2017

Introduktionsdokument för programmerare

INTRODUKTION TILL PROGRAMMERINGSRELATERADE ARBETSUPPGIFTER

ARGONOVA SYSTEMS | Albanoliden 5, 506 30 Borås

Introduktionsdokument för programmerare

Introduktion till programmeringsrelaterade arbetsuppgifter

Av *Mikael Wacker*

2017

Innehållsförteckning

Inledning

Huvudsaklig programmering sker i PHP, HTML och CSS. Med det sagt är vi på Argonova Systems en webbyrå som händelsevis även kan ta in andra case. Andra språk kan förekomma vid exempelvis app-utveckling. Då är det också vanligare med andra plattformar än vår egna. Vanligast förekommande i det här avseendet är WordPress, vilket det här dokumentet kommer att utgå från att du redan känner till.

All programmering är mer eller mindre ”problemlösande”. Fast det finns olika nivåer av problemlösning och på Argonova finner du både problemlösning i form av rena kodmässiga problem, men även som systemnivårelaterade problem. Med det senare nämnda menar vi problem i form av hur ett system ska struktureras upp och/eller övergripande se ut, för att kunna lösa en kunds behov och önskemål.

Front- respektive *back-end* är vanligt förekommande gruppindelningar. Här på Argonova kan du ha en tydligare profilering mot endera, fast du förväntas ändå alltid att ha koll på båda delar.

Uppdelningen mellan vår egna plattform och exempelvis då WordPress, ser grovt beskrivet ut som att skräddarsydda lösningar byggs i vår egna plattform, medans mer klick-klick-färdig-webbplatser mest fördelaktigt sätts upp i WordPress. Men undantag kan förekomma.

Vår egna plattform

Argonova har en egen plattform som vi utåt sett har valt att kalla *Aroma CMS*. Internt däremot har plattformen genom tiden alltid refererats till som *argoPlatform*. ArgoPlatform påbörjades runt 2007-2008 och har sedan dess vidareutvecklats av ett dussintal olika programmerare genom åren. Något man får leta efter när man tittar på koden. Det beror nämligen på de kodstandarder som direkt togs fram och som håller än idag. Själva poängen är såklart att du från fil till fil ska kunna tro att det är samma person som programmerat alla. Det här gör koden konsekvent och lättare att ta till sig vilket underlättar enormt i underhållet av sajterna. ArgoPlatform hade aldrig kunnat vara det som den är idag utan dessa kodstandarder och symmetriska struktur.

ArgoPlatform är en ”sort of MVC”-uppbyggd plattform. I grunden av dess struktur återfinns en *modulgrupp* som hanterar data, en *vygrupp* som hanterar visuellt kopplat till modulgruppen och en *kärngrupp*. Ovanpå eller utöver detta har vi sedan också *mallar*, *layouter* och *avdelningar*. Allt detta sammantaget har resulterat i en plattform med stora förutsättningar att skapa skräddarsydda system. Andra plattformar så som Wordpress har förstås liknande förutsättningar, och ofta stora bibliotek av färdiga moduler. Att däremot skräddarsy dessa andra plattformar brukar vanligtvis och endast vara hållbart upp till en viss nivå av komplexitet. Därav ser vi alla plattformars olika fördelar, utifrån de olika situationernas behov och budget.

Att beskriva argoPlatform:s struktur och arbetssätt i ren text kan vara lite svårt att hänga med i, men låt oss göra ett försök. Index.php i webbroten är själva utgångspunkten. Därifrån hämtas sedan olika delar ur plattformen för att konstruera en sida. Exempelvis, bland det första som sker, så vill systemet få koll på den aktuella sökvägen (**route**). Det här hanteras genom något vi kallar för **clRouter**. Genom **route**:n får systemet sedan fram en tillhörande **layout**. Layouten innehåller sedan en **template**-fil, en **layout**-fil samt ett gäng **vyer**. Systemet tar då först vyerna och renderar dem i layout-filen. Layout-filen med de renderade vyerna, renderas i sin tur sedan in i template-filen. Vilket resulterar i en färdig HTML-sida på skärmen - Viktigast att ta med sig ifrån den här beskrivningen är att **layout** förekom i två olika skepnader. – 1. En layout som finns sparad i databasen ihop med en sökväg, och 2. en layout-fil som innehåller HTML-strukturen för sidans innehåll. Till exempel om den enskilda undersidan ska ha en eller två kolumner med text- & bildinnehåll. Template-filen är den fil där du hittar exempelvis <html>, <head> och <body>.

