

La gestion de projet informatique trois-tiers : tickets, versions et intégration continue	
Date	20 juin 2011
Auteur	Guillaume ZAVAN
Destinataires	Claire HANEN, Bertrand LE CUN

Sommaire

1. Introduction.....	3
1.1 À propos.....	3
1.2 Problématique.....	3
1.2.1 Définition du sujet.....	3
1.2.2 Conventions.....	4
1.3 Plan.....	4
2. Premier tiers : tickets.....	5
2.1 Principe.....	5
2.2 Définitions.....	5
2.2.1 Forme.....	5
2.2.2 Cycle de vie.....	6
2.2.3 Caractéristiques.....	8
2.3 Effets.....	8
2.4 L'exemple : JIRA.....	11
2.4.1 Introduction.....	11
2.4.2 Tableau de bord.....	11
2.4.3 Ticket.....	12
2.4.4 Autres fonctionnalités.....	14
2.4.5 Retour d'expérience.....	14
2.5 Synthèse.....	15
3. Second tiers : versions.....	16
3.1 Principe.....	16
3.2 Définitions.....	16
3.2.1 Dépôt.....	16
3.2.2 Numéro de version.....	16
3.2.3 Propagation.....	17
3.2.4 Externalisation.....	18
3.2.5 Branches.....	18
3.3 Effets.....	19
3.4 L'exemple : Subversion.....	21

3.4.1 Introduction.....	21
3.4.2 Fenêtre principale.....	22
3.4.3 Historique & Comparaisons.....	24
3.4.4 Visualisation des branches.....	25
3.4.5 Retour d'expérience.....	25
3.5 Synthèse.....	26
4. Troisième tiers : intégration continue.....	27
4.1 Principe.....	27
4.2 Effets.....	27
4.2.1 Pré-requis.....	27
4.2.2 Compléments.....	29
4.3 Effets.....	30
4.4 L'exemple : Hudson.....	31
4.4.1 Introduction.....	31
4.4.2 Tableau de bord.....	32
4.4.3 Gestion d'un dépôt.....	34
4.4.4 Retour d'expérience.....	35
4.5 Synthèse.....	35
5. Usage combiné.....	36
5.1 Théorie.....	36
5.1.1 Récapitulatif.....	36
5.1.2 Cas d'association.....	38
5.1.3 Normalisation.....	39
5.2 Usage pratique.....	40
5.2.1 Tickets.....	40
5.2.2 Versions.....	41
5.2.3 Intégration continue.....	41
5.2.4 Résumé.....	42
5.3 Synthèse.....	42
6. Conclusion.....	43
7. Annexes.....	44
7.1 Abréviations.....	44
7.2 Bibliographie.....	44

1. Introduction

1.1 À propos

Le présent document se veut à la fois un état de l'art et une analyse critique de trois techniques déjà fort répandues dans les entreprises spécialisées en service informatique, mais relativement méconnues des autres secteurs d'activité ainsi que du grand public.

Le choix de ce sujet comme mémoire de fin d'étude s'explique de plusieurs manières.

- Premièrement, la volonté d'aborder une problématique alliant la technique, indissociable du métier d'informaticien, à la pratique, absolument nécessaire à la conduite efficace d'un projet dans le domaine de l'IT.
- Deuxièmement, l'immersion dans le milieu du conseil en informatique, mêlant de manière uniforme plusieurs corps de métier par ailleurs parfaitement disparates : marketing, design, conception, montage, développement ...
- Troisièmement, le souhait d'étudier un sujet au cœur des évolutions du secteur : l'industrialisation et la centralisation des projets informatiques, impliquant un nombre toujours croissant d'intervenants, et donc une progression dans la méthodologie organisationnelle.

Ces différents facteurs, alliés à une pratique intensive des technologies présentées, suffisent à justifier le sujet du présent document.

1.2 Problématique

1.2.1 Définition du sujet

On s'intéresse au sujet suivant.

La gestion de projet trois-tiers : tickets, versions et intégration continue

Avant de se pencher plus amplement sur la méthodologie adoptée par ce mémoire ainsi que sur les règles fixées pour sa rédaction, il convient de définir avec précision les termes ainsi que le sens précis du sujet qu'il aborde.

La « gestion de projet » se définit, de manière relativement intuitive, comme « une démarche visant à organiser de bout en bout le bon déroulement d'un projet » [1]. Il convient de préciser que le présent document se concentre uniquement aux projets menés dans le domaine informatique, et donc dotés de caractéristiques particulières : forte volubilité des ressources, libre granularité des livrables, ou encore large variété d'intervenants [2]. Le terme « trois-tiers » fait évidemment référence aux trois éléments cités dans la suite du sujet, mais pas seulement : il s'applique au terme « gestion de projet » et induit donc que celle-ci puisse être divisible en parts de taille équivalentes ou égales. Il s'agit, on l'aura compris, d'une catégorisation des outils utilisés pour le suivi du projet, et non de la discipline en elle-même. La « gestion de projet trois-tiers » correspond donc à un trinôme de techniques permettant de gérer, sur la durée et sous tous les angles, le déroulement d'un projet informatique.

Le terme de « ticket », ou « étiquette », se définit comme une « marque fixée, fiche placée sur un objet pour en indiquer le contenu, le prix, la destination, etc. » [3] ; le cas échéant, l'objet concerné correspond à une tâche entrant en jeu dans le déroulement du projet : correctif, évolution, planification ou autre. Le mot « version » correspond à « chacun des états d'un texte, d'une œuvre littéraire ou artistique qui subit des modifications » [4] ; au sens informatique du terme, la définition s'étend à l'ensemble de livrables d'un projet : blocs de code, médias, documentation. Enfin, « l'intégration continue » équivaut au « rattachement à une même unité de production, de toutes les opérations qui

conduisent de la matière première (intégration amont) à la réalisation ou même à la diffusion du produit fini (intégration aval) » [5] d'une manière « qui ne présente aucune rupture, aucune discontinuité dans l'espace ; ininterrompu(e) » [6] ; autrement dit, de quelle façon il convient de joindre les livrables du projet sans créer d'interruption dans l'état du produit fini.

Prenant en compte ces différentes définitions, on synthétisera le sujet précédemment annoncé comme une étude de la maîtrise d'œuvre dans le domaine informatique au moyen de trois outils :

- La division et le fichage des éléments intervenant dans la réalisation du produit fini ;
- L'archivage par versions successives de ces mêmes éléments ;
- La mise en valeur sans interruption dans le temps de ces différentes versions.

1.2.2 Conventions

La rédaction du présent mémoire se plie aux conventions suivantes :

- Les abréviations utilisées, exception faites des marques ou objets de copyright, sont obligatoirement indiquées en majuscules et explicitées dans le lexique (partie 7.1), accompagnées si nécessaire de leur traduction en français.
- Les références vers des œuvres externes, qu'elles soient soumises ou non à des droits d'auteur, sont systématiquement indiquées au moyen d'une entrée dans la bibliographie (partie 7.2). Le numéro de l'entrée se trouve alors indiqué entre crochets : [N°].
- Les extraits recopiés de manière littérale depuis une œuvre externe, soumise ou non à des droits d'auteur, sont encadrés par des guillemets et suivi d'une référence vers la bibliographie (voir ci-dessus).
- Les références traduites de l'anglais ou résultat d'interprétations (dans le cas de données autres que littérales : études, statistiques) sont indiquées comme telles dans la bibliographie.
- Les références provenant de contenus à caractère participatif (tels que des wiki) sont dûment datées dans la bibliographie.
- Les captures d'écran illustrant notamment les solutions logicielles présentées peuvent avoir subi des retouches visant à masquer des contenus (fichiers, adresses) à caractère confidentiel.

1.3 Plan

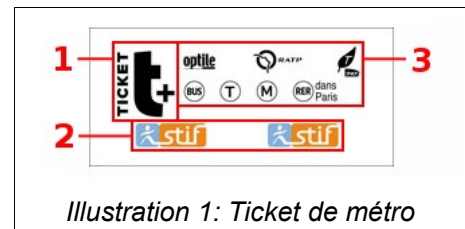
Conformément au sujet présenté précédemment, ce mémoire s'axe autour de quatre parties principales : une pour chaque tiers évoqué dans le sujet plus une abordant les interactions rendues possibles d'un tiers à un autre, mais également vers les différents acteurs du projet. On décomptera en sus l'introduction, la conclusion et les annexes, plus un sommaire.

2. Premier tiers : tickets

2.1 Principe

Appliqué aux technologies de l'information, le principe d'un ticket diffère peu de son sens commun (défini en 1.2.1). Afin de comprendre, dans un premier temps, ses caractéristiques les plus classiques, on se propose d'étudier brièvement un exemple simple et arrivé à maturité : le ticket de métro parisien (Illustration 1).

Premier point étudié, l'objet auquel se réfère le ticket. Dans le cas présent, il s'agit d'un droit contractuel à emprunter les transports en commun ; en temps que tel, il possède un périmètre légal pré-défini et explicite [7] auquel son possesseur doit se plier. De même, chaque ticket s'applique à un référentiel précis dont le détail doit être connu de son (ses) propriétaire(s) et émetteur(s). Second point d'intérêt, la forme prise par le ticket. Il s'agit d'un document écrit, généralement cartonné, doté d'informations textuelles et pictographiques (évoquées ultérieurement) mais également d'une bande magnétique apte à stocker de l'information [8] (heure d'utilisation, péremption ...) : l'état d'un ticket peut donc évoluer selon l'usage qu'on en fait. Troisième et dernier point abordé, les éléments inscrits au dos du ticket. Il s'agit de son type (1), de son émetteur (2) et des zones auxquelles il donne accès (3) ; le cas échéant, ils constituent l'ensemble des indications nécessaires au porteur. On constate que le ticket met en avant les informations les plus utiles de façon à simplifier au maximum son utilisation.



En résumé, un ticket prend la forme d'un document :

- Possédant un environnement de référence connu des individus à qui il sert ;
- En mesure d'évoluer et de stocker une quantité variable d'informations ;
- Mettant en avant un ou plusieurs aspects essentiels sur sa nature et son état.

Dans le cadre d'un projet informatique, ces trois caractéristiques se retrouvent dans le concept de « ticket de support ». On peut le définir comme « un élément contenu dans un système de suivi des problèmes, détenteur d'informations sur des interventions de support, effectuées par les équipes techniques ou tiers, pour le compte de l'utilisateur final ayant reporté un incident lui empêchant de travailler [...] correctement » [9], ou plus largement comme « [...] une correspondance dans laquelle un client notifie une entreprise d'un problème avec une commande ou demande un type précis d'assistance. » [10]. Il s'agit donc d'un ticket, tel que définit précédemment, ne possédant pas d'existence matérielle (autre que les données qu'il utilise), géré au moyen d'un logiciel dédié et destiné à faire circuler des éléments d'information relatifs à une tâche donnée.

2.2 Définitions

2.2.1 Forme

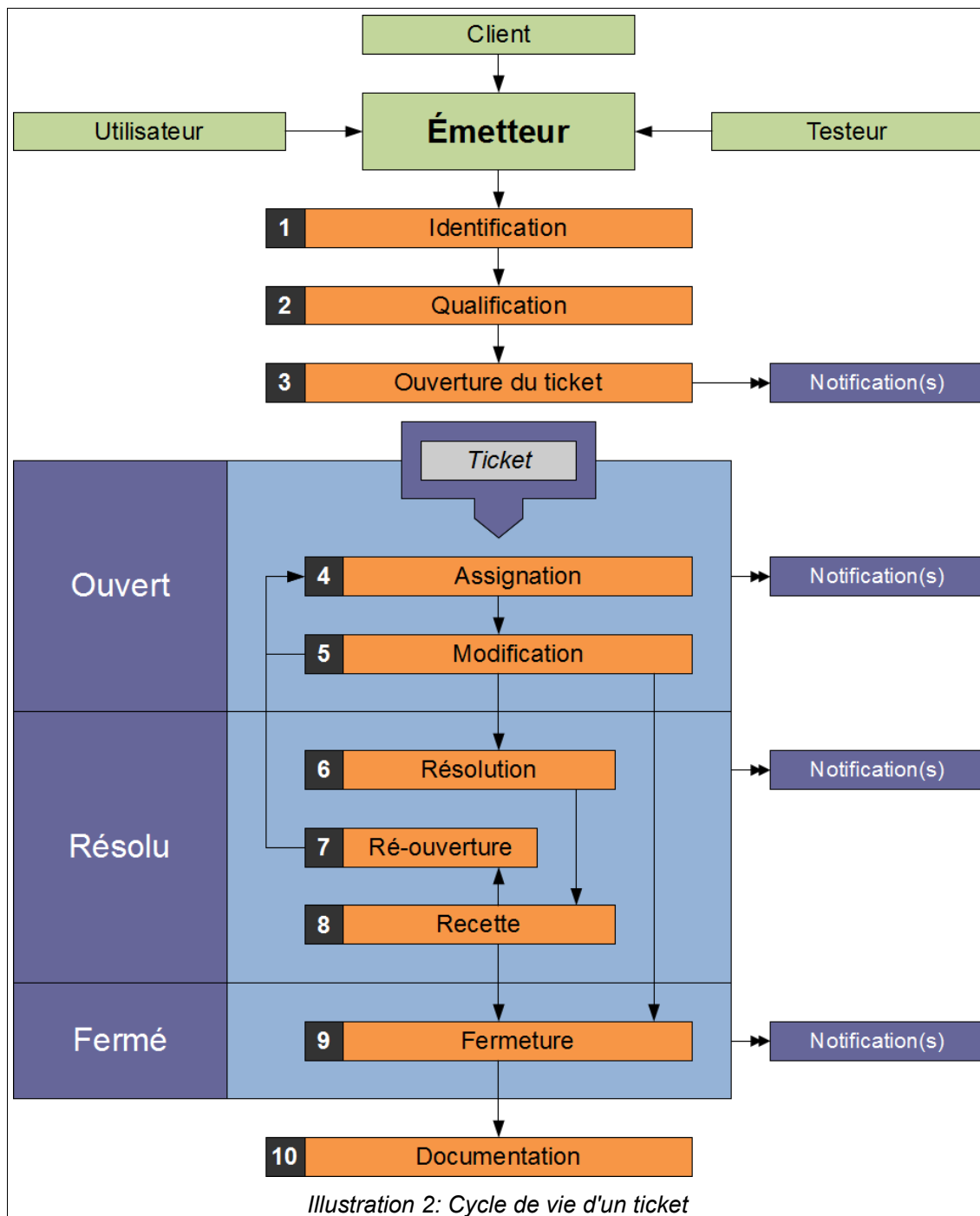
L'intégration d'un logiciel de suivi de problèmes, capable donc de générer et transmettre les tickets de support, requiert un investissement plutôt faible, variant notamment selon le nombre d'utilisateurs et le client choisi (le marché se constituant pour un tiers de solutions libres et pour deux tiers de solutions propriétaires [11]). De manière générale, le gros du budget de déploiement correspond à l'achat ou la location d'un hébergement réseau – intranet ou extranet – compatible avec le logiciel choisi (les langages utilisés variant en fonction de celui-ci).

Une fois déployé, les logiciels de suivi de problèmes sont, dans l'écrasante majorité des cas, accessibles au moyen d'une interface web [12], et intègrent généralement l'envoi de courriels ainsi que d'autres IHM. Les fonctionnalités indirectement liées à la gestion des tickets de support varient

également beaucoup : notifications RSS ou Twitter [13], protocoles d'identification très variés [14] et ainsi de suite.

2.2.2 Cycle de vie

Le schéma ci-dessus propose une représentation du cycle de vie d'un ticket de support [15,16].



L'émetteur du ticket entretient nécessairement un lien, direct ou non, avec le projet : qu'il soit membre de la MOA, MOE, ou simple utilisateur. Selon les entreprises et sa politique en matière de SI, les restrictions d'accès au système pourront être soumises à contraintes ou purement interdites – il s'avère d'ailleurs que l'utilisateur final n'a que rarement la possibilité de créer ses propres entrées.

Les tickets de support peuvent se diviser en deux catégories distinctes :

- Les tickets relatifs à une évolution : incorporation d'un élément inexistant dans le projet ;
- Les tickets relatifs à un correctif : modification ou réparation d'un élément déjà développé dans le cadre du projet.

Les différentes étapes du cycle de vie diffèrent légèrement en fonction de la nature initiale du ticket. On se propose d'étudier plus en détail chacune d'entre elles dans le tableau suivant :

N°	Étape	Évolution	Correctif
1	Identification	Cadrage de l'évolution : environnement, localisation, montage ...	Pointage du problème : URL, capture d'écran, code d'erreur, règle de gestion ...
2	Qualification	Description précise de l'évolution, éventuellement via une spécification	Description précise du problème visant à permettre sa reproduction
3	Ouverture	Enregistrement du ticket	Enregistrement du ticket
4	Assignment	Attribution du ticket à un intervenant capable de cadrer et, au besoin, fragmenter l'évolution	Attribution du ticket à un intervenant capable de le préciser ou de le résoudre
5	Modification	Ajout de précisions relatives à l'évolution ou mise à jour de son état d'avancement	Complétion des informations sur le problème ou inscription des correctifs effectués
6	Résolution	Finalisation de l'état d'avancement et résolution	Résolution du problème
7	Ré-ouverture	Régression dans l'état d'avancement	Non-résolution du problème
8	Recette	Validation des fonctionnalités comprises dans l'évolution	Vérification de la non-reproduction du problème et de l'absence d'effets de bord
9	Fermeture	Archivage du ticket	Archivage du ticket
10	Documentation	Mise à jour des documents existants	Mise à jour des documents existants

Tableau 1: Étapes du cycle de vie

Les suites possibles d'une étape à une autre sont quant à elles décrites dans le tableau suivant :

N°	Étape	Issue(s) possible(s)	Cause
1	Identification	Qualification	Périmètre ou localisation définie
2	Qualification	Ouverture	Description achevée
3	Ouverture	Assignment	Ticket ouvert
4	Assignment	Modification	Mise à disposition pour complétion
5	Modification	Assignment	Redirection du ticket vers un intervenant apte à l'achever
		Résolution	Réalisation du travail attendu
		Fermeture	Invalidation du ticket
6	Résolution	Recette	Mise à disposition pour validation
7	Ré-ouverture	Assignment	Ré-assignment suite à une réouverture
8	Recette	Fermeture	Validation des déploiements effectués
		Ré-ouverture	Invalidation des déploiements effectués
9	Fermeture	Documentation	Archivage du ticket
10	Documentation	-	-

Tableau 2: Suites logiques du cycle de vie

Il existe donc, on le voit, deux situations dans lesquelles une boucle peut se produire au sein du cycle de vie, éventuellement à plusieurs reprises :

- Un intervenant détenteur du ticket se révèle incapable de le résoudre seul : il apporte alors des développements ou informations supplémentaires, et le ré-assigne à un tiers plus apte à clore sujet ;
- Un intervenant en charge de la recette constate que l'évolution se révèle inachevée ou le problème irrésolu ; il ré-ouvre alors le ticket et le redirige vers un tiers apte à prendre en charge les développement restants [17].

2.2.3 Caractéristiques

Un ticket de support porte donc un panel d'informations dont le contenu se précise à chaque étape du cycle de vie. La forte volubilité du support permet le stockage rapide et peu coûteux d'un nombre important de données ; ainsi, « chaque fois qu'un utilisateur du système effectue un changement, le système de suivi des problèmes enregistre l'action et son auteur, de façon à maintenir un historique des actions effectuées » [18]. Les informations et méta-informations enregistrées sont en conséquence nombreuses et de nature hétérogène.

