



Bonnes pratiques



Par l'exemple de



Bonnes pratiques PHP ... Par l'exemple

Objectifs et plan de la présentation

➤ Présenter les pratiques PHP et règles de l'art du développement PHP, avec retour d'expérience

Principe de la présentation

- 8 thèmes abordés
- Pour chaque thème, 3 paragraphes évoqués :



Pourquoi



Comment



Résultats et retours d'expérience

Thèmes abordés

- Thème 1: Documentation technique
- Thème 2: Design pattern Accès aux données (DAO et ORM)
- Thème 3: Design pattern MVC et moteurs de templates
- Thème 4: Performances
- Thème 5: Traduction
- Thème 6: Portabilité
- Thème 7: Sécurité
- Thème 8: Tests unitaires/non régression



Bonnes pratiques PHP ... Par l'exemple

Choix de l'exemple

➤ Un EPR/CRM de 8 ans d'expérience et 800 000 lignes de code:



Couverture fonctionnelle

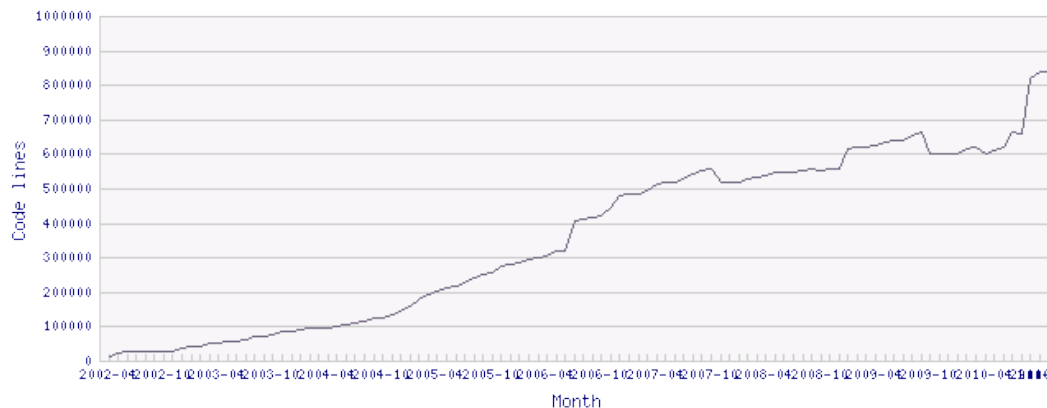
- ERP/CRM (Facturation, Commandes, Produits, Stock, Annuaire, Agenda, Projet, etc...)

Couverture technique

- PHP 4 à 5.3
- MySql 3+ / PostgreSQL 8.3+
- LDAP, RSS, SOAP, ...

Historique

- Début des développements en 2003 (PHP 4 / MySql 3)
- 800 000 lignes de codes (commentaires exclus), une vingtaine de développeurs actifs



Bonnes pratiques PHP ... Par l'exemple

Thèmes 1/8 : Documentation technique

 Pourquoi ?

- Faciliter la lecture de code issus d'autres développeurs
- Etude d'impacts
- Accélérer le développement en bénéficiant d'auto-complétion lors de l'écriture du code.

Bonnes pratiques PHP ... Par l'exemple

Thèmes 1/8 : Documentation technique



Comment ?

- Utilisation d'outils et conventions de documentation (JavaDoc, phpDocumentor, Doxygen)

```
<PHP> - dolibarr/htdocs/commande/class/commande.class.php - Eclipse SDK
File Edit Source Refactor Navigate Search Project Run Window Help

commande.class.php
1 <?php
2 /* Copyright (C) 2004-2010 Laurent Destailleur  <eldy@users.sourceforge.net> */
3
4 /**
5  * \file      htdocs/commande/class/commande.class.php
6  * \ingroup   commande
7  * \brief     Fichier des classes de commandes
8  * \version   $Id: commande.class.php,v 1.54 2010/10/27 22:41:40 eldy Exp $
9  */
10 require_once(DOL_DOCUMENT_ROOT."/core/class/commonobject.class.php");
11
12
13 /**
14  * \class     Commande
15  * \brief     Class to manage customers orders
16  */
17 class Commande extends CommonObject
18 {
19     var $variable1;
20     var $variable2;
21
22     /**
23      * Renvoie la reference de commande suivante non utilisee en fonction du module
24      * de numerotation actif defini dans COMMANDE_ADDON
25      * @param    soc      objet societe
26      * @return   string   reference libre de nouvelle commande
27      */
28     function getNextNumRef($soc)
29     {
30         global $conf;
31
32         $file = $conf->global->COMMANDE_ADDON.".php";
33     }
34 }
```

Conventions Doxygen

Conventions JavaDoc



Bonnes pratiques PHP ... Par l'exemple

Thèmes 1/8 : Documentation technique



Résultats et retour d'expérience

- Plusieurs conventions/outils testés => Le choix final est la convention JavaDoc complétée de tags Doxygen (Entêtes fichier).
- Génération de la documentation faite avec Doxygen.

The screenshot shows a web browser window displaying the Dolibarr CommandeFournisseur Class Reference page. The page is titled "CommandeFournisseur Class Reference" and includes a navigation menu with tabs for Main Page, Related Pages, Modules, Classes, Files, and Directories. The Classes tab is selected, showing a list of class members. The page content includes a description of the class, an inheritance diagram, and a list of public member functions.

CommandeFournisseur Class Reference

Class to manage predefined suppliers products. [More...](#)

▼ Inheritance diagram for CommandeFournisseur:

```
graph BT
    CommonObject --> Commande
    Commande --> CommandeFournisseur
```

List of all members.

Public Member Functions

CommandeFournisseur (\$DB)
Constructeur.
fetch (\$id, \$ref=)
Get object and lines from database.
log (\$user, \$statut, \$date, \$comment=)
Add a line in log table.
valid (\$user)
Validate an order.
getLibStatut (\$mode=0)
Return label of the status of object.
LibStatut (\$statut, \$mode=0)

Bonnes pratiques PHP ... Par l'exemple

Thèmes 2/8 : Design patterns DAO et ORM

 Pourquoi ?

- Définir un cadre de développement homogénéisant la production de code
- Faciliter la lecture de code issus d'autres développeur.
- Permettre l'accélération du développement par génération de code

Thèmes 2/8 : Design patterns DAO et ORM



Comment ?