För hantering av säkerhet och behörighetsnivåer innehåller argoPlatform **ACO**:er med tillhörande **ACL**:er. **AC**cess**O**bject som i sin tur innehåller **AC**ess**L**ist. En **ACO** kan exempelvis vara *writeNews* och innehålla en **ACL**: *admin* = allow, *guest* = deny. Vilket för oss in på nästa angränsande fakta, nämligen att **alla** som besöker hemsidan klassas som en användare och där en icke inloggad som bara besöker hemsidan alltså klassas som en *guest*.

För *frontend* kan det nämnas att argoPlatform använder **SASS**, **S**yntactically **A**wesome **S**tyle **S**heets, i form av **SCSS**. **SCSS** är en syntax av **SASS** som är kompatibel med **CSS**. **SCSS** ses som en förlängning på **CSS** och därför fungerar alla **CSS-**regler utan problem i **SCSS**. ArgoPlatform använder **SCSS** för att på ett kraftfullare sätt kunna minska ned antalet upprepningar.

Kodstandarder

Det finns två stycken styrdokument:

- coding_standards.pdf - **Argonova Coding Standards** by Peter Nguyen 2008-07
- development_guidelines.pdf - **Development Guidelines** by Peter Nguyen 2008-08

Båda dokumenten finns inbäddade i detta dokument.

Att skriva kod

Kodstandarderna är till för att skapa likformig kod eftersom att alla tjänar på det. Utöver dokumenten för hur kod ska se ut finns hela argoPlatform att titta på som ett exempel för näst intill alla situationer. Kommer du ihåg poängen med att man skulle tro att samma person programmerat alla filer? Vid eventuell osäkerhet kring kod, behöver det inte vara märkvärdigare i de flesta lägen än att snegla på samma eller likvärdig fil i argoPlatform eller på en annan produktion.

Att skriva ”snygg kod” som tar någon minut längre brukar oftast vara ett vinnande koncept i det längre perspektivet. Däremot ska inte ”snygg kod” blandas ihop med ”snyggast lösning”. För ”snyggast lösning” behöver däremot inte vara ”rätt lösning”. Utan ”rätt lösning” är *det du med trygghet känner att du kan stå för och hinner inom utsatt tid* - vilket väldigt ofta är synonymt med kundens budget! Men bara för att du i en situation behöver välja en så kallad ”ful-lösning” bara för att hinna med, så betyder inte det att du kan strunta i att skriva snygg kod. För snygg kod och snygg lösning är alltså två olika saker.

Koden du skriver behöver enkelt tala om vad som sker, eventuella kommentarer talar om *varför*.

Commits till argoPlatform

Att bidra till plattformens utveckling är en del av området för en programmerare. Under ett projekts gång kan det hända att något nytt eller användbart utvecklas, även efter att projektet har slutförts. Det kan då hända att det som har utvecklats ska in i vår **SVN** för plattformen. Det är också högst troligt att det som utvecklats innehåller kundspecifika lösningar. Därför behöver man ställa sig några frågor innan man gör en *commit*:

- Innehåller det som ska commitas något ”kundspecifikt”?
- Är det som ska commitas generellt (fungerar i alla projekt)?
- Kan det påverka något befintligt i plattformen?
- Följer koden argoPlatforms kodstandard?

