Adam Trachtenberg
Managing Innovation
December 4, 2002

The battle for middleware: PHP versus the world

Introduction

In 1996, PHP began as an individual's personal hobby. Today, it is the most popular middleware language on the Internet. Web servers on almost 10,000,000 unique domains support PHP.[i] This number is greater than for any other language. This paper traces the environmental and strategic reasons behind PHP's rise.

Middleware sits between (i.e. in the middle of) the web server and all other backend applications, like databases, user authentication systems, and e-mail programs. It enables a site to move from fixed static pages, which return the same content to every person, to a dynamic application, capable of using programming logic to produce customized content for the user. All major web sites require middleware.

While many languages can function as middleware, this paper focuses on the five most popular competitors: ASP/VBScript, ColdFusion, Perl, PHP, and Java.[1] In particular, it focuses on PHP, an open source language that has no commercial backing. There are no

[1] I am most deserving of the ire from Python/Zope users, but while there has always been a loyal community of Python programmers, I do not believe that there was ever a time when Python was seriously considered as a possible choice as the dominant web programming language. Also, ASP users are not required to use VBScript; however, this is the most popular language used with ASP.

full-time paid developers working on PHP, nor are there corporate marketers or strategists guiding its grow. This leads to an interesting question: How can a volunteer project out-compete Microsoft and Sun?

Two factors are largely responsible for PHP's growth: its position as a language that's both web-specific and open source, and the demand-side increasing returns captured by the early creation of an Apache web server specific version of PHP.

Language Design versus Language Ownership

Some programming languages are the equivalents of Swiss Army Knifes; they're the jack-of-all-trades solution to problem solving. You can use them to open wire bottles, cut rope, gut fish, and file nails. Others are like Le Creuset's Screwpull corkscrew. They cost over $200 and all they do is open wine bottles. However, they're really good at their job.

Open source languages provide the user with the complete source code to the language. Not only is the language free of charge, but the user also is granted the freedom to alter the language to suit their needs. Closed source languages, in contrast, fall under the control and development of a fixed entity, usually a corporation. The languages' future are controlled by corporate business strategy. If a new strategy clashes with a developer's future directions, this causes problems.

Graphing these languages on a two-by-two matrix, shows that only PHP is both a specialized web language and open source:

		Language Design	
		Specialized	Generic
Ownership	Open	PHP	Perl
	Closed	ColdFusion	ASP/VBScript Java

This unique positioning gives PHP an advantage over the other four languages. With web development in the mainstream, the benefits of PHP's specialized design outweigh its drawbacks. And, the edge provided by a polyglot middleware make open source a superior development method in this heterogeneous environment called the Internet.

Language Design

Many factors go into the process of selecting a language for a project. In an established environment, they center on existing knowledge and task suitability. Developers prefer to use a popular language that they program fluently. Additionally, it should be easy to solve the problem using this language.

If the task cannot be solved using this language, then a developer decides if this problem will be a frequent task. If so, then it's a worthwhile to learn the specialized language. If not, then she’ll struggle on and reuse the more generic language.

This trade-off assumes programming languages are static. After all, why can’t the generic language transform itself to work in the new environment? Actually, just like spoken languages, computer languages shift to accommodate new trends and can even "borrow" words and concepts from other languages and incorporate them into their lexicon.

However, all computer languages have an essential nature. They may be able to alter portions of their makeup, but they cannot undergo a complete revamp. There's a reason why nobody programs web sites in COBOL. A general language may be able to support a new area, but it cannot be as effective as a language specially designed for that field.

In this aspect, it is not a stretch to compare computer languages with corporations. Under certain innovations, existing corporations maintain control. Nevertheless, upstart firms can often undermine these companies and nimbly dance around them to capture value. Therefore, the question is: When is it better to be a specialized language and when it is better to be a generalized language?

To answer that question, we need a more complete understanding of the difference between the two forms. Two quotations best describe the thinking behind the design and objectives of a specialized language and a general one. The first quotation comes from Rasmus Lerdorf, the creator of PHP:

> PHP became popular because it eliminated most of the tediousness of writing CGI scripts.... The problem it solves is ugly. Ugly problems often require ugly solutions. Solving an ugly problem in a pure manner is bloody hard. PHP's aim is to make solving the web problem easy.[ii]

Larry Wall, the creator of Perl, sees this trade-off from another perspective:

> I think languages like awk and PHP hobble themselves in the long run by attaching themselves to a particular ecological niche, particularly when a generalist like Perl can effectively occupy the same niche. So I've never felt tempted to even try PHP.[iii]

Clearly, PHP is a specialized language and Perl is more general. Actually, ColdFusion and PHP are the only two web-specific languages. ASP/VBScript is a subset of Visual Basic, a Microsoft language for Windows programming. Perl is a text processing language. Java was created for programming cable set top boxes.