Martin Fowler a identifié 3 patterns d'accès aux données pour un langage Objet :

- **Le Table And Row Data Gateway**

Une classe par table, voir une classe par ligne de table. La classe ne contient que du code d'accès aux lignes ou colonnes de tables. Le code métier doit alors être ajouté dans d'autres classes. Les classes possèdent juste les méthodes CRUD.

Exemple: Certains Frameworks d'ORM comme **Prado**

- **Le Active Record**

Identique au précédent, mais on se permet d'ajouter aux méthodes CRUD quelques fonctions métiers dans les classes, à conditions que ces fonctions aient un rapport avec la table ou ses lignes.

- **Le Data Mapper**

Les classes représentent les entités du problème et non les données. Il faut donc doubler, tripler... ces classes avec des classes Mapper pour accéder aux données. Plus "puriste" sur le papier car plus proche du métier, ce mode a aussi l'inconvénient d'être plus complexe sur le plan pratique.

Exemple: Certains Frameworks d'ORM comme **Propel**



Bonnes pratiques PHP ... Par l'exemple

Thèmes 2/8 : Design patterns DAO et ORM

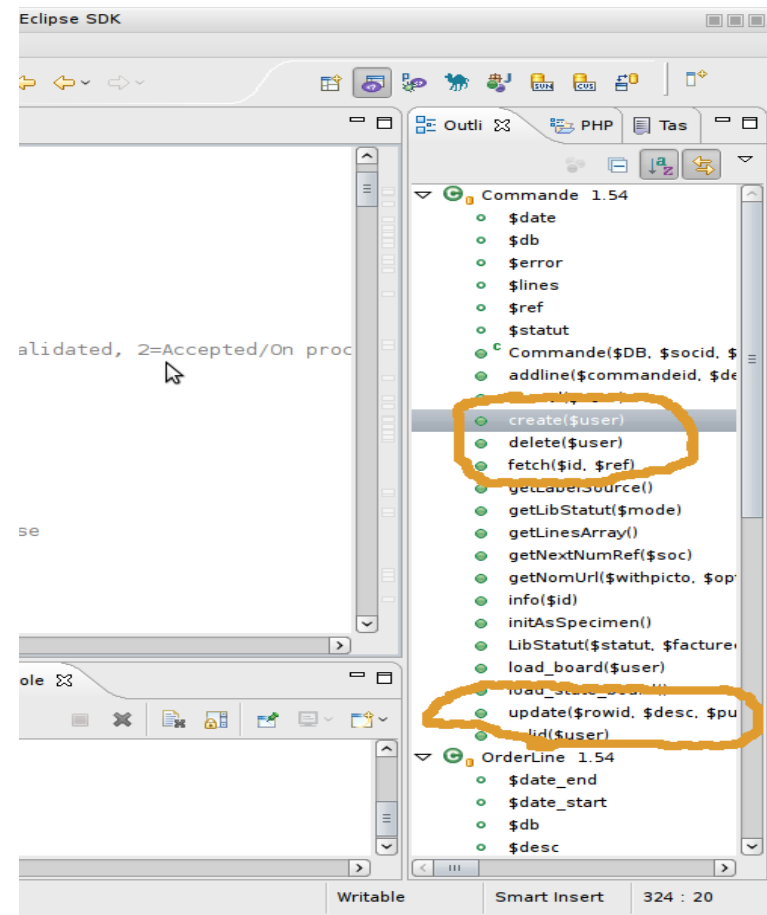


Résultats et retours d'expérience

- Le Pattern **Table And Row Data** est trop limité en terme d'accès aux données.
- Le Pattern **Data Mapper** crée une forte abstraction qui, à l'usage, génère aussi de l'obscurité de code.
- Le Pattern **Active Record** a finalement été généralisé car offrant le meilleur facteur productivité * lisibilité.

Exemple sur la classe Commande =>

Cela a de plus permis le développement d'un script générateur de classe.



Bonnes pratiques PHP ... Par l'exemple

Thèmes 3/8 : Design patterns MVC



Pourquoi ?

- Définir un cadre de développement homogénéisant la production de code
- Faciliter la lecture de code issus d'autres développeurs
- Permettre l'accélération du développement par génération de code
- Permettre la personnalisation du visuel sans compétences PHP



Bonnes pratiques PHP ... Par l'exemple

Thèmes 3/8 : Design patterns MVC



Comment ?

- Le design pattern MVC « by file », « by framework »

Chaque écran possède un fichier de template et un fichier de code actions.

Exemple: La plupart des Frameworks de présentation (**Smarty**, ...)

- Le design pattern MVC « by file », « by PHP »

Identique au précédent, mais le moteur de templates est en fait un simple include d'un fichier **template.tpl.php** qui manipule les variables PHP.

- Le design pattern MVC « by part », « by PHP »

C'est le même fichier qui contient le code action et la présentation. La séparation se fait dans le découpage du fichier (le haut contient le code, le bas la présentation)



Bonnes pratiques PHP ... Par l'exemple

Thèmes 3/8 : Design patterns MVC



Comment ?

Avantage/Inconvénient	MVC by file by Framework	MVC by file by PHP	MVC by part by PHP
Garantie technique d'isolation code/présentation	+	-	-
Lisibilité HTML	+	+	-
Lisibilité Code	-	+/-	+
Évolutivité IHM complexes/Ajax...	-	+	+
Fonctionnalités	+/-	-/+	-/+
Performances	- (120% smarty)	+ (101%)	+ (100%)

Bonnes pratiques PHP ... Par l'exemple

Thèmes 3/8 : Design patterns MVC



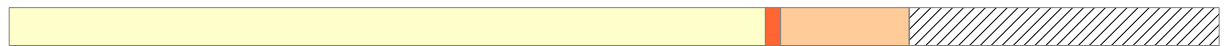
Résultats et retours d'expérience

- Abandon de Smarty :

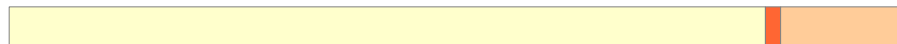
→ La dégradation de maintenabilité sur le code et limitations dues à la complexité des IHM ont été jugés non compensés par le gain sur la maintenabilité de la présentation.