Därefter är det också väldigt viktigt att commiten innehåller tydliga och bra kommentarer. Alla kommentarer ska vara på engelska och inledas med ett typbeskrivande ord.

Grundläggande begrepp

Flertalet begrepp har redan hunnit omnämnas, fast det finns betydligt fler. Här följer därför en lista över de allra mest vanliga:

- SVN:en = Det *repository* som vi i skrivande stund sparar vår argoPlatform i.
- TRS = Tidsrapporteringssystemet.
- AromaCMS = Den ”mall” som administrationen för argoPlatform använder.
- PoEdit = Ett program vi använder för översättning av gettext-strängar.

Webbhotell & domänhantering

Argonova använder i huvudsak GleSYS och Binero som webhotell. Under 2017 fasas dock Binero ut mer och mer. Under samma period har även domänadministrering förflyttats från Binero till Resello. Domänadministrering kommer i de flesta fall skötas av någon annan än programmerare, men du kommer att behöva grundläggande kunskaper i allmän domänhantering och insikt i hur våra rutiner ser ut.

GleSYS är ett webbhotell med en något högre förväntad kunskapströskel i driftfrågor. Vid uppsättning av en ny webbplats tillhandahålls exempelvis aldrig någon databas, utan det förväntas du själv sätta upp. Vi ser det här som positivt och var en av anledningarna till att vi valde just GleSYS; friheten till egen kontroll. Varje server har också minst 1st SSH-användare dock utan fullständig root-access. Detta på graund av att vi har tjänsten *Managed hosting*. Men via SSH kan du ändå få en del saker gjort. Managed hosting innebär bland annat att GleSYS ansvarar för själva driften av servern, sköter vissa kostnadsfria ingrepp och ger viss support.

Flytt av webbplatser

Vid flytt av webbplatser finns det en del saker att tänka på. Maildrift är viktigast, sedan kommer maildriften och därefter måste du också tänka på maildriften. Skämt åsido, det är lätt att glömma maildriften när webbfiler och databas tar mycket uppmärksamhet. Kom ihåg maildriften. Förhoppningsvis ombesörjs själva domänflytten av någon annan än programmeraren. Men att mailen är uteslutande viktigast för våra kunder går inte upprepa för många gånger. Maildrift, maildrift.

När det kommer till själva den fysiska webbplatsen så brukar plattformen fungera ”out of the box” hos 9 av 10 webbhotell. Däremot kan du ibland behöva placera plattformen inuti webbroten och då krävs det några få anpassningar av **index.php** samt **cfBase.php**. *One.com* är bara ett exempel på när det krävs.

Övrigt kodrelaterat att tänka på som programmerare

Grundläggande program du kommer att behöva utöver en kodeditor är PoEdit, WebDrive, Photoshop och FileZilla. För direkt filöverföring föredras FileZilla då *.svn*-filer kan skippas – de skall inte med till kundernas webbutrymmen. Photoshop och även Office-paketet tillhandahålls av oss under din anställning. Vilken kodeditor du använder är upp till dig själv. Vad beträffar FTP är en rekommendation att tänka på att du enkelt ska kunna skifta mellan hundratals kunder varje vecka. Vissa editorer har inbyggda FTP-klienter av olika kvalité.

Argonova Coding Standards

by Peter Nguyen 2008-07

Purpose

By adapting to a coding standards, it will help a developer to have a consistent coding style. This will further increase maintainability and readability. When working in a team, it is even more critical to have every developer in that team using the same coding standards to avoid errors and confusion. The coding convention provided here is not meant to be a universal standard or an everlasting one. It is therefore important that a developer is flexible enough to adapt to new changes.

