plus une heure à chercher un bug, vous le trouvez en quelques minutes. Parfois, il peut arriver que vous n'arriviez pas à faire marcher un test. C'est que vous avez vraisemblablement un problème bien plus important, et il faut revenir en arrière pour être sûr que vos tests sont bons, ou pour voir si la conception a besoin d'être reprise.*
>
> **Kent Beck, *L'Extreme Programming expliqué* [18]**

Cela ne va pas seulement doper votre productivité immédiate, mais également rendre votre logiciel adaptable. En effet, quand une nouvelle fonction doit être ajoutée, ou qu'un changement doit être effectué, vous serez capable de le faire plutôt que de dire « Désolé, ce logiciel est trop fragile. Si on touche quelque chose à un endroit, cela risque de casser ailleurs, et nous n'avons aucun moyen de le savoir ».

> *Les tests allongent la vie du programme (si les tests sont exécutés et maintenus). Quand vous avez les tests, vous pouvez effectuer plus de modifications et pendant plus longtemps que si vous n'en aviez pas. Si vous continuez à écrire les tests, votre confiance dans le système va s'accroître au fil du temps.*
>
> **Kent Beck, *L'Extreme Programming expliqué* [18]**

Points clés

- Tester ne concerne pas seulement la qualité du logiciel. C'est également un des facteurs de productivité les plus efficaces, à la fois sur le court-terme et le long-terme.
- Considérez les tests comme faisant partie intégrante de votre activité quotidienne de développement.
- Mettez systématiquement en œuvre des tests unitaires automatisés pour vos composants logiciels.
- Arrangez-vous pour que les tests unitaires soient très rapides. Individuellement, chacun devrait prendre moins d'une seconde.
- Pensez à automatiser les phases de test de plus haut niveau (par exemple les tests fonctionnels), et intégrez-les au processus de test réalisé par la plate-forme de construction.

Recommandation 11 – Déployez tranquille : externalisez les références vers des ressources externes

Pendant sa vie, votre application va tourner dans plusieurs environnements (par exemple le poste de développement, le serveur d'intégration, l'homologation, divers environnements de déploiement, la production, etc.). Si votre code ou d'autres fichiers au sein de votre package applicatif contient des références vers des ressources différentes selon les environnements, il vous faudra le modifier et reconstruire votre application pour chacun d'eux.

Par exemple, si la modification survient dans votre code source, vous devrez lancer une construction complète pour régénérer un livrable à jour. Si la modification est apportée à un fichier de configuration à l'intérieur de votre livrable, vous serez peut-être capable de le faire sans passer par un cycle de construction complet, mais vous devrez ouvrir votre package et modifier son contenu avec précaution.

En plus d'être néfaste à votre productivité, cela est également contraire à l'objectif même d'avoir des environnements de recette. En effet, pour minimiser les risques de régressions accidentelles, l'application que vous déployez doit être aussi proche que possible de l'application testée et validée. Dans certaines entreprises, cette politique est renforcée en s'assurant que le package applicatif qui obtient le feu vert en recette n'est pas modifié avant d'arriver en production (certification par signature MD5).

De plus, selon les environnements, le besoin de modifier le contenu du package applicatif peut créer des problèmes d'ordre organisationnel.

Alors que la livraison de l'application nécessite des compétences de l'équipe de développement, la connaissance des paramètres de production est restreinte à l'équipe de production (mot de passe critique, habilitations confidentielles, etc.). Par conséquent, les deux équipes devront travailler ensemble pour déployer l'application.

Le codage en dur pose un autre problème : les ressources externes utilisées par votre application ont tendance à évoluer avec le temps. Par exemple, il arrive qu'un mot de passe pour une base de données change, ou qu'un service web migre d'un serveur à un autre, utilisant une autre URL. Encore une fois, si vous devez effectuer ces modifications au sein de votre livrable, voire au fin fond de votre code, vous devrez reconstruire et redéployer toute l'application.

