Multi-layer Software Configuration: Empirical Study on Wordpress

Mohammed Sayagh, Bram Adams
Polytechnique Montreal, Canada
{mohammed.sayagh, bram.adams}@polymtl.ca

***Abstract*—Software can be adapted to different situations and platforms by changing its configuration. However, incorrect configurations can lead to configuration errors that are hard to resolve or understand, especially in the case of multi-layer architectures, where configuration options in each layer might contradict each other or be hard to trace to each other. Hence, this paper performs an empirical study on the occurrence of multi-layer configuration options across Wordpress (WP) plugins, WP, and the PHP engine. Our analyses show that WP and its plugins use on average 76 configuration options, a number that increases across time. We also find that each plugin uses on average 1.49% to 9.49% of all WP database options, and 1.38% to 15.18% of all WP configurable constants. 85.16% of all WP database options, 78.88% of all WP configurable constants, and 52 PHP configuration options are used by at least two plugins at the same time. Finally, we show how the latter options have a larger potential for questions and confusion amongst users.**

I. Introduction

Configuration is the means to adapt a software application to different contexts and environments. For example, the Linux kernel can be customized to different users by selecting only the features that are of interest. Similarly, the kernel can be customized to a specific hardware platform by providing the details of processor, hard disk, and other devices. Each software system has its own mechanism for configuration, ranging from hardcoded constants to global variables, property files or dedicated databases, typically with a graphical user interface to hide the underlying storage mechanism.

An incorrect value of a configuration option could result in incorrect behavior of a system, which we refer to as configuration errors. Such errors occur often, typically are severe in nature, hard to debug, but they are actionable [1]. Such bugs are severe because they can have a catastrophic impact. For example, due to a misconfiguration, Facebook was left inaccessible for about two hours[1], depriving more than 500 million users from access to the Facebook website. Configuration errors are also hard to debug, since they need expertise in the failing application. However, if one is able to track down the cause of such an error, then the error is actionable since a maintainer just needs to update the configuration, usually without recompiling.

The major challenge for resolving software configuration errors is to find the violating configuration options, a challenge that is aggravated in multi-layer systems. Multi-layer systems consist of multiple layers, each of which hides the complexity of a lower layer, and has its own objects and configuration mechanisms. Since the behaviour of the system as a whole requires neighbouring layers to collaborate, one needs to understand each layer's configuration as well as how configuration options in each layer interfere with each other.

Let's consider the case of WP (Figure 1), which is currently the most popular content management system, and a typical example of a multi-layer system consisting of a LAMP stack (Linux, Apache, MySQL and PHP), the WP PHP application and a myriad of WP plugins. One example of a cross-layer configuration error was the inability of WP plugins to send emails, due to a misconfiguration in lower layers related to the PHP configuration option *sendmail_path*[2]. A second example was the case where the *NextGen* plugin was no longer able to upload images[3] until someone pointed out that the script downloading the images was blocked by a configuration option in the PHP layer (*memory_limit*). In both examples, configuration options in lower layers impacted the behaviour of the top layer plugins.

Whereas existing work focuses on software configuration and configuration errors within a single layer of a software system, this paper represents a first empirical study towards understanding the configuration options used by and shared between different layers in the WP multi-layer system, as well as potential links with comprehension problems of users. The results can then be used in a follow-up study on multi-layer configuration errors. In particular, we address two preliminary research questions to understand the evolution of configuration options across time, and two questions analyzing the usage of options across the studied layers:

RQ1: What is the proportion of usage of each configuration mechanism in each layer?
WP uses configurable constants and database configuration options equally, while WP plugins prefer (87%) configuration options stored in the database.

RQ2: How does configuration mechanism usage evolve across time in each layer?
Generally, the number of configuration options grows across time, especially when new features are added

[1]https://www.facebook.com/notes/facebook-engineering/more-details-on-todays-outage/431441338919

[2]https://wordpress.org/support/topic/plugin-contact-form-7-wont-connect-to-smtp-server

[3]https://wordpress.org/support/topic/plugin-nextgen-gallery-error-exceed-memory-limit

in a layer. Configuration options that are not used anymore are removed after a while.

RQ3: How many configuration options defined in lower layers are used by WP plugins?
On average, 1.49% to 9.49% of all WP database configuration options and 1.38% to 15.18% of all WP configurable constants are used by the plugins. Furthermore, 1.30 PHP configuration options are used by plugins, but only 0.40 ones are modified. These large numbers are confirmed by the Stack-Overflow and WP Exchange fora, where 12.19% of all plugin conversations and 9.49% of all WP conversations related to configuration mention multi-layer configuration options.

RQ4: How many plugins share the same configuration options of lower layers?
78.88% of all WP configurable constants and 85.16% of all WP database options are used by at least two plugins. For PHP, 52 PHP configuration options are used by at least two plugins. We found a strong correlation of up to 0.55 between the number of plugins using a configuration option and the number of fora conversations mentioning it.

The paper is organized as follows: section II presents the background and related work, and section III presents our methodology. Section IV provides the results of our study, while section V discusses the threats to validity. Finally, section VI concludes the paper and presents future work.

II. Background and Related Work

In this section, we provide background information about software configuration, multi-layer systems, and the WP ecosystem, and we discuss related work.

A. Software Configuration

Configuration is a mechanism to adapt software systems to a context or an infrastructure and is used to customize a system's behaviors. A configuration option is a pair consisting of an option name and its value, where the value has a specific type (typically boolean or categorical, but sometimes numeric or even a string). For example, a PHP configuration option *file_uploads*, which is used to allow file upload or not, could have a value of either *On* or *Off*, while an option like *error_log* could have any string as its value.

Different configuration mechanisms exist, which typically differ in binding time and storage mechanisms. Values could be bound to configuration variables at compile-time, load-time (virtual machines) or run-time, while the values could be stored in the source code (macro constant or global variable), database, property files or in other supports.

A configuration error then is a set of configuration options that lead to unexpected behaviour, typically causing errors, even though the source code itself is correct. Yin et al. [2] provide an empirical study of a commercial storage system and four open source systems on configuration errors, and were able to classify 546 configuration errors into five major categories. Arshad et al. [3] provide a characterization of configuration problems for two Java EE application servers, GlassFish and JBoss, by analyzing 281 bugs-reports. Hubaux et al. [4] conduct two surveys respectively among Linux and eCos users to understand configuration challenges. Jin et al. [5] analyze two open source applications and one industrial application to quantify the challenges that configurability creates for software testing and debugging. Other studies focus on predicting configuration bugs. Using textual information in bug reports, Xia et al. [6] built a model to predict whether a bug is a configuration error or not.

