Bien chez soi, bien moins cher

Mise en place d'une plateforme de vente à distance pour CONFORAMA

Témoignage de l'industrialisation des développements PHP autour d'un projet de site marchand

Groupe CONFORAMA – Chiffres clés

- Acteur majeur de l'équipement de la maison à prix discount en Europe
- Une offre complète de meubles, d'articles de décoration, d'électroménager et d'électronique de loisirs
- Une disponibilité des produits via les magasins et Internet
- CA 2010 : 3 092 Millions €
- Conforama est présent dans 7 pays : France, Espagne, Portugal, Italie, Croatie, Luxembourg,et Suisse
- Effectif : 13 400
- 251 magasins dans le monde, dont :
  - France : 198
  - International : 53 magasins

Groupe CONFORAMA – Positionnement internet

- Objectif : 10% du chiffre d'affaires en 2013
- Site français de référence pour l'équipement de la maison
  - 2 clients sur 3 visitent notre site Internet avant d'aller faire un achat en magasin
- Un large accès à l'ensemble de notre offre, partout en France
- Groupe Conforama : 5 millions de visiteurs par mois
- Difficultés aujourd'hui
  - Coûts important de la plateforme actuelle (WebSphere Commerce)
  - Faible réactivité des développements car externalisés
  - Problème de logistique d'expédition car logique de « retail » fortement ancrée chef CONFORAMA
- Projet : confié à LMDV la création et la gestion d'une nouvelle plateforme de vente à distance pour CONFORAMA