Au contraire, si les références externes de ce type sont définies à l'extérieur de votre package applicatif, vous pourrez les modifier sans le toucher. JNDI, une extension standard de la plate-forme Java, fournit différents mécanismes pour injecter ces paramètres à l'exécution, par exemple ils peuvent être inclus dans la configuration de chaque serveur d'application. Le fragment suivant de code XML d'une configuration Tomcat définit une chaîne de connexion à une base de données :

```
<?xml version='1.0' encoding='utf-8'?>
<Context displayName=»My Web Application» docBase=»/usr/local/myWebApp/
myWebApp-1.0.war» path=»/myWebApp»>
  <ResourceParams name=»jdbc/myDataSource»>
    <parameter>
      <name>url</name>
      <value>jdbc:oracle:thin:@myDBServer:MyTNSNAME</value>
    </parameter>
....
  </ResourceParams>
</Context>
```

Ce fragment de code XML est à l'extérieur du package applicatif (c'est-à-dire en dehors du .war)

Dans l'application, vous pouvez rechercher cette chaîne de connexion en utilisant l'API JNDI, ou bien vous pouvez demander au framework Spring d'injecter automatiquement la chaîne de connexion à l'intérieur de l'un de vos objets, comme le montre l'exemple ci-dessous :

```
<bean id=»DataSource» class=»org.springframework.jndi.JndiObjectFactory-
Bean»>
    <property name=»jndiName» value=»java:/comp/env/jdbc/myDataSource»/>
</bean>
```

Ce fragment de code XML est à l'intérieur du package applicatif (c'est-à-dire dans le .war)

Comme le montre cet exemple, plutôt que de coder en dur la chaine de connexion dans le package, on la référence par un nom logique (java:/comp/env/jdbc/wspds). La valeur effective de la chaîne de connexion à la base de données (jdbc:oracle:thin:@16.40.64.62:1521:SID) est définie à l'extérieur du package applicatif sur serveur d'application. Elle peut être définie par un fichier, comme dans l'exemple, ou bien par d'autres moyens appropriés à votre serveur (console d'administration, etc.).

Trucs et astuces

Cette démarche ne se limite pas aux valeurs qui dépendent de l'environnement ou aux références à des ressources externes. Elle peut être utilisée pour tout autre paramètre technique qui vous paraît susceptible de varier au cours de la vie de l'application. En séparant votre exécutable de ce type de paramètres, vous éviterez souvent des cycles coûteux de développement/validation/déploiement.

Points clés

- Ne définissez pas les valeurs dépendantes de l'environnement à l'intérieur de votre package applicatif.
- Utilisez des noms logiques dont les valeurs effectives sont définies à l'extérieur de votre package applicatif.

Recommandation 12 – Soyez attentif à votre productivité

Une bonne pratique pour améliorer la productivité : traitez-la explicitement. Faites-en un sujet de réflexion et de discussion au sein des équipes projet, plutôt que quelque chose d'implicite qui n'est jamais traité à part entière. Organiser des discussions entre les équipes autour du thème de la productivité sensibilise au sujet et aide à lever et régler les problèmes rapidement. Vous réussirez en outre à créer une intelligence collective, en promouvant la découverte, la discussion et l'échange des pratiques de productivité.

Voici quelques recommandations qui nous semblent pertinentes pour atteindre cet objectif :

- **Accordez du temps**, lors des réunions d'équipes, pour débattre ouvertement de la productivité (problèmes, opportunités, pratiques, etc.).
- **Échangez des astuces de productivité avec vos collègues** : des raccourcis claviers utiles, des pratiques de remaniement de code, les nouvelles fonctionnalités dans les outils de développement, etc. Construire une culture autour de la productivité est enrichissant, aussi bien pour atteindre les objectifs du projet que pour développer de précieuses compétences.
- **N'acceptez pas l'inconfort**. Ne l'acceptez pas, c'est tout. Quand vous sentez que quelque chose dans votre processus freine votre productivité (qu'il s'agisse d'une tâche fastidieuse, répétitive, un retard contrariant ou tout ce qui peut rendre le développement pénible), ne l'acceptez pas. Plutôt que de laisser le malaise s'installer, parlez-en à l'équipe et réglez ensemble le problème pour retrouvez la productivité et le plaisir de travailler… Parce que vous le valez bien !
- **Adaptez les pratiques de développement et les recommandations techniques à la culture et aux compétences de votre équipe.** Si vous possédez un environnement parfait, idéal, dernier cri, mais que votre équipe ne peut pas en tirer avantage ou n'arrive pas à le maîtriser, c'est catastrophique.