PHP

PHP code tags

PHP code tags should always be in the "long tag" format, "<?php ?>" and not the shorthand "<? ? >"

```
<h1><?php echo $sTitle; ?></h1>
<div id="content"><?php echo $sContent; ?></div>
```

Indenting

Indenting helps to make more readable codes by identifying control flow and blocks of code. Always use **real tabs** instead of spaces as this allows the most flexibility across editors.

```
if( $iHours < 24 && $iMinutes < 60 && $iSeconds < 60 ) {
	return true;
} else { return
	false;
}
```

Arrays

Large arrays with named keys and values should also be indented for readability.

```
$aPages = array(
	'Home' => 'home.php',
	'Products' => array(
		'Category 01' => 'category01.php',
		'Category 02' => 'category02.php'
	),
	'Contact us' => 'contact.php'
);
```

Line length

Avoid having lines that span more than 80 characters in length. This is to minimize the need for horizontal scrolling and improve readability.

Vertical alignment

Vertical alignment can help to produce more readable codes, but this should be used sparely because it also make the code more difficult to maintain. Addition or removal of lines of codes may lead to adjustment of the whole code block if they are vertical aligned. Vertical alignment could also be broken when the editor changes from one tab length to another, if real tab character is used.

```
$sTitle           = 'About us';
$sSubTitle        = 'About the company';
```

Spacing

Although most programming languages are unconcerned with the presence or absence of whitespaces, making good use of spacing is good programming style. You should always put spaces at these locations in the code:

- After commas
- After the opening and before the ending parenthesis at the first pair of parenthesis in an expression.
- Before opening braces
- After semicolon in a “for” loop
- Before and after operators

```
$aFruits = array( 'banana', 'apple', 'orange' );
$iCount = count( $aFruits ); for( $i = 0; $i < $iCount; $i++ )
{ if( $i == 1 || ($i != 2 && $i != 3) && is_numeric($i) ) {
echo $aFruits[$i];
      }
}
$iSum = $iPrice - $iDiscount * $iPrice;
$sLongName = $sPrefix . $sName . $sSuffix;
```

Quotes, heredoc and nowdoc

Use single and double quotes when appropriate. Single quotes should be used in cases listed below:

- If you are not evaluating anything in a string
- If you have HTML codes or other content with double quotes in a string. • In array keys

```
$sHtml = '<div id=”logo”><img src=”/images/logo.gif” alt=”Logotype” /></div>';
$sErr = “An error occurred with error number $iErrNr and message $sErrMsg”;
$aCar['color'] = 'blue';
$_SESSION['user_id'] = 9;
```

You should almost never have to escape quotes in a string because you can always alternate between single and double quotes.

Heredoc and nowdoc

Heredoc and nowdoc are useful for multiline strings. The con of it is that the closing identifier must end on its own line with no other character, except possibly a semicolon (;). Therefore, it will break the

indenting rule if used in an indented code block. As a alternative, you may use the multiline features of both single or double quotes.

```
echo <<<EOT
      <div id=”content”>
            <h1>$sTitle</h1>
      </div>
EOT;

// Breaks indenting if(
!empty($_GET['msg'] ) {
switch( $_GET['msg'] ) { case
'no_access':
                  echo <<<EOT
                        <div class=”err”>You have no access to this page</div>
                        <p>Please sign in first</p>
EOT; break;
      }
}

// Use quotes instead if(
!empty($_GET['msg']) ) {
switch( $_GET['msg'] ) { case
'no_access':
                  echo '
                        <div class=”err”>You have no access to this page</div>
                        <p>Please sign in first</p>';
                  break;
      }
}
```

Naming

A naming convention will reduce the effort to read and understand source codes, enhances code appearance and also gives other additional benefits that won't be discussed here. There are different styles of naming convention and the choice of which to use are a very controversial issue. But the most important rule of a naming convention is consistency. Developers in a team should always follow one naming convention and stick to it, otherwise it may lead to confusion and code that are harder to maintain.

Identifiers should also have descriptive names in relation to its content or function. There are mainly two methods for naming identifiers in programming languages.

