

# **SOFTWARE ARCHITECTURE**

→ *Reviewing Team E • WordPress*

**Authors (*Team D*):**

Hans Peersman  
Jeroen van der Velden  
Nikos Mytilinos  
Renato Hijlaard

**Student #:**

3117413  
3251268  
3423794  
3700399

**January 2011**

**UTRECHT UNIVERSITY**  
*Faculty of Science*  
**DEPARTMENT OF INFORMATION & COMPUTER SCIENCE**

## ***Table of Contents***

Table of Contents .....	2
Introduction.....	3
1. General remarks related to the architecture document .....	4
1.1 Section: 4.3.3 Usability vs Security .....	4
1.2 Section: 5.1.3 Patterns.....	4
1.3 Section: 5.2.5.1 SQL Injection .....	4
2. Small Requirement Change .....	5
3. Large Requirement change.....	7
4. Risk analysis .....	9
4.1 Risk 1: Plug-ins .....	9
4.2 Risk 2: Procedural code .....	9
4.3 Risk 3: MVC & WPDB class .....	9

## ***Introduction***

With the present document a project review is made regarding the findings of the review of the architectural description made by group E. The document is structured as follows. The second chapter will start with describing general findings and remarks related to the architectural description; concerns that arise and which questions are left unanswered but we consider as important to be included in an architectural description. The second chapter will continue with presenting a small requirements change which would only cause a small change in the architecture. This small requirements change will be further elaborated by a use case. In the third chapter a requirements change will be presenting who would cause the architecture to fall apart. This requirements change will also be further elaborated by a use case. The fourth chapter performs a risk analysis to point out aspects of the software system that needs to be re-architected according to specific requirements.

The final goal of the review document is to provide team E with additional guidelines and improvements regarding their architecture document, and in the same time to pinpoint changes that could be made to WordPress through “re-architecting” according to specific requirements changes.

## **1. General remarks related to the architecture document**

Within this chapter general remarks related to the architectural description of WordPress will be made.

### **1.1 Section: 4.3.3 Usability vs Security**

Within Chapter 4.3.3 in the architecture document about WordPress, a comparison is being made between *Usability* and *Security*; however this chapter seems to be lacking of adequate explanation for several points of view. Usability is usually gained by adding additional plug-ins but the document fails to explain how WordPress protects itself against malicious plug-ins. One of the biggest problems of open-source software and third party plug-ins nowadays is the activities of hackers who add malicious code that cannot be seen by common users. The document does not explain how WordPress protect itself against these kinds of plug-ins except for “*only install trusted plug-ins and be careful*” which can be characterized as a vague statement. Additionally, the document does not mention how the plug-in scanner works nor which kind of access is required for plug-ins in order to successfully work (e.g. Administrator access). Moreover, the document describes plug-ins could cause delays to a website or internal server errors, however it remains unclear what this has to do with the security aspect. Is there any reason for this? Do individual plug-ins or the core building blocks of WordPress affect each other from the security point of view? Why does the performance related issue exactly arise?

### **1.2 Section: 5.1.3 Patterns**

According to the document in question, WordPress Architecture is based on four types of building blocks. However none seem to include the plug-ins which are the essential part of WordPress. Which building block manages these plug-ins? The document describes how the plug-ins are interacting with the core machine and how they are handled in the output, e.g. in the form of templates, but fails to describe concretely the exact building blocks and how these are affected.

Secondly, the document mentions that WordPress does not strictly follow the Model-View Controller (MVC) and the use of procedural code instead of Object Oriented (OO) code. What was the reason to make this comparison and are there any side effects of the corresponding WordPress' choices in the architecture? The advantages and tradeoffs remain unclear until a comparison is made between Joomla and WordPress of which the MVC model causes Joomla to be seen as a better reliable system with better maintainability.

### **1.3 Section: 5.2.5.1 SQL Injection**

Why does WordPress not use the PHP Data Objects (PDO) included by PHP? Earlier in the document it is being recommended to always upgrade to the latest version of WordPress but doesn't this advice also count for having the latest version of PHP? PHP is the essential underground for WordPress and therefore one would also suspect that the latest version advice would also be applicable for this. The document does not describe the advantages or the impact on quality aspects/tradeoffs of using WordPress' private `wpdb` layer instead of the PHP's PDO. In addition, a reader cannot obtain information about how vulnerable WordPress is in combination with an older version of PHP.

## 2. Small Requirement Change

Within this chapter a small requirement change will be made towards the update functionality of WordPress. By default, WordPress does not have the ability to automatically check for new versions and proceed to upgrade to the latest version. However, since this process is easy to automate, plug-ins have already been made available to support such a functionality.

Plug-ins supporting the automatic upgrade of WordPress contain 8 basic steps:

1. Back up files and make available for download
2. Back up database and make available for download
3. Download latest files from WordPress.org/latest.zip and unzip
4. Put site in maintenance mode
5. De-activate all active plug-ins and remember plug-ins
6. Upgrade WordPress files
7. Re-activate plug-ins
8. Put site in regular mode

In order to initiate this process, the plug-ins call UPDATE.PHP which can be found within the WP-INCLUDES directory which is automatically loaded by WordPress. UPDATE.PHP contains of a simple set of functions to check the currently used WordPress version and plug-ins against the newest version available on API.WORDPRESS.ORG.