Le tableau suivant présente une liste non-exhaustive des informations communément liées à un ticket de support (les champs dotés d'une étoile sont enregistrés de façon automatique) :

Donnée	Description
Libellé	Titre du ticket
Description	Descriptif exhaustif du problème ou de l'évolution
État	En fonction du processus de résolution : ouvert, ré-ouvert, résolu, fermé, etc.
Rapporteur	Informations sur l'émetteur du ticket
Attribution	Informations sur le détenteur courant du ticket
Catégorie	Type de demande (évolutions et correctifs pouvant également se subdiviser)
Date de création*	Date d'émission du ticket
Date de mise à jour*	Date de la dernière mise à jour du ticket
Date d'échéance	Date limite souhaitée pour la résolution
Date de résolution*	Si résolu, date de la résolution
Date de fermeture*	Si fermé, date de la fermeture
Composant	Élément(s) affecté(s)
Environnement	Environnement d'exécution
Estimation	Durée nécessaire au traitement
Pièce(s) jointe(s)	Fichier(s) lié(s) au ticket
Étiquette(s)	Étiquette(s) ou tag(s) affecté(s) au ticket
Commentaire(s)	Commentaire(s) placé(s) sur le ticket
Historique*	Liste des opérations effectuées ultérieurement

Tableau 3: Caractéristiques d'un ticket

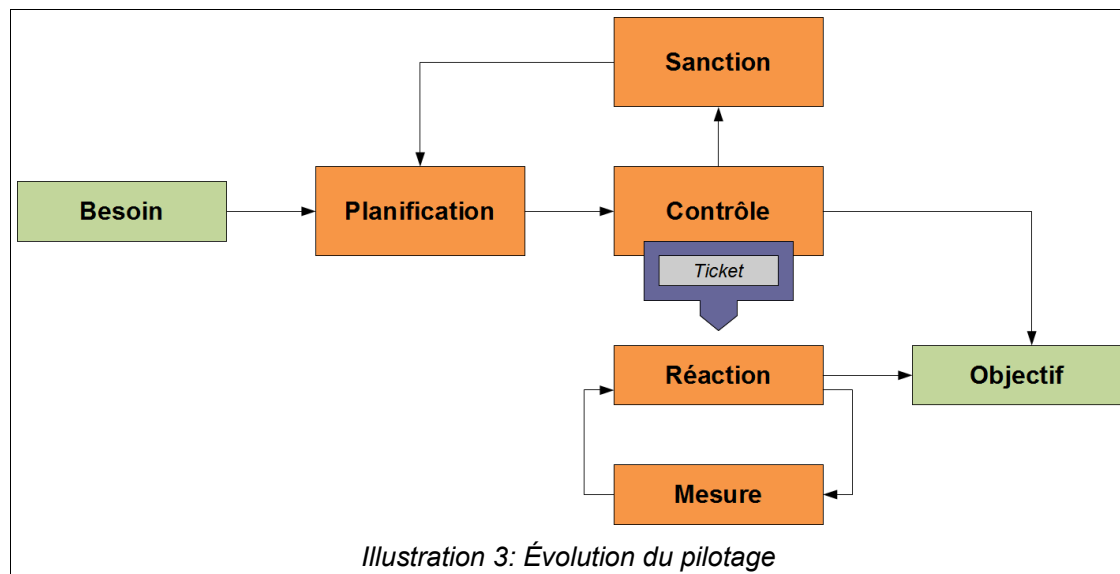
Évidemment, les caractéristiques mentionnée ci-dessus peuvent varier d'un système à un autre. La section 2.4 du présent rapport présente une application, JIRA, dédiée à la gestion de tickets, dont la caractérisation diffère légèrement de la liste présentée.

2.3 Effets

Les prochains paragraphes proposent un récapitulatif des avantages, inconvénients et plus généralement conséquences notoirement engendrée par l'usage de tickets de support dans une entreprise ou un organisme spécialisé dans les technologies de l'information.

Optimisation du suivi

Le principal avantage du ticket de support, mais également sa raison d'être [19], tient évidemment dans l'énorme contribution qu'il apporte au suivi des problèmes au sein de l'entreprise. Premièrement, il apporte une plus-value à chaque étape du processus de production : à bas niveau, il simplifie la conception et le déploiement des correctifs ; à haut niveau, il accélère le travail d'organisation des ressources. Deuxièmement, il permet, dans le cadre d'une politique appropriée, d'offrir une visibilité constante et en temps réel aux clients et tiers externes à l'entreprise sans pour autant leur donner d'emprise sur le processus en lui-même.



Impact sur la logique de pilotage

L'usage de tickets de support engendre un impact direct sur la méthodologie de travail abordée au sein des équipes. Dans un schéma organisationnel classique, le contrôle du travail effectué (sous forme de COPRO, COPIL, ou autre), nécessairement soutenu par des documents écrits, ne peut se faire qu'à intervalles espacés, et souffre donc d'un manque de souplesse se traduisant le plus souvent, en cas de problème, par des sanctions. Via un système de ticket, les points posant problème sont immédiatement documentés, datés, identifiés et peuvent donc être traités au plus vite par les exécutants finaux. Ce modèle ne présente néanmoins pas que des avantages ; il engendre notamment une tension implicite à chaque étape du travail susceptible, dans les cas les plus graves, de nuire aux utilisateurs eux-mêmes.

Usage à des fins de reporting

L'utilisation des tickets de support se révèle en parfaite adéquation vis-à-vis de la culture du *reporting*, basé sur l'acte de « communiquer sur des états d'activité en fonction d'objectifs et de cibles préalablement fixées entre une personne et son supérieur hiérarchique » [21], et s'inscrivant « [...] dans une longue tradition du management par le contrôle » [22]. En effet, l'archivage systématique de l'intégralité des actions effectuées dans le système permet une estimation, sans doute imprécise, mais réelle, du travail effectué par les intervenants utilisant le système. Si les conséquences ne se ressentent guère, voir pas du tout, en haut de la chaîne (hiérarchique et/ou organisationnelle), elles peuvent rapidement engendrer une pression, voir une gêne, parmi les exécutants finaux. Il s'agit d'un état de fait d'autant plus préjudiciable que ces derniers sont au cœur même de l'activité, et donc de la productivité de l'entreprise. Toutefois, une stratégie RH

adaptée, ainsi qu'un usage modéré de l'outil en temps que vecteur de traçabilité permettent d'estomper considérablement ces effets indésirables.

Augmentation des itérations

L'illustration 4 présente deux scénarios basés sur une trame commune : l'envoi de deux rapports relatifs à un même problème se manifestant différemment selon le cas. Dans une structure classique, le problème (1) transite nécessairement par le responsable en charge du projet concerné (2) ; celui-ci en identifie la source et le transmet (3) au développeur le plus approprié pour résolution (4). Lorsqu'un second rapport relatif au même problème (5) lui parvient, il peut le classer immédiatement comme doublon (6, 7.1) ou, au pire, le transmettra au développeur (6, 7.2) qui conclura de la même façon (8). Dans une structure équipée de tickets de support, les deux problèmes (9, 13) risquent d'être acheminés auprès d'intervenants différents sans détection de doublon (10, 11, 14, 15), jusqu'à résolution (12) ou invalidation (16), souvent par le développeur lui-même.. On constate par le biais de ce schéma le caractère particulièrement chronophage que peuvent revêtir l'utilisation de tickets de support, notamment au sein de structures impliquant un large nombre d'intervenants et une organisation interne inadaptée.

Altération des processus

Le principal inconvénient engendré par l'usage des tickets de support reste dans les changements importants qu'il entraîne au sein du processus de production. Ainsi, l'installation d'un système de ce type ne peut se faire sans formation préalable, autant en interne (équipes de production, cadres) qu'en externe (clients, partenaires). L'ampleur des répercussions dépend directement de la complexité du logiciel choisi ; ainsi, plusieurs d'entre eux supportent l'enregistrement par courriel (chaque message envoyé au support ouvre un ticket caractérisé de manière automatique), dont le fonctionnement se révèle totalement transparents aux rapporteurs.

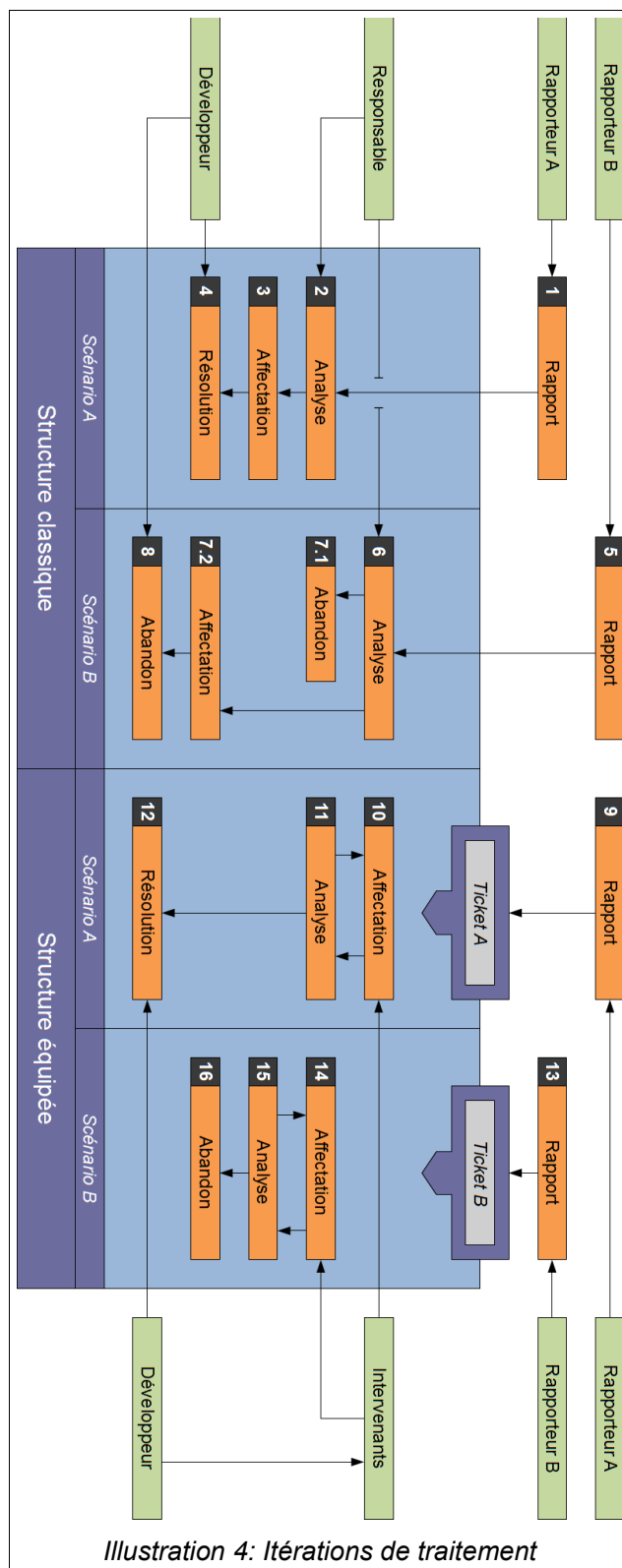


Illustration 4: Itérations de traitement

2.4 L'exemple : JIRA

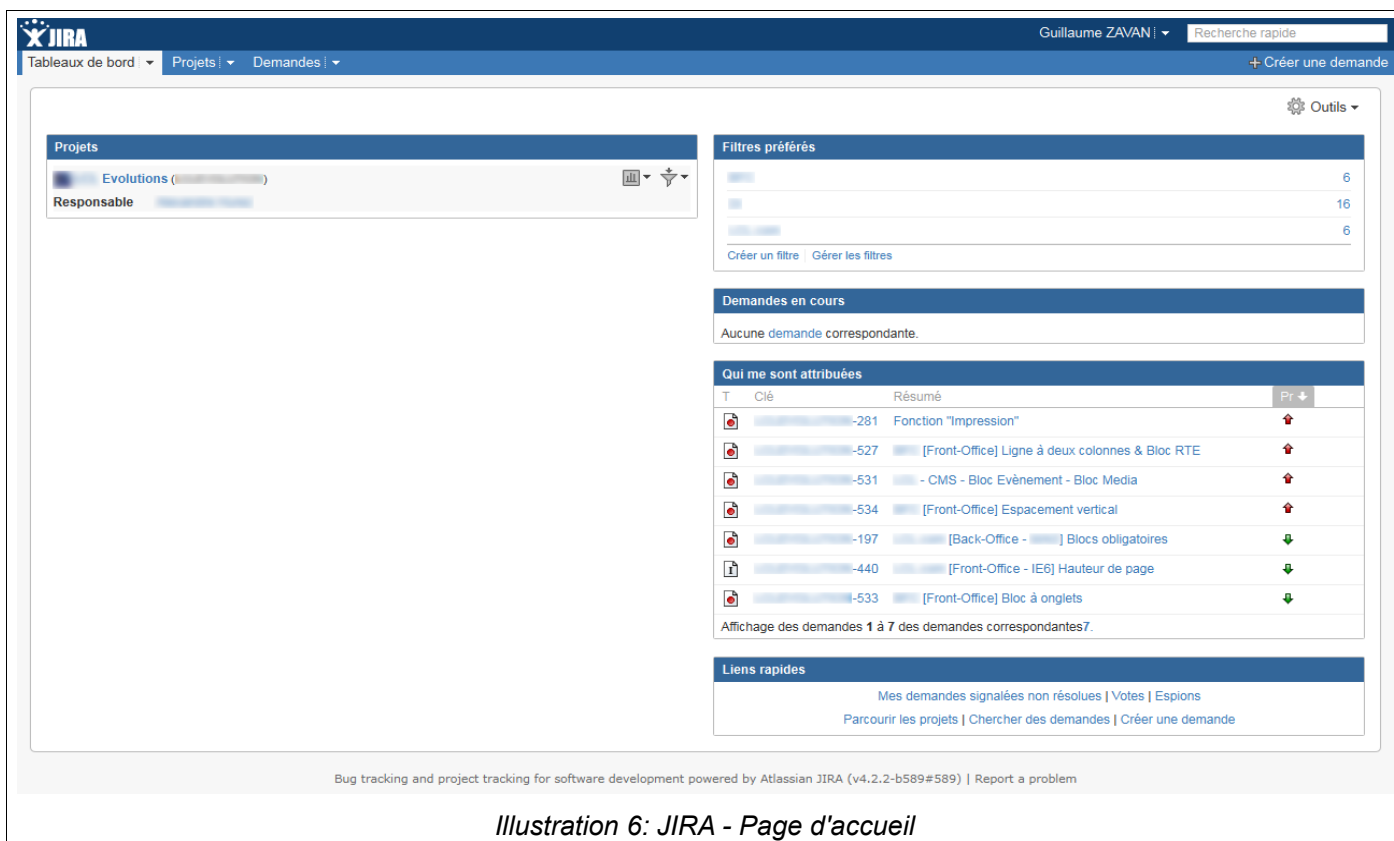
2.4.1 Introduction

Afin de conclure la première partie de ce document dédiée aux tickets de support, il convient de se pencher sur une des manière dont il peut être déployé en entreprise et donc, plus exactement, sur une des solutions logicielles existantes sur le marché : JIRA.



Développé et commercialisé par Atlassian Software Systems [23], ce logiciel propriétaire permet « [...] le suivi de bugs, le suivi de problèmes et la gestion de projet » [24]. Son prix se décline en fonction du type de licence accordé, de gratuit – dans le cas d'une utilisation non-commerciale – à plus de 12.000\$ – pour des projets à but lucratif impliquant plus de 100 intervenants [25]. L'éditeur affiche un portefeuille client impressionnant, gouvernemental (FBI, Parlement Européen, police Allemand ...) et privé (Pixar, EADS, Oracle ...) [26]. Il intègre la majeure partie des fonctionnalités offertes par ses concurrents [27] et en fait donc un exemple idéal de logiciel dédié aux tickets de support.

2.4.2 Tableau de bord

The screenshot shows the JIRA dashboard interface. At the top, there's a navigation bar with the JIRA logo, user name "Guillaume ZAVAN", and a search bar. Below this, there are tabs for "Tableaux de bord", "Projets", and "Demandes". The main content area is divided into several sections: "Projets" on the left, "Filtres préférés" on the right, "Demandes en cours" in the center, and "Qui me sont attribuées" below it. The "Qui me sont attribuées" section contains a table of assigned issues. At the bottom, there's a "Liens rapides" section with links to various JIRA features. The footer of the dashboard mentions "Bug tracking and project tracking for software development powered by Atlassian JIRA (v4.2.2-b589#589) | Report a problem".

T	Cle	Résumé	Pr
	-281	Fonction "Impression"	
	-527	[Front-Office] Ligne à deux colonnes & Bloc RTE	
	-531	- CMS - Bloc Evènement - Bloc Media	
	-534	[Front-Office] Espacement vertical	
	-197	[Back-Office -] Blocs obligatoires	
	-440	[Front-Office - IE6] Hauteur de page	
	-533	[Front-Office] Bloc à onglets	

Illustration 6: JIRA - Page d'accueil

L'illustration 6 présente la page d'accueil du logiciel JIRA interprétée via un navigateur classique. Une fois passée l'indispensable phase d'identification auprès du système (au moyen d'un formulaire classique, à l'aide d'OAuth [28] ou encore de LDAP [29]), l'utilisateur accède à son panneau de contrôle.

On constate en premier lieu un choix de sobriété dans l'interface, de façon à mettre en avant les éléments au cœur du système :

- A gauche, les projets au sein desquels intervient l'utilisateur ;

- A droite, les ticket et outils, accessibles par recherche, filtres pré-définis ou attribution personnelle ;
- Dans l'en-tête, une navigation simple, un outil de recherche rapide et un accès aux paramètres.

Initialement déstabilisante, la navigation se révèle à l'usage très intuitive et facilitatrice, rendant possible le cadrage d'un nombre important d'informations sans complexification inutile. Par ailleurs, les fonctionnalités du logiciel permettent la création d'espaces de travail sur mesure, en fonction des besoins ressentis par l'utilisateur.


2.4.3 Ticket

L'illustration 7 (page suivante) présente l'interface de gestion des tickets. Là aussi, l'accent porte sur la lisibilité et la facilité d'accès aux informations ; on y distingue :


- Un en-tête comprenant le nom du projet ainsi que le titre et le numéro du ticket ;
- Une barre d'outils permettant d'exécuter rapidement les principales commandes relatives au ticket proprement dit ;
- Un panel d'informations, dont :
 - Le type de ticket (bug, évolution, contenu, etc.) ;
 - La priorité affectée au ticket ;
 - La version affectée et, si nécessaire, corrigée ;
 - Le ou les composants affectés ;
 - L'environnement de travail concerné ;
 - L'état de progression du ticket (ouvert, résolu, invalidé, ré-ouvert ...) et son éventuelle fermeture ;
 - L'origine (rapporteur) et l'attribution du ticket ;
 - Les principales dates-clef dans son cycle de vie.
- Un descriptif textuel, doté d'outils de mise en page évolués permettant de clarifier au maximum la demande ;
- Une liste des fichiers joints au ticket ;
- Et enfin, un menu à onglets permettant de naviguer parmi les informations moins cruciales, telles que l'historique, les commentaires, etc.

Les options offertes, autant en possibilités de classification (par projet, composant et version), qu'en termes de mise en page, permettent aux intervenants de donner un maximum de lisibilité aux tickets générés dans le logiciel. JIRA propose d'ailleurs une solution intermédiaire et plutôt efficace au scénario présenté dans l'illustration 4 (section 2.3) : la notion de composant permet l'attribution par défaut des tickets selon une configuration pré-établie ; deux avantages en découlent :


- Pertinence améliorée lors des premières ré-attributions ;
- Possibilité pour un tiers extérieur au projet (client, testeur, etc.) d'adresser avec précision et de façon transparente ses problèmes.