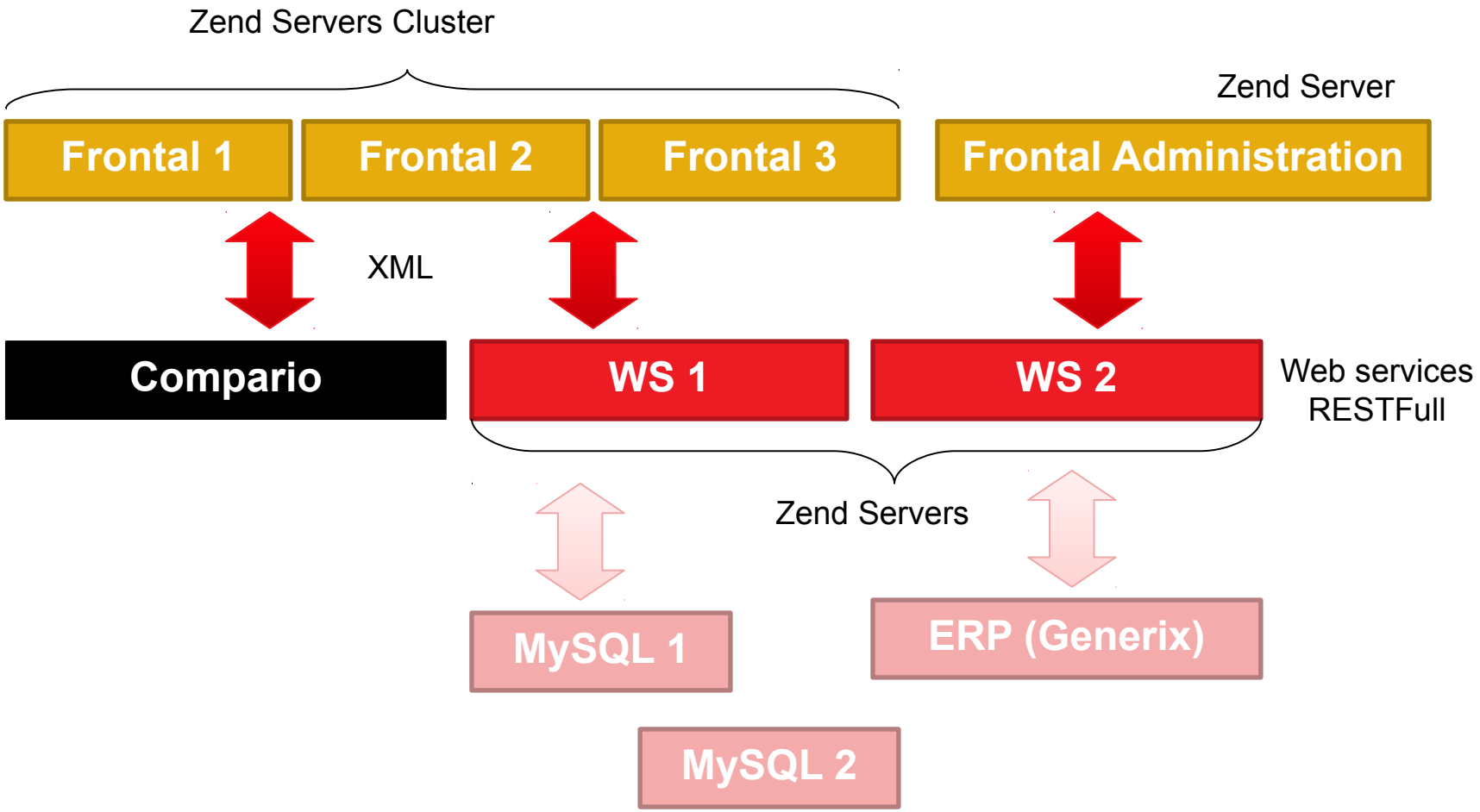
Nouvelle plateforme de VAD

- Rôle de La Maison de Valérie
  - Développer la nouvelle plateforme web
  - Prendre en charge la logistiques d'expédition
  - Ouvrir à la vente multi-canal : web, téléphone, courrier, et magasins
  - Mettre en place un pôle de compétence PHP pour développer la plateforme e-commerce en interne

Le pôle de compétences PHP

- Objectifs du pôle
  - Augmenter la réactivité des développements
  - Amélioré la robustesse et la disponibilité du site
  - Favoriser l'évolution et la faciliter la maintenance
  - Améliorer la qualité des applications
  - Internationaliser la plateforme
  - Développer des applications PHP ou sites web annexes
- Nécessité d'industrialiser les développements
- Constitution d'une équipe de développeurs
  - Aujourd'hui 6 développeur spécialistes de Zend Framework

Architecture du projet



Grandes lignes de l'industrialisation

- Le projet et l'équipe
  - Mise en place de la méthode Scrum
- Le code
  - Solution Zend
  - L'aventure GIT
  - Doctrine ORM
- La qualité
  - Convention de codage
  - Tests statiques
  - Tests unitaires
- Intégration continue
  - Déploiement automatisé via PHING

Méthode SCRUM

Le projet et l’équipe

Scrum – Ce qui a été mis en place

- Scrum meeting
  - Tous les jours 9h – 9h20
  - Echange sur le travaux de la veille et ceux prévus pour aujourd'hui
  - Remontée au scrum master des difficultés techniques
  - Partage sur l'état du code disponible sur le dépôt distant
- Sprint et stories
  - Découpage du projet en tâches unitaires
  - Regroupement des tâche en sprint de 1 à 2 semaines
  - Tenu d'un backlog pour suivre l'avancement des tâches
  - Sprint review et sprint launch à chaque charnière entre sprint

Scrum – Conclusions -1

- Cohésion équipe - projet
  - Les notions de sprint et de story permettent de focaliser l'équipe sur les objectifs à court terme.
  - Le meeting permet de renforcer la cohésion de l'équipe et de faire circuler l'information
- Organisation du travail
  - Travailler en sprint peut faire perdre de vue l'avancement globale du projet
  - Travailler le contenu des release le plus en amont possible.
  - L'influence d'éléments externes à l'équipe (métiers, prestataires ou services externalisés) met en péril le bon déroulement des sprints
- Backlog
  - Un backlog peut masquer les besoins de spécifications
  - Pas d'outil dédié.

Scrum – Conclusions - 2

- Le scrum master
  - Travail à temps plein si le projet est long et l'équipe importante
  - L'organisation du travail prend souvent le pas sur le reporting du travail effectué
  - Pas d'outils dédié
- Et demain chez Conforama ?
  - Améliorer le reporting
  - Améliorer l'implication des éléments extérieurs à l'équipe scrum : Problème de la sous-traitance du design, intégration dans la démarche globale de la DSI (bug tracking, partage de connaissance..)
  - Trouver ou développer un outil de gestion Scrum et plus particulièrement du backlog.