Now, the answer to the question of when should you specialize versus generalize is clear: it comes from the size of the market. Wall is incorrect when he claims "Perl can effectively occupy the same niche" because the "niche" of web programming and middleware is so large, it's a viable area in its own right. And, as discussed, a developer is far more likely to learn a new language in an emerging field because it strengthens her skill set.

Therefore, ColdFusion and PHP experienced an attacker's advantage. They customized themselves to solve the problem of web development. And, they weren't required to satisfy the needs of an existing community of programmers or restricted by pre-existing design choices.

Language Ownership

Open source languages provide the user with the complete source code to the language. Not only is the language free of charge, the user is also granted the freedom to alter the language to suit their needs. There is no commercial funding for the language; the community as a whole (with some self-selected leaders as a guide) must take a shared sense of ownership. But, there is always the freedom for a person to branch away from the collective and adapt the code to suite his specific needs.

Closed source languages, in contrast, fall under the control and development of a fixed entity, usually a corporation. The corporation is responsible for guiding the direction of the language, adding new features, and fixing bugs. The company pays the salaries of the developers, allowing those programmers the ability to devote their full attention to solving those problems. But, in exchange, the language's future is controlled by the company's business strategy. Then, when a new strategy clashes with a developer's future directions, this presents a problem.

Perl and PHP are open source; ASP/VBScript, ColdFusion, and Java are closed source. Open source languages are superior to closed source ones. Corporations need to capture value, so it is in their benefit to maintain tight controls over the language. But, despite their attempts at high appropriability, the presence of an equally good open source substitute negates much of the corporation’s ability to extract profits because open source languages eliminate the possibility of hold up and provide the developer with control.

Due to the nature of middleware, it is important for a developer that the language communicate with as many web servers and other backend applications as possible. Developers do not want the decision to alter one of those other components to be tied to a change in middleware because the existing middleware does not support the new component. Additionally, changing middleware is more expensive than changing any other backend component. In brief, this argument goes like this:

> Written in a computer language, middleware code is by its very nature unique to the choice of middleware platform. Shift to a new platform, necessitates a company to port the code to the new environment. This is a large switching cost. First programmers need to learn the new language. Once that is accomplished, they must spend many man-hours translating the code from the old language to the new. This is a non-trivial task middleware languages are completely different from each other in structure and idiom. To use the metaphor of human languages, the relationship among web languages is akin to Latin and Mandarin and Farsi and Basque instead of Spanish and French and Italian and Portuguese.

In *The Cathedral and the Bazaar*, Eric S. Raymond's first lesson of open source is: “Every good work of software starts by scratching a developer's personal itch.”[iv] This not only applies to new projects, but also to project extensions. So, if an open source middleware language doesn't speak to Oracle, a developer can write and contribute an Oracle module to do just that. And, if another developer wants to add in a new feature, he can code that function to go along with the rest. Closed source software, in comparison, does not allow its users to add new extensions. So, the number of web servers and databases and other programs it can communicate with will be limited to the number of people allocated to these problems.

Demand-side Increasing Returns

Computer programming languages exhibit the characteristics of demand-side increasing returns. In short, the more people who use a language, the more new programmers are willing to learn and use it. This occurs for many reasons:

Risk Reduction

This is a simple trust metric. It's the old argument: “If everyone is using it, it can't be bad.” In the case of web programming languages, a popular language demonstrates versatility. If both beginning programmers and high-performance sites use a language, then it satisfies multiple niches. So, if both parties are happy, developers feel their project is unlikely to run into problems. Also, popular languages are marketable skills.

Support

Receiving help and support is easier when more people use a language. Communications on web development are primarily through community mailing lists and web sites. A larger user base increases the chances that a question is answered. Additionally, more information is already available because others have already documented solutions to problems.

The larger the market, the more professional computer books will be published. This is necessary. While open source communities may produce high quality software, they tend to ignore documentation. And, corporations cannot self-publish their way into the documentation business. High quality documentation only comes through outside parties.

Complimentary Goods

Since the cost of copying computer code is zero, in a networked environment code sharing and reuse plays a critical role. Why reinvent the wheel, if someone will hand you a set of high-performance tires? Starting with a base of core components reduces a project's development time. A vibrant community increases code sharing; as language usage goes up, the more high-quality components become available.