Many studies have been conducted to resolve configuration errors. Keller et al. [7] proposed the tool *ConfErr* that aims at quantifying the resilience of a software system to configuration errors caused by spelling mistakes, structural errors, and semantic errors. Zhang et al. [8] built a tool to identify the root cause of a configuration error in Java programs. Zhang et al. [1] provide the tool *ConfSuggester*, which suggests the configuration option responsible for introducing a bug in a new version. The suggestion is generated based upon the control flow of a system. Elsner et al. [9] propose a framework to detect configuration inconsistencies. It allows a user to specify the possible inconsistencies in a software application, which will be combined with a model built from the configuration files to find the inconsistencies. Attariyan et al. [10] built the tool *ConfAid*, which aims at pointing out the root cause of configuration errors, again by analyzing the control flow. Tartler et al. [11] propose an approach to resolve the inconsistencies between the configuration model and its implementation in the Linux source code.

While the above research provides a set of configuration options that should be changed in order to fix a bug, Wang et al. [12] present an ordered set of configuration options to change in order to fix a configuration error, based on user feedback. Similarly, Xiong et al. [13] propose an approach that yields the configuration options to change and a range of possible values. Lillack et al. [14] evaluate the tool *Lotrack*, which explains for each code fragment which load-time configuration options should be active for it to be executed. Nadi et al. [15] propose a static approach to extract and validate configuration constraints from C code, which would be hard for non-experts of a system to do manually. They also evaluate the approach's accuracy on four highly configurable open source systems. Rabkin et al. [16] use logs and traces to map each program point to the configuration options that could introduce an error. Jin et al. [17] built PrefFinder, using an NLP engine to provide the possible values of a configuration option. However, none of these related papers study multi-layer configurations.

While we do not study multi-layer configuration errors, we do a preliminary study of the prevalence of multi-layer configuration options. High prevalence would suggest that corresponding errors are likely and hence should be studied.

B. WP Ecosystem

Similar to Drupal and Joomla, WP is one of the most popular and powerful [18] Content Management Systems

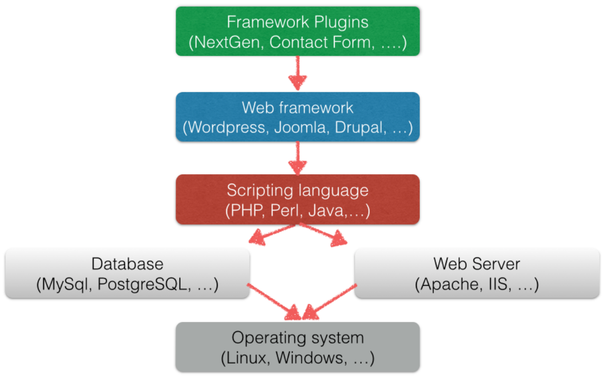


Figure 1: The layers of a typical WP installation. We focus on the configuration options of the top three layers.

(CMS) for creating blogs. It powers more than 60 million websites, i.e., 61% of all websites created by a CMS[4], and 23.3% of all websites in existence[5]. One of the most important factors in WP' success is its variety of plugins. WP plugins (such as *NextGen*, and *Contact Form*) allow users to add to their websites any functionality that they can imagine, since a plugin basically consists of PHP scripts that can access any of the lower layers of the WP architecture. WP has thousands of plugins, together downloaded more than 748 million times[6].

WP typically runs on top of a multi-layer LAMP stack (Figure 1), consisting of a Linux operating system, Apache web server, MySQL relational database and PHP scripting language[7]. The vertical layout of Figure 1 shows how the different layers communicate with each other. The plugin layer communicates with the *Web framework* layer, which relies on the lower layer *Scripting language* that is used to connect to the *database* and *web server*. These elements in turn rely on the operation system, which hides the hardware complexity.

Some studies have focused on multi-language or multi-layer web applications. As PHP is a dynamic language used to create web pages, Nguyen et al. [19] propose a static analysis to find undefined variables and functions in all web pages generated by any HTML, JS, PHP, or SQL script. Eshkevari et al. [20] study the problem of interference (conflicting entity names, hooks, database code, variables, and risky includes) between WP and 10 plugins, and propose an approach to resolve it. Nguyen et al. [21] elaborated a prototype PHP interpreter to detect WP plugin conflicts out of the large number of possible combinations (2^{50}) of activated WP plugins. They found that among all plugin combinations, 29% of WP statements and 89% of WP variables' values are shared. None of these papers study configuration options.

C. WP Configuration Mechanisms

WP and its plugins use two mechanisms for configuration. The first one consists of storing the configuration options in

[4]http://w3techs.com/technologies/overview/content_management/all/
[5]http://w3techs.com/technologies/details/cm-wordpress/all/all
[6]https://wordpress.org
[7]http://searchenterpriselinux.techtarget.com/definition/LAMP

```
if (! defined('FTP_FORCE')) define ('FTP_FORCE', true);
```

```
$method = defined('FS_METHOD') ? FS_METHOD : false;
```

Figure 2: Two examples of configurable constants that can be redefined in *wp-config.php*

Table I: WP and plugins of the Small Data Set used in RQ1 and RQ2.

WP/Plugin	Versions (#)	# Downloads	Popularity Rank of Last Version
WP (platform itself)	1.5 - 4.0 (26)		
all-in-one-seo-pack	0.6.2.6 - 2.2.4.1 (232)	21.23M	2
updraftplus	0.7.4 - 1.9.5 (175)	1.70M	11
Google XML Sitemaps	2.5 - 4.0.8 (46)	16.20M	12
NextScripts	1.6.1 - 3.2.3 (11)	1.44M	15
wp-pagenavi	1 - 2.87 (23)	5.28M	27
Page Builder by SiteOrigin	1.2.10 - 2.0.3 (32)	1.15M	29
MailPoet	2.5.2 - 2.6.9 (26)	3.29M	33
Redirection	2.1.29 - 2.3.11 (19)	2.09M	34
The Events Calendar	1.5 - 3.9.1 (44)	1.35M	48
BBPress	2 - 2.5.4 (37)	1.66M	57
Download Manager	2.1.3 - 2.7.5 (7)	0.92M	58
broken-link-checker	0.1 - 1.10.4 (118)	3.14M	72
Captcha	2.12 - 4.0.7 (87)	2.42M	77
Flyzoo	1.4.2 - 1.4.5 (4)	0.32M	117
WP-Members	2.1.0 - 2.9.7 (49)	0.64M	132

a database, while the second one consists of overriding PHP constants in the WP source code. We respectively refer to them by "database options" and "configurable constants".