Delimiter separated words: Also known as the “underscore” method. In this approach, words in an identifier are separated by a delimiter, usually an underscore “_” or a hyphen “-”. For example, most of the core functions in PHP are named in this way.

```
$bIsArray = is_array( $aArray );
error_reporting( E_ALL );
$s_user_gender = 'male';
```

Letter-case separated words: Also known as “camelCase” (or “camelBack”) method, which use capitalization as word boundaries. It is more compact than the “delimiter separated” method, however it

can spoil readability when acronym and initialism must be used as words. Another disadvantage is that there may be problems if a developer want to run the code through a spellchecker.

```
$sUserTitle = getUserTitle( $iUserId );
```

Variables

Variable names should use the camelCase method for compact coding and less typing. Besides that, the variable should also have a prefix to describe its type. Although PHP support type juggling, it is necessary to add a type prefix for other developers to quickly understand your code and use the correct function for a certain variable type. The prefix should be one of the following:

- s for strings
- i for integers
- f for floats
- b for boolean
- a for arrays
- o for objects

```
$sDbName = 'app_db';
$iUserAge = 18;
$fPercentage = 45 / 100;

if( is_array($aFruits) ) {
}

$bIsLoggedIn = true;
if( $bIsLoggedIn === true ) {
}
$oDbPaging = new dbPaging();
$oDbPaging->showNav();
```

Functions and methods

Should use the camelCase method.

```
function getUserAge( $iUserId )
{ } $oDbPaging->showNav();
```

Classes

As with variables, class names should use the camelCase method, but without the need for a prefix.

```
class dbPaging {
}
$oPaging = new dbPaging;
```

Brace style

Braces should always be included except for single statements codes where braces may be optional. The opening braces should always be on the first line of the code block. This include functions and methods.

```
if( $x == $y ) {
	//block of code
} if( $x == $y ) echo
$x;
```

```
function
foo() { }
```

```
switch ($x) {
}
```

Looping and control structures

```
if( $x == $y ) { }

elseif( $x == $z ) {

} else {

}

switch( $x ) {
	case 0:
		// code
	break; case 1:
		// code
	break; default:
		// code
} foreach( $aArray as $key => $value

) { } for( $i = 0; $i <= 10; $i++ ) {

} while( $i < 10 ) {

}
```

Ternary operators

Ternary operators are fine to use and are preferred in some situations (ex. concatenation of string and variables), as long as you don't make them too complicated.

```
$sVar = !empty( $sString ) ? $sString : '';

$sHtml = '
	<h1>' . ( !empty($sTitle) ? $sTitle : $sTitleDefault ) . '</h1>';
```

Include files

To include files and scripts, “require_once” should always be used and in some special cases “include_once” may also be used.

```
require_once “test.php”;
```

Regular expressions

The “Perl-compatible” regular expressions should be used, and the functions which start with “preg_”.

```
function isUrl( $sUrl ) {
    $sPattern = '/^(https?):\/\/((?:[-a-z0-9]+\.)*)([-a-z0-9]+)\.(com|edu|gov|
int|mil|net|org|biz|info|name|museum|coop|aero|[a-z]{2})(\/[-a-z0-
9_:@&?#=+,.!\/
~*%$]*)?$/i';
    return preg_match( $sPattern, $sUrl ); }
```

SQL statements

All SQL commands should be in UPPERCASE to distinct them from table and column names.

```
$sSql = "SELECT DISTINCT(article_cat_id), COUNT(DISTINCT(article_id)) AS i_count_articles
FROM tbl_article GROUP BY article_cat_id";
```

Commenting

Commenting your code is critical both when you work alone or in a team. Even though you may understand everything you code at the moment, the situation will probably has changed a month or maybe a year later. That's when you will be glad that you properly commented when you coded. It also helps other developers to understand your code. There are also tools that automate the process of creating a documentation based on properly inserted comments (ex. phpDocumentor)