Growth of Apache

The rise and fall of popular web servers is an interesting subject by itself. Over the last seven plus years, the top players have completely overturned. The companies that led the market are gone, while servers that did not exist before have seemingly entrenched positions.

In August 1995, the web server market had one major leader -- NCSA with 57% -- a second tier player -- CERN with 20% -- and a fractured set of alternatives -- all with less than a 4% share. Netscape and Apache had just launched and Microsoft had yet to introduce a web server of their own.

Seven years later, Apache has steadily grown and has a 60% share, while Microsoft's IIS has a 29% share. Netscape, after peaking with a 20% market share in late 1996, has slowly fallen and now has only 1.33% of the market. NCSA and CERN have a total combined share of 0.01%. [v]

So, despite the dominance of Internet Explorer on the desktop, Microsoft still has a long way to go to control the web server. So, with hindsight, Apache was the way to go. A savvy middleware could leverage Apache's growth to help spur its own growth. But, how can one language get a first-mover advantage over another?

The key comes from a performance limitation in the original CGI design. To solve this problem, web server programmers designed an interface allowing middleware systems to tightly couple themselves with the web server. This required work on the part of the middleware developer, but gave significant returns for end users. PHP was the first middleware solution to take advantage of Apache's module feature.

Lerdorf launched the Apache module of PHP in April 1996[vi]. Three months later, a Perl Apache module, known as mod_perl, was released. At that time, Perl was incrediably popular under CGI, far more so than PHP. So, were these three months, the crucial difference? No.

Unlike PHP, which is a specialized web language, Perl is a generic text processing language. So, to help ease web development in Perl, people used an external helper extension, called CGI.pm. Due to the pre-existing complexities in the language, CGI.pm was not made compatible with mod_perl until March of 1997.[vii] This made it difficult for people to convert their code from CGI to mod_perl because they needed to use a different extension. Additionally, mod_perl's architecture is not user friendly and imposes some restrictions on the code above the language defaults. Still in place today, these restrictions make porting code difficult, and also raise the barrier to entry for new Perl programmers looking to adapt mod_perl.

Next, there is Java. Despite a flurry of popularity, Sun largely positioned Java as a client-side language. Server-side use -- in the form of an Apache module -- also began in July of 1997, but did not launch a version 1.0 until two years later -- in June of 1999.[viii]

Additionally, Apache runs on both Unix and Windows, but its Unix origins enable superior performance on that platform. So, the majority of Apache sites are Unix-based and middleware that cannot run on Unix is at a sever disadvantage. In keeping with Microsoft's strategy, IIS only runs on Windows. ColdFusion has a similar Windows restriction. For this reason alone, neither of these two languages could compete with PHP.

Conclusion

PHP's growth into "the web language" came from its strategic positioning within the middleware market. As a specialized web-centric language, PHP helped developers solve common problems with a minimum of fuss. As an open source language, it reduced their risk of hold-up and provided developers with the valuable option of writing their own code in the case of a shift in direction. This combination helped easy adoption of PHP on a standalone basis.

Additionally, once the early adopters chose PHP, the demand-side increasing returns of middleware helped perpetuate PHP's success. Many of these returns come from the innate characteristics of middleware itself. However, its unclear if this would be sufficient to cross the chasm. So, PHP used a complimentary asset as a Trojan horse: Apache.

The first mover advantage of the PHP Apache module turned the spread of Apache into the spread of PHP. Installing Apache also meant installing PHP. End users, finding PHP pre-installed on their machines, chose the use the language, for the reasons outlined above.

PHP's position is a result of the union of all these forces.

[i] PHP Usage Stats. http://www.php.net/usage.php (25 Nov. 2002).

[ii] Lerdorf, R. Re: [PHP-DEV] short_open_tag. PHP Development listserv. php-dev@lists.php.net (15 Oct. 2002).

[iii] Wall, L. http://interviews.slashdot.org/article.pl?sid=02/09/06/1343222 (2 Dec. 2002).

[iv] Raymond, E. S. *The Cathedral and the Bazaar, Revised Edition*. O'Reilly & Associates (1999).

[v] Netcraft Web Server Survey http://www.netcraft.co.uk/survey/ (2 Dec. 2002).

[vi] Lerdorf, R. PHP/FI Apache Module Success! New Apache HTTP Daemon new-httpd@apache.org (18 Apr. 1996).

[vii] mod_perl: History http://perl.apache.org/about/history.html (2 Dec. 2002).

[viii] History of Changes. http://java.apache.org/jserv/changes.html. (2 Dec. 2002).