Guillaume ZAVAN | Recherche rapide

Tableaux de bord | Projets | Demandes | Créer une demande


 / -525
[Front-Office] Impression

Modifier | Attribuer | Commentaire | Plus d'actions | Fermer la demande | Rouvrir la demande | Flux de travaux | Affichages




Informations
Type: Bug
Priorité: Important
Affecté la/les version(s): / Nouveaux modules
Composants: 
Étiquettes: Aucune
Environnement: Tous

Personnes
Attribution: Guillaume ZAVAN
Rapporteur: Guillaume ZAVAN
Voter (0) Observer (0)

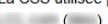
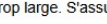
Dates
Echéance: 03/mai/11
Création: 03/mai/11 02:58 PM
Mise à jour: 04/mai/11 12:34 AM
Résolue: 03/mai/11 04:46 PM

Description
Problème
Le rendu des impressions pour  présente quelques défauts.


Sources


Description	URL
Démonstration	http://...
Rendu normal	 .png
Rendu d'impression	 .png
Bannière de remplacement	 .png


Détails
On souhaite corriger trois points de détails sur le rendu d'impression :

- Lien**
Le lien apparaît en double, car il est inscrit sur la bannière et retourné sous forme textuelle. Remplacer le fichier de bannière par celui fourni avec ce JIRA pour résoudre le problème.
- CSS invalide**
La CSS utilisée pour les contenus des pages d'impression n'est visiblement pas la bonne : les contenus apparaissent sous l'aspect . Vérifier et remplacer le(s) chemin(s) appelé(s).
- Largeur de la fenêtre**
La pop-up affichée est trop large. S'assurer qu'elle prenne la taille de  px en largeur.

Pièces jointes



32 kB 03/mai/11 02:58 PM



638 kB 03/mai/11 02:58 PM





132 kB 03/mai/11 02:58 PM

Activité

Toutes | Commentaires | Journal de travail | Historique | Activité | Subversion Commits | Git Commits

 a ajouté un commentaire - 03/mai/11 04:17 PM - Limité à Developers
Corrigé : <http://...>

 Guillaume ZAVAN a ajouté un commentaire - 03/mai/11 04:46 PM
Corrigé.

 Hudson Application on hudson a ajouté un commentaire - 04/mai/11 12:34 AM
Integrated in  #597
 -525

Commentaire

Bug tracking and project tracking for software development powered by Atlassian JIRA (v4.2.2-b589#589) | Report a problem

Illustration 7: JIRA - Ticket

2.4.4 Autres fonctionnalités

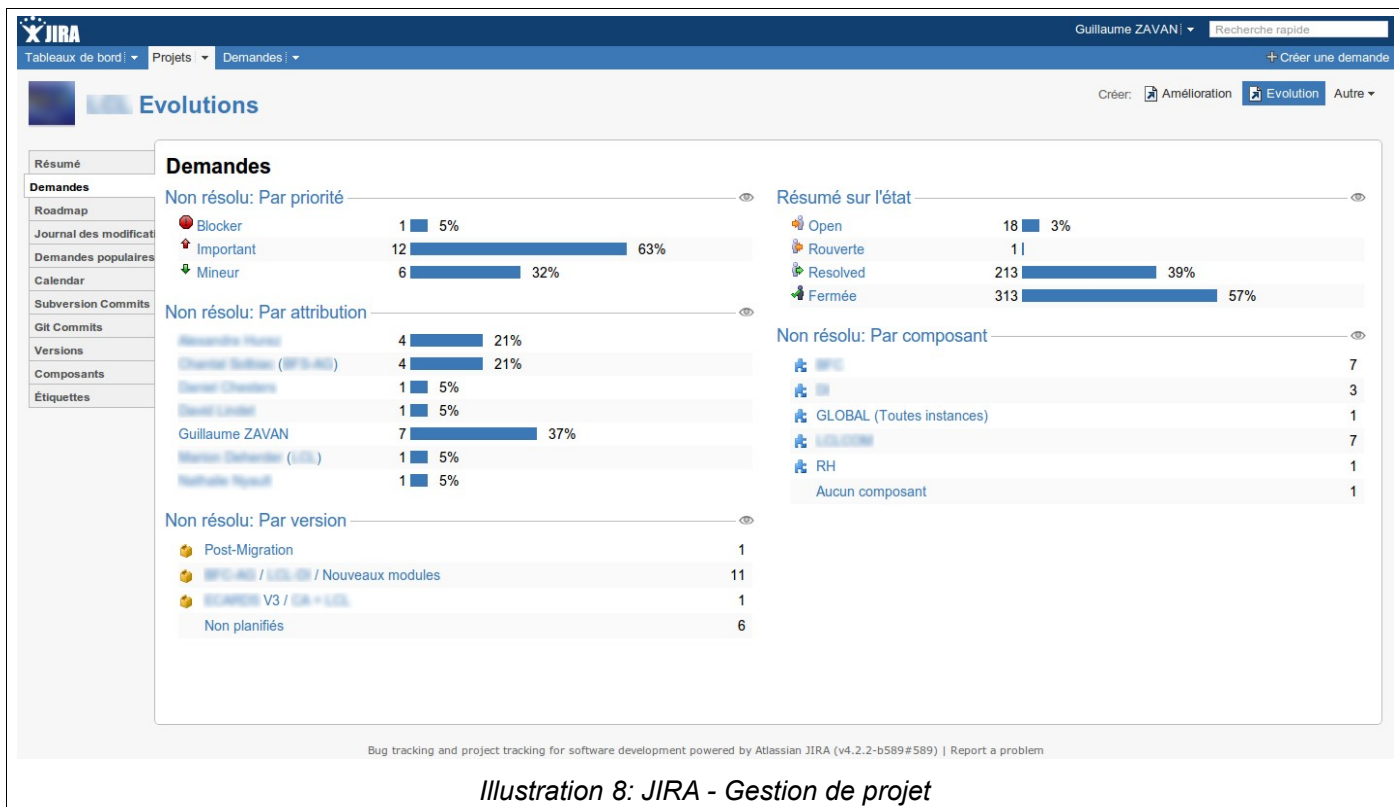


Illustration 8: JIRA - Gestion de projet

JIRA propose, outre les éléments détaillés précédemment, divers outils destinés au cadrage et à l'organisation des projets dont, en vrac :

- Un utilitaire statistique poussé (illustration 8, ci-dessus), permettant d'évaluer rapidement l'état d'avancement d'un projet donné ;
- Un moteur de recherche développé basé sur son propre langage de requêtes (proche de la syntaxe SQL) ;
- Des modules d'export vers de nombreux formats (XML, RSS, Office, etc.) pouvant servir au développement de plugins ;
- Des outils d'organisation générale : calendriers, flots de travail, étiquetage ;

2.4.5 Retour d'expérience

Alors, que peut-on retenir de l'outil JIRA en dehors des avantages et des inconvénients communs aux tickets de supports, présentés précédemment dans ce mémoire ?

Au banc des points forts, la simplicité d'utilisation, paradoxalement alliée à une très grande richesse sur le plan fonctionnel, font de JIRA un excellent choix pour le déploiement de tickets au sein d'un projet dès dix intervenants. Une fois passée la phase d'initiation, rapide mais obligatoire, le logiciel se révèle à la fois réactif et précis ; sa mise à disposition auprès d'intervenants externes se fait d'ailleurs sans problème aucun, sous réserve de mettre à disposition une aide contextuelle, déjà fournie, et des tutoriels d'utilisation.

Seule ombre au tableau, les notifications envoyées par e-mail. Commodes qu'en de rares situations, elles s'avèrent trop fréquentes pour être lisibles. Dans le cadre du suivi de projet, on finit par ne plus porter attention qu'aux messages de création ou de fermeture, les autres finissant immédiatement à la corbeille.

2.5 Synthèse

En résumé, les tickets de supports constituent pour l'entreprise une arme à double tranchant dont l'effet dépend directement des précautions prises durant leur déploiement.

Utilisés intelligemment, ils prennent une place essentielle dans les processus organisationnels de l'entreprise, et ce à tous les niveaux de la chaîne de production. Ils améliorent l'autonomie des intervenants, offrent une meilleure répartition de la charge de travail, et abaissent considérablement les obstacles au suivi des ressources. Enfin, ils remplissent parfaitement leur tâche première : documenter, de façon continue et en prenant en compte une multitude de profils techniques, les évolutions à réaliser dans le cadre du projet.

Toutefois, les pièges présentés par leur utilisation sont nombreux. Déployés trop rapidement, sans formation préliminaire des intervenants, ils risquent – au mieux – de ne pas trouver leur place dans l'entreprise et – au pire – d'alourdir considérablement le suivi des problèmes. Utilisés abusivement, ils peuvent prendre le pas sur les rapports humains, par ailleurs indispensable à la tenue du projet ; phénomène d'autant plus courant au sein de grandes structures. Mal configurés, ils engendrent des itérations inutiles dans le traitement des tâches, l'apparition de doublons, et la multiplication des retours indésirables.

En conséquence, il ne suffit pas de prétendre à l'utilisation d'un système de ce type pour en tirer de pleins bénéfices. La préparation des équipes, internes ou externes, le développement de pratiques adaptés, et le choix d'un logiciel correspondant au besoin sont autant de facteurs indispensables à l'exploitation efficace des tickets de support.

3. Second tiers : versions

3.1 Principe

En informatique, le terme « version » possède une signification proche de la définition proposée en introduction de ce document : ainsi, on lui donne couramment le sens de « chacun des états d'un logiciel en cours de développement [...] » [30]. Dans le cas d'un logiciel d'ors et déjà distribué au grand public, les versions se basent généralement sur un code simple : chiffres (1.0, 1.1, etc.), date (8.04), année (95), et ainsi de suite [31].

On peut dès lors poser légitimement la question de la granularité à donner à ces versions. En effet, un logiciel, fût-il simple, naît d'une mécanique complexe, certes invisible à l'utilisateur (exception faite des options de paramétrage qu'il offre), mais bien réelle pour les individus intervenant dans sa réalisation : qu'il s'agisse de composants algorithmiques (code), multimédias (images, sons, vidéos), sécuritaires (certificats) ou encore littéraires (contenus, configuration). Sachant qu'une seule modification dans un de ces éléments engendre nécessairement une évolution dans l'état du produit fini, doit-on considérer qu'il s'agit d'une version à part entière ?

La réponse dépend avant tout de l'utilisation faite de cette information.

- Pour l'utilisateur final, non, car seules les versions publiques (livraisons) comptent. À priori stabilisées et vérifiées avant distribution, elles sont de nature moins nombreuses que les états réels du logiciel durant son développement ;
- Pour le développeur, oui, car le repérage et l'identification des comportements défectueux ou inachevés nécessite autant de précision que possible. Il s'agit avant tout de pouvoir analyser à quel moment quels fichiers subissent des modifications.

Le périmètre de ce mémoire couvre essentiellement le second cas ; néanmoins, nous n'éluderons le premier pour autant, car il compte tout autant dans le processus de production et le travail de développement.

3.2 Définitions

3.2.1 Dépôt

Afin de permettre et surtout d'automatiser le suivi des versions d'un logiciel, il convient en premier lieu de définir un espace apte à accueillir les composants entrant en jeu dans son fonctionnement : le dépôt (ou *repository*). Il s'agit – dans le cas qui nous intéresse – d'un « emplacement à partir duquel des bases de données spécifiques, fichiers, ou documents sont obtenus pour réinstallation ou distribution sur un réseau » [32].

Concrètement, le dépôt de version se caractérise le plus souvent de la façon suivante :

- Il fonctionne au moyen d'un serveur distant, accessible par réseau local ou par internet, soumis à des restrictions d'accès précises (en lecture et en écriture) ;
- Il se base sur un protocole dédié d'accès aux données privilégiant généralement la fiabilité au détriment du fonctionnel ;
- Il ne peut être utilisé qu'au moyen d'un client adapté.

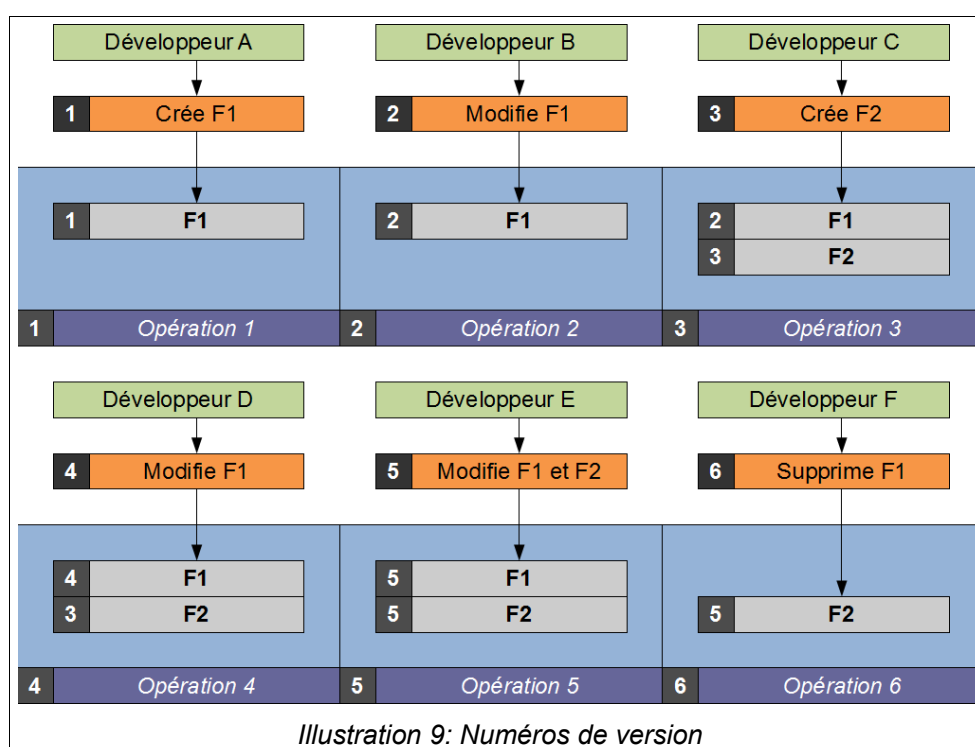
Néanmoins, ces différents aspects peuvent varier selon le logiciel et la configuration utilisés [33].

3.2.2 Numéro de version

Une fois un dépôt choisi et configuré selon les contraintes du projet, il convient d'aborder la notion, évoquée précédemment, des numéros de version.

Dans la majeure partie des protocoles dédiés à la gestion de versions, ils prennent la forme d'un entier, encodé ou non [34]. Ils s'appliquent, d'une part, à chaque fichier présent dans le dépôt et, d'autre part, au projet lui-même. Leur attribution, présentée dans l'illustration 9, répond le plus souvent aux règles suivantes :

- Tout changement dans l'état du dépôt (ajout, mise à jour, suppression) incrémente son numéro de version de 1 : il s'agit dès lors du chiffre de référence pour les opérations à l'origine de ce changement ;
- Les fichiers affectés par la mise à jour se voient attribués le numéro de référence tout juste attribué au dépôt ; dans le cas d'une suppression, le fichier disparaît purement du dépôt.



Ce mécanisme, associé à un maintien automatique de méta-informations sur chaque opération enregistrée par le dépôt (ou propagation, voir 3.2.3), telles que la date, l'utilisateur, les commentaires éventuels, permet de conserver une image précise des évolutions successives du projet.

3.2.3 Propagation

La notion de propagation (approximativement traduite de l'anglais *commit*) se situe aux cœur des problématiques engendrées par la question des versions et – on le démontrera dans le dernier chapitre de ce document – joue un rôle fondamental dans la gestion de projet à trois tiers.

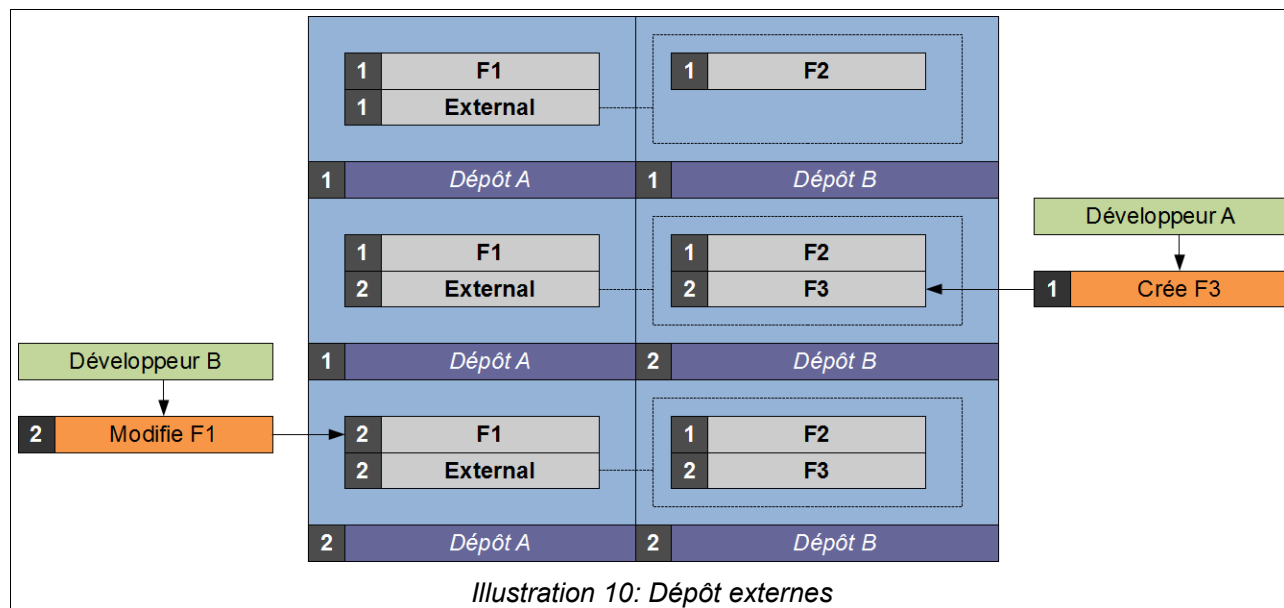
Identique, dans son sens premier, au terme utilisé en bases de données, il consiste en une synchronisation entre les sources locales possédées par l'utilisateur et le dépôt (sur l'illustration 9, il s'agit des étapes 1 à 6 effectués par les développeurs A à F). Il permet de diviser les évolutions des différents composants en groupes de périmètres variables (une propagation peut se traduire autant par une simple modification que par de multiples ajouts, mises à jour et suppressions simultanées) organisés et suivis dans le temps.

La grande majorité des protocoles dédiés à la gestion de version incluent un fonctionnalité d'archivage automatique, fonctionnant par duplication des versions les plus anciennes, ou par mémorisation des changements [35]. Extrêmement facilitatrice pour la recherches d'effets de bords ou

la sélection de sources fiables pour duplication (voir 3.2.5, plus bas), ce système démultiplie les ressources nécessaires à l'hébergement du dépôt ; ainsi, les fichiers les plus lourds (maquettes, vidéos, etc.) sont souvent conservés à un emplacement externe, pour éviter l'archivage de données inutilement encombrantes.

3.2.4 Externalisation

Une partie des protocoles dédiés à la gestion de versions permet l'imbrication de dépôts par externalisation [36]. Il s'agit peu ou prou de la création, au sein d'un dépôt donné, d'un lien symbolique vers un emplacement situé sur un autre dépôt.