Database options are stored in the table *wp_options*, while configurable constants are PHP constants that can be overridden by a user in a central configuration file *wp-config.php*. Every usage of the latter constants in the source code is preceded by an if-check (as shown in Figure 2) that checks whether the constant has been defined already. If not, it defines the constant with a default value. Since the *wp-config.php* file is loaded first by WP, any constant defined in that file by the user will have precedence over the default value.

III. Approach

This section presents the methodology used to answer the research questions of the introduction.

A. Data Selection

Since RQ1 and RQ2 require manual analysis to complement the quantitative findings, we used a more focused data set for it ("Small Data Set"). For RQ3 and RQ4, we rely less on qualitative analysis and are able to study a larger-scale data set ("Large Data Set").

1) Small Data Set (RQ1 and RQ2): For RQ1 and RQ2, we analyzed the source code of the WP layer (in the remainder of the text, we refer to "WP") and 15 WP plugins. The selection of plugins is based on the following two criteria:

- Criterion 1: Plugins should have a dedicated set of methods to extract configuration options from the database.
- Criterion 2: To avoid the need for intra- or inter-procedural data flow analysis, the parameter of the methods used to extract a plugin's configuration options should be specified as a literal.

While all plugins use the same methods to access the database options of the WP layer, each plugin can have its own methods to access its own configuration options. Since these methods are not known a priori (basically requires manual analysis), and these methods could change across time, the manual validation of criteria 1 and (especially) 2 took a substantial amount of time.

Based on these criteria, we randomly selected plugins for analysis from the popular plugins listed in the ”add plugins” administrator page of a WP website, and obtained 15 plugins that satisfied the criteria (see Table I). Although these plugins do not cover the top 15 most popular plugins, we can see that all plugins have at least 320,000 downloads, with a maximum of 21.23 million downloads for *all-in-one-seo-pack*. The plugins have between 4 and 232 versions.

2) Large Data Set (RQ3 and RQ4): To analyze the interaction between different layers in the third and the fourth research questions, we use the last version of WP at the time of writing (4.0) and the 484 most popular plugins. This list of plugins can be obtained by any WP user from the administrator pages of a WP website. As we obtained in the first and second research questions the names of all WP configuration options, we just have to check the usage of those names in the plugins' source code. Furthermore, we found that for the methods used to access WP database options, the WP option name is passed as a literal argument in 98% of all the method calls. Therefore, we were able to use the 484 most popular plugins without limitations or specific criteria.

B. Identification of Configuration Options and Their Usage

1) Data Sources: We obtained the versions of WP and each plugin selected for the Small Data Set from the corresponding Subversion (SVN) repositories. In those repositories, WP and its plugins make all their versions accessible via SVN tags.

To obtain the 484 most popular plugins, we selected and installed the top 484 popular plugins (at the time of writing the paper) in our WP administrator environment.

2) Manual Identification of Access Methods:: Each plugin can have its own method to access the configuration options from the database. To understand these methods, we installed and activated each plugin, then looked at the database options added by the plugins of the Small Data Set, as well as how these configuration options are accessed in the source code of the plugin. Some plugins use the same methods as WP to access their own database configuration options, while others have their own methods. The number of methods differs from one plugin to another, and ranges from one to five.

As the plugins could change their methods across versions, we analysed the last version of each plugin, computed the number of configuration options for all plugins, then we checked manually for the previous versions if there was a big difference in the number of configuration options between two versions, which could be due to the modification of the methods to access database options. If so, we took the change in access methods into account.

C. Measuring The Proportion of Usage of Each Configuration Mechanism (RQ1/RQ2)

To obtain the configuration options used in the WP layer and the plugins layer, we performed the following two steps. First, to get the configurable constants of the plugins and WP, we scanned the source code for constants for which there is an if-check, as shown in Figure 2. For WP itself, we also scanned any constant already defined in *wp-config-sample.php*.

Second, to get the database configuration options of the plugins, we scanned the source code to find method calls to the methods that we know are being used by the plugin to access options (from our manual analysis in the previous section).

D. Measuring Direct Usage of Configuration Options (RQ3/RQ4)

To find the WP configuration options used by the plugins, and the PHP configuration options used by WP and the plugins, we used two approaches. The first approach measures direct usage of configuration options, i.e., textual occurrences of any of the known WP configurable constants (based on the list that we obtained in the two previous research questions) or explicit calls from a plugin or WP to obtain the value of a specific database option. This analysis uses regular expressions. The second approach (discussed in the next subsection) measures options accessed indirectly via nested method calls (e.g., a plugin calls a method of WP that accesses a configuration option).

To find accesses to PHP configuration options, we check the source code of WP and each plugin for calls to the methods *ini_get* or *ini_set*.

E. Measuring Indirect Usage of Configuration Options (RQ3/RQ4)

Given the gap in layers between the definition and usage of an option, RQ3 and RQ4 also need to deal with indirect usage of options. For example, instead of accessing a configuration option inside the source code of a plugin, the plugin might call a function of another plugin or maybe WP that accesses the configuration option.

We use the open source PHP-Parser[8] to build control flow graphs of all methods inside WP and each plugin, then perform a filtering of the control flow graphs. The filtering retains only statements that read or write configurable constants and database options of any of the three layers. Methods that are called but do not manipulate configuration options themselves are filtered out as well.

At the end of this filtering, the graphs contain all the data about configuration options and indirect access via method calls that we need. For each method, we can transitively follow the graph's edges to find all WP and PHP options that it reads or writes. For example, if the function x calls y and z, and y calls w, then by using the graph generated in this step, we return all the configuration options used in x, y, w, and z.

[8]https://github.com/cwi-swat/PHP-Parser

Note that direct usage of configuration options represents a lower bound of option usage for a given plugin or WP, with indirect usage corresponding to the worst case of additional options that could be accessed. We expect real usage to be closer to the lower bound, but included both bounds for completeness.