Comments that are on a single line should use the double slash “//”. Multiline comments of text or code blocks should use the “/*...*/” commenting style. Comments that contains documentation should be in the form of “docblocks”.

```
// Single line comment

/* Multiline comments with codes

$sVar = '';

*/


/**

*       Documentation for class foo

*        */ class foo {

}
```

HTML

All HTML syntax must follow the XHTML 1.0 specification (http://www.w3.org/TR/xhtml1/). The HTML code must also be properly indented. Child elements should be indented one tab more than their parent element. See ***Indenting*** in the PHP section.

```
<div id=”container”>
      <div id=”header”></div>
      <div id=”content”>
            <div id=”main”></div>
            <div id=”misc”></div>
      </div>
      <div id=”footer”></div>
</div>
```

CSS

Spacing, braces, indenting should follow the same convention as mentioned in the **PHP** section. Selectors should be named according to the “delimiter separated” method. When there is a list of selectors, each selector should be on its own line for readability.

```
body { background-image: transparent url('images/bg.jpg') no-repeat;
} body, div, p, td { font-family: Arial,
Helvetica, sans-serif
      font-size: 12px;
}
#forumParent { font-
      size: bold;
}
```

Javascript

As most syntax of Javascript is similar to PHP, the same convention should be followed.

MySQL

Naming in MySQL should be done according to the “delimiter separated” method and all characters must be in lowercases.

Databases

There may be times when you don't have the possibility to name the database. Other times, you should try to follow to the convention and add the prefix “db_” when you name the database.

Tables

Tables in the database should have the prefix “ent” (short for “entity”). This will help readability in long SQL statements. Examples of table names are “entUser”, “entForumParent”. For tables that are related to each other (ex. parent-child relation) then this should be indicated in the name by having everything but the last word the same, for example “entForumParent” and “entForumChild”.

Columns

Column names should use the last word in its table name as a prefix. This is to avoid ambiguous field names. It also helps to quickly identify which fields belong to which table etc. For example, for the table “entUser”, the column names can be like this “userId”, “userNick”, “userEmail” etc.

Files and directories

File and folder names should use the “camelCase” naming method. There are certain folder names that are recommended to be used for a certain type of files. Here are a list of some recommended names:

- “images” for all kind of images (layout images, user images, icons etc.)
- “functions” for all functions and libraries files
- “css” for all CSS files
- “js” for all Javascript files

Development Guidelines (PHP & MYSQL)

by Peter Nguyen 2008-08

Purpose

The purpose of these guidelines is to help developers avoid mistakes and code more efficient. The guidelines also provide suggestion on how to solve different problems. Instead of re-inventing the wheel, there may already be other people who has come up with a solution that you can apply.

Goals and priorities

When programming an application, each developer should try to achieve the following goals in the same priority order:

1. Easy to maintain and extend
2. Effective and short production time
3. High performance

Appliance

IT is a fast developing industry, thus technologies and softwares evolves and changes constantly. Therefore you should be aware that the guidelines in this documentation may not apply to earlier or future versions. A good developer must have the flexibility to change and adapt to new standards and technologies. This guidelines currently apply to the following versions:

- PHP 4 & PHP 5
- MySQL 4 (not OOP), MySQL 5

Coding methods

Principles

KISS ("Keep it simple stupid")

http://en.wikipedia.org/wiki/KISS_principle

This is often not as simple as it sounds. Larger applications tends to grow big and complex and there are many solutions to a problem. But by trying to choose the simplest solution, you will have a greater chance to avoid complications.

Convention over configuration

http://en.wikipedia.org/wiki/Convention_over_Configuration

By following the coding standards and convention provided in this document, you can avoid to include unnecessary configurations in your applications. This will both improve maintainability and performance.

Multi-tier architecture / MVC

http://en.wikipedia.org/wiki/Three-tier_(computing) http://en.wikipedia.org/wiki/Model-view-controller

The obvious advantage of using a multi-tier architecture is that you have separated layers which can function independently but communicate with each other and exchange information. This make it possible to make changes in one layer without affecting the others which reduce time for code maintainance and improve code reusing.