L'illustration 10 présente un exemple d'externalisation : un dépôt (A) inclut dans ses sources un second dépôt (B). On constate que la modification du dépôt externe n'entraîne pas de conséquence sur la version du conteneur ; l'intégration se fait dès lors de façon totalement transparente. Le principal intérêt présenté par cette technique apparaît dans les projets impliquant de nombreux intervenants : il devient alors possible de procéder à une division des dépôt par compétence ou par composant, et de protéger plus efficacement les sources contre des manipulations ou des effets de bords indésirables.

3.2.5 Branches

La notion de branche constitue une réponse pratique aux problématiques liées à la nécessité de fournir, à intervalle régulier, des livrables à destination des utilisateurs finaux. Elle se définit comme « [...] la duplication d'un objet soumis à un contrôle de version [...] de façon à permettre des modifications parallèles le long de chaque branche » [37] ; l'objet mentionné étant, le cas échéant, tout ou partie du dépôt. Une branche consiste donc en un pseudo-dépôt, indépendant du dépôt d'origine, et pouvant donc suivre ses propres développements depuis un point donné.

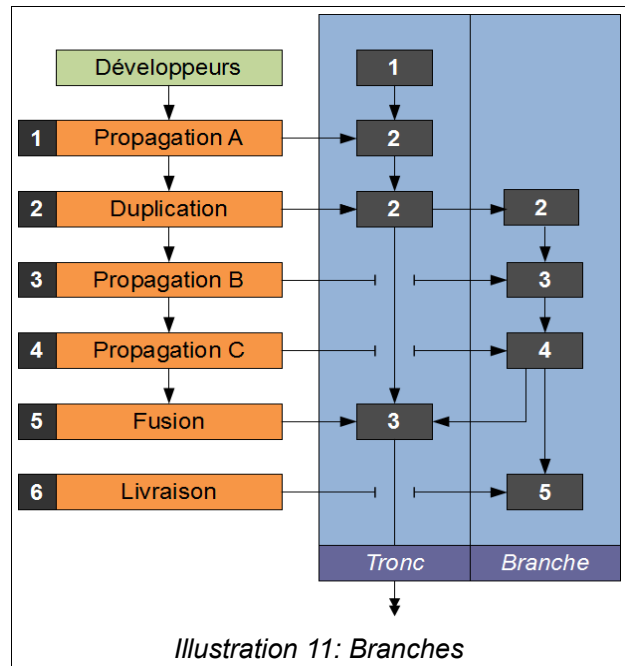
Il convient dès lors de distinguer trois notions fondamentales dans ce mécanisme :

- Le tronc (ou *trunk*), qui désigne le dépôt initialement utilisé dans le projet, et sensé représenter les dernières évolutions du fonctionnelles et techniques du logiciel développé ;
- Les branches (homonyme à l'anglais, *branches*), représentatives d'autant de schémas d'évolution divergeant depuis une version choisie du dépôt d'origine ;

- Les emprunts (arbitrairement traduit de l'anglais *tag*), images fixées du dépôt originel ou d'une de ses branches à instant T, généralement utilisées pour désigner un ensemble suffisamment abouti et stable pour être livré.

L'illustration 11 présente un cas classique d'utilisation des branches. À partir d'un dépôt classique, un développeur effectue une duplication (2) engendrant l'apparition d'une branche à part. Il peut dès lors travailler dessus sans modifier outre mesure l'état du tronc commun. À terme, il peut envisager d'en constituer une empreinte (6) et, éventuellement, d'intégrer tout ou partie des changements dans le tronc commun (5).

Il paraît important de distinguer la duplication par branche d'une copie pure et simple (également autorisée par plusieurs systèmes de suivi de versions sous le nom de *fork* [38]). En effet, étant donné qu'à chaque propagation, le dépôt archive l'intégralité des modifications effectuées, la création d'une branche n'exige pas de copie physique des fichiers : les fichiers non-altérés sont, le cas échéant, appelés directement depuis les données mémorisés sur le tronc commun. Il paraît dès lors possible de coder au moyen d'un nombre important de branches parallèles.



Des limites existent néanmoins à l'utilisation de ce système. En premier lieu, le risque qu'une branche subisse trop de modifications pour pouvoir être intégrée au tronc commun pèse considérablement dans la balance, tout particulièrement sur les projets dont les composants présentent une forte dépendance ; il peut dès lors arriver que l'opération de fusion nécessite plus de travail que la reprise à zéro des évolutions réalisées sur la branche. En second lieu, la nécessité d'informer et de suivre les équipes de production pour éviter toute confusion entre les différentes constitue une étape indispensable lors de l'ajout ou la suppression de branches. Il convient donc dans les deux cas d'assurer un contrôle, minimal mais régulier, de l'utilisation faite de ce mécanisme afin de pallier à tout effet de bord indésirable qui pourrait en découler.

3.3 Effets

Si l'utilisation d'un système dédié à la gestion de versions entraîne nécessairement diverses contraintes matérielles et procéduraires, on ne peut nier l'apport considérable qu'il apporte dans la gestion d'un projet.

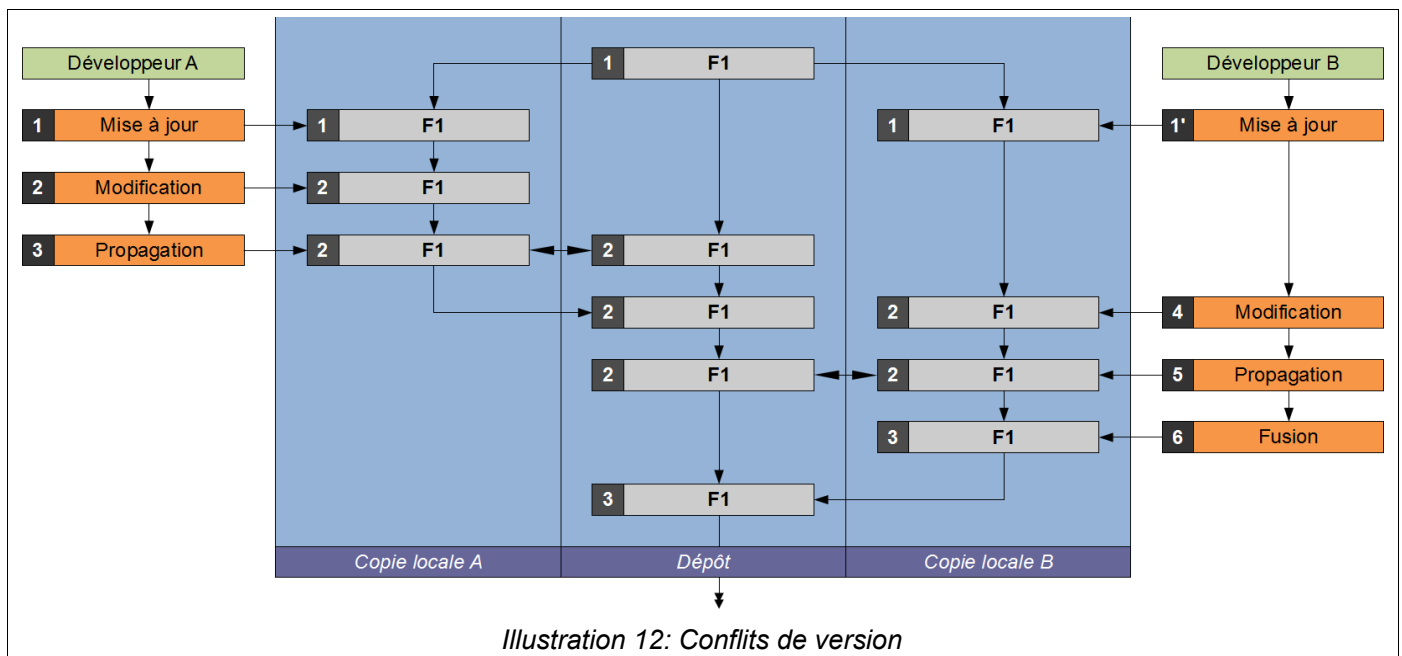
Tout d'abord, il s'agit d'un vecteur privilégié pour la transmission de l'information : un intervenant peut dès lors, par le biais des archives réalisées de manière automatique par le dépôt, comprendre une partie, si ce n'est la totalité du contexte ayant amené à chaque modification des sources. Alliées à une connaissance plus généraliste du projet, ces informations facilitent la compréhension du code et accélèrent les investigations sur les dysfonctionnements les plus anciens, ignorés ou transparents au moment de leur apparition. Dans l'autre sens, il rend également possible la restauration des fichiers ayant développé un comportement anormal suite à une modification.

Deuxièmement, ce système simplifie et sécurise les tentatives de développement parallèles, expérimentales ou dont l'issue laisse craindre la génération d'effets de bord. Au moyen des branches, la création de versions alternatives pour le projet en cours, totalement isolées du tronc commun, revêt une simplicité déconcertante ; l'utilisation de répertoires externes, non-dupliqués lors de la création de la branche, autorisant la syndication d'un ou plusieurs composants du programme.

Troisièmement, la répartition des dépôts au moyen de l'externalisation permet de concrétiser la division des composants en fonction des intervenants appelés à travailler dessus, et selon une classification laissée à l'appréciation des responsables du projet :

- Par niveau de sécurité, les dépôts les plus sensibles étant soumis à des contraintes plus strictes en lecture et en écriture ;
- Par fonctionnalité, chaque dépôt correspondant à un lot d'outils intégré à un ensemble plus vaste ;
- Par qualification technique, de façon à classer les intervenants et leur autorisations selon leur corps de métier (ingénieurs, graphistes, administrateurs, éditorialistes, etc.) ;
- Par environnement, dans le cas d'applications découlant sur plusieurs usages indépendants ;
- Et ainsi de suite.

En contrepartie, on ne peut passer sous silence deux principaux points d'ombre pouvant devenir rapidement un handicap lors d'un usage en production.



Le premier, présenté dans l'illustration 12, concerne les conflits pouvant se produire entre deux versions simultanées d'un même fichier, propagées par deux utilisateurs différents. Sur le schéma, les développeurs A et B possèdent tous deux une copie locale d'un même fichier F1 ; A effectue une modification (2) qu'il propage sur le serveur (3). Dans le même temps, B effectue une autre modification (4) et tente de la propager (5) ; le serveur détecte alors un conflit (la version du fichier sur le dépôt différent de la version localement modifiée) et exige une étape de fusion (6) débouchant sur une troisième version, théoriquement intermédiaire, du fichier. On imagine facilement les complications pouvant se produire quand :

- Les données concernées sont de type binaires et non-supportées par les outils de fusion classiques ;
- Le fichier, par sa taille ou son rôle, se soumet à de nombreuses modifications simultanées par des utilisateurs différents.

Le second point pouvant rapidement poser problème concerne la question du matériel nécessaire à héberger le dépôt. En effet, si l'on considère le cas d'un fichier binaire de grande taille, on

constate qu'une copie doit être réalisée à chaque modification du fichier, de façon à permettre l'archivage de ses différentes versions ; problème d'autant plus épineux si celui-ci subit des modifications régulières (dans le cas, par exemple, d'une maquette ou d'éléments multimédias).

3.4 L'exemple : Subversion

3.4.1 Introduction



Afin d'illustrer le fonctionnements d'un système dédié à la gestion de version, il convient de se pencher sur un protocole mature, compatible avec les fonctionnalités évoquées précédemment, et jouissant d'une large popularité en entreprise : le choix le plus judicieux paraît sans aucun doute être Subversion (SVN). Créé en 2000 par CollabNet [39], intégré en 2009 au projet Apache et distribué depuis sous licence éponyme [40], ce protocole jouit encore aujourd'hui d'une large popularité, malgré son abandon progressif pour d'autres outils permettant notamment les propagations en mode non-connecté [41].

Concernant le client utilisé pour accéder au répertoire, on se réfèrera à un logiciel relativement méconnu mais très performant, SmartSVN, commercialisé par Syntevo et doté d'une interface graphique très développée. Il en existe néanmoins de nombreux autres ; par ailleurs, on notera que le protocole SVN fonctionne également en mode console et via un navigateur classique (en lecture uniquement).

Enfin, on étudiera le dépôt (public) du logiciel Notepad++ [42].

3.4.2 Fenêtre principale

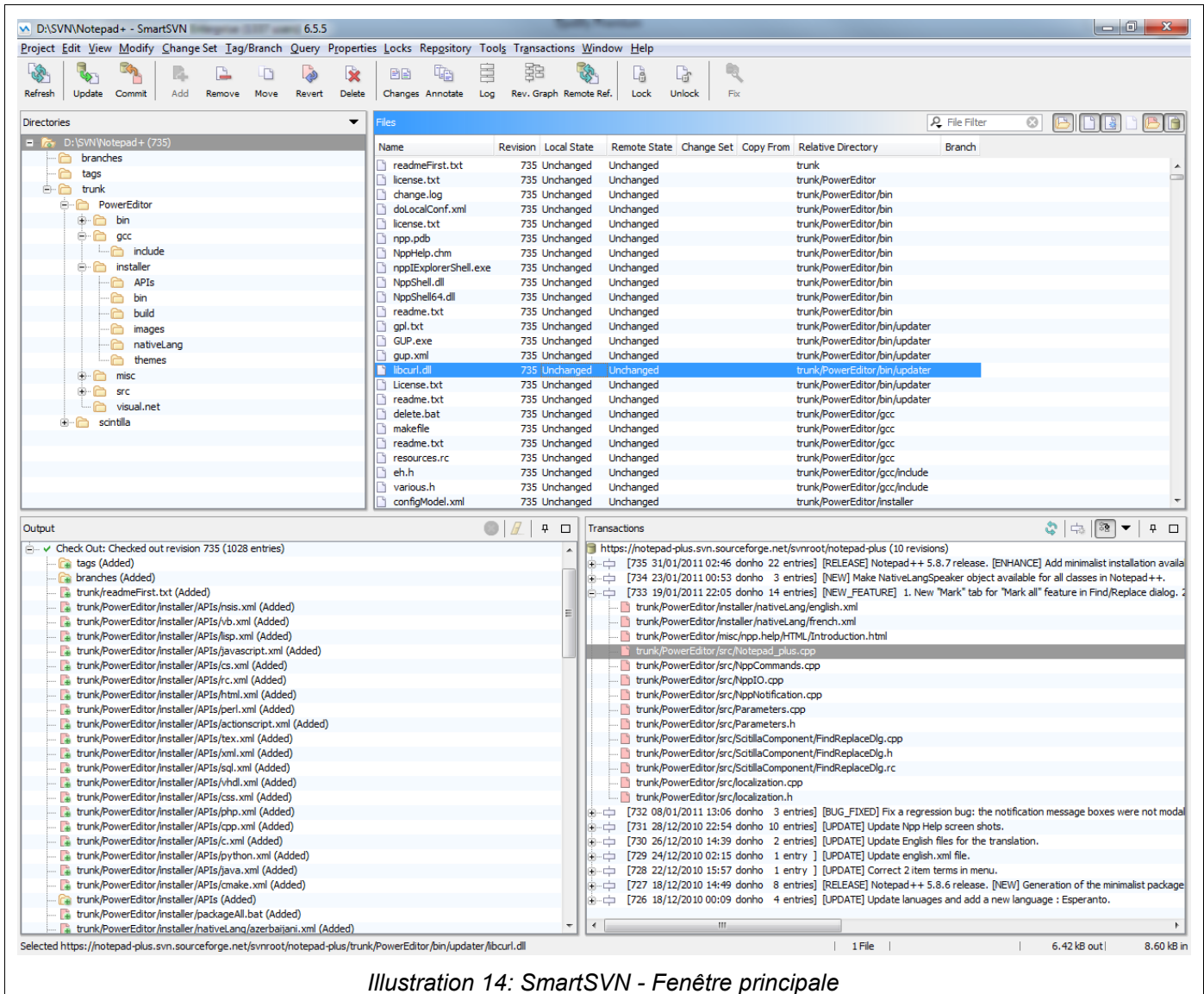


Illustration 14: SmartSVN - Fenêtre principale

L'illustration 14 présente l'interface de travail principale de SmartSVN. Elle se divise en quatre panneaux distincts :

- En haut à gauche, l'arborescence du dépôt local, au sein de laquelle les liens d'externalisation apparaissent de façon transparente ;
- En haut à droite, la liste des fichiers présents dans le dossier courant ainsi que les informations sur leur état dans le dépôt :
 - Numéro de version ;
 - État local comparé à l'état distant (ajouté, modifié, supprimé, restauré, etc.) et inversement ;
 - Chemin complet au sein du dépôt ;
 - Branche.

- En bas à gauche, l'historique simple des interactions effectuées avec le dépôt (propagations, mises à jour, etc.) ;
- En bas à droite, les dernières propagations enregistrées par le dépôt ainsi que les composants modifiés ;
- Enfin, dans la partie supérieure, un panel d'outil permet d'exécuter les principales commandes supportées par le protocole :
 - Mise à jour du dépôt vers le répertoire local ;
 - Propagation ;
 - Ajout, suppression, déplacement, restauration ;
 - Affichage des changements entre les versions locales et distantes ;
 - Logs et graphiques de révisions ;
 - Options de verrouillage.

L'interface de base ne présentant guère de surprises par rapport aux fonctionnalités proposées par le protocole, on se focalisera d'avantage sur les différents outils offerts par le logiciel.

3.4.3 Historique & Comparaisons

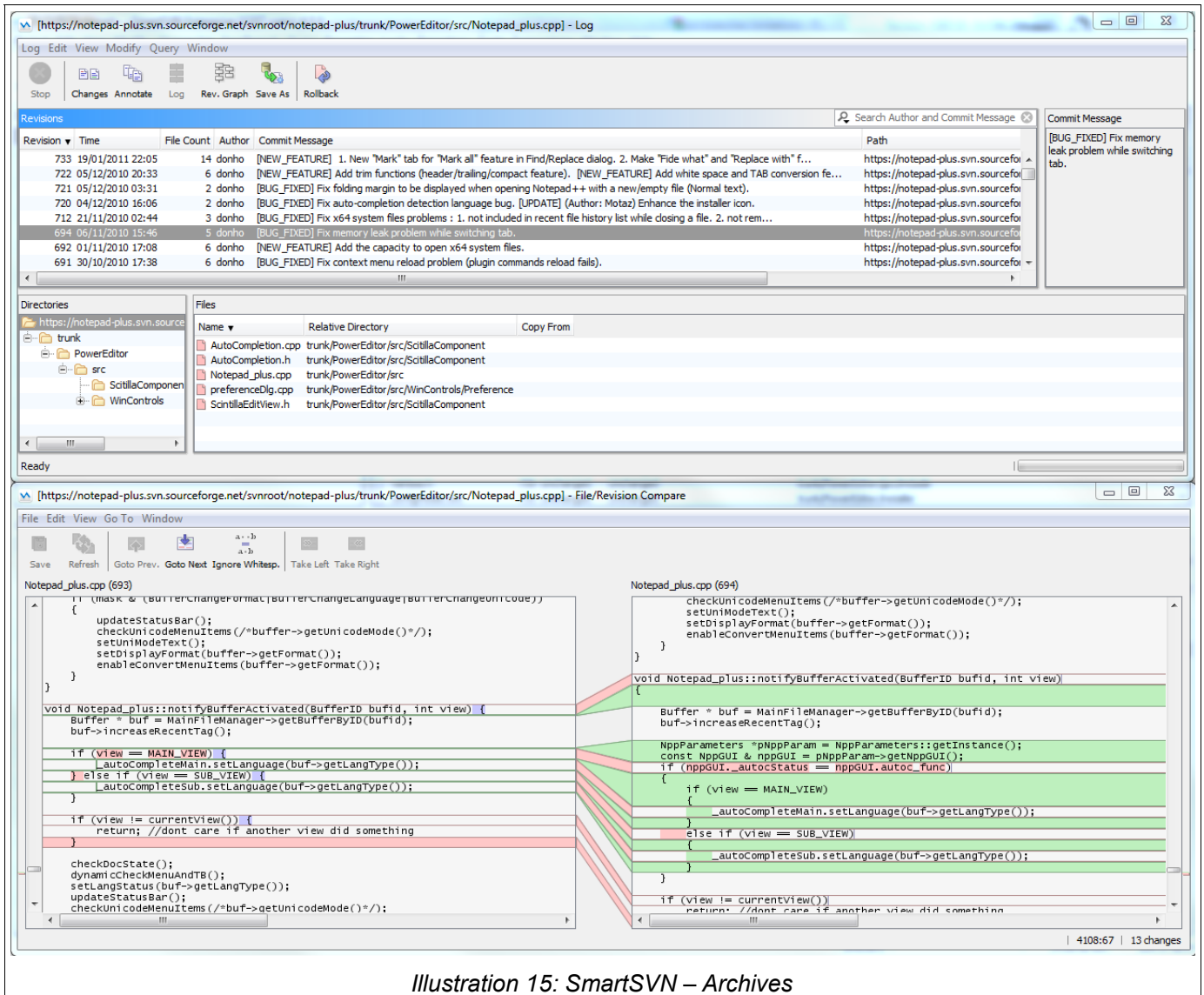


Illustration 15: SmartSVN – Archives

L'illustration 15 présente le principe d'archivage illustré par l'étude de l'historique d'un fichier choisi au hasard dans le dépôt.