→ Les performances de Smarty très inférieures.

MVC by file by framework
Without cache (Smarty)



MVC by file by framework
With cache (Smarty)



MVC by file by PHP



MVC by part by PHP



- Utilisation du MVC by part by PHP en général, utilisation du MVC by file by PHP sur les portions d'écrans répétitives uniquement.

→ Autonomie très rapide des développeurs



Bonnes pratiques PHP ... Par l'exemple

Thèmes 4/8 : Performances



Pourquoi ?

- Parce que le temps de réponse est le deuxième facteur (après l'ergonomie) de productivité des utilisateurs.
- Parce qu'une application Web (client léger) doit être aussi rapide qu'une application Client-Serveur.
- Parce que ne bénéficiant pas de fonction natives de serveurs applicatifs, les développeurs PHP ne cherchent pas à optimiser l'usage des ressources.



Bonnes pratiques PHP ... Par l'exemple

Thèmes 4/8 : Performances



Comment ?

- Optimisation du côté du navigateur et réseau
- Optimisation du code
- Utilisation de serveur de cache



Bonnes pratiques PHP ... Par l'exemple

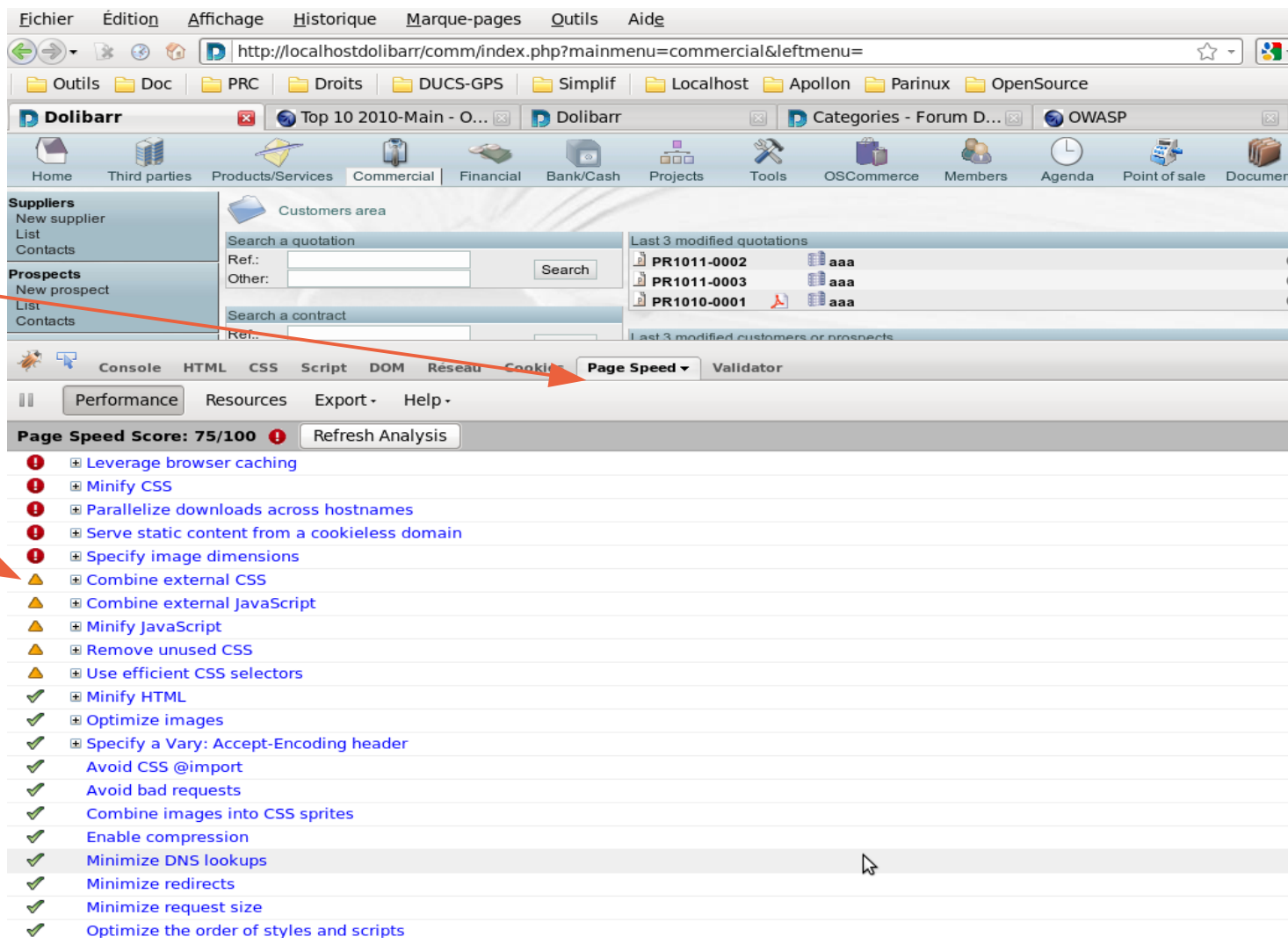
Thèmes 4/8 : Performances

 Comment ? (Optimisation navigateur)

- Utilisation du plugin Firebug + PageSpeed (Google) ou Yspeed (Yahoo)

Un bouton pour activer
l'analyse de la page.

Une liste de résultat
avec conseils correctifs



Fichier Édition Affichage Historique Marque-pages Outils Aide

http://localhostdolibarr/comm/index.php?mainmenu=commercial&leftmenu=

Outils Doc PRC Droits DUCS-GPS Simplif Localhost Apollon Parinux OpenSource

Dolibarr Top 10 2010-Main - O... Dolibarr Categories - Forum D... OWASP

Home Third parties Products/Services Commercial Financial Bank/Cash Projects Tools OSCommerce Members Agenda Point of sale Document

Suppliers
New supplier
List
Contacts

Prospects
New prospect
List
Contacts

Customers area

Search a quotation
Ref.:
Other:

Search

Search a contract
Ref.:

Last 3 modified quotations

PR1011-0002	aaa
PR1011-0003	aaa
PR1010-0001	aaa

Last 3 modified customers or prospects

Console HTML CSS Script DOM Réseau Cookies **Page Speed** Validator

Performance Resources Export - Help -

Page Speed Score: 75/100 Refresh Analysis

- ❗ Leverage browser caching
- ❗ Minify CSS
- ❗ Parallelize downloads across hostnames
- ❗ Serve static content from a cookieless domain
- ❗ Specify image dimensions
- ⚠ Combine external CSS
- ⚠ Combine external JavaScript
- ⚠ Minify JavaScript
- ⚠ Remove unused CSS
- ⚠ Use efficient CSS selectors
- ✅ Minify HTML
- ✅ Optimize images
- ✅ Specify a Vary: Accept-Encoding header
- ✅ Avoid CSS @import
- ✅ Avoid bad requests
- ✅ Combine images into CSS sprites
- ✅ Enable compression
- ✅ Minimize DNS lookups
- ✅ Minimize redirects
- ✅ Minimize request size
- ✅ Optimize the order of styles and scripts

Bonnes pratiques PHP ... Par l'exemple

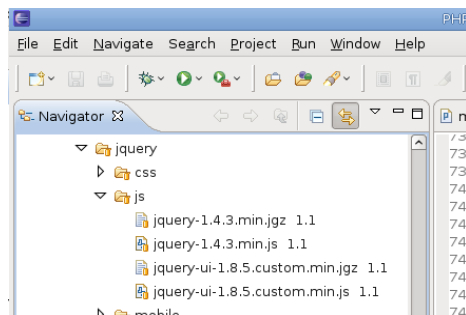
Thèmes 4/8 : Performances



Résultats et retours d'expérience (Optimisation navigateur)

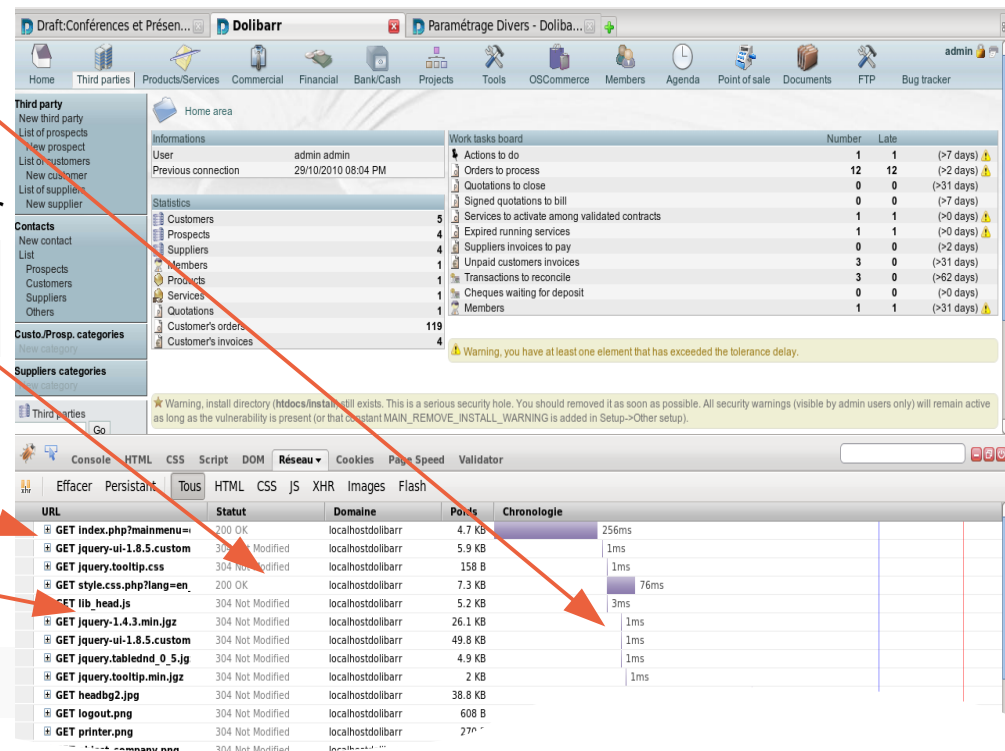
- Modification de l'ordre des js et css augmente le nombre de threads // du navigateur
- Forçage du cache navigateur par directives serveur

```
# Cache for static resources (Enable expiration and set it to one month)
ExpiresActive On
ExpiresByType image/x-icon A2592000
ExpiresByType image/gif A2592000
```
- Fusion des js propres au projet
- Compression gz + Minification js



+ AddType text/javascript .jgz
AddEncoding gzip .jgz

+ \$mini="";\$ext='.js';
if (isset(\$OPTIMIZE)) { \$mini='.min'; \$ext='.jgz'; }
print '<script src="/jquery/js/jquery-1.4.3'.\$mini.\$ext.'"></script>';



- Gain de 20% (dans la vue Réseau de Firebug) sur le rendu navigateur
- Passage de la note PageSpeed de 46/100 à 76/100