F. Measuring Configuration Options' Occurrences in Discussion Fora (RQ3/RQ4)

To understand the actual impact of the findings of RQ3 (usage of options of deeper layers) and RQ4 (multiple plugins using the same options), we analyzed the StackOverflow and WP Exchange[9] fora respectively from July 2008 and November 2011 to March 2015, amounting to 7,259,572 and 46,509 conversations in total. We got the data of these fora in the form of xml files from the StackExchange archives[10].

We used StackOverflow and WP Exchange fora, because StackOverflow is one of the most popular fora for developers across all programming languages, while WP Exchange is a popular forum with WP developers. We did not analyze the WP support forum[11], because we could not distinguish conversations related to WP from those related to WP plugins.

For RQ3, we analyzed whether multi-layer configuration represents a real problem in practice, by measuring the percentage of conversations related to multi-layer configuration issues. For this, we searched for the names of WP and PHP configuration options inside forum conversations on plugins. We used the conversations' tags to distinguish between conversations related to WP and conversations related to plugins. A conversation related to WP plugins typically contains the keyword *"plugin"* like *"Wordpress-plugin"*.

For RQ4, we analyzed the same fora to find if there is a correlation between the number of plugins using a configuration option and the number of conversations about the option, which could give an indication about the existence of multi-layer configuration problems or confusion by users.

IV. Results

In this section, we present for each research question the motivation, the approach used, and the results.

(RQ1) What is the proportion of usage of each configuration mechanism in each layer?

Motivation. We analyze in this research question the use of each configuration mechanism, with the goal of understanding which one is the most popular and hence needs closer analysis.

Approach. We used the approach of Section III-C.

Results. **The average number of configuration options across all plugins and WP is 76.** FlyZoo and Page Builder have the lowest number of options (less than 10), followed by *Redirection* and *Captcha*. *NextScripts* and WP have more than 300 options, followed by *updraftplus* and *all-in-one-seo-pack*.

On average 87% of a plugin's configuration options are stored in the database. Figure 3 shows the distribution of the

[9]http://wordpress.stackexchange.com
[10]https://archive.org/download/stackexchange
[11]https://wordpress.org/support/view/all-topics

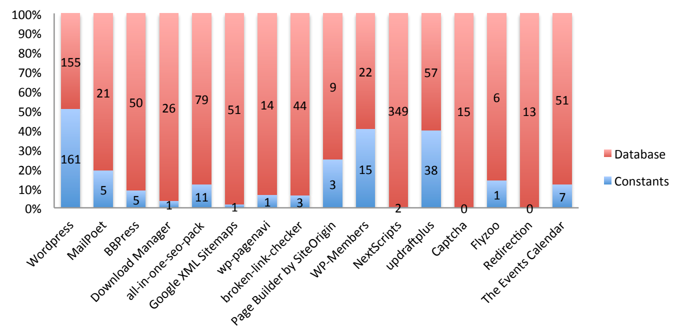


Figure 3: Distribution of the number of configurable constants and database options for WP and the 15 analyzed WP plugins.

Table II: Categories of database options.

Categories	Examples
General	siteurl, blogname, admin_email
Writing	use_smilies, mailserver_url
Reading	posts_per_page
Discussion	show_avatars, comment_registration
Media	thumbnail_size_w, large_size_h
Permalinks	permalink_structure, category_base

number of configuration options across both mechanisms, for each plugin. It shows that WP uses 161 configurable constants as configuration option, and stores 155 configuration options in its database. In contrast, all WP plugins mostly use the database as a configuration mechanism, with an average of 87%, compared to 13% for configurable constants.

While the plugins use database options more than constants to configure their behavior, the percentage of usage of each mechanism differs from one plugin to another. There are some plugins where the configurable constants present 40% of all configuration options, such as *WP-Members* and *updraftplus*. Other plugins use the constants mechanism to configure around 20% and 25% of their configuration options, like *MailPoet* and *Page Builder by SiteOrigin*. The plugin *Flyzoo* uses constants as configuration mechanism for 14% of its configuration options, approximately the same as the plugin *The Events Calendar*. Finally, we have nine plugins where the configurable constants represent less than 10% of all configuration options. The plugins *Captcha* and *Redirection* do not use configurable constants at all, they use only the database as a mechanism to store configuration options.

Discussion. To understand the dominance of database configuration options, we manually analyzed the documentation

Table III: Categories of WP configurable constants with examples by wpengineer.com [12].

Categories	Examples
General	AUTOSAVE_INTERVAL, WPLANG
Status	APP_REQUEST, DOING_AJAX
Path, dirs, and links	WP_LANG_DIR, ABSPATH
Database	DB_HOST, DB_USER
Multisite	MULTISITE, DOMAIN_CURRENT_SITE
Cache and script compressing	COMPRESS_SCRIPTS, ENFORCE_GZIP
Filesystem and connections	FTP_HOST, WP_PROXY_PORT
Themes	HEADER_IMAGE, TEMPLATEPATH
Debug	SCRIPT_DEBUG, WP_DEBUG
Security and cookies	COOKIE_DOMAIN, NONCE_KEY

of the plugins and categorized the database options among different use case categories. We used the existing categorization of WPengineer.com[12] for configurable constants (Table III), as inspiration for our categorization of database options.

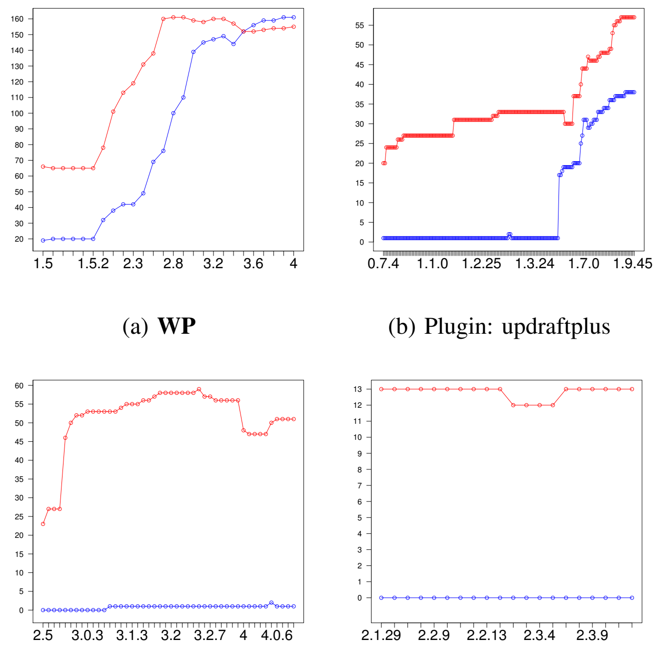
Overall, the database configuration options are those that are shown in a plugin's public web interface, i.e., they are meant to be changed and customized by plugin users via the administrator pages. For example, we found a commit that removed a configuration option from the database[13], because it didn't have any UI anymore to change it. On the other hand, the constants, by definition, require changes to the code. While those constants conveniently can be overridden in the *wp-config.php* file, they require manual exploration of the source code to be detected and to understand the default value.