La fenêtre du haut permet de consulter les changements subis par le fichier au fil du temps. Les opérations sont organisées chronologiquement par propagations successives ; l'interface affiche pour chacune d'entre elles :

- Le numéro de version associé ;
- L'heure et la date ;
- Le nombre de fichiers concernés ;
- L'auteur ;
- Le commentaire associé, souvent rendu obligatoire par les administrateurs du dépôt ;
- Le chemin vers le fichier.

Un simple clic permet en outre d'afficher les fichiers concernés.

La fenêtre du bas correspond à la mise en parallèle de deux versions successives d'un même fichier sur le dépôt. Il s'agit d'un logiciel comparatif classique, permettant la mise en exergue des éléments ajoutés, modifiés et supprimés d'un état à un autre par l'utilisation de couleurs. Il permet par ailleurs les opérations de fusion, utiles à l'intégration d'une branche sur tronc commun évoquée précédemment.

3.4.4 Visualisation des branches

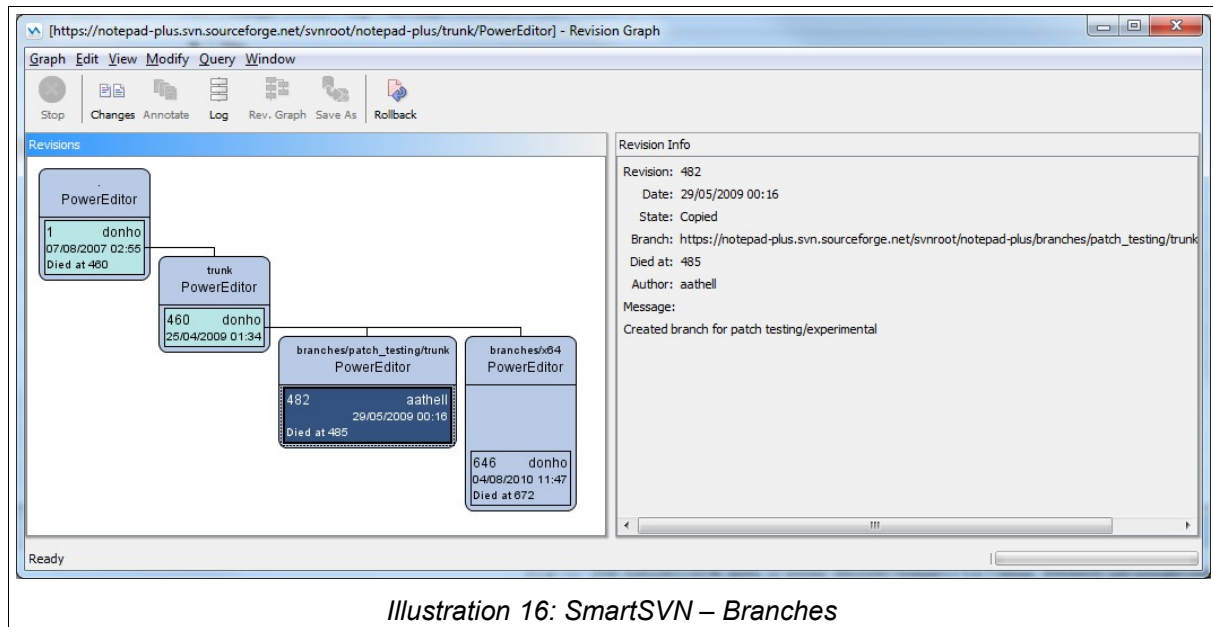


Illustration 16: SmartSVN – Branches

Le troisième et dernier outil abordé dans cette section permet la visualisation graphique des différentes branches créées depuis la naissance du dépôt. L'illustration 16 en présente un cas d'utilisation classique.

À gauche, un graphique sous forme d'arbre indique les branches créées depuis la racine ; on y retrouve de gauche à droite et de haut en bas les opérations de duplication réalisées, ainsi que leurs noms, leurs rôle et leur état courant. Un simple clic permet d'afficher, à droite, diverses informations sur la branche en elle-même :

- Le numéro de version lors de la duplication ;
- La date et l'heure de l'opération ;
- Le type d'opération, emprunte ou duplication ;
- L'adresse dans le dépôt ;
- La dernière propagation connue ;
- L'auteur de l'opération ;
- Les commentaires éventuels.

Tout comme n'importe quel répertoire ou fichier, une branche possède son propre historique et se comporte du reste comme un dépôt parfaitement indépendant de sa source originale.

3.4.5 Retour d'expérience

À l'usage, SmartSVN se révèle aussi simple que complet à l'utilisation dès les concepts fondamentaux du protocole assimilés. Ainsi, un premier apprentissage à partir des commandes de base ne constitue aucunement un luxe pour pouvoir profiter pleinement du logiciel et de ses capacités.

De manière plus générale, l'utilisation d'un dépôt SVN s'avère d'une grande simplicité pour les clients finaux. Le nombre restreint de commandes utiles (mise à jour, propagation, ajout, et suppression) et la syntaxe épurée qu'elles présentent en permet une compréhension rapide ; ce facteur revêt une importance d'autant plus grande qu'il facilite la transition au sein des structures souhaitant s'équiper : en terme de coût de formation, d'une part, et de temps nécessaire à la transition, de l'autre.

3.5 Synthèse

En conclusion, que peut-on retenir des systèmes dédiés à la gestion de versions ?

Désormais datés – SCCS, un des premiers protocoles de ce type, approche de son quarantième anniversaire [44] – mais incontournables dans tout projet informatique correctement organisé, ils apportent une sécurité et une lisibilité indispensables à la gestion du projet. Ils garantissent la traçabilité du changement, simplifient les développements expérimentaux ou risqués, et améliorent la division des ressources entre les intervenants. Leur utilité s'étend d'ailleurs depuis peu auprès d'un public plus large, par le biais notamment de services permettant à tout un chacun de conserver une partie de ses documents personnels sur des espaces distants et auto-gérés tels que Dropbox, Google Docs, et autres.

On peut néanmoins formuler quelques reproches à ce standard de production généralement adopté sans difficulté par un nombre croissant d'utilisateurs. Outre la question des conflits de version, évoquée précédemment, le problème des ressources consommées par ce type de système s'inscrit en problématique majeure dans un environnement tourné vers l'informatique écologique (*Green IT*) : les ressources consommées, autant pour le dépôt que pour ses copies locales, occupent évidemment bien plus de place qu'un simple serveur partagé entre les intervenants.

En résumé, même si les principaux défauts immédiats que posent les systèmes de ce type peuvent se pallier au moyens de pratiques adéquates (fragmentation des dépôts entre les intervenants adaptés, limitations de taille pour les fichiers, groupage des propagations), rien n'assure qu'ils n'évolueront pas vers une forme différente à celle que nous connaissons aujourd'hui. Les solutions logicielles décentralisées, permettant un développement déconnecté pour un partage ultérieur des travaux réalisés, pourraient constituer une issue possible à ces questions.

4. Troisième tiers : intégration continue

4.1 Principe

L'intégration continue se veut une solution aux problèmes posés par la mise à jour de composants distincts et indépendants dans un ensemble plus vaste. Encore mal définie [45], elle peut se voir comme « [...] une pratique de développement logiciel par laquelle les membres de l'équipe [de production] intègrent leur travail fréquemment, voir quotidiennement – amenant à de multiples intégrations journalières » [46] visant à « [...] améliorer la qualité logicielle et à réduire le temps nécessaire à le livrer, en remplacement la pratique traditionnelle du contrôle de qualité appliquée après complétion des développements » [47].

Elle s'appuie sur deux pré-requis fondamentaux (détaillés dans la section 4.2.1) :

- L'utilisation d'un dépôt de versions centralisé ;
- La définition d'une politique de compilation standardisée.

L'objectif poursuivi a globalement une finalité informative : les erreurs techniques, mais également fonctionnelles, doivent pouvoir être identifiées et documentées dès que possible afin de bénéficier d'une résolution priorisée et, autant que possible, rapide. Souvent sous-estimés par les équipes de production, les problèmes d'intégration peuvent prendre une tournure désastreuse – que Ward Cunningham, inventeur du concept de wiki et pionnier de l'*Extrem Programming*, désigna en 2009 sous le sobriquet d'« enfer de l'intégration » [48] ; d'ailleurs, on constate que, ironiquement, le rôle d'arrière-plan qu'ils occupent dans le projet est justement à la source des conséquences qu'ils peuvent engendrer : détectés à la dernière minute, ils interfèrent dans les plannings de production et génèrent des retards souvent coûteux.

Excroissance des théories d'*Extrem Programming* et de *Test Driven Development* – technique consistant à injecter directement dans le code une batterie de tests afin d'automatiser la validation de tout ou partie des aspects fonctionnels du programme –, le concept d'intégration continue n'a pas encore atteint un niveau de maturité suffisant à en donner une définition absolue et définitive. La suite de ce chapitre consiste donc plus en un état de l'art qu'en un exposé global sur le sujet – à l'instar des deux technologies précédemment abordés.

4.2 Effets

4.2.1 Pré-requis

Le déploiement de l'intégration continue dans un protocole de production nécessite donc deux éléments fondamentaux : l'utilisation d'un dépôt de version et la mise en place de compilations automatisées.

Le premier point s'explique facilement. Étant donné que l'intégration continue vise à détecter au plus tôt les dysfonctionnements techniques liés au partage d'une multitude d'éléments – blocs de code, éléments de configuration, etc. –, il apparaît logique de pouvoir avant toute chose distinguer et identifier les évolutions successives du programme : le choix d'un dépôt de versions semble donc tout indiqué. Par ailleurs, ce dépôt doit être partagé de tous les intervenants du projet et posséder une branche principale de développement (*mainline* ou *trunk*) reflétant la version la plus récente du programme [50].

Le deuxième point constitue l'essence même de l'intégration continue. Afin de pouvoir vérifier la validité des ajouts successivement effectués par les développeurs au sein de l'application, il convient de réaliser aussi souvent que possible une vérification globale des modifications apportées ; la meilleure façon de satisfaire cette contrainte passe par l'instauration d'un protocole de test, généralement désigné sous le nom de « compilation automatique » (*build automation*) mais s'étendant en fait à un domaine plus large que la compilation seule.

L'illustration 17 propose un descriptif des différentes étapes possibles lors d'une validation standard menée dans un contexte d'intégration continue. On y distingue quatre étapes successives :

- La compilation à proprement parler ;
- Le démarrage de l'application ;
- Le protocole de tests automatisé ;
- La génération de documentation.

Chaque étape concerne une ou plusieurs facettes du projet associées à un ou plusieurs corps de métier. Ainsi :

- Durant la compilation, on valide :
 - Le code ;
 - Les structures de données.
- Durant le démarrage de l'application ou son déploiement sur un serveur dans un contexte de développement web :
 - La configuration ;
 - Les modules externes ;
 - Les médias ;
 - Et, dans une certaine mesure, les structures de données.
- Durant le protocole de test (explicité plus tard) :
 - Le respect de la logique métier ;
 - Le respect des spécifications ;
 - La prise en charge des contenus prévus.
- Enfin, durant la phase de documentation (également évoquée ultérieurement) :
 - Le comportement global en intégration ;
 - Les notes de version.

Évidemment, les étapes ainsi que les intervenants peuvent varier selon les caractéristiques du projet ; il s'agit néanmoins d'une structure de base couramment rencontrée dans ce type de pratique.

L'idée d'une politique de compilations standardisée consiste à réaliser de façon impromptue et ponctuelle l'intégration complète composants du projet afin de vérifier que le comportement des uns n'interfère pas avec l'activité des autres. Martin FOWLER, consultant reconnu pour son travail sur le sujet et co-signataire du Manifeste Agile [51], préconise dans un article de 2009 [52] la stratégie suivante : chaque propagation sur le dépôt doit entraîner un *build* (terme difficilement traduisible, pouvant être défini comme « [...] le processus de conversion des fichiers

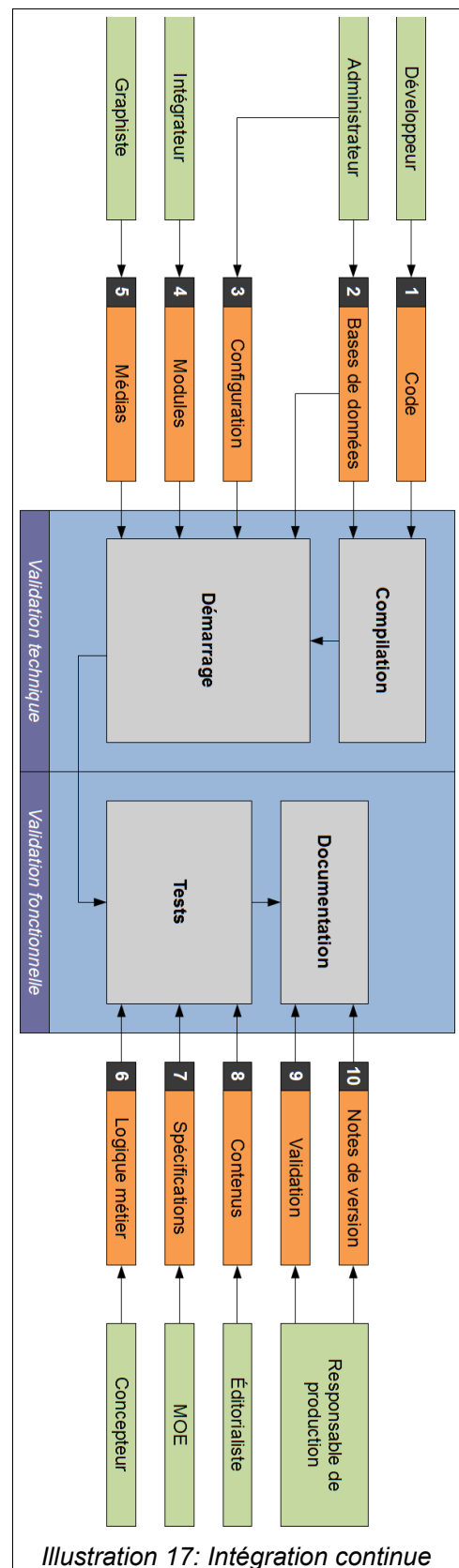


Illustration 17: Intégration continue

sources vers une entité exécutable pouvant être lancée sur un ordinateur [...] » [53]) du dépôt dans son ensemble ; les intervenants doivent soumettre leur travail à minima quotidiennement. Dans de nombreuses structures, cette opération s'effectue, en supplément ou non, durant la nuit, de manière à disposer dès le matin d'un rapport complet sur de possibles problèmes apparus la veille.

Généralement, l'intégration continue se fait au moyen d'un service déployé sur le dépôt ; plusieurs solutions logicielles, supportant la plupart des protocoles dédiés à la gestion de versions, existent à cette fin [54]. Leur choix s'effectue, ici encore, selon les caractéristiques du projet : logiciel dédié à la compilation et aux tests, interface souhaitée, possibilités d'intégration, etc.

4.2.2 Compléments

On l'aura compris, l'intégration continue n'a clairement pas atteint une forme définitive et ne cesse d'évoluer au gré de l'usage qu'en font aujourd'hui ses utilisateurs. Pour preuves, plusieurs bonnes pratiques (*best practices*) ont au fil du temps fait leur apparition ; la suite de cette section en présente quelques-unes des plus courantes.

Implémentation de tests unitaires

Déjà évoquée dans la section précédente, cette technique, utilisée par ailleurs en développement orienté test (*Test Driven Development*) [55], consiste à implémenter au sein même du code source une batterie de tests validant l'aspect fonctionnel de l'application. Pour les langages de haut niveau, et plus particulièrement orientés objet, ils peuvent prendre la forme de classes dédiées à cet effet ; à plus bas niveau, une étude des comportements-type, en boîte noire (*black-box testing*) [56] par exemple, peut être envisagée. Généralement conçus grâce à une technique dédiée (type xUnit [57]), ils émettent des résultats interprétables par le serveur d'intégration continue.

Validation sous environnement vierge

Afin de permettre l'exécution de tests standardisés, notamment sur les structures de données utilisées par le programme, il s'avère souvent utile d'utiliser un contenu factice apte à simuler un environnement de travail réel (cette nécessité a d'ailleurs donné naissance aux objets simulacres, ou *mocks*, utilisés en tests unitaires Java [58]). Or, si ces vérifications basées sur un environnement artificiel permettent couvrir un spectre plus large que dans un contexte « à vide », elles ne doivent pas les éluder pour autant : dans le cas contraire s'installe un risque d'imprévu lors du déploiement en production (communément appelé à tourner au cauchemar).

Maintien d'une image valide

Une version du programme échouant lors du processus d'intégration (à la compilation, au démarrage ou pendant les tests standardisés) ne possède, à priori, que peu de chances de figurer dans les emprunts livrables du dépôt (voir 3.2.5). Une réponse adéquate à ce constat consiste en le maintien automatisé d'une branche « valide » correspondant à la dernière version validée sans erreurs à l'intégration. Il apparaît dès lors possible de disposer à tout moment d'une instance fonctionnelle et validée du programme.

Maintien d'une compilation rapide

L'installation d'une solution d'intégration continue multiplie le nombre de compilations menées directement ou à partir du dépôt principal. Simple à expliquer, l'ampleur de ce phénomène dépend de deux facteurs : un, la fréquence des tests prévus par la politique définie à leur sujet ; deux, la taille du projet – poids des applications et nombre d'intervenants. Le maintien d'une compilation rapide permet de limiter les effets de ce problème ; or, dans le cadre d'un projet de taille importante, celle-ci passe nécessairement par une intégration à paliers (*staged build* [59]).