Solution Zend

Le CODE

Solution Zend – Zend Studio

- Usages
  - Convention de codage
  - Tout le nécessaire pour coder en PHP / Zend Framework
  - Intégration de PHPUnit, Zend Framework
- Difficultés
  - Gestion de dépôt GIT dans version 8
  - Intégration de Code Sniffer, ou autre outil du même type

Solution Zend– Zend Server

- Usages
  - Zend Page Cache
  - Zend cache
  - Job Queue

Solution Zend - Zend Cluster manager

- Usage
  - Session clustering
  - Gestion du cluster via l'application (déploiement à chaud)

L’aventure GIT

Le code

L’aventure GIT – Valeurs ajoutées

- Dépôt décentralisé
  - Travail en mode déconnecté : train, nuit, week-end, panne réseau
  - Liberté du développeur
- Agilités des branches
  - Refonte du code en toute tranquillité d’esprit
  - Gestion du code délirant en supprimant aisément une branche entière.
- Mode Agile et industrialisé
  - Scrum : sprint, stories et release on des réalités dans GIT : tag, branches…
  - Intégration continue : Le dépôt distant de GIT sert de base à des tests automatisés via PHPunderControl
  - Liaison potentielle avec PHING pour faciliter les déploiements en production ?
  - Combinaison avec le BugTracker pour le suivi de bug ?

L’aventure GIT – Difficultés rencontrées

- Un système trop flexible
  - Mise en place de process d’utilisation
- Difficulté d’acquisition par les développeurs
  - Outil peu répandu dans la communauté des développeurs
  - Pas d’intégration de GIT au sein de l’IDE => outil supplémentaire
- Création du poste de Sources Managers
  - Responsable du bon fonctionnement de GIT
  - Doit former les nouveaux membres de l’équipe
  - Responsable du code déployé en production et du code partagé sur le dépôt distant

L’aventure GIT – Conclusions

- Mise en place longue et douloureuse (6 mois)
  - Trouver le bon process d’utilisation
  - Trouver les bons outils
  - Former les développeurs
  - Former les source-managers
- Un résultat éclatant
  - Performance indiscutable !
  - Liberté du développement
  - Fort potentiel d’intégration dans la chaine de production

Doctrine ORM (1.2)

Le code

Doctrine ORM – Objectifs

- Oublier la base de données
- Simplifier nos modèles
- Prendre en charge la migration de la base de façon automatisée
- Profiter des caches et des gains de performances
- Utilisation de templates permettant de factoriser des comportements de modèle.

Doctrine ORM – Conclusion

- On ne peut plus s'en passer !
  - Relations et contraintes de clé étrangère simplissime
  - Maintenance de la base de données simplifiée
  - Template / behaviors simples et puissants
- Code sensible
  - Surcharge de méthodes trop souvent permise et source importante de bugs.
- Documentation un peu légère
- Prise en mains délicate
  - Contrainte de clé étrangère non respectée sur des sauvegardes de données en cascade.
  - Apprentissage un peu long
  - Difficulté d'intégration à Zend Framework en raison des incompatibilité d'autoloading des classes.

Convention de codage

La qualité

Convention de codage

- Pourquoi standardiser le code
  - Pour permettre à tout développeur de prendre en main rapidement ce qu'il n'a pas codé
  - Pour assurer la compatibilité du code sur différents OS (espace, tabulation, retour à la ligne…)
  - Pour bâtir un PHPDoc complet
  - Pour rendre le code plus clair : exemple : les propriétés privées ou protégées son préfixées par un underscore
- Standardiser quoi ?
  - La structure de répertoire => Suivi de la norme MVC de Zend (sauf pour le design)
  - Les code PHP => norme GN de SQLI simplifiée
  - Demain le JS, l'HTML et le CSS..