In order to automate the process as provided by WordPress the requirements of UPDATE.PHP should be changed. It should not only check if a new version is available but also include the steps made by the plug-in for an automated upgrade (Figure 1).

```
function wp_version_check() {
    if ( defined('WP_INSTALLING') )
        return;

    global $wp_version, $wpdb, $wp_local_package;
    $php_version = phpversion();

    $current = get_site_transient( 'update_core' );
    if ( ! is_object($current) ) {
        $current = new stdClass;
        $current->updates = array();
        $current->version_checked = $wp_version;
    }

    $locale = apply_filters( 'core_version_check_locale', get_locale() );
```

**Figure 1.** A required change in WordPress to achieve automated upgrade.

In case the function WP\_VERSION\_CHECK() notices that a new version of WordPress exists, it should call a new function called WP\_VERSION\_UPGRADE(), which should be added to UPDATE.PHP. In this function all the steps, from 1 till 8 should be executed.

Secondly the file WP-SETTINGS.PHP should also be updated. Apart from a check if the site is in maintenance mode through WP\_MAINTENANCE(), it should also check if the system is in upgrade mode. A new variable should then be added.

Third, a new variable should be made in the `WP-CONFIG.PHP` file, where people can have the possibility to either enable or disable the possibility of automated updating of WordPress.

### **3. Large Requirement change**

In this section we will describe a change in the requirements that can only be achieved by profound modifications in the architecture of WordPress. Our choice to create the conditions for such an effect is to create a version of WordPress in Microsoft's .NET Framework. It is a very popular software framework which supports several programming languages and allows the creation of web applications like WordPress. We will investigate the impact of the change by focusing on the characteristics of WordPress' architecture that are analyzed in the architecture document.

In the section concerning the quality aspects of the system, there is a description about pingback, trackback and stop spam capabilities. All these aspects are high level entities and should be rather regarded as functions, therefore it is not useful to investigate the impact of the .NET adoption from the architectural point of view. Nevertheless, usability, security and use of patterns are architectural aspects that will be seriously affected.

First of all, usability of the system would be excessively affected while the two different platforms support different techniques for visualizations. The design and the user interface components should be modified according to .NET capabilities to render visualizations on the user's screen. Flexibility is mentioned as a key aspect of usability and is realized mainly by the implementation of 3<sup>rd</sup> party plug-ins by indented developers across the globe. Although both PHP and .NET are using object-oriented programming languages, the former does not support all concepts of the object-oriented approach. For instance, there is no support for encapsulation and overloading concepts, or for private, public, or protected functions in classes, which can have a significant impact on the way that the plug-ins are produced. Upon the change to .NET, flexibility will be adapted to include the outcome of more effective developing. Moreover, .NET allows the creation of session states not only by using cookies as it is happening in PHP, but also by cookie-less session state definitions that allows extensible definitions of custom settings. The consequence is again increased flexibility. On the same time though, despite the fact that PHP is free software, .NET Framework requires a commercial Microsoft Windows server system. Flexibility at this point is hindered, while purchase and integration of commercial software is required for the support the .NET Framework. Another important issue is that .NET web applications are precompiled and therefore are occupying resources on the server, whereas PHP web applications are interpreted on-the-fly.

Security is analyzed as another important attribute of WordPress' architecture. Both of the platforms are vulnerable to attacks aiming to steal or modify data or to send spam and distribute malware. On the contrary to PHP, .NET Framework contains integrated mechanisms to deal with security risks. In case we are interesting to increase security by enabling these mechanisms, changes in the way developers are working are again required to support the consequent change in architecture. The extra security level suggests additional validation and verification checks, and finally the transit from "limited access" perspective to the "full access" one. Passwords, file permissions, database security and data backups are already included in the .NET Framework; the swift in the architecture will provide new means to set safety, security and compliance as components of the adapted architecture.

Further on, WordPress is examined how compatible it is with the MVC software architecture with not very positive results. A change from PHP to .NET Framework will possibly

cause an even more diversion from the MVC approach. First of all, the support for MVC in .NET Framework appeared in a relatively recent version, while the support in PHP has started a long time ago, a fact that signifies the higher maturity of the architecture in PHP. The comparison between PHP and .NET variations of MVC can also show that the former requires more changes in the existing architecture of the system. Replacement of plain SQL queries with the more advanced LINQ queries prescribes numerous changes in the existing database source code and demands developing of new database elements.

## **4. Risk analysis**

In this section we highlight the risks the current architecture of WordPress brings with it. The risks are limited by the findings within the architectural document.

### **4.1 Risk 1: Plug-ins**

In the document in question it is mentioned that plug-ins pose a security risks (*4.3.3 Usability vs. Security*). WordPress has over 12.000 plug-ins, maintained by the community itself. What happens if WordPress has an update? Is backward compatibility always kept for the current plug-ins? How can we be sure a WordPress update will not affect the plug-ins that are currently installed? Does WordPress check compatibility with plug-ins when new major changes are made? This may cause severe consequences to the functionality of some plug-ins, that after an upgrade might not be able to run anymore. Furthermore if backward compatibility is kept, this might clutter the code in the long term. There is nothing mentioned in the document about this issue. The fact that WordPress is mostly based on plug-ins makes this a very important risk.

### **4.2 Risk 2: Procedural code**

Section *5.1.3 Patterns* mentions that WordPress uses procedural code instead of patterns. In section *6.1.4 Technical differences between WordPress and Joomla* the authors state that the procedural code severely affects the maintainability of WordPress. For this reason the procedural code should also be seen as a risk.

### **4.3 Risk 3: MVC & WPDB class**

Building on the previous risk, the use procedural had as a result that WordPress does not conform to the MVC pattern. In section *5.2.5.1 SQL injection* the authors talk about the WPDB class. This is a custom database abstraction layer. It is supposed to prevent malicious queries from executing by checking if they are properly escaped. Unfortunately it is not mandatory to use this class. This means that plug-in developers can choose not to include in their implementations. The architectural decision not to use MVC may pose serious security risks for unknowing persons using WordPress.