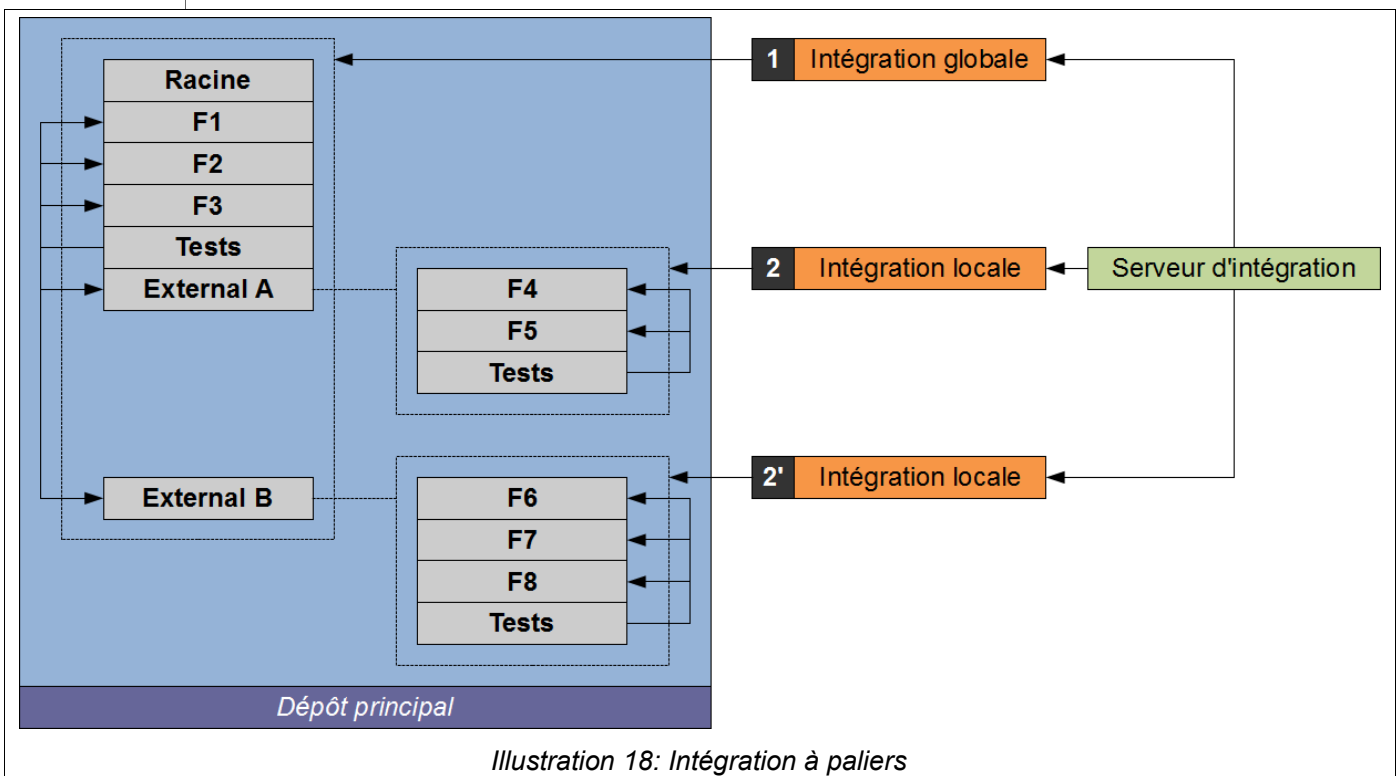


Illustration 18: Intégration à paliers

L'illustration 18 propose un aperçu de cette technique. Celle-ci consiste à diviser le dépôt principal (*mainline*) en une arborescence de sous-dépôts (par externalisation ou répartition en branches) pouvant individuellement être soumis à des tests d'intégration continue. Il convient ensuite d'adapter le comportement du serveur de manière à limiter les ressources requises : les tests engendrés par des propagations se limitent au(x) palier(s) concerné(s) et ceux lancés de manière ponctuelle (une ou deux fois par jour) englobent l'ensemble du programme.

Génération et diffusion des résultats

En parallèle de l'apport technique – somme toute relativement minime – délivré par l'intégration continue émerge une finalité essentiellement informative. Étant donné, on l'a vu, que chaque intervenant, indépendamment de son niveau ou de son métier, peut accidentellement provoquer des anomalies lors du processus d'intégration, il semble logique qu'ils puissent également prendre connaissance des tests menés en prévision. Une des préconisations en la matière tient donc dans la diffusion la plus large possible des résultats obtenus lors des compilations standardisées [60], par le biais d'un réseau intranet ou par messagerie.

4.3 Effets

Une politique d'intégration continue constitue un choix optionnel, mais judicieux, pour tout organisme en charge d'un projet informatique dont les intervenants ne connaissent et surtout ne maîtrisent pas toutes les facettes.

À l'instar des techniques évoquées dans les deux chapitres précédents, elle ne peut être ni décidée, ni déployée à la légère : elle implique nécessairement des choix de gestion cruciaux qui, insuffisamment considérés, peuvent impacter considérablement sur les apports de son mécanisme. Néanmoins, une fois convenablement implantée dans le processus de production, elle apporte autant une sécurité supplémentaire face aux intempéries de dernière minute qu'une visibilité bienvenue sur l'ensemble du projet. Identifiés dans un délai maximal de l'ordre d'une journée, les problèmes

d'intégration deviennent dès lors une priorité commune, placée directement sous la responsabilité des intervenants mis en cause par leurs ajouts au projet.

Cette mise en exergue des responsabilités individuelles peut également constituer une ombre au tableau : facteur de pressions supplémentaires, elle peut attiser des conflits pré-existants au sein des équipes, tels que ceux liés au niveau de compétence, l'implication, ou le soin apporté par chaque intervenants dans son travail. Par ailleurs, le processus d'intégration continue peut, faute d'un suivi correctement effectué, étouffer le choix d'une stratégie de développement global pour encourager celui d'une progression au jour le jour, sans réelle concertation sur le pilotage général du projet.

4.4 L'exemple : Hudson

4.4.1 Introduction

Parmi la vingtaine de solutions logicielles dédiées à l'intégration continue [61], Hudson se distingue autant par sa gratuité que par sa popularité : utilisé « [...] par plus de 17.000 installations serveur à travers le monde à la fois dans de petites, moyennes et grandes compagnies, dont eBay, HP, MySQL, JBoss, Xerox, Yahoo, LinkedIn ou Goldman-Sachs » [62], il constitue un exemple idéal pour les principes précédemment évoqués.

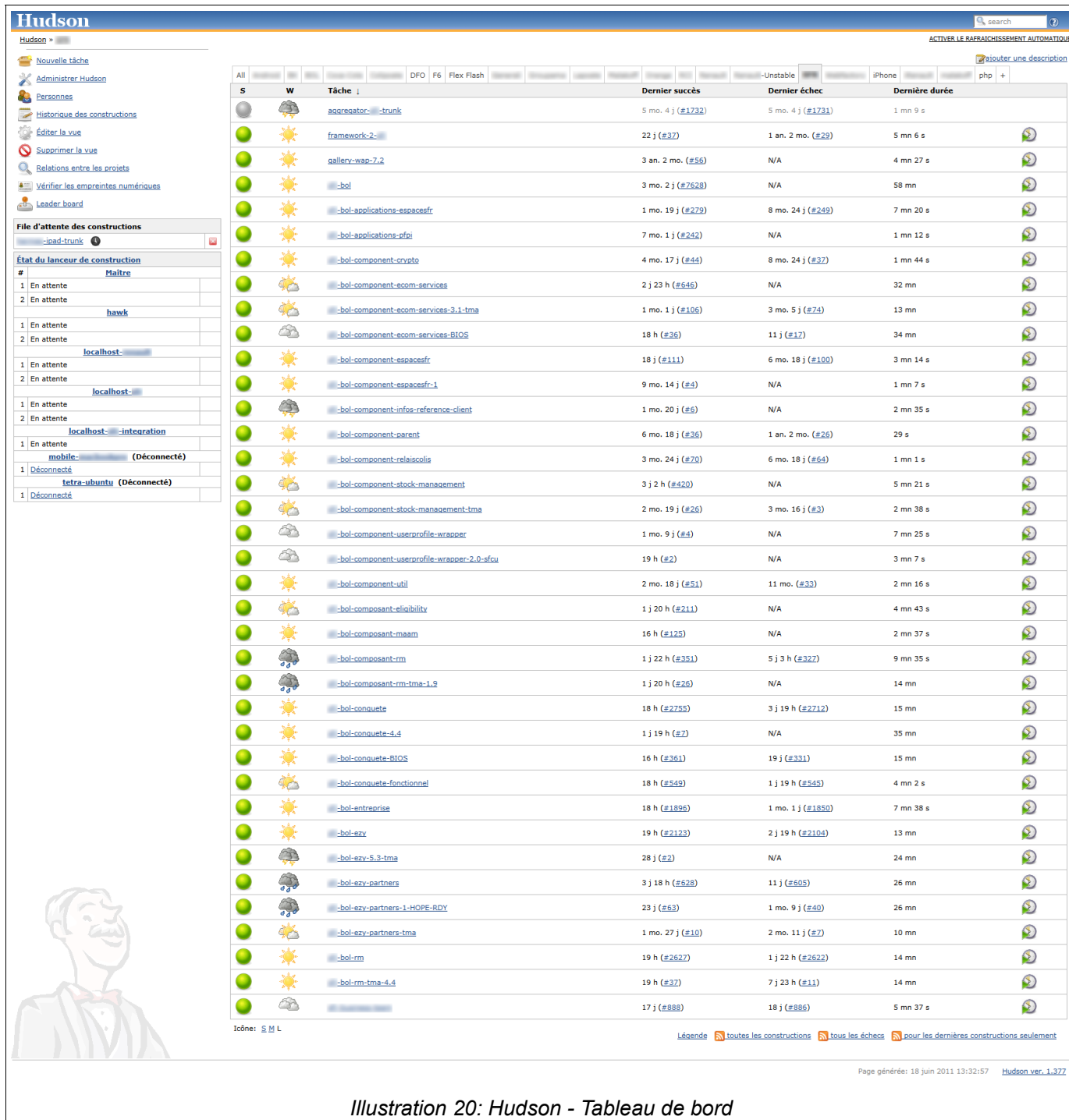


Placé en novembre 2010 sous le contrôle d'Oracle (donnant naissance au projet Jenkins suite à une affaire d'ordre juridique [63]), ce programme développé en Java requiert un serveur JEE dûment configuré pour fonctionner [64]. Sa finalité se divise en deux lignes : compiler et tester le projet, d'une part, et permettre le suivi des tâches lancées depuis l'extérieur, de l'autre [65] ; on retrouve ici les constantes courantes en matière d'intégration continue : valider et informer. Pour y parvenir, Hudson fait appel à un large spectre de fonctionnalités [66], dont :

- L'utilisation de courriels, flux RSS ou encore protocoles de messagerie instantanée ;
- L'intégration des technologies JUnit et TestNG, le premier fonctionnant en boîte blanche et le second en boîte noire ;
- L'externalisation des compilations – et donc des ressources nécessaires – sur les postes des intervenants ;
- Le suivi des fichiers par empruntes ;
- L'utilisation de liens permanents et standardisés ;
- Le support de plugins variés.

Outre ces différents outils, il propose une interface web complète qui, après identification, permet aux intervenants de visualiser et d'intervenir sur les tâches d'intégration en cours ou passées.

4.4.2 Tableau de bord



The screenshot displays the Hudson web interface. On the left, there is a sidebar with navigation links such as 'Nouvelle tâche', 'Administrer Hudson', 'Personnes', 'Historique des constructions', 'Éditer la vue', 'Supprimer la vue', 'Relations entre les projets', 'Vérifier les empreintes numériques', and 'Leader board'. Below these is a section for 'État du lanceur de construction' showing the status of various builders like 'Maitre', 'hawk', 'localhost', 'mobile', and 'tetra-ubuntu'. The main area is a large table of builds. The table has columns for 'S' (status), 'W' (workspace), 'Tâche' (task), 'Dernier succès' (last success), 'Dernier échec' (last failure), and 'Dernière durée' (last duration). Each row represents a build, with icons indicating its status (green for success, yellow for unstable, red for failure) and a link to its details. The builds are organized hierarchically by task name, such as 'aggregator', 'framework-2', 'gallery-wap-7.2', and various 'bol' components. At the bottom right, there is a legend for the status icons and a footer indicating the page was generated on 18 juin 2011 at 13:32:57, version 1.377.

Illustration 20: Hudson - Tableau de bord

L'illustration 20 présente le tableau de bord principal d'Hudson. Doté d'une interface très simple, il propose une liste des dépôts indépendamment de leur taille ou de leur place dans une hiérarchie à paliers. Pour chacun d'entre eux, le logiciel indique :

- Le résultat obtenu lors de la dernière compilation (succès, instable ou en échec) ;
- La stabilité générale du dépôt (en fonction du taux de succès obtenus lors des dernières validations effectuées) ;

- Le nom du dépôt ;
- La date du dernier succès ;
- La date du dernier échec ;
- La durée de la dernière itération de compilation et test effectuée par le serveur.

En outre, divers liens permettent d'accéder rapidement aux outils principaux du système : interfaces dédiées à un dépôt donné (détaillée section 4.4.3), lancement manuel d'une itération, accès aux logs, et ainsi de suite. Par ailleurs, un panneau situé sur la gauche permet de connaître en temps réel les itérations en cours ou prévues.

De manière plus générale, l'interface, très optimisée, permet une navigation simple et intuitive au sein d'un système de vues personnalisable et intuitif.

4.4.3 Gestion d'un dépôt

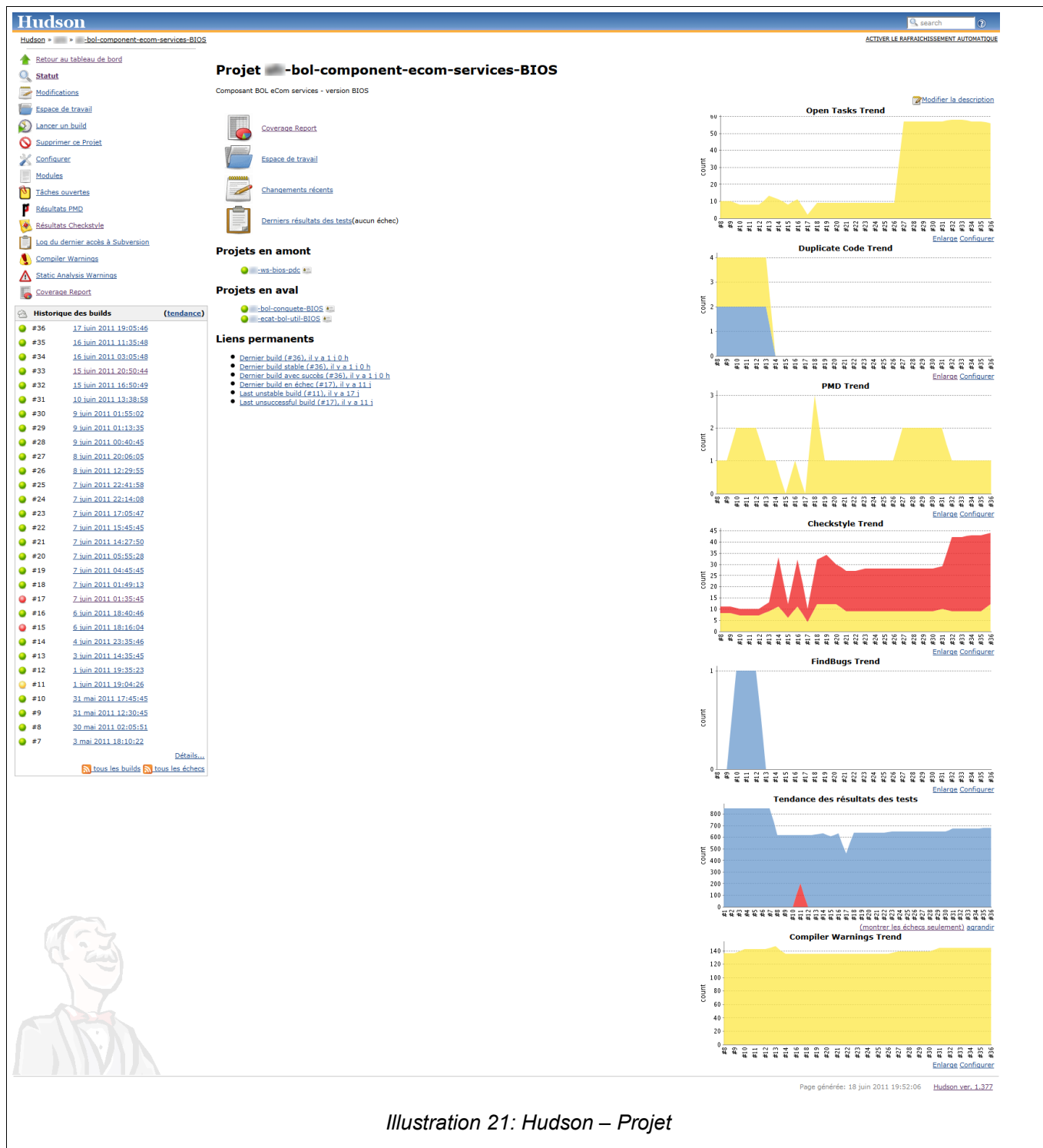


Illustration 21: Hudson – Projet

L'illustration 21 présente l'écran dédié à la gestion d'un dépôt donné. Il se divise en trois parties : à gauche, une navigation et un historique de l'activité ; au centre, des informations et outils généraux ; à droite, plusieurs graphiques présentant la résistance du dépôt aux principaux tests menés sur son contenu :

- Détection des balises TODO au sein des sources ;
- Détection des éléments de code dupliqués ;
- Validation conceptuelle via PMD [67] ;
- Validation syntaxique via Checkstyle [68] ;
- Validation fonctionnelle via Findbugs [69] ;
- Tests standardisés JUnit ;
- Avertissements générés par le compilateur.

Chacun d'entre eux dispose d'une interface indépendante et détaillée, permettant d'identifier rapidement les causes et effets des dysfonctionnements relevés.

Par ailleurs, un historique des itérations effectuées par le serveur (désignées dans la version française sous le nom de « construction », de l'anglais *build*) permet de consulter simplement les dernières validations effectuées sur le dépôt : date, origine, résultats, contexte (local ou en aval d'un autre palier), environnement (prédéfini ou vierge), etc.

4.4.4 Retour d'expérience

Pour l'utilisateur final, l'une des surprises d'Hudson tient sans doute dans l'aspect particulièrement permissif qu'il présente. L'accès à l'information ne connaît nulle restriction (tout le monde peut accéder à tous les rapports pour tous les dépôts) et la participation des intervenants passe au premier plan, grâce à un panel d'outils public développé : le lancement à distances des cycles de compilation et de validation, la possibilité de commenter les rapports générés par le système, ou même de configurer le comportement des tests, encourageant ainsi les apports de tout un chacun.

Malgré tout, on pourra critiquer les régressions importantes qu'un tel mécanisme engendre en termes de sécurité. Si le serveur d'intégration reste indépendant des dépôts de données à proprement parler (dont il maintient généralement une copie), il n'empêche que de nombreuses informations pouvant revêtir un caractère confidentiel (par exemple, des rapports d'erreur pointant des failles exploitables en production) deviennent dès lors accessibles à tous les intervenants du projet. Si ce point ne pose guère de problème dans un contexte de développement participatif, il peut également constituer un souci de taille en entreprise et au sein de projets nécessitant un niveau élevé de sécurité (secteur bancaire, médical, militaire, ou autre).