  Par exemple, bien que nous recommandions généralement d'utiliser un système de mapping objet-relationnel pour vos interactions à la base de données, c'est une démarche qui requiert un certain niveau d'expérience qui ne correspond pas forcément à la culture et aux compétences de votre équipe. Votre équipe pourrait préférer démarrer avec l'API JDBC puis passer progressivement à Hibernate, ou toute autre technologie alternative.
- **Développez une compréhension profonde de la technologie que vous utilisez et des techniques de programmation associées.** Chaque ligne de code que vous produisez, chaque instruction Java doit être parfaitement claire pour vous, être justifiée, et rester entièrement sous votre contrôle. Inversement, ne reproduisez pas aveuglément du code et des modèles qu'on vous a montrés ou auxquels vous avez accès. Sans une complète compréhension de votre propre production, vous ne serez pas capable de déboguer ou maintenir votre propre code.

Points clés

- Traitez explicitement la question de la productivité
- Créez une culture de la productivité
- N'acceptez pas l'inconfort
- Adaptez les pratiques à vos compétences et à votre culture
- Développez une compréhension et une maîtrise profonde de la technologie

Références

[1] **Java Enterprise Edition v. 5**
http://java.sun.com/javaee/technologies/javaee5.jsp

[2] **The Java Language Specification, Third Edition**
http://java.sun.com/docs/books/jls/third_edition/html/j3TOC.html

[3] **Java 2 Platform, Enterprise Edition v. 1.4**
http://java.sun.com/j2ee/1.4/

[4] Le **Framework Spring**
http://www.springframework.org

[5] **Hibernate – Persistance relationnelle pour Java et . NET**
http://www.hibernate.org

[6] **JPA –API de Persistance Java**
http://java.sun.com/javaee/overview/faq/persistence.jsp

[7] **Eclipse Plugin Central**
http://www.eclipseplugincentral.com

[8] **Maven – Un outil pour la gestion et la compréhension des projets logiciels**
http://maven.apache.org

[9] **Subversion – Un système open source de contrôle de version**
http://subversion.tigris.org

[10.a] **Archiva, gestionnaire de référentiel d'artefact de construction**
http://maven.apache.org/archiva/

[10.b] **Nexus, gestionnaire de référentiel d'artefact de construction**
http://nexus.sonatype.org/

[11] **L'intégration continue**
Martin Fowler, http://martinfowler.com/articles/continuousIntegration.html

[12] **Hudson : un moteur extensible d'intégration continue**
http://hudson.dev.java.net

[13] **LuntBuild – Automatisez et gérez vos builds**
http://luntbuild.javaforge.com

[14] **Bamboo, Serveur d'intégration continue**
http://www.atlassian.com/software/bamboo/

[15] **JIRA: Suivi des bugs, suivi des tâches, et gestion de projets**
http://www.atlassian.com/software/jira/

[16] **Eclipse – Une plate-forme de développement ouverte**
http://www.eclipse.org

[17] **JUnit**
http://www.junit.org

[18] **L'Extreme Programming expliqué : Embrassez le Changement**
Kent Beck et Cynthia Andres, Addison-Wesley Professional.