Bonnes pratiques PHP ... Par l'exemple

Thèmes 4/8 : Performances



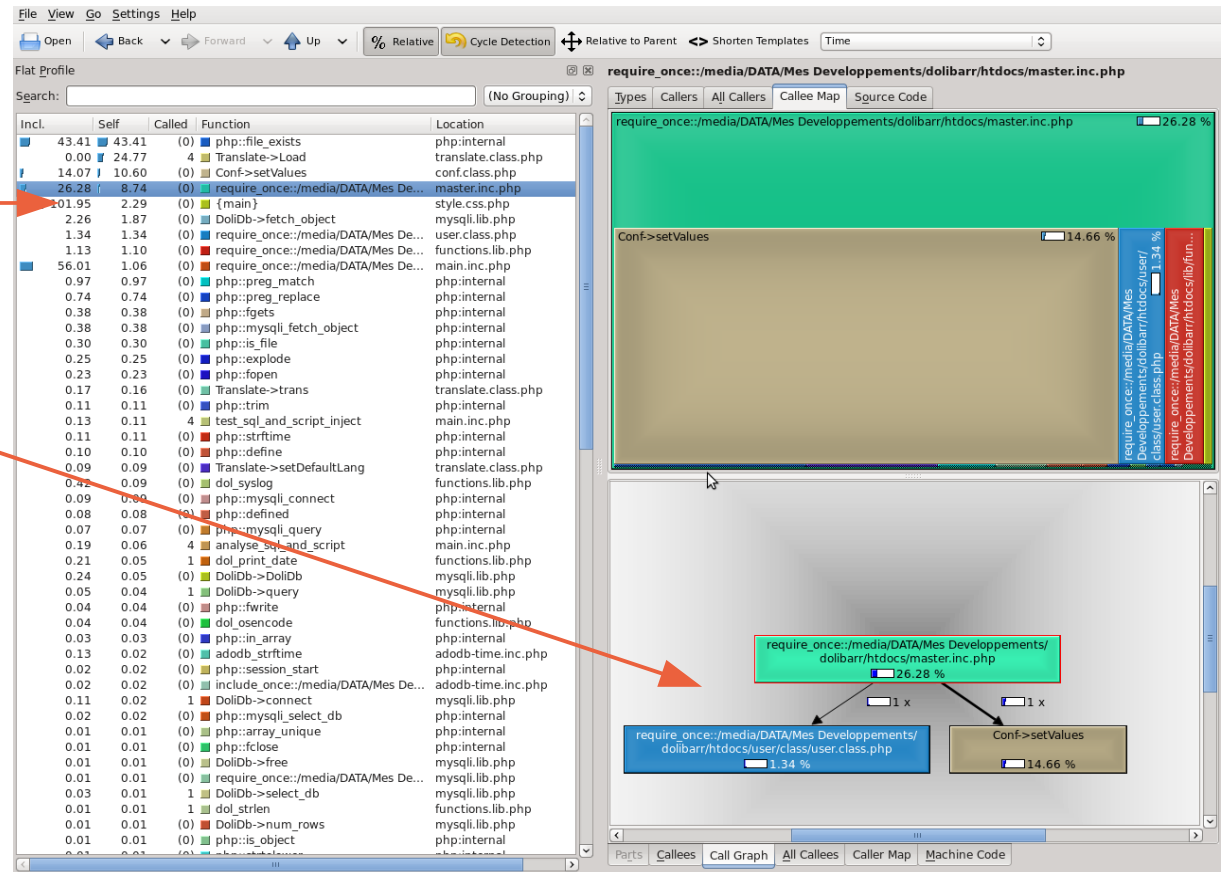
Comment ? (Le coding)

● Utilisation de Xdebug et XcacheGrind

- Activation du mode trace de xdebug dans un fichier cachegrind.
- Analyse du fichier cachegrind avec un lecteur (KcacheGrind ou WinCacheGrind)

Liste des fonctions et temps

Vue descendante dans les fonctions



Bonnes pratiques PHP ... Par l'exemple

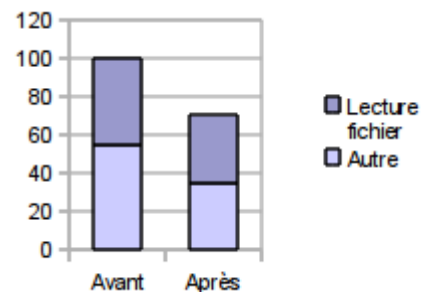
Thèmes 4/8 : Performances



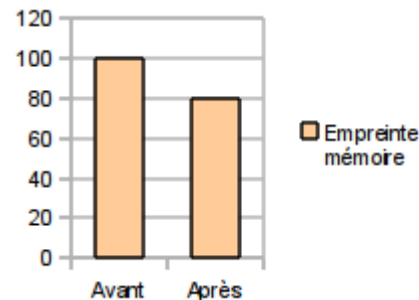
Résultats et retours d'expérience (Le coding)

- Passage du temps de réponse des pages d'un indice 100 à un indice 70
- Réduction de l'empreinte mémoire d'un indice 100 à un indice 80
- Mise en évidence que 50% du temps était consacré à une tâche de lecture des fichiers .lang (Fichier des traductions Dolibarr).

Temps de réponses



Mémoire



Bonnes pratiques PHP ... Par l'exemple

Thèmes 4/8 : Performances



Comment ? (Le serveur de cache)

- Utilisation du serveur MemCache (A ne pas confondre avec le cache OpCode PHP ou le cache des moteurs de templates)
- Utilisation de l'implémentation PHP du client (Memcache ou Memcached)

```
<?php

$memcache = new Memcache;
$memcache->connect('localhost', 11211) or die ("Connexion impossible");

$version = $memcache->getVersion();
echo "Version du serveur : ".$version."<br/>\n";

$tmp_object = new stdClass;
$tmp_object->str_attr = 'test';
$tmp_object->int_attr = 123;

$memcache->set('key', $tmp_object, false, 10) or die ("Echec de la sauvegarde des données sur le serveur");
echo "Les données ont été stockées dans le cache (les données expireront dans 10 secondes)<br/>\n";

$get_result = $memcache->get('key');
echo "Données depuis le cache :<br/>\n";

var_dump($get_result);

?>
```

Bonnes pratiques PHP ... Par l'exemple

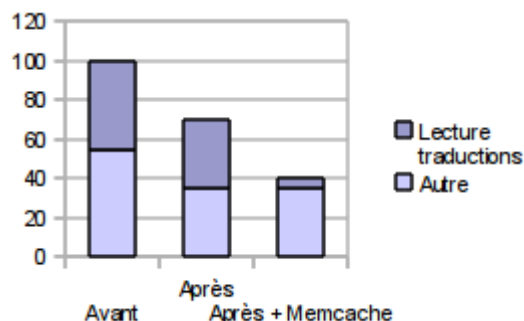
Thèmes 4/8 : Performances



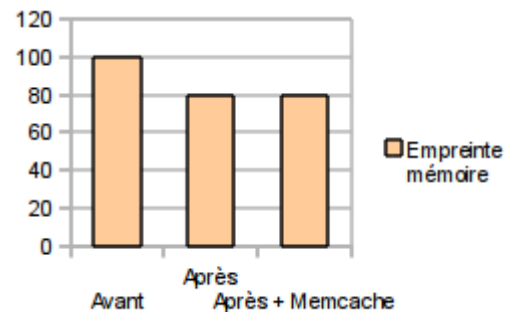
Résultats et retours d'expérience (Le serveur de cache)

- Utiliser sur la seule phase de lecture des fichiers de paramètres et traduction => Remplacement de fonctions d'I/O disques en I/O mémoire.
- Réduction par 10 du temps de la phase de lecture des fichiers traduction, d'où une réduction de presque 50% du temps de traitement.