4.5 Synthèse

On l'aura compris, la question de l'intégration continue ne connaît pas encore de réponse définitive. Les solutions logicielles existantes, encore jeunes, se basent plus sur la participation d'utilisateurs et de contributeurs passionnés par le sujet que sur des constats solides, établis et surtout durables. D'autre part, l'absence de cohésion dans les recherches en la matière, mise en exergue par le conflit ayant donné naissance à Jenkins, nuit fortement à la visibilité de cette technique novatrice.

Mais ces handicaps n'empêchent pas un constat simple : l'intégration continue commence à marquer les mentalités. De plus en plus présente en entreprise, elle répond à un besoin réel, particulièrement pointé par les méthodes agiles : la réactivité face à l'imprévu. Trop souvent organisée de manière trop stricte, à l'aide de processus inadaptés car inspirés du domaine industriel (à l'instar de la méthode MERISE [70]), la conduite de projet informatique gagne à prendre en souplesse.

Néanmoins, il convient de ne pas confondre effet de mode et révolution technologique. Si la généralisation des réseaux, l'émergence des langages de programmation de haut niveau ou la formalisation progressive des processus de production (notamment par le biais des normes de qualité) peuvent être perçues comme de réelles évolutions du domaine informatique, rien n'assure que l'intégration continue, dans sa forme courante, saura trouver une place à part entière dans la conduite de projet de demain.

5. Usage combiné

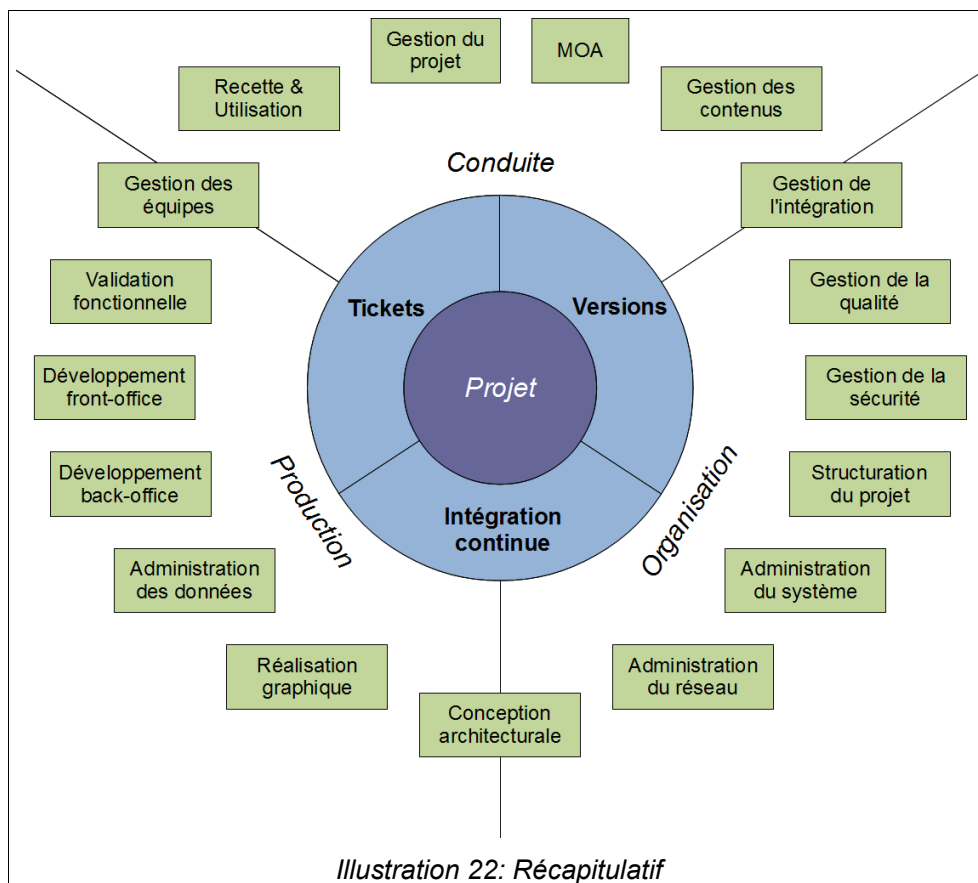
5.1 Théorie

5.1.1 Récapitulatif

Avant de se pencher sur les possibilités offertes par l'utilisation combinée des trois techniques présentées dans les chapitres précédent, il convient de se pencher sur une représentation plus globale des sujets évoqués. Pour se faire, on choisit de diviser le projet en trois sphères de compétences :

- La conduite, englobant l'ensemble des tâches liées à la validation, la livraison et l'utilisation du projet ;
- L'organisation, relative aux problématiques d'intégration, d'architecture ou encore de qualité ;
- La production, désignant le travail de développement à proprement parler, mais aussi à l'administration des données et l'encadrement des équipes.

Mises en commun avec les principaux postes existants dans le domaine informatique, elles donnent la représentation suivante du projet.



Deux points sont néanmoins à noter. Premièrement, il survient couramment, et particulièrement dans les projets de taille restreinte, qu'un même individu occupe plusieurs des postes représentés sur le schéma. Deuxièmement, peut arriver, cette fois plutôt dans de grandes structures, qu'un des postes mentionné constitue un projet à part entière ; solution efficace notamment dans le cas d'une

mutualisation d'un même poste entre plusieurs services – c'est souvent le cas de l'administration du système ou du réseaux.

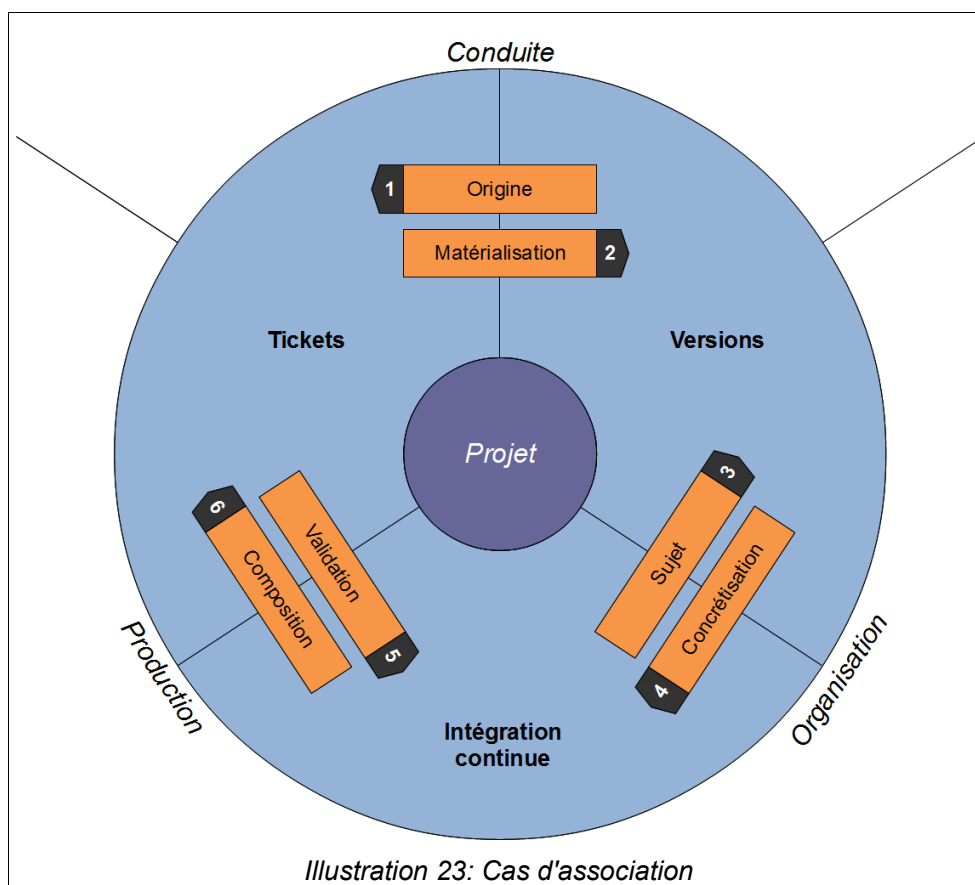
Alors où placer, dans ce contexte, les trois techniques étudiées ? Destinées à assurer la jonction, on l'a vu, entre des corps des métiers très différents, elles ne peuvent résumer leur influence à une sphère spécifique du projet. Toutefois, elles ne s'y généralisent pas non plus entièrement. On propose dès lors l'hypothèse suivante :

Sphère	Utilise	Technique	Causes
Conduite	Oui	Tickets	Communication avec l'utilisateur final Suivi d'avancement
	Oui	Versions	Réalisation des livrables Initiation et gestion des évolutions
	Non	Intégration continue	Absence de suivi quotidien Indépendance des problématiques d'ordre technique
Organisation	Non	Tickets	Insensibilité aux problèmes d'ordre fonctionnel Indépendance globale sur le plan technique
	Oui	Versions	Maintien d'un dépôt à jour Suivi des évolutions
	Oui	Intégration continue	Maintien d'une version valide et stable Suivi de la qualité
Production	Oui	Tickets	Résolution concrète des anomalies Répartition des charges de travail
	Non	Versions	Indépendance des problèmes organisationnels Travail à échelle locale
	Oui	Intégration continue	Gestions des anomalies au jour le jour Responsabilité individuelles face aux problèmes de compatibilité

Tableau 4: Sphères d'intervention

Ce modèle engendre le constat suivant : chaque sphère d'activité fait usage de deux des trois techniques étudiées, la dernière ne présentant qu'un intérêt limité, voir inexistant. On peut toutefois y soulever une critique : il perd son exactitude dans un contexte hiérarchique fort, au sein duquel les différents intervenants doivent adapter leur comportement en fonction de leur place respective, et non de leur métier propre. Ce genre de structure tend néanmoins à disparaître progressivement, son fonctionnement s'adaptant mal à aux projets informatiques – la communication directe fait d'ailleurs partie des quatre axes prônés par le manifeste agile [71].

5.1.2 Cas d'association



Penchons-nous à présent sur les cas d'association, à proprement parler, survenant entre ces trois techniques. On reprend la structure de l'illustration 22 en mettant de côté la question du rôle des intervenant, et en se concentrant sur le type d'informations échangées d'un système à un autre. Leur nature varie d'une transaction à une autre ; le tableau 5 propose un résumé synthétique expliquant pour chacune d'entre elles sa raison d'exister.

Demandeur	Fournisseur	N°	Cause
Tickets	Versions	2	Le ticket s'applique nécessairement à une ou plusieurs version du programme qui doivent donc être référencées
	Intégration continue	5	La résolution du ticket passe obligatoirement par la vérification du correctif déployé en intégration
Versions	Intégration continue	4	Pour exister, une version d'un dépôt donné doit se soumettre à au moins un test d'intégration
	Tickets	1	L'existence d'une version donné du dépôt répond à une problématique référencée par tickets
Intégration continue	Tickets	6	Un test d'intégration s'applique à une ou plusieurs anomalies ou évolutions référencées par ticket
	Versions	5	Le serveur d'intégration utilise une version clairement identifiée du dépôt à tester

Tableau 5: Nature des informations

Même si, une fois encore, on peut reprocher à ce modèle le défaut de ne pas pouvoir s'adapter à l'ensemble des projets informatiques, on retiendra tout de même l'idée suivante : les interactions entre

les techniques étudiées existent à échelle globale, sous des formes divergentes. Il peut s'agir de données brutes (cas 2 et 3 : versions), de méta-informations (cas 4 et 5 : intégration continue) ou d'informations abstraites (cas 1 et 6 : tickets).

5.1.3 Normalisation

On s'appuie désormais sur deux principes établis :

- Les systèmes étudiés se destinent à couvrir plusieurs domaines d'activité ;
- Les systèmes étudiés communiquent entre eux sous des formes variables.

Or, une question reste en suspens : étant donné que les trois mécanismes génèrent de façon autonome des éléments d'informations à destinations de leur environnement, quel étalon utiliser pour générer ces échanges ?

Si le ticket apparaît en premier dans le processus – car pointant du doigt une anomalie ou une évolution jusque là inexistante –, il ne peut être considéré comme la source de ce trafic : en effet, son cycle de vie n'inclue pas nécessairement la modification des sources du système dans le cas, par exemple, d'une invalidation. Les itérations menées par le serveur d'intégration continue ne sont pas non plus à l'origine de ces échanges : en effet, leur exécution répond à une politique standardisée et ne découlent (généralement) pas d'une intervention humaine. Reste donc la notion de version, et plus précisément l'évènement donnant naissance à cette dernière : la propagation.

Cet événement situé à la base même des développements suffit en effet à justifier l'ensemble des informations circulant d'un système à un autre. Le scénario le plus courant suit la forme suivante :

- Première étape : un développeur soumet son travail au moyen d'une propagation sur le dépôt approprié ; il précise le ou les numéros de tickets décrivant l'action effectuée.
- Deuxième étape : le serveur intégration détecte un changement de version ; il effectue une itération destinée à valider les modifications effectuées.
- Troisième étape : le système de gestions de tickets reçoit une notification du serveur d'intégration pointant les éléments modifiés ainsi que les étiquettes concernées.

À chaque étape, des liens se créent dans les deux sens. Ainsi, une fois la propagation achevée, ils s'enrichissent des informations suivantes :

Détenteur	Pair	N°	Information
Tickets	Versions	2	Numéro(s) de version(s) contenant les correctifs ou évolutions décrites
	Intégration continue	5	Validité fonctionnelle des évolutions
Versions	Intégration continue	4	Validité technique des évolutions
	Tickets	1	Numéro(s) de ticket(s) décrivant les évolutions soumises
Intégration continue	Tickets	6	Historique d'intégration des tickets
	Versions	5	Contenu et changement précis

Tableau 6: Typologie des échanges

Le concept de propagation prend donc une importance inattendue : en sus de son rôle cellulaire, représentatif des évolutions successives du serveur, il acquiert dès lors un aspect fonctionnel et informel utile à tous les stades du processus de production.

La section suivante présente plusieurs exemples de cette communication dans un environnement composé des trois exemples précédemment donnés : JIRA, Subversion et Hudson.

5.2 Usage pratique

5.2.1 Tickets

http://.../secure/ViewProfile.jspa -->

Demande (Afficher en ligne)

Clé: **EVOLUTION-560**

Type de demande: Sous-tâche

Etat: Ouvertes

Priorité: Mineur

Attribution: Guillaume ZAVAN

Rapporteur: Guillaume ZAVAN

Environnement: Tous

Opérations

- ☐ Afficher tout
- ☐ Afficher les commentaires
- ☐ Afficher l'historique

[Front-Office] Filet entre deux colonnes

Mise à jour: 17/juin/11 12:35 PM Création: 10/juin/11 10:56 AM

Le commentaire suivant a été ajouté à cette demande : [Permalien]

Auteur: Hudson Application on hudson

Date: 18/juin/11 12:32 AM

Commentaire:

Integrated in v5 #609

EVOLUTION-560

Projet: Evolutions

Composants:

Affecte les versions: -AG / / Nouveaux modules

Versions corrigées: -AG / / Nouveaux modules

Pièces jointes: 1.png

Description

Problème

Le filet affiché entre deux colonnes situées sur la même ligne en cœur de page n'a pas la couleur adéquate :

Sources

Description	URL
Pré-production	http://...
Développement	http://...
Capture d'écran	1.png

Illustration 24: Serveur d'intégration > Tickets

L'illustration 24 provient d'un courriel envoyé par le système JIRA suite à l'intégration d'un ticket par le serveur d'intégration continue. Celui-ci contient trois éléments d'information : le numéro du ticket concerné, le numéro attribué à la construction (*build*) et le résultat en découlant (le cas échéant, un succès, indiqué par la puce verte).

Repository	Revision	Date	User	Message
Principal	#108949	Fri Jun 17 12:19:05 CEST 2011		EVOLUTION-560

Files Changed

MODIFY [1.png](#)

Illustration 25: Dépôt > Tickets

L'illustration 25 présente un commentaire rédigé par le dépôt de versions sur un ticket donné. Il pointe plusieurs informations :

- La branche concernée par la propagation ;
- Le numéro de version associé ;
- La date et l'auteur de l'opération ;
- Les fichiers affectés.

5.2.2 Versions

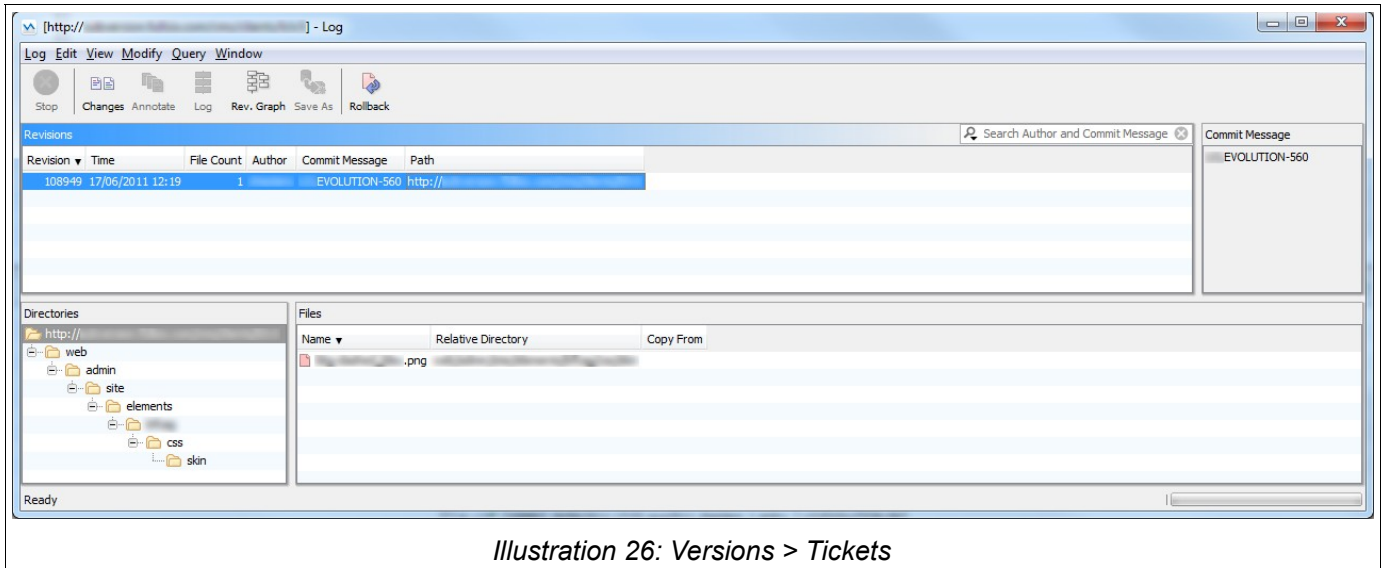


Illustration 26: Versions > Tickets

L'illustration 26 présente les informations fournies par SVN au sujet d'une propagation donnée. On remarque sur la droite la présence d'un commentaire (indiqué par le développeur) indiquant le ou les tickets concernés par l'évolution : il apparaît dès lors possible d'obtenir un descriptif du travail fourni sans passer par la lecture du code à proprement parler.