Convention de codage – Conclusion

- Acquisition facile par l'équipe si elle sait pourquoi elle doit le faire
- Bon challenge pour l'équipe de limiter les entorses à la norme
- Le code est effectivement bien plus clair et facile à appréhender
- Manque un validateur de ces normes dans l'IDE (PHPCS dans Zend studio ?)

Tests statiques

La qualité

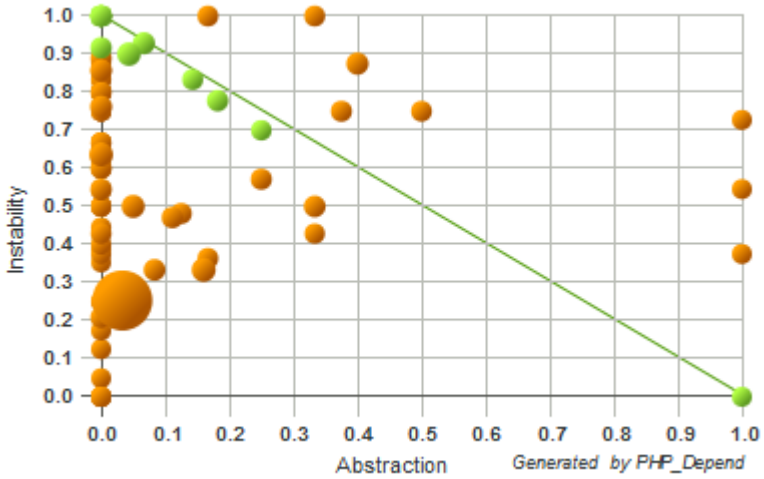
Tests statiques - PHP Code sniffer

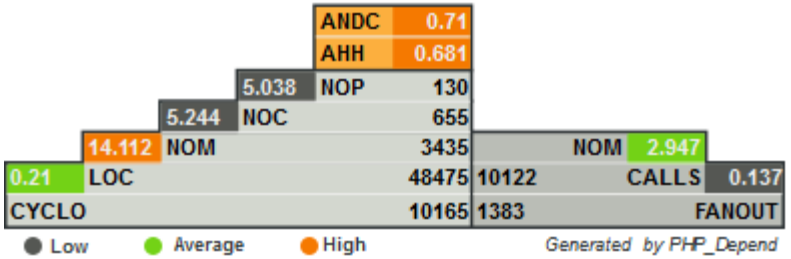
- Rôle et implémentation
  - Pour valider le suivi des conventions de codage
  - PHPCS est lancé quotidiennement sur le code du dépôt distant de GIT
- Exemple
  - Code du 18/11/11 : 3285 erreurs sur 238 fichiers PHP analysés…. (rush de fin de projet)
  - Meilleur résultat obtenu : moins de 300 erreurs
- Analyses
  - Compte rendu hebdomadaire en réunion technique
  - Arbitrage sur les exceptions à la règles
  - Amélioration des règles : Trouver l'équilibre entre norme et fluidité de développement
- Demain
  - Disposer de PHPCS sur les postes des développer pour partager du code plus propre
  - Le must : intégrer les conventions à Zend Studio.

Tests statiques – PHP Depends

- Pourquoi ?
  - Pour maitriser la complexité du code
  - Favoriser la maintenance et la pérennité du code en maitrisant le niveau de corrélation des classes
- Comment l'utilise-t-on aujourd'hui
  - Analyse quotidienne sur le code du dépôt GIT distant
  - Pas encore analysé en réunion technique hebdomadaire.
- Demain ?
  - Analyse plus poussée et partagée avec l'équipe de développement
  - Mise en place de refonte de code pour améliorer la complexité.

Tests statiques – PHP Depends - Exemple





- Que voit-on ?
  - Beaucoup de lignes de code par méthode (LOC / NOM = 14.112)
  - Héritage fort (ANDC = 0.71 et ANH = 0.68) (usage des frameworks ?)
  - Faible niveau d'abstraction : voir si des interfaces se dessinent…

Tests statiques - Conclusions

- On exploite surtout PHPCS et le résultat est très positif il mérite d'être mieux intégrer à l'environnement de travail
- PHP Depends doit être mieux pris en compte dans la conception
- D'autres outils sont en place et demandent encore à être exploités :
- Demain la cellule de QA devra analyser l'ensemble des résultats de ces outils pour donner un autre éclairage sur le code.