As presented in Table II, we identified six database option categories. The first category represents the *"general configuration options"*, which refer to configurations used all over a website, such as the *blogname* or *siteurl*. The second category corresponds to *"writing"* options used to write the website pages and posts, such as the option *use_smilies*, which is a boolean variable used to display emoticons as graphic icons. The third category is *"reading"*, which presents all options related to displaying the posts, such as the number of posts per page (option *posts_per_page*). The fourth category corresponds to *"discussion"* options for the articles published on the website, such as *comment_registration*, which is a boolean variable used to decide whether commenters need to be registered. The fifth category corresponds to *"media"* options, i.e., the allowed dimensions of images. The last category corresponds to *"permalinks"*, which allows to customize the URL structure of blog posts and archives.

On the other hand, if we compare to the categorization by WPengineer.com for configurable constants in WP (Table III), configurable constants instead tend to manage more technical behavior, or correspond to configuration options that do not change regularly. For example, the general category contains technical options such as *DOING_AUTOSAVE*, which is used to specify whether *"WordPress is doing an autosave for posts"*[12]. There are other configurable constants that refer to development tasks, such as a category for debug mode, security, paths, directories, and links.

(RQ2) How does configuration mechanism usage evolve across time in each layer?

Motivation. The goal of this research question is to understand whether WP layers change the mechanism used to specify configuration options, or whether new options tend to prefer one mechanism or the other. It also sheds light on whether the number of configuration options plateaus early on, or whether the number keeps on increasing. The latter case would not only result into more options, but also potentially into more interactions between options, which could introduce more configuration errors.



(a) **WP** (b) Plugin: updraftplus

(c) Plugin: Google XML Sitemaps (d) Plugin: Redirection

Figure 4: Evolution of the number of configuration options for both mechanisms across WP and the studied WP plugins versions, i.e., database (red) and configurable constants (blue), Our full results are online[15].

Approach. For both configuration mechanisms, we use the same approach as in RQ1 for each plugin's version.

To analyze the results, we performed a number of activities. First, we used SLOCcount [22] to compute the number of lines of code of WP and its plugins to find if the increased number of options is related to large amounts of code (and hence features) being added. We also analyzed the names of the configuration options added in those versions with the goal of understanding whether or not the configuration options are related to new features. Finally, we also used the tool DiffMerge[14] to compare the source code of a version introducing many configuration options with its predecessor.

Results. **For 60% of the plugins, the number of configuration options grows across time for both mechanisms.** As presented in Figure 4 (the full plots for all plugins can be found online[15]), the number of database options in the last version of a plugin is higher than in the first version, except for the plugin *Redirection* where the number remained the same. Similar observations hold for the configurable constants, except for those of the six following plugins (not shown): *MailPoet, BBPress, NextScripts, Captcha, Flyzoo*, and *Redirection*.

As we saw for RQ1, the number of configuration options stored in the database is higher than the number of configurable constants, and Figure 4 shows that this number remains higher across all versions of the plugins, except for WP, where

[12]http://wpengineer.com/2382/wordpress-constants-overview/
[13]https://core.trac.wordpress.org/changeset/27916
[14]http://www.diffmerge.net
[15]http://mcis.polymtl.ca/∼msayagh/Paper/SCAM15/Figures/

the number of database options eventually dropped below the number of configurable constants.

Each plugin, except for *Redirection*, sees growth in its usage of at least one of the two configuration mechanisms. For WP, the number of options grows rapidly for both mechanisms, similar to the plugin *updraftplus* and *WP-Members*. For the other plugins, the number of configuration options stored in the database grows more rapidly than the number of configurable constants. The *Redirection* plugin sees no growth, and temporarily even lost one database configuration option between version *2.3.2* and version *2.3.5*.

For some versions, we can observe that the number of configuration options has an important decrease. For example, the number of configuration options decreases between versions *3.4.1* and *4* of the plugin *Google XML Sitemaps* for database configuration options. The same happens between versions *1.1* and *1.2.2* of the plugin *broken-link-checker*.

Discussion. To understand the above findings better, we studied the Spearman correlation between the evolution of the number of configuration options and the size (i.e., number of source code lines) of WP and plugins versions. Table IV shows that the correlation results are strong for WP and all plugins cases, except for *MailPoet* (moderate) and *Redirection* (moderate and negative). In the latter case, the number of configuration options is stable across all versions except two, one of which shows a decrease. We also plotted the number of options per line of code[16], which showed us that nine plugins have a decreasing trend, i.e., the size of these plugins increases more rapidly than their number of options. On the other hand, *Download Manager, NextScripts, Page Builder by SiteOrigin*, and *WP-Members* proportionally add more options than they increase in size and features, while *Flyzoo* and *Redirection* more or less retain a constant proportion.

As examples of these strong correlations, we found that between versions *2.0.3* and *2.1* of the plugin *BBPress*, its developers added 11 configuration options, since many features were added, represented by 7738 new lines of code. Just the addition of the component *bbConverter*[17] to the plugin introduced seven configuration options. In the plugin *all-in-one-seo-pack*, its developers added the following components: *Sitemap* and *Social Meta module*. Furthermore, some information was made configurable in the plugins *WP-Members* and *The Events Calendar*. For example, instead of hardcoding the different parts of an email like the body, or the mail footer, these now became configurable.

The growth of the number of configurable constants is due to making additional constants configurable or adding new components (containing configurable constants). By analyzing the difference between versions *2.8.10* (2 configurable constants) and *2.9.0* (16 configurable constants), we found that version *2.9.0* makes certain PHP constants of version *2.8.10* configurable, by testing if they are defined before their existing definitions (cf. Fig. 2). For the *updraftplus* plugin, we found that newly added components have 16 configurable constants.

Table IV: Correlation between number of configuration options and number of lines of code of WP and the analyzed plugins.