[19] **NSIS (Nullsoft Scriptable Install System)**
http://nsis.sourceforge.net

[20] **Subclipse**
http://subclipse.tigris.org

[21] **Le livre blanc sur les performances de Java SE 6**
http://java.sun.com/performance/reference/whitepapers/6_performance.html

[22] **JDBC 4.0 Specification**
http://jcp.org/aboutJava/communityprocess/final/jsr221/index.html

[23] **Spring Security (Acegi)**
http://static.springframework.org/spring-security/site/index.html

[24] **Le MVC avec Spring**
http://static.springframework.org/spring/docs/2.0.x/reference/mvc.html

[25] **Grails**
http://grails.codehaus.org

[26] **Jersey**
http://jersey.dev.java.net

[27] **Services Web : Choix Architecturaux**
OCTO Technology. http://www.octo.com/com/download/cahier_servicesweb.pdf

À propos d'OCTO Technology

Fondé en 1998, OCTO Technology est un cabinet français d'Architectes spécialisés dans l'architecture des systèmes d'information et les méthodologies agiles pour le développement de logiciels. OCTO a établi une approche pionnière dans les technologies avancées et le développement de logiciels et, pour chaque projet, fournit des solutions actionnables répondant aux besoins de nos entreprises clientes. La communication, le respect et le partage des connaissances sont les valeurs essentielles de l'entreprise, partagées et transmises par ses cofondateurs, et visibles à travers nos nombreuses publications. OCTO construit continuellement une communauté d'excellence dans le développement de logiciel agile, l'Intégration et la Sécurité en Entreprise, le Test Driven Development, Java, .NET, et l'Open Source. Nos clients bénéficient de l'expérience collective acquise grâce aux divers projets sur lesquels ont travaillé nos architectes.

À propos des auteurs

Les auteurs de ce document sont des consultants OCTO Technology, Philippe Mougin, Guillaume Duquesnay, Arnaud Héritier et Benoît Lafontaine.

Remerciements spéciaux à Olivier Mallassi, Laurent Brisse, Damien Joguet, Nelly Grellier pour leur support tout au long de la rédaction de ce document.

Merci à Roxana Gharagozlou pour la qualité visuelle du produit fini.

Pour tous renseignements complémentaires : www.octo.com ou contactez nous à :

publications@octo.com.

Remerciements

Remerciements spéciaux à la SGCIB et plus particulièrement à M. Tesici et M. Cosenza d'avoir autorisé la publication de ce livre blanc dont le contenu est inspiré d'une mission effectuée à la SGCIB en 2007.

Bibliographie OCTO Technology

Dans la collection : « Une Politique pour le Système d'Information »

Une Politique pour le Système d'Information

Descartes - Wittgenstein - (XML)

Une Politique pour le Système d'Information

Gestion des identités

OCTO Technology publie régulièrement le fruit de ses expériences et de ses recherches sous forme de Livres Blancs :

- Services Web : Choix Architecturaux (2007)
- Architecture Orientée Services (SOA), Une politique de l'interopérabilité (2005)
- Architecture de Systèmes d'Information - Gouvernance de la Donnée (2004)
- Architecture d'Applications - la solution .NET (2003)
- Architectures de Systèmes d'Information (2002)
- Le Livre Blanc de la Sécurité (2001)
- IRM, Gestion de la Relation Client sur internet (2000)
- EAI, Intégration des Applications d'Entreprise (1999)
- Les Serveurs d'Applications (1999)

OCTO Technology est également l'auteur de trois ouvrages parus aux Editions Eyrolles :

« Le projet e-CRM - Relation client et Internet » (2002)

« Les Serveurs d'Applications » (2000)

« Intégration d'Applications - l'EAI au cœur du e-business » (2000)

Paris - Rabat - Lausanne

50, Avenue des Champs-Elysées - 75008 Paris - Tél : [33] 1 58 56 10 00 - Fax : [33] 1 58 56 10 01
59, Av Fal Oueld Oumeir, Agdal, Rabat - Tél : [212] 37 77 88 43 - Fax : [212] 37 77 88 44
Avenue du Théatre, 7, CH-1005 Lausanne - Tél : [41] 21 312 94 15 - Fax: [41] 21 312 94 16
Info@octo.com - www.octo.com - http://blog.octo.com
www.universite-du-si.com