5.2.3 Intégration continue

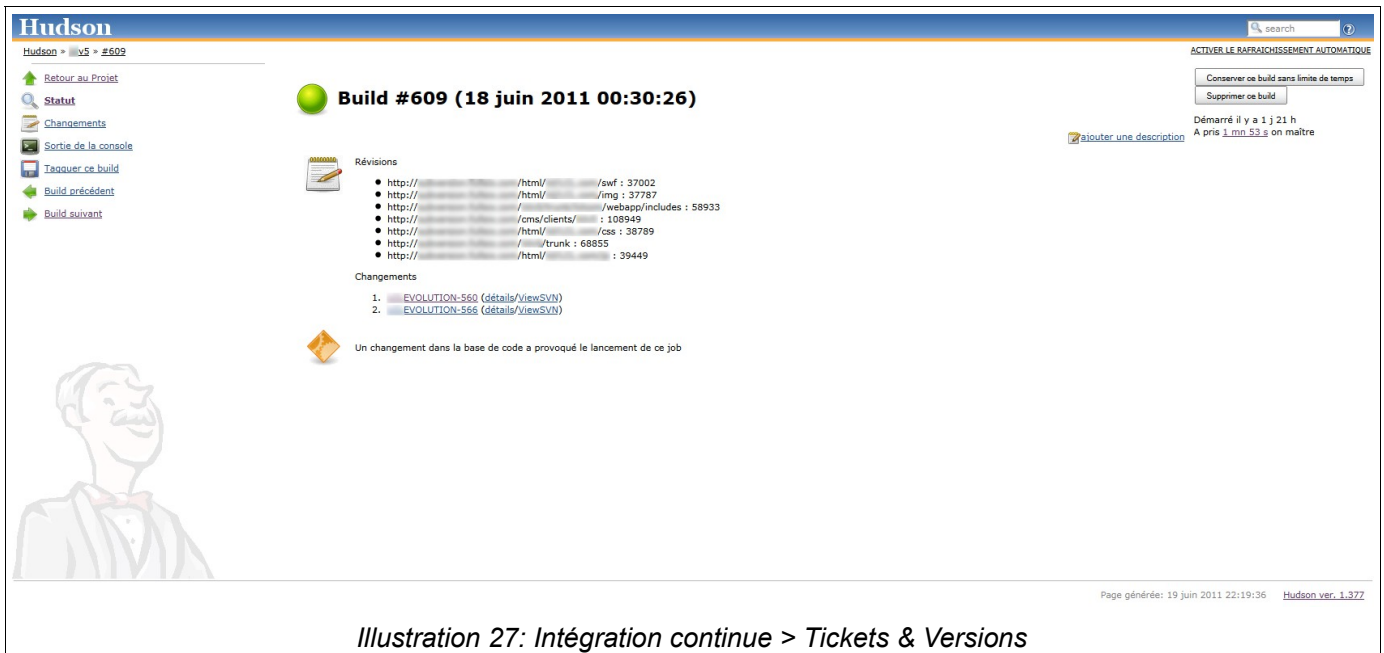


Illustration 27: Intégration continue > Tickets & Versions

L'illustration 27 présente un dernier exemple de communication entre les systèmes. Issu de l'interface web proposée par Hudson, il présente les résultats d'une itération effectuée sur le projet courant. Les informations liées à l'environnement sont ici pointées très clairement : en haut, la liste des dépôts concernés et leur version au moment de l'intégration ; en bas, la liste des changements, indiquant la ou les propagation(s) testée(s) ainsi que les tickets associés.

5.2.4 Résumé

Par le jeu des connexions possibles d'un élément à un autre, on s'aperçoit de la facilité avec laquelle il devient possible de naviguer du matériel (dépôt) au fonctionnel (tickets), du fonctionnel au qualitatif (intégration continue), et vice-versa. Les solutions logicielles récentes ont d'ailleurs pour la plupart intégré ce besoin, et proposent divers mécanismes – plugins, URL fixes, etc. – permettant d'assurer l'interopérabilité du mécanisme avec l'extérieur.

5.3 Synthèse

Les interactions possibles entre les techniques étudiées révèlent donc un potentiel considérable, modifiant fondamentalement la vision classique du projet. Elles permettent non seulement de comprendre mais d'expliquer les évolutions successives qu'il connaît, et ce indépendamment de la sphère d'appartenance des différents intervenants. La simplicité des outils, leur utilisation en temps réel offrent un terrain idéal à une communication rapide et efficace du cœur des équipes jusqu'à l'utilisateur final. Le gain de temps pour la rédaction de la documentation s'avère lui aussi considérable : l'historisation des événements rend possible l'automatisation de travaux tels que la réalisation des notes de version (*release notes*), les rapports d'incident, ou encore le suivi de l'activité.

Qu'on ne s'y trompe pas : la configuration d'un complexe de ce type demeure, au contraire de son utilisation, une tâche délicate ne pouvant être menée à la légère. Qu'il s'agisse de la formation des équipes – qui, à tous les niveaux, doivent adapter leur comportement en conséquence –, des coûts supplémentaires en termes de ressources matérielles – un serveur pour l'intégration, un pour les dépôt et un pour les tickets – ou encore de l'entretien nécessaire – mise à jour des projets, changement dans la politique d'intégration –, le travail à réaliser pour parvenir à un environnement idéalement configuré reste un facteur bloquant pour de nombreuses structures.

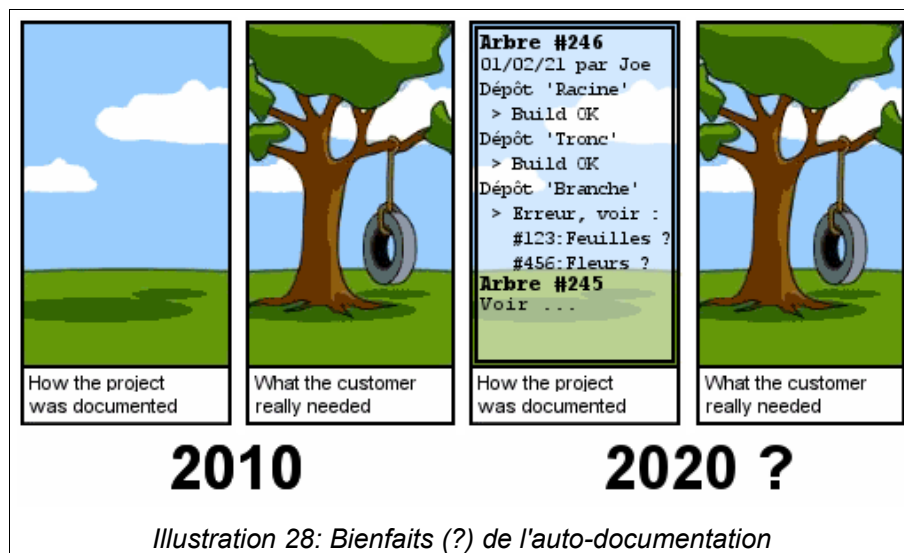
6. Conclusion

En substance, que peut-on retenir de la gestion de projet à trois tiers ?

Encore jeune, elle séduit un nombre croissant d'organismes désireux de revoir en profondeur leur façon de gérer les projets informatiques. Les apports considérables qu'elle crée en termes de sécurité (test standardisés, archivage), de communication (qualification des anomalies, documentation) ou d'accessibilité (interfaces simples, nombreux médias) justifient sans doute les investissements conséquents qu'elle requiert : en matériel, d'une part, et en compétences de l'autre. L'intérêt qu'elle génère pour d'importants groupes (tels que Oracle pour Hudson ou Apache pour Subversion) tend d'ailleurs à confirmer que son fonctionnement s'avère suffisamment mature pour justifier son déploiement au sein de structures sensibles.

Mais s'agit-il pour autant d'une technique éprouvée et formelle ? Si les travaux récents en matière de programmation agile ou de qualité logicielle laissent à penser que oui, on se gardera d'effectuer un jugement hâtif sur la question. En effet, la participation considérable des utilisateurs finaux dans les solutions logicielles utilisées, l'omniprésence des plugins au sein de ces dernières ou encore l'arrivée ponctuelle d'outils novateurs (par exemple, Git) sur le marché laissent à penser que d'importantes évolutions pourraient prochainement secouer la conception que nous en avons aujourd'hui.

Enfin, on restera sur une morale trop souvent vérifiée en matière de SI : les outils ne sont rien sans l'usage qu'on en fait. Si la gestion de projet trois tiers, dont nous avons tenté une description dans ce document, s'avère généralement un bienfait considérable pour les structures qui l'adoptent, son usage, et seul son usage, décide des effets qu'elle entraîne. Une propagation non-commentée, un ticket imprécis et dopé aux fautes de grammaire ou encore une intégration lancée au petit bonheur la chance ne présentent aucun intérêt dans un contexte de gestion à trois tiers. La succès de ces outils dépendra en premier lieu des pratiques que l'ensemble des acteurs du secteur désigneront et surtout prouveront indispensables à leur fonctionnement optimal.



7. Annexes

7.1 Abréviations

COPIL	Comité de pilotage
COPRO	Comité de projet
IHM	Interface homme-machine
IT	Information technology <i>Technologie de l'information, informatique</i>
MOA	Maitrise d'ouvrage
MOE	Maitrise d'œuvre
SVN	Subversion

7.2 Bibliographie

- 1 Wikipedia**
« Gestion de projet » du 5 mai 2011
http://fr.wikipedia.org/wiki/Gestion_de_projet
- 2 Le journal du net**
« Gestion de projet informatique : définition, outils et retours d'expérience »
<http://www.journaldunet.com/solutions/dsi/projets-informatiques/>
- 3 Dictionnaire Larousse**
<http://www.larousse.fr/dictionnaires/francais/%C3%A9tiquette>
- 4 Dictionnaire Larousse**
<http://www.larousse.fr/dictionnaires/francais/version>
- 5 Dictionnaire Larousse**
<http://www.larousse.fr/dictionnaires/francais/int%C3%A9gration>
- 6 Dictionnaire Larousse**
<http://www.larousse.fr/dictionnaires/francais/continu>
- 7 Site des transports en Île-de-France**
Conditions d'utilisation du ticket T+
<http://www.transport-idf.com/frontal?controller=TitresTransport&titre=billet>
- 8 Commission Nationale Informatique & Libertés**
La traçabilité des déplacements
<http://www.cnil.fr/en-savoir-plus/fiches-pratiques/fiche/article/la-tracabilite-des-deplacements/>
- 9 Wikipedia**
« Issue tracking system » du 15 mai 2011 (traduit de l'anglais)
http://en.wikipedia.org/wiki/Ticket_%28IT_support%29
- 10 Internet Marketing Definitions**
« Support ticket » (traduit de l'anglais)
<http://www.internetmarketingdefinitions.com/SupportTicket>
- 11 Wikipedia**
« Comparison of issue tracking systems », § « General », du 15 mai 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems
- 12 Wikipedia**
« Comparison of issue tracking systems », § « Input interfaces », du 15 mai 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems

- 13 Wikipedia**
« Comparison of issue tracking systems », § « Notification interfaces », du 15 mai 2011
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems
- 14 Wikipedia**
« Comparison of issue tracking systems », § « Authentication methods », du 15 mai 2011
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems
- 15 Wikipedia**
« Issue management », § « Workflow », du 15 mai 2011
http://en.wikipedia.org/wiki/Issue_management
- 16 Wikipedia**
« Issue tracking system », § « Product or software development », du 15 mai 2011
http://en.wikipedia.org/wiki/Issue_tracking_system
- 17 Wikipedia**
« Issue management », § « Product or software development », du 15 mai 2011
http://en.wikipedia.org/wiki/Issue_management
- 18 Wikipedia**
« Issue tracking system », § « Architecture », du 15 mai 2011 (traduit de l'anglais)
http://en.wikipedia.org/wiki/Issue_tracking_system
- 19 Search CRM**
« Trouble ticket », C.G. VERHOOG
<http://searchcrm.techtarget.com/definition/trouble-ticket>
- 20 Tableau de bord de gestion**
« Mesurer la performance pour un pilotage pro-actif », Alain FERNANDEZ
http://www.nodesway.com/tableau_de_bord/mesurer_la_performance.htm
- 21 Qualité online**
« Définition pour le management de la qualité : "reporting" »
<http://www.qualiteonline.com/glossaire-R-321-def.html>
- 22 Piloter**
« Qu'est ce que le reporting ? », Alain FERNANDEZ
<http://www.piloter.org/business-intelligence/reporting.htm>
- 23 Atlassian Software Systems**
Site officiel : <http://www.atlassian.com/>
- 24 Wikipedia**
« JIRA » du 21 mai 2011 (traduit de l'anglais)
<http://en.wikipedia.org/wiki/JIRA>
- 25 Atlassian Software Systems**
« JIRA », « Pricing » (interprétation)
<http://www.atlassian.com/software/jira/pricing.jsp>
- 26 Atlassian Software Systems**
« Customers »
<http://www.atlassian.com/about/customers.jsp>
- 27 Wikipedia**
« Comparison of issue-tracking systems » du 21 mai 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_issue_tracking_systems
- 28 Atlassian Software Systems**
« Allowing OAuth Access » du 21 mai 2011
<http://confluence.atlassian.com/display/JIRA043/Allowing+OAuth+Access>
- 29 Atlassian Software Systems**
« Configuring the LDAP Connection Pool » du 21 mai 2011
<http://confluence.atlassian.com/display/JIRA043/Configuring+the+LDAP+Connection+Pool>

- 30 **Office Québécois de la langue Française**
« Version (informatique) », définition de l'office de la langue française
<http://www.granddictionnaire.com>
- 31 **Wikipedia**
« Software Versioning » du 4 juin 2011
http://en.wikipedia.org/wiki/Software_versioning
- 32 **Search Oracle**
« Repository » du 4 juin 2011 (traduit de l'anglais)
<http://searchoracle.techtarget.com/definition/repository>
- 33 **Wikipedia**
« Comparison of revision control software » du 4 juin 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- 34 **Wikipedia**
« Comparison of revision control software », § « Features » du 5 juin 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- 35 **Wikipedia**
« Comparison of revision control software », § « Technical information » du 5 juin 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- 36 **Wikipedia**
« Comparison of revision control software », § « Advanced features » du 5 juin 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_revision_control_software
- 37 **Wikipedia**
« Branching (software) » du 11 juin 2011 (traduit de l'anglais)
http://en.wikipedia.org/wiki/Branching_%28software%29
- 38 **Wikipedia**
« Fork (software development) » du 11 juin 2011
http://en.wikipedia.org/wiki/Fork_%28software_development%29
- 39 **Wikipedia**
« Apache Subversion » du 11 juin 2011
http://en.wikipedia.org/wiki/Apache_Subversion
- 40 **Apache Subversion**
« FAQ » du 11 juin 2011
<http://subversion.apache.org/faq.html>
- 41 **Développez**
« Qu'est ce qui ne vas plus avec Subversion ? »
<http://www.developpez.com/actu/29731/Qu-est-qui-ne-va-plus-avec-Subversion-Le-systeme-de-gestion-de-versions-est-de-plus-en-plus-decrite-et-remplace-par-d-autres-outils/>
- 42 **Source Forge**
« Notepad++ »
<http://sourceforge.net/projects/notepad-plus/develop>
- 43 **SVN Book**
« Version control with Subversion »
<http://svnbook.red-bean.com/nightly/fr/index.html>
- 44 **Wikipedia**
« Source Code Control System » du 12 juin 2011
http://fr.wikipedia.org/wiki/Source_Code_Control_System
- 45 **Javalobby**
« The Changing Definition of Continuous Integration », Eric MINICK
<http://java.dzone.com/articles/changing-definition-continuous>
- 46 **Martin Fowler**
« Continuous Integration », Martin FOWLER (traduit de l'anglais)
<http://www.martinfowler.com/articles/continuousIntegration.html>

- 47 **Wikipedia**
« Continuous Integration » du 12 juin 2011 (traduit de l'anglais)
http://en.wikipedia.org/wiki/Continuous_integration
- 48 **Cunningham & Cunningham, Inc.**
« Integration Hell », Ward CUNNINGHAM
<http://c2.com/cgi/wiki?IntegrationHell>
- 49 **Wikipedia**
« Manifeste Agile » du 12 juin 2011
http://fr.wikipedia.org/wiki/Manifeste_agile
- 50 **Martin Fowler**
« Continuous Integration », § « Maintain a Single Source Repository », Martin FOWLER
<http://www.martinfowler.com/articles/continuousIntegration.html#MaintainASingleSourceRepository>
- 51 **The Agile Manifesto**
Kent BECK, Mike BEEDLE, Arie van BENNEKUM, Alistair COCKBURN, Ward CUNNINGHAM, Martin FOWLER, James GRENNING, Jim HIGHSMITH, Andrew HUNT, Ron JEFFRIES, Jon KERN, Brian MARICK, Robert C. MARTIN, Steve MELLOR, Ken SCHWABER, Jeff SUTHERLAND & Dave THOMAS
http://www.hristov.com/andrey/fht-stuttgart/The_Agile_Manifesto_SDMagazine.pdf
- 52 **Martin Fowler**
« Continuous Integration », Martin FOWLER (traduit de l'anglais)
<http://www.martinfowler.com/articles/continuousIntegration.html>
- 53 **Wikipedia**
« Software Build » du 13 juin 2011 (traduit de l'anglais)
http://en.wikipedia.org/wiki/Software_build
- 54 **Wikipedia**
« Comparison of Continuous Integration Software » du 13 juin 2011 (interprétation)
http://en.wikipedia.org/wiki/Comparison_of_Continuous_Integration_Software
- 55 **Agile Alliance**
« Test-Driven Development », § « What is Test-Driven Development ? », Christoph STEINDL
<http://cf.agilealliance.org/articles/system/article/file/1423/file.pdf>
- 56 **Real Search Group**
« Testing Overview and Black-Box Testing Techniques », Lauren WILLIAMS
<http://agile.csc.ncsu.edu/SEMaterials/BlackBox.pdf>
- 57 **Wikipedia**
« xUnit » du 13 juin 2011
<http://en.wikipedia.org/wiki/XUnit>
- 58 **Mock Objects**
« A Brief History of Mock Objects », Steve FREEMAN
<http://www.mockobjects.com/2009/09/brief-history-of-mock-objects.html>
- 59 **Martin Fowler**
« Continuous Integration », § « Keep the Build Fast », Martin FOWLER
<http://www.martinfowler.com/articles/continuousIntegration.html#KeepTheBuildFast>
- 60 **Martin Fowler**
« Continuous Integration », § « Everyone Can See Whats Happening », Martin FOWLER
<http://www.martinfowler.com/articles/continuousIntegration.html#EveryoneCanSeeWhatsHappening>
- 61 **Wikipedia**
« Comparison of Continuous Integration Software » du 18 juin 2011
http://en.wikipedia.org/wiki/Comparison_of_Continuous_Integration_Software
- 62 **Methods & Tools**
« Hudson – Your Escape from Integration Hell », Simon WIEST (traduit de l'anglais)
<http://www.methodsandtools.com/tools/tools.php?hudson>

- 63 Wikipedia**
« Jenkins (software) » du 18 juin 2011
http://en.wikipedia.org/wiki/Jenkins_%28software%29
- 64 Wikipedia**
« Hudson (software) » du 18 juin 2011
http://en.wikipedia.org/wiki/Hudson_%28software%29
- 65 Hudson Wiki**
« Meet Hudson », § « What is Hudson ? » du 18 juin 2011
<http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson>
- 66 Hudson Wiki**
« Meet Hudson », § « Features » du 18 juin 2011
<http://wiki.hudson-ci.org/display/HUDSON/Meet+Hudson>
- 67 SourceForge**
« PMD »
<http://pmd.sourceforge.net/>
- 68 SourceForge**
« Checkstyle »
<http://checkstyle.sourceforge.net/>
- 69 SourceForge**
« Findbugs »
<http://findbugs.sourceforge.net/>
- 70 Wikipedia**
« Merise (méthode informatique) » du 18 juin 2011
http://fr.wikipedia.org/wiki/Merise_%28informatique%29
- 71 Wikipedia**
« Manifeste Agile » du 19 juin 2011
http://fr.wikipedia.org/wiki/Manifeste_agile