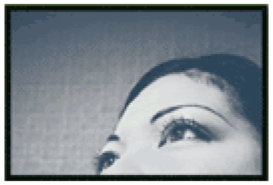
Review

Grant
COMMUNICATIONS LLC



Site Assessment™

Rx WordPress & Third Party CMS Systems

Components Affecting Organic Search Engine Performance

Updated: June 26, 2014

Get Results

WordPress Blog and other CMS (Content Management Systems)

General Observations as a Result of our Recent 450 Tests:

Reverse-Engineering Google:

One of the foundational reasons for us to embark on a year-long study of Google was our client network performance after the Panda and Penguin updates, and specifically the drop-off of organic rankings for those clients utilizing a third party CMS (Content Management System).

From the WordPress website; "*WordPress is web software you can use to create a beautiful website or blog.*" The foundation to CMS software is the blog application itself. The word "blog" comes from "Web Log", or diary, of daily observations posted onto the web for comments and feedback. Consider this as one of the origins of FaceBook, which acts as a central controller for people to post their daily logs. Google was one of the original early adopters of this platform with their "Blogspot" application, controlled through Google itself. So having a blog as your website isn't inherently bad…

We have a number of clients for whom we developed or maintain their WordPress, Joomla and other third partyCMS. For the most part, their overall performance on Google is lackluster at best, even with advanced Dublin Core and OpenGraph code support and complex inbound link structures. **Why?**

For the most part, the clients originally intended to constantly update their blog with news, product releases, case studies and other typical documents that provided additional support for both their clients and prospects. After consulting with them, we advised the clients that this would be highly unlikely; the average site on our network is only changed once or twice per year, and are generally personnel or sales rep related. Additionally, we shared our experience with static as compared to third party CMS system organic Google performance on the Search Engine Ranking Pages (SERPs), and cautioned them as to the probable impact on their Google organic placement when their static presence went away.

What Did We See?

For the most part, limited changes can be made to some of the code and physical file asset elements due to WordPress and other 3rd Party CMS software limitations. In many cases, core 3rd party and/or Widget security updates will overwrite file changes, negating your efforts. This can (and will) directly affect your ability to control search engine placement without careful adherence to the standards you CAN control. Chief among controllable elements is the element **4.55-Last Date Text/ Content Added**.

How does this affect a WordPress (or other CMS system) website?

Google keeps track of last updates and size of changes. If a blog's text content is updated fairly regularly (we're tracking a minimum of once a month minimum), you're OK. Many of the elements you can't control because of the platform are weighted, so you won't be overly impacted by the use of the blog *in the way it was originally introduced*. Just updating images (that may not be properly compressed anyways) doesn't count.

Using the software for adding new content (see prior notes on update frequency), or to minimize the cost of site development and ongoing administration (usually for the reason “to give us internal control for *large updates in the future”)*, Google will see this and respond accordingly. Google appreciates the time (or resources) invested in making the web a compelling media. This compelling web makes it easier for Google to sell ads, the basis of their business model. When it sees a blog with the last update posted two years prior, and using a canned theme that has been used and reused hundreds or thousands of times, it sees a lessening of the value of the web. Consider this as an alternative to a ‘duplicate content’ penalty.

Again, we rarely ***if ever*** see any regular updates to typical corporate sites. Company product lines are stable, and no one has the time to write up the extra content that Google expects if you’re using a CMS system.

At the completion of our study, we reviewed the eight main areas of website performance and the elements contained within each area. Using analyzed metrics and adding inferential logic, we developed a target list of the specific areas and elements that wouldn’t comply with the Google PageSpeed tool or their advanced WpTest modeling system, ***based on Google’s own definitions***. We then analyzed against static sites; in some cases, using the clients original static assets before redeployment. Our negative initial reactions to CMS development weren’t our viewpoint anymore; this was from measured and analyzed areas and elements, ***using Google’s own toolsets***.

Finally, we ran ranking reports on the sites core key phrases, then compared to the site rankings prior to being converted from a static site to a database-driven application. In all cases where content updates were limited to non-existent, SERP performance had declined dramatically, even after we had added advanced ‘best of breed’ server-side code and other support programs. Those clients who posted on a bi-weekly or more frequent basis did not see the same decline.