Temps de réponses



Mémoire



Bonnes pratiques PHP ... Par l'exemple

Thèmes 5/8 : Traductions

 Pourquoi ?

- Permettre une internationalisation et étendre sa base potentielle d'utilisateur.
- Gérer les spécificités locales (utilisateurs en DOMTOM, autres pays francophones, ...)

Thèmes 5/8 : Traductions



Comment ?

Localisation des dates :

- Utilisation de la fonction PHP `setlocale()` + `strftime()`, `mktime()`, `gmtime()` ?
ou
- Utilisation d'un fichier de configuration du format de date ?

Localisation des symboles numériques + monnaies :

- Utilisation de la fonction PHP `setlocale()` + `number_format()` ?
ou
- Utilisation d'un fichier de configuration du format de date ?

Localisation des chaînes de traductions :

- Utilisation des fonctions de conversion des pages codes ?
- Utilisation de `GetText` et le standard `po` ?
ou
- Utilisation d'une méthode maison ?



Bonnes pratiques PHP ... Par l'exemple

Thèmes 5/8 : Traductions



Résultats et retours d'expérience

Localisation des dates :

- Utilisation de la fonction PHP ~~setlocale()~~ + ~~strftime()~~, ~~mktime()~~, ~~gmtime()~~
→ Fonctions dépendant de la configuration système (variable UTC) et PHP (php.ini). Non modifiable par thread mais global (process)..
- Utilisation d'un fichier de configuration du format de date

Localisation des symboles numériques + monnaies :

- Utilisation de la fonction PHP ~~setlocale()~~ + ~~number_format()~~ ?
→ Fonction dépendant de la configuration système (variable LANG) et PHP (php.ini). Non modifiable par thread mais global (par process).
- Utilisation d'un fichier de configuration du format de date ?

Localisation des chaînes de traductions :

- Utilisation des fonctions de conversion des pages codes ?
→ Partir sur du full UTF-8 (Code source, Output HTML, Base de données)
- Utilisation de GetText et le standard po ?
→ GetText dépendant de la configuration système (variable LANG, UTC) et PHP (php.ini). Non modifiable par thread mais global (par process).
→ Standard po se base sur une chaîne source et non une clé.
- Utilisation d'une méthode maison



Bonnes pratiques PHP ... Par l'exemple

Thèmes 6/8 : Portabilité



Pourquoi ?

- Avoir une application indépendante de l'OS, base de données, paramétrage PHP...
- Pouvoir développer sous un OS (Windows, MAC...) et mettre en production sous un autre (Linux, ...) sans surprise.
- 3 grandes problématiques de portabilité :
 - Base de données
 - Système de fichiers
 - Paramétrage PHP



Bonnes pratiques PHP ... Par l'exemple

Thèmes 6/8 : Portabilité



Comment ?

Base de données :

- PDO, DAO autre framework ORM, autre ?

Systèmes de fichiers :

- Rien à faire pour la gestion de / ou \ (Natif PHP)
- Pas de solution native pour une compatibilité du code sur tout systèmes de fichiers (Windows ISO – Linux UTF8)

Paramétrage PHP :

- La gestion des écarts des options PHP (error_reporting, register_global, magic_quotes) doit se faire de manière programmatic.



Bonnes pratiques PHP ... Par l'exemple

Thèmes 6/8 : Portabilité



Résultats et retours d'expérience (Base de données)

Trop de lacunes à PDO, une classe de conversion ordres SQL mysql → base x et de conversion code erreur mysql → code générique a été développée.

Couche de portabilité Avantage/Inconvénient	PDO	Dolibarr	Autre frameworks
Portabilité ordres DDL (insert, update, select, delete)	Oui	Oui	?
Portabilité ordres DML (create, truncate, ...)	Non	Oui	?
Portabilité des codes erreurs	Incomplète	Oui	?
Gestion des transactions imbriquées	Non	Oui	?

Bonnes pratiques PHP ... Par l'exemple

Thèmes 6/8 : Portabilité



Résultats et retours d'expérience (Système de fichier)

- Les fonctions de manipulation fichiers de PHP ne tiennent pas compte de la page de code du système de fichier (FAT, NTFS, ext3, ext4, ...).

Exemple :

```
$monfichier='fichier accentué.txt'; fopen($monfichier, 'w') ;
```

ne stockera pas correctement le fichier si le source manipule des données UTF-8 et que le système de fichier est en ISO (et vis-versa).

→ Toutes les fonctions fichiers ont donc été surchargées :

```
$monfichier='fichier accentué.txt'; dol_fopen($monfichier, 'w') ;
```

```
function dol_fopen($monfichier) { return fopen(dol_osencode($dir)); }
```

```
function dol_osencode($str) {
```

```
    $tmp=ini_get("unicode.filesystem_encoding");
```

```
    if (empty($tmp) && ! empty($_SERVER["WINDIR"])) $tmp='iso-8859-1';
```

```
    if (empty($tmp)) $tmp='utf-8';
```

```
    if (! empty($globalforceosencoding)) $tmp=$globalforceosencoding ;
```

```
    if ($tmp == 'iso-8859-1') return utf8_decode($str);
```

```
    return $str; }
```

// Disponible PHP 6.0

// Par défaut Windows

// Par défaut autre



Bonnes pratiques PHP ... Par l'exemple

Thèmes 6/8 : Portabilité



Résultats et retours d'expérience (Paramétrage PHP)

Solutions pour éviter que le paramétrage PHP impacte le comportement de l'application :

- **error_reporting** → Forcer le choix par instruction PHP dans un include d'en-tête.

```
error_reporting(E_ALL ^ E_NOTICE);
```

- **register_globals** → Utilisation d'une fonction GETPOST()

```
$monfichier='fichier accentué.txt';  
fopen($monfichier, 'w') ;
```

- **magic_quotes** → Compensation par instructions PHP dans un include d'en-tête.

```
if (function_exists('get_magic_quotes_gpc')) { // magic_quotes_* removed in PHP6  
if (get_magic_quotes_gpc()) {  
    $_GET    = array_map('stripslashes_deep', $_GET);  
    $_POST   = array_map('stripslashes_deep', $_POST);  
    $_REQUEST = array_map('stripslashes_deep', $_REQUEST);  
    $_COOKIE = array_map('stripslashes_deep', $_COOKIE);  
}  
@set_magic_quotes_runtime(0); } ;
```

```
function stripslashes_deep($value) {  
return (is_array($value) ? array_map('stripslashes_deep', $value) : stripslashes($value)); }
```

Bonnes pratiques PHP ... Par l'exemple

Thèmes 7/8 : Sécurité



Pourquoi ?