Object Orientated Programming (OOP)

Design patterns

http://en.wikipedia.org/wiki/Design_pattern_(computer_science)

Design patterns are especially important when programming in OOP. They can be very helpful in providing solutions to different design problems. But you should also be aware that a design pattern may not always be the best solution. Here are examples of patterns:

- **Singleton** - Ensure a class only has one instance, and provide a global point of access to it.
- **Factory method** - Define an interface for creating an object, but let subclasses decide which class to instantiate. Factory Method lets a class defer instantiation to subclasses.
- **Adapter -** Convert the interface of a class into another interface clients expect. Adapter lets classes work together that couldn't otherwise because of incompatible interfaces.
- **Observer** - Define a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically.
- **Template** - An abstract class defines various methods, and has one non-overridden method which calls the various methods.

Global variables and registry pattern

The registry pattern (with or without singleton) is another way to provide global access to variables and objects. In PHP on the other hand, this can be accomplish by using the “global” keyword or the superglobal variable $GLOBALS (recommended). There is then no reason to create a class to handle the same functionality.

Avoid "God" classes

A common mistake when programming in OOP is the a developer tends to put too many functionality into a single class, thus creating a "God" class that handle many things. This is a step against multi-tier architecture and will make the class harder to maintain and reuse. Instead break the class down into smaller classes where each class handle only one entity. For classes that are DAO (Data Access Object), a good rule of thumb is that an entity may be defined as a table in the database.

Use of "public", "private" and "protected" keywords

When creating a class, always explicit add the "private" keyword to class properties and methods which don't need to be accessed outside of the class. The "protected" keyword must also be used for defining class properties and methods that can only be accessed by the class itself and all inherited classes. For class properties that may be accessed from outside the class, the "public" keyword must be declared. Class methods on the other hand, can have the "public" keyword omitted in PHP 5.

```
class simpleClass { public $sPublic = "Public";
        private $sPrivate = "Private";
        protected $sProtected = "Protected";

        private function myPrivate() {
        } protected function myProtected() {
        } function myPublic() {
        }
}
```

Variables and objects declaration

Always declare variables and objects before use to reduce notices and other type of errors. Beside suppressing errors, it will also help to reduce bugs that can arise when the same variable name may be used in different scripts.

Output buffering

Output buffering should always be used when there is something to output. This is to prevent header errors for example when you using header redirecting. Another advantage is that you can use output buffering to create to cache your output.

Template systems

While others are promoting the use of different template engine like Smarty or patTemplate, people tends to forget that PHP is a template system in itself. The use of a template engine will just add another syntax that the developer or designer has to learn. It will also decrease performance compared to using PHP native functions.

Another alternative to templates is the use of XSLT to transform XML to HTML. Although XSLT is a powerful template tool, this will only introduce another unnecessary parsing layer as properly coded XHTML is a XML derivative. On the other hand, XSLT may be useful when you need to transform for example a XHTML page to another XML format.

User authentication

An user authentication system should be based on roles and groups as this will make it easier to assign different privileges to a multiple users.

When using session checking, the authentication script should check a user for a matching combination of both the user's IP-address and the session ID, this is to minimize the risk for session hijacking. For applications that requires high security (applications with money transfer for example), the SSL protocol should be used as much as possible.

Localization (Gettext)

For localization of applications, it is recommended to use the GNU developed internationalization library Gettext. The advantage of Gettext is that it is fast (PHP module) and easy to implement and maintain for both developers and translators. To use Gettext in the code, just wrap any string to be translated in the gettext() function (or use the underscore _() function for shorter syntax). If there is no translation, the text will be left as it is in the text, which is very convenient. Gettext also have tools to scan through the source code and extract all strings to be translated.

```
echo _('This is my title'); // This text will be translated.
```

When translating the text, the translator can use an regular text editor or a specialized editor like poEdit or gted (requires Eclipse).