Tests unitaires - Objectifs

- Objectifs
  - Limiter les régressions sur le code notamment après un « git merge »
  - Aider au développement des classes profondes sans visibilité directe sur le résultat d'une page web. (TDD)
  - Détecter les bugs avant qu'il n'apparaissent en production
  - Tester le système avant la mise en production, au cours du processus de déploiement.
- Outils utilisés :
  - PHPUnit : test de base de donnée et unitaire sur le classes
  - Zend_Test : validation des résultat des controlleurs
  - Selenium (en cours d'acquisition par l'équipe) : validation des scénarii incontournables du site : connexion, tunnel de commande…

Test unitaires – Exemple

- Nombre de test en place : environ 200 aujourd'hui
- Test de « controllers » via Zend_Test
- Code en relation avec l'ERP réalisée principalement en TDD

Test unitaire – Conclusions

- Avoir des classes testables…. Pas si facile.
- Prise de recul important sur le code : pratique extrêmement vertueuse.
- Vrai valeur ajouté sur la robustesse du code
- Automatisation aisée.
- PHPUnit gourmand.

Intégration continue

Intégration continue – Objectifs et outils

- Déployer rapidement l'application sur l'environnement de production
- Déployer l'application sur des environnements multiples : production, stagging, test…
- Automatiser :
  - Les copies de fichiers
  - Le paramétrages de l'application spécifique à l'environnement cible
  - Les tests au cours du déploiement
  - La modification de la structure de la base de données
  - Le rollback en cas de problèmes
- Déployer plus souvent
  - Release à chaque fin de sprint.
  - Les tests sur le code en cours de développement
- Outils
  - PHPUnderControl sur le dépôt de code distant
  - PHING pour rédiger des script de déploiement automatiser

Intégration continue – PHPUnderControl

- Tests effectué sur le code en développement :
  - PHPCS : analyse du suivi des conventions de codage
  - PHP Depends : analyse statique de la complexité du code
  - PHPCP, PHPMD : pas encore pris en compte dans l'analyse
  - Tests Unitaire PHPUnit : Stratégie pas encore définie
- Conclusions
  - Outils pas encore mûr. Complexe à mettre en œuvre
  - Des efforts de notre côté pour mieux intégrer l'outil dans nos process de développement
  - Demain : trouver un outil mieu intégré: tests, génération d'archive pour les releases, relation avec le backlog et / ou le bugtracker pour un meilleur suivi de la qualité de code.

Intégration continue – PHING

- Utilisation aujourd'hui : Déploiement du code sur l'ensemble du cluster de production
- Demain
  - Lancement des tests au cours du déploiement : gestion du rollback automatisé
  - Intégration des propriété de Doctrine pour upgrader ou down grader la structure de la base de données à chaud.
  - Création d'un fichier d'historisation du code déployé
  - Pilotage du cluster Zend pour déployer à chaud sans interruption de service (mise à jour progressive des nœuds)

Conclusions générales

- Tous ces aspects nous apporté
  - Rigueur du code et du travail des développeurs en général
  - Accumulation de documentation et savoir-faire autour de notre application
  - Augmentation de compétence des développeurs : autoformation, curiosité, ouverture d'esprit
  - Recul sur le code
  - Productivité des développeur et indicateur de qualité
  - Aisance dans la mise en production grâce à l'intégration continue
- A poursuivre
  - Conception SOLID pour facilité la testabilité du code
  - Meilleur intégration des outils les uns avec les autres pour centraliser l'information
  - Mieux intégrer les métiers à nos process.

On aurait pas su faire sans eux :

- Alterway : conseil en industrialisation
  - Usage de GIT
  - Réflexions sur les test unitaires
- Flavea
  - installation de la plateforme d'intégration continue (PHPUC)
  - Appui au développement de l'interface d'administration de la plateforme.
- Zend

Bien chez soi, bien moins cher

Merci !

Sophie BEAUPUIS – PHP Tour Lille 2011