- Éviter les failles de sécurité. S'armer contre les dangers.

L'OWASP recense et classe ces dangers (<http://www.owasp.org>).
Il convient d'avoir une solution pour chacun d'eux.

- A1: Injection (SQL, HTTP)
- A2: Cross-Site Scripting (XSS)
- A3: Broken Authentication and Session Management
- A4: Insecure Direct Object References
- A5: Cross-Site Request Forgery (CSRF)
- A6: Security Misconfiguration
- A7: Insecure Cryptographic Storage
- A8: Failure to Restrict URL Access
- A9: Insufficient Transport Layer Protection
- A10: Unvalidated Redirects and Forwards

Voir page (http://www.owasp.org/index.php/Top_10_2010-Main)



Thèmes 7/8 : Sécurité



Comment ?

- **A1: Injection (SQL, HTTP)** → Instructions PHP dans un include d'en-tête

```
if (test_sql_and_script_inject($_GET)) die('hack non permis'); // A faire sur POST, REQUEST, COOKIE  
  
function test_sql_and_script_inject($val) { $sql_inj = 0;  
$sql_inj += preg_match('/delete[\s]+from|update.+set|.../i', $val);  
return $sql_inj; }
```
- **A2: Cross-Site Scripting (XSS)** → Instructions PHP dans un include d'en-tête

```
$sql_inj += preg_match('/<script/i', $val);
```
- **A3: Broken Authentication and Session Management** → Utiliser les sessions PHP dans un include d'en-tête
- **A4: Insecure Direct Object References** → Instructions PHP dans pages

```
$result = restrictedArea($user, 'facture', $facid, "", 'fk_soc', $fieldid);  
  
function restrictedArea($user, $features='societe', $objectid=0...) {  
...test conformité de l'id et du user...  
If (nonconforme) die('hack non permis'); }
```
- **A5: Cross-Site Request Forgery (CSRF)** → Instructions PHP dans un include d'en-tête

```
$token = md5(uniqid(mt_rand(), TRUE)); // Nouveau jeton à chaque accès  
if (isset($_SESSION['newtoken'])) $_SESSION['token'] = $_SESSION['newtoken'];  
$_SESSION['newtoken'] = $token;  
if (isset($_POST['token']) && isset($_SESSION['token'])) {  
if (($_POST['token'] != $_SESSION['token'])) { die('hack non permis'); } }
```



Thèmes 7/8 : Sécurité



Comment ?

- **A6: Security Misconfiguration** → Utilisation des bonnes options PHP

En production :

`safe_mode=on` (même si sera abandonné), `open_basedir`, `expose_php=off`,
`display_errors = off`, `allow_url_fopen=off`, `allow_url_include=off`

- **A7: Insecure Cryptographic Storage** → Stockage en base du md5(motdepasse)

Pour valider, ne pas faire :

`mot_de_passe_saisie=fdecryptage(mot_de_passe_crypte_en_base)`

mais faire :

`fcryptage(mot_de_passe_saisie)=mot_de_passe_crypte_en_base`

Et utiliser une fonction fcryptage non réversible (exemple md5)

- **A8: Failure to Restrict URL Access** → Même solution que A4

- **A9: Insufficient Transport Layer Protection** → Activation du HTTPS

- **A10: Unvalidated Redirects and Forwards** → Ne pas faire de redirect depuis une valeur issu d'un paramètre utilisateur.



Bonnes pratiques PHP ... Par l'exemple

Thèmes 7/8 : Sécurité



Résultats et retours d'expérience

Euh... ?!?!

Quand on parle de sécurité, n'est-ce pas un retour ?

Bonnes pratiques PHP ... Par l'exemple

Thèmes 8/8 : Tests unitaires

 Pourquoi ?

- Améliorer la qualité
- Réduire le travail répétitif des tests
- Éviter les régressions

Bonnes pratiques PHP ... Par l'exemple

Thèmes 8/8 : Tests unitaires



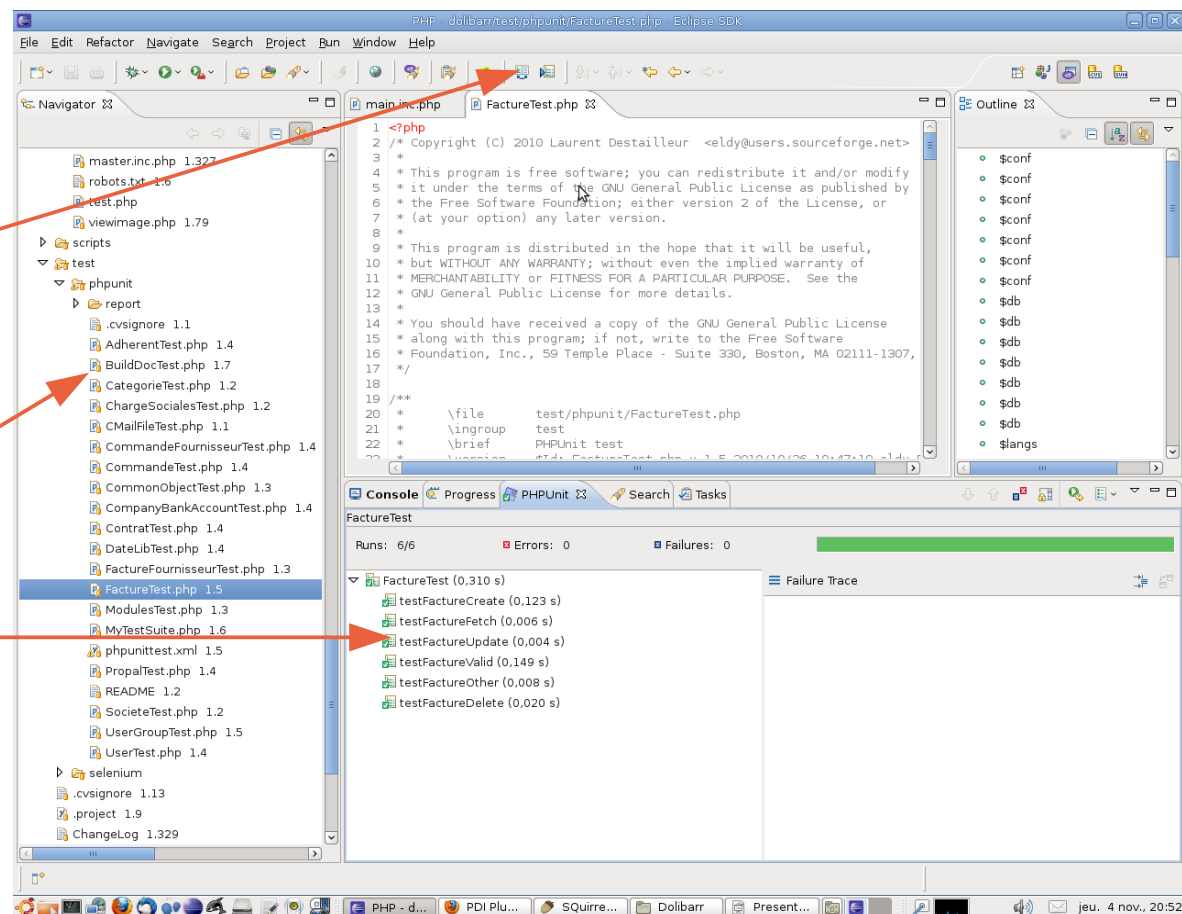
Comment ?