User inputs

All user inputs must be treated as potential security risks. The inputs must be validated to check that it contains both a correct value and correct type. In addition, all inputs which will go into the database must be sanitized and escaped (ex. with mysql_real_escape_string() for MySQL) to prevent SQL injection. The “is_” functions are very useful here.

```
if ( !empty($_GET['user_id']) AND ctype_digit($_GET['user_id']) ) { if (
        !empty($_POST['user_presentation']) ) {
                $sSql = “UPDATE tbl_user SET user_presentation = '” .
mysql_real_escape_string( $_POST['user_presentation'] ) . “' WHERE user_id = '” . $_GET['user_id'] . “'”;
                mysql_query( $sSql );
        }
}
```

While escaping user inputs into the database is good, strings that would be used for output later on should also go through some HTML filtering method to minimize the risk for XSS (Cross-sitescripting) or just bad input that may ruin the site layout. A good PHP module for filtering HTML and Javascript is HTMLPurifier. Another quick way is to use the strip_tags() and htmlentities() functions.

Preferred functions

file_exists()

If you only want to check the existence of a file or directory, you should use file_exists() instead of is_dir() or is_file() because it's a little bit faster.

str_replace()

Avoid using regular expressions like preg_replace() to replace strings because it's more resource demanding, instead use str_replace() as much as possible. This function can also takes arrays as parameter.

empty()

There are often times when you want to check a variable if it is defined and contains value. In PHP, checking an undefined variable will raise an notice error. To avoid this, you should use the empty() function for this purpose.

```
// This will raise an notice error if $_POST['user_name'] is not set if (
$_POST['user_name'] != 0 && strlen($_POST['user_name']) > 10 ) { }

// This way will not raise an notice error, but still not good enough if ( isset($_POST['user_name']) &&
$_POST['user_name'] != 0 && strlen($_POST['user_name']) > 10 ) { }

// This is better if ( !empty($_POST['user_name']) && strlen($_POST[' user_name'])
> 10 ) {
}
```

ctype_digit()

To check whether an user input, which is of string type, to contain only numeric characters (if you expect such), you can use the function ctype_digit().

```
if ( !empty($_POST['user_age']) && ctype_digit($_POST['user_age']) ) { $sSql = "UPDATE
        tbl_user SET user_age = '" . $_POST['user_age'] . "'"; mysql_query( $sSql );
}
```

is_numeric()

This function is especially useful for checking user inputs that should be of numerical type but not necessary only numeric characters (in such case, the function ctype_digit() is preferably). It works on both strings and numerical values and therefore can be used for $_POST and $_GET variables.

```
if ( !empty($_POST['product_price']) && is_numeric($_POST['product_price']) ) {
        $sSql = "UPDATE tbl_product SET product_price = '" . $_POST['product_price'] . "'";
        mysql_query( $sSql );
}
```

MySQL

Normalization and denormalization

You should always try to normalize the data in MySQL as much as possible to decrease redundancy and in some cases, improve performance. This rule however, does not apply for large-data application.

Indexes

Always create indexes on columns that are used often in "WHERE" statements. You can use "EXPLAIN" on your queries to see which columns should be indexed in a table. Although it requires more disk spaces, indexes will help to speed up queries.

Correct column types

Always select the correct column types when creating tables. This will help to improve performance and reduce disk usage. For example if a column will only store a string of maximum 10 characters, that field should use the column type “VARCHAR(10)”. If a column will store a string of 10 characters, it should be of the column type “CHAR(10)” etc.

Store IP-address as INT(10)

IP-addresses should be converted to stored in column with the type INT(10). Use MySQL functions “INET_ATON()” and “INET_NTOA()” to convert an IP-address to its numerical values. This enable you for example to select a range of IP addresses.

```
$sSql = “SELECT user_id FROM tbl_user WHERE user_ip > INET_ATON('192.168.0.1') AND user_ip < INET_ATON('192.168.0.255')”;
```