WP/Plugin	Correlation
Flyzoo	0.9933902
The Events Calendar	0.9884512
NextScripts	0.9860491
Download Manager	0.9814041
broken-link-checker	0.9762101
WP	0.9643059
updraftplus	0.9582439
BBPress	0.9447637
Captcha	0.937967
WP-Members	0.9121678
Google XML Sitemaps	0.8971884
all-in-one-seo-pack	0.8252223
Page Builder by SiteOrigin	0.728631
wp-pagenavi	0.7014725
MailPoet	0.491287
Redirection	-0.3714881

There is one important case of decrease of configuration options between versions *3.4.1* and *4* of the plugin *Google XML Sitemaps*. Analysis of the source code showed the following comment: ”*restores some default options which were not needed anymore in v4.*”.

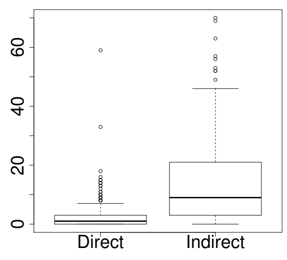
(RQ3) How many configuration options defined in lower layers are used by WP plugins?

Motivation. Now that we better understand the scale and evolution of configuration option usage within layers, RQ3 analyzes the usage across different layers (see Figure 1). Such usage potentially can be error-prone, since configuration options of lower layers need to be defined in a different location than the plugins' options, and a change to the value of a lower layer option could impact multiple plugins at once. This RQ focuses on the plugins' usage of lower layer options, whereas RQ4 measures the amount of lower layer options used by more than one WP plugin.

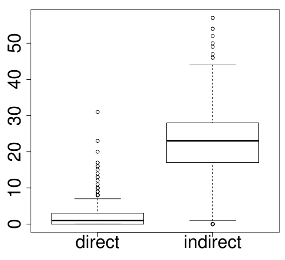
Approach. We calculate the plugins' direct and indirect usage of WP and PHP options using the approach of Section III-D and III-E. For database options of WP and for PHP options, we split up ”usage” of an option into reading and writing (based on the names of access methods), whereas for constants we only consider reading (since by definition a constant can only be defined once). Note that for RQ3 and RQ4 we use the Large Data Set.

Results. **Each plugin reads on average 1.49% to 9.49% of all WP database options, and 1.38% to 15.18% of all WP configurable constants.** While WP plugins read on average 2.32 WP database options directly, they read 13.73 options indirectly, corresponding respectively to 1.49% and 9.49% of the WP database options. Similarly, they read on average 2.41 WP configurable constants directly and 22.22 ones indirectly (i.e., 1.38% and 15.18%). Figure 5 confirms that the plugins read more configuration options indirectly than directly (both database options and constants). Note that the set of direct reads is not a subset of the set of indirect reads (although

[16]http://mcis.polymtl.ca/∼msayagh/Paper/SCAM15/LOC_vs_Options/
[17]https://wordpress.org/plugins/bbconverter/

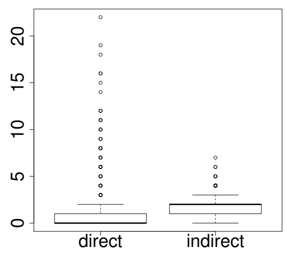


(a) Database options

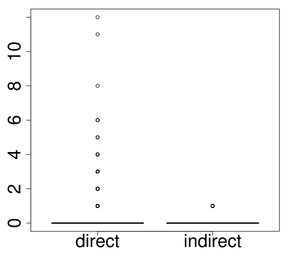


(b) Configurable constants

Figure 5: The number of WP options read by plugins.



(a) Reading



(b) Writing

Figure 6: The number of PHP options used by plugins.

overlap is possible), since options in the latter category need to be read at least once in an indirect way to be considered as indirectly read.

The maximum number of WP database options read directly by one plugin is 59 (*JetPack* plugin), whereas for the indirectly read ones it is 57 (*Worker* plugin). The maximum number of WP configurable constants read directly by one plugin is 31 for *BuddyPress*. The plugin *JetPack* indirectly reads 70 configurable constants, which represents the highest number of configurable constants read indirectly.

Each plugin directly writes on overage 0.11 WP database configuration options, and indirectly on average 0.32. While 441 plugins out of 484 do not directly write any configuration option, 35 plugins directly write one WP database option, five plugins write two WP database options, two plugins write three, and only one plugin writes five.

89 plugins (18% of all the plugins analyzed) indirectly write one or more WP database configuration options, where the maximum number of written WP configuration options is nine.

The plugins also read on average 1.30 PHP configuration options, and write 0.40 options, whereas WP reads 20 PHP options, and writes 8 options. Figure 6 shows that the plugins read the PHP options more indirectly than directly. It also shows that the number of writes is low (median of 0).

Discussion. Database options and configurable constants are mostly being read by their own plugins. This can be seen by the fact that the plugins use just 1.42% of the WP database options directly, and just 1.49% of the configurable constants. Moreover, few PHP configuration options are read by plugins, just 1.30 on average, and few of them (0.40) are modified. The use of PHP configuration options by WP is also negligable.

Although this seems good news, the number of configuration options used indirectly is significantly higher than the number of options used directly. Hence, the modification of one configuration option could impact the behavior of many plugins at once, possibly without developers being aware (since the dependencies are indirect). Figure 5 also shows that configurable constants are used more indirectly than database options. We study this in more detail in the next RQ.

To better understand the degree to which cross-layer configuration option usage could cause problems to users in terms of understanding and maybe errors, we analyzed the StackOverflow and WP Exchange fora. Out of 60,502 StackOverflow conversations related to only WP (not plugins), 5,907 (9%) are related to configuration issues, and 816 of these 5,907 conversations (13%) are related to PHP configuration options. Stackoverflow also contains 8,417 conversations related to WP plugins, where 679 conversations (8%) contain at least one WP or PHP configuration option, i.e., are related to multi-layer configuration issues. The WP Exchange forum[9] contains 9,756 conversations related to WP configuration issues out of 46,509 (21%), where 504 (5%) are related to PHP configuration options. From the 8,426 conversations related to plugins in the WP Exchange forum[9], 1,375 conversations (16%) are related to WP or PHP configuration.

Although the percentages of plugin conversations related to multi-layer configuration (8% and 5%) seem low, it is important to keep in mind that we do not know the number of plugin conversations talking about configuration in general (as the plugins' options follow different naming conventions). Since this number will be much lower than 8,417 or 8,426, the percentage of *configuration* discussions that consider options across layers will be much higher than 8% or 5%.