- Utilisation de PHPUnit via l'utilisation du plugin Eclipse PTI (<http://www.phpsrc.org/eclipse/pti/>) qui intègre dans Eclipse PHPUnit (entre autres)
- Utilisation de xDebug
- Stockage des classes de tests unitaires dans le projet SVN ou CVS.

Bouton de bascule classes
Bouton déclenchement test

Liste des classes PHPUnit

Vue résultats des tests



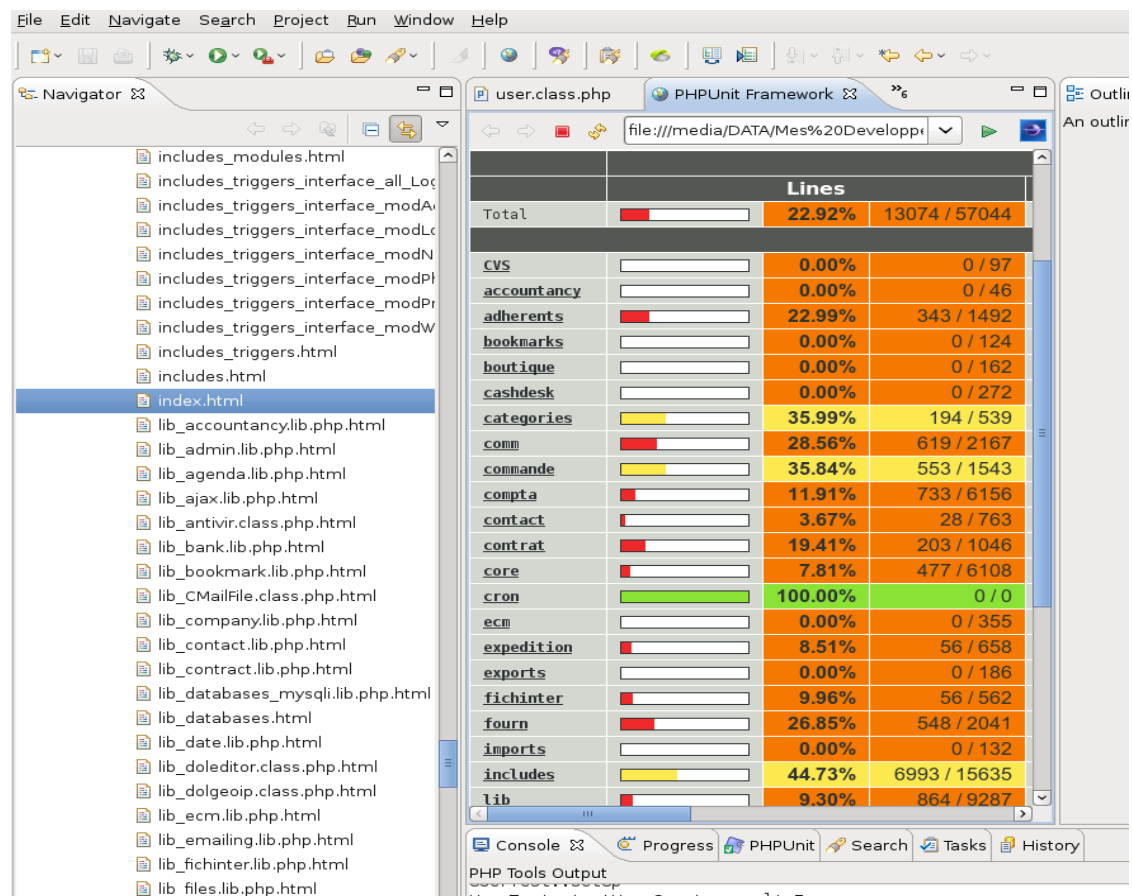
Bonnes pratiques PHP ... Par l'exemple

Thèmes 8/8 : Tests unitaires



Résultats et retours d'expérience

- Une couverture de code des classes métiers passant de 0 à 22% en 4j de développement, pour une exécution des tests automatisés en 30 secondes.
- Une vision graphique HTML de couverture de test à la ligne de code prêt grâce à PHPUnit et xDebug (en ligne de commande)



Bonnes pratiques PHP ... Par l'exemple

Conclusion

➤ Merci...



De nombreux autres thèmes non évoqués...

- Utilisation de Framework PHP clés en main (Symfony, Zend...)
- Utilisation de la génération de code (MDA)
- Les outils d'intégration continue

Pour d'autres retour du projet...

- Le socle de développement de Dolibarr:
http://wiki.dolibarr.org/index.php/Environment_and_development_tools
- Le portail officiel du projet: <http://www.dolibarr.org>
et la documentation : <http://wiki.dolibarr.org>
- ou le stand Dolibarr sur le salon Forum PHP 2010...