Conclusion

Our conclusion: If a CMS system is employed and used in the manner in which the application was originally developed, negative element performance was discounted and body element weightings were elevated. Specific element considerations with explanations follow. Conversely, if the CMS was neglected, full negative weightings to each element that wasn’t in compliance takes effect. To improve performance, the site owner must initiate ongoing campaigns to offset the negative elements through costly and time consuming inbound link campaigns and Social media interaction

On the following are the specific elements identified from the Google testing suites, listed by area affected.

Partial Listing

Body Content

4.4-Text to Code
To allow a CMS to be extremely adaptable to the needs of different markets, just about every contingency in terms of layout and Widget addition flexibility has already been added. For everyone, whether you use most of the available objects or not. This creates "**code bloat**" automatically.

4.5-HTML Requests
As above, every contingency for modern websites has been prebuilt. Every available resource will continually be loaded, in the event you enable some new modules. Or not. All of this capability needs to be loaded with the first page, module by module. Add to this the Widgets, or mini programs and applications, that run concurrently wit the blog itself. These operate from javascript and CSS platforms that integrate seamlessly into the blog. Unfortunately, these all require additional requests to run, many from the originating Widgets server (see **1.30-Serve resources from a consistent URL** for more details).

4.37-Missing ALT Tags
Even when updating sites regularly, changing from the default empty image ALT tag is difficult to remember. How many times has the key phrase already been inserted into ALT tags on this page? And what phrase was assigned to this page in the first place? Aside from understanding the HTML code view in WordPress, if an ALT tag generator hasn't been installed.

Server

1.6-Page Fully Loaded
Due to the need to be adaptable for just about any possible site configuration, thousands of related files are served either on initial load, or as an application is called for as a DOM or HTTP request. This automatically increases overall load time by default.

1.23-Avoid bad requests
During initial CMS development, different plugins or Widgets, will be tested for compatibility with the system as well as what will fit client requirements. Upon final approval, the original test files remain as orphans that may be preloaded by the CMS.

1.29-Minimize request size
Many developers for CMS systems typically provide their applications for free as a way to gain individual reputation. This can be monetized in a variety of ways. If it was developed for free, limited time was spent in minimizing the necessary resources. If a paid subscription, we have noted more attention is paid to the details of load time, minification and overall size.

1.30-Serve resources from a consistent URL
Widgets and specific CMS application controls are served from their respective servers, which circumvents this Google PageSpeed minimum standard.

Media & Compression

6.10- CSS Files
6.15- Javascript Files
Widgets each call for their own CSS and JS files for display and print, due to their need for updates.

6.36- Minify HTML
The core WordPress files that are rendered on a server are prebuilt templates using AJAX (Asynchronous Javascript and XML) in processing core PHP scripts interacting with a database. Any modifications to the source inflates the probability of the site becoming disabled. After an update, the core files will have all reverted to their original syntax anyways, making this irrelevant.

6.42- Combine images into CSS sprites
Highly ineffective, due to the scattering of both common and Widget files.

6.48- Minify CSS
6.54- Minify Javascript
Any changes made to Widget files would be overwritten during the widgets next update, removing minification.

6.43- Defer Parsing of Javascript
This is the big drawback. WordPress is built on the AJAX core, and requires Javascript calling to jQuery to render the administrative interface and database hooks to be set in the <head> area of the code to allow the blog to be operable. Google prefers javascript to run at the end of the document, directly above the closing </body> code.

6.23- Total Image Size
The current standard WordPress developer isn't concerned with file size, due to broadband connectivity. Additionally, the requisite software requires additional expense and training, costing most of a projects profitability. Then, site owners or WordPress hosting companies won't divulge FTP credentials to the server due to security protocols, making any newly compressed image uploading and hooks twice as involved.

Code & Architecture

2.6-Total Page Size (All Files)
2.14-DOM Elements-
2.17-# HTTP Requests-
2.22-Static HTML
2.24-Avoid CSS @import
2.25-Avoid landing page redirects
2.27-Remove query strings from static resources

Internal Links

5.18-Phrases in Navigation links
5.23-Broken Link %
5.25-Unfriendly Link %

Meta Code

3.29-Dublin Core Tags