Therefore, an important percentage of conversations in both fora is related to multi-layer configuration issues, which suggests that it is an important issue for WP users.

(RQ4) How many plugins share the same configuration options of lower layers?

Motivation. In this research question, we analyze interference between plugins caused by dependency on a common configuration option of WP or PHP. Such interference could indicate risky configuration options of lower layers that might impact many plugins at once.

Approach. Similar to the previous research question, we measure both direct and indirect configuration option usage. Since here we are interested in any usage of configuration options by one or more plugins, we do not distinguish between reading and modification of options, but merge both into ”usage”. For our study of Spearman correlations of forum conversations, we merged the data of both fora into one.

Results. **78.88% of all WP configurable constants and 85.16% of all WP database options are used by at least two plugins.** In Figure 7, 25 out of 161 WP configurable constants are used directly by more than 10 plugins, while

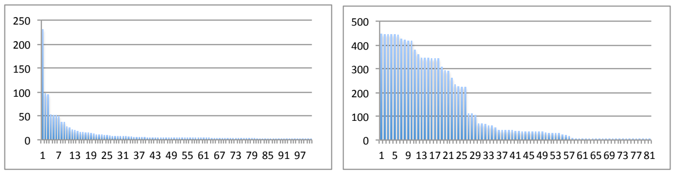

(a) direct (b) indirect

Figure 7: The number of plugins (Y axis) using a given configurable constant (ordered on the X axis).

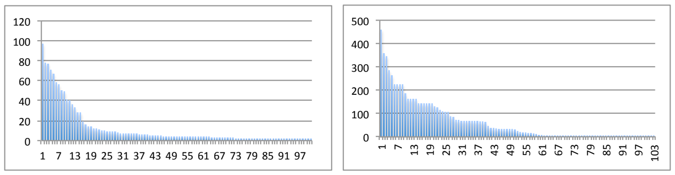

(a) direct (b) indirect

Figure 8: The number of plugins (Y-axis) sharing the same database configuration option (ordered on the X axis).

Table V: The correlation between the number of plugins using an option and the number of conversations mentioning it.

Configuration options of	Case	Correlation
WP configurable constants	Direct use	0.5533475
WP configurable constants	indirect use	0.2268492
WP Database options	Direct use	0.3643119
WP Database options	indirect use	0.2795615
PHP configuration options	Direct use	0.5593974
PHP configuration options	Indirect use	0.4427347

75 configurable constants are used indirectly by more than 16 plugins (29 even by more than 104 plugins). The highest number that we found were the 231 different plugins that directly use the configurable constant *ABSPATH*. For indirect usage, *WP_DEBUG* is used by 447 different plugins out of the 484 plugins in the Large Data Set, and the *VHOST*, *MULTISITE*, and *SUNRISE* options are used by 445 different plugins. At the other extreme, 21 configurable constants are used directly by just two different plugins, and five are used indirectly by two different plugins.

Turning now to WP database options, Figure 8 shows how 24 out of 155 database options are used directly by more than 10 plugins, whereas 59 are used indirectly by more than 10 plugins. The option used directly by the highest number of plugins is *active_plugins*, which is used by 97 different plugins out of 484 plugins, and the second most important configuration option is *siteurl* (used by 78 plugins). The option *blog_charset* is used indirectly by the highest number of plugins (458 different plugins). At the other extreme, 28 WP database configuration options are used directly and seven ones are used indirectly by only two different plugins.

52 PHP configuration options are used by at least two different plugins. In Figure 9, 13 PHP configuration options are used directly by more than 10 plugins, while five PHP configuration options are used indirectly (across WP functions) by more than 28 different plugins. The PHP configuration *memory_limit* is used directly by 75 different plugins, *safe_mode* is used by 71 plugins. However, the most used options are *arg_separator.output* and *mbstring.func_overload*, which are used respectively by 300 and 297 different plugins. There are also three PHP configuration options used by just three different plugins, and nine PHP configuration options that are used indirectly from the WP source code.

Discussion. As 101 (62%) out of 161 of WP configurable constants are used directly and 81 (50%) are used indirectly by at least two different plugins, the modification of a configurable constant inside the WP layer might impact many other plugins indirectly. For example, the configurable constant *ABSPATH* is used directly by 47% of the plugins studied, and indirectly by 78%. If its value would become corrupt, it could impact the behavior of at least 78% of the plugins.

Similarly, there are some WP configuration options stored in the database that are used by many plugins, which means that their modification could impact the plugins' behavior as well. For example, *blog_charset* is used indirectly by 94% of the plugins studied. An error with this option could impact what the plugins display on the screen. The above of course holds for the PHP configuration options as well, which come from an even deeper layer.

We found that there is a weak to moderate correlation (between 0.22 and 0.55) between the number of times a WP configuration option is being used directly by plugins and the number of conversations discussing the option, as shown in Table V. The correlation results for indirect usage of WP configurable constants and WP database options are weaker. All correlations are positive, hence the more plugins use an option of a deeper layer, the more discussions there are about the option, hence the more information people require about it. This suggests that the reuse of configuration options of deeper layers can pose problems for WP users and hence requires more research.

V. Threats to Validity

Regarding threats to external validity, since we analyzed only the WP ecosystem, and within this ecosystem only a limited amount of plugins, we cannot generalize the results to other systems. However, our results provide a first large analysis of the use of configuration options in multi-layer systems, since in total we considered 484 plugins, 15 of which

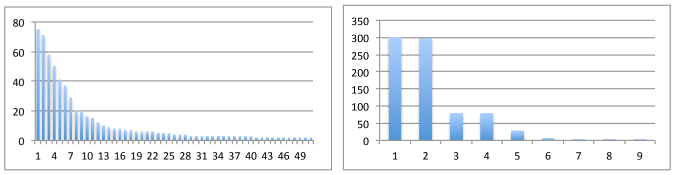

(a) direct (b) indirect

Figure 9: The number of plugins using the same PHP configuration option (ordered on the X axis).

were manually analyzed across time. In future work, we plan to analyze other multi-layer systems, in the same as well as other domains.

Regarding threats to internal validity, we analyzed only 15 plugins for the first and the second research question due to the manual analysis required, especially due to the criterion for literals in the method calls used to access the plugins' database options for RQ1 and RQ2, and the need to manually find these methods. However, these plugins are all popular plugins from a variety of organizations and domains.

VI. Conclusion

Multi-layer systems like WP have a potential for configuration errors due to interference between configuration options in different layers. As a first step towards analyzing such errors, this paper performed an empirical study on the prevalence of multi-layer configuration options in WP, WP plugins and the PHP system. We found that except for WP itself, WP plugins prefer storing configuration options in a database, in order to make them easily available to the end user for configuration. Furthermore, configuration options and usage evolve across time, especially when new features are added. Across layers, we found that each plugin uses on average 1.30 PHP configuration options and modifies 0.40 options, while it uses on average 1.49% to 9.49% of all WP database options and more than 1.38% to 15.18% of all WP configurable constants. Furthermore, 78.88% of all WP configurable constants and 85.16% of all WP database options are used by at least two plugins at the same time, which can be between two and 447 plugins for configurable constants, two and 458 plugins for WP database configuration options, and between two and 300 plugins for PHP configuration options.

Finally, there is more indirect use of configuration options than direct use, which could make the detection and fixing of configuration errors more difficult. We indeed found initial evidence of this potential through the relatively high percentage of conversations in Stackoverflow and WP Exchange fora talking about options of deeper layers. Hence, we would suggest Wordpress to provide a mechanism to warn plugin developers or users for the impact of cross-layer configuration modifications. Future work needs to build on these results to help users detect and fix configuration problems across multiple layers.

References

[1] S. Zhang and M. D. Ernst, "Which configuration option should i change?" in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 152–163.

[2] Z. Yin, X. Ma, J. Zheng, Y. Zhou, L. N. Bairavasundaram, and S. Pasupathy, "An empirical study on configuration errors in commercial and open source systems," in *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, ser. SOSP '11. ACM, 2011, pp. 159–172.

[3] F. Arshad, R. Krause, and S. Bagchi, "Characterizing configuration problems in java ee application servers: An empirical study with glassfish and jboss," in *Software Reliability Engineering (ISSRE), 2013 IEEE 24th International Symposium on*, Nov 2013, pp. 198–207.

[4] A. Hubaux, Y. Xiong, and K. Czarnecki, "A user survey of configuration challenges in linux and ecos," in *Proceedings of the Sixth International Workshop on Variability Modeling of Software-Intensive Systems*, ser. VaMoS '12. New York, NY, USA: ACM, 2012, pp. 149–155.

[5] D. Jin, X. Qu, M. B. Cohen, and B. Robinson, "Configurations everywhere: Implications for testing and debugging in practice," in *Companion Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE Companion 2014. New York, NY, USA: ACM, 2014, pp. 215–224.

[6] X. Xia, D. Lo, W. Qiu, X. Wang, and B. Zhou, "Automated configuration bug report prediction using text mining," in *Computer Software and Applications Conference (COMPSAC), 2014 IEEE 38th Annual*, July 2014, pp. 107–116.

[7] L. Keller, P. Upadhyaya, and G. Candea, "Conferr: A tool for assessing resilience to human configuration errors," in *Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008. IEEE International Conference on*, June 2008, pp. 157–166.

[8] S. Zhang, "Confdiagnoser: An automated configuration error diagnosis tool for java software," in *Software Engineering (ICSE), 2013 35th International Conference on*, May 2013, pp. 1438–1440.

[9] C. Elsner, D. Lohmann, and W. Schroder-Preikschat, "Fixing configuration inconsistencies across file type boundaries," in *Software Engineering and Advanced Applications (SEAA), 2011 37th EUROMICRO Conference on*, Aug 2011, pp. 116–123.

[10] M. Attariyan and J. Flinn, "Automating configuration troubleshooting with dynamic information flow analysis," in *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, ser. OSDI'10. USENIX Association, 2010, pp. 1–11.

[11] R. Tartler, D. Lohmann, J. Sincero, and W. Schröder-Preikschat, "Feature consistency in compile-time-configurable system software: Facing the linux 10,000 feature problem," in *Proceedings of the Sixth Conference on Computer Systems*, ser. EuroSys '11. ACM, 2011, pp. 47–60.

[12] B. Wang, L. Passos, Y. Xiong, K. Czarnecki, H. Zhao, and W. Zhang, "Smartfixer: Fixing software configurations based on dynamic priorities," in *Proceedings of the 17th International Software Product Line Conference*, ser. SPLC '13. ACM, 2013, pp. 82–90.

[13] Y. Xiong, A. Hubaux, S. She, and K. Czarnecki, "Generating range fixes for software configuration," in *Software Engineering (ICSE), 2012 34th International Conference on*, June 2012, pp. 58–68.

[14] M. Lillack, C. Kästner, and E. Bodden, "Tracking load-time configuration options," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. ACM, 2014, pp. 445–456.

[15] S. Nadi, T. Berger, C. Kästner, and K. Czarnecki, "Mining configuration constraints: Static analyses and empirical results," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. ACM, 2014, pp. 140–151.

[16] A. Rabkin and R. Katz, "Precomputing possible configuration error diagnoses," in *Automated Software Engineering (ASE), 2011 26th IEEE/ACM International Conference on*, Nov 2011, pp. 193–202.

[17] D. Jin, M. B. Cohen, X. Qu, and B. Robinson, "Preffinder: Getting the right preference in configurable software systems," in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: ACM, 2014, pp. 151–162.

[18] S. K. Patel, V. Rathod, and J. B. Prajapati, "Performance analysis of content management systems-joomla, drupal and wordpress," *International Journal of Computer Applications*, vol. 21, no. 4, pp. 39–43, 2011.

[19] H. V. Nguyen, H. A. Nguyen, T. T. Nguyen, A. T. Nguyen, and T. Nguyen, "Dangling references in multi-configuration and dynamic php-based web applications," in *Automated Software Engineering (ASE), 2013 IEEE/ACM 28th International Conference on*, Nov 2013, pp. 399–409.

[20] L. Eshkevari, G. Antoniol, J. R. Cordy, and M. Di Penta, "Identifying and locating interference issues in php applications: The case of wordpress," in *Proceedings of the 22Nd International Conference on Program Comprehension*, ser. ICPC 2014. ACM, 2014, pp. 157–167.

[21] H. V. Nguyen, C. Kästner, and T. N. Nguyen, "Exploring variability-aware execution for testing plugin-based web applications," in *Proceedings of the 36th International Conference on Software Engineering*, ser. ICSE 2014. New York, NY, USA: ACM, 2014, pp. 907–918.

[22] D. A. Wheeler, "Sloc count user's guide," 2004.