Short history of PHP troubles

The first release of PHP was in 1995 and it's usage grew rapidly. Regular version updates providing additional functionalities were pushed until 2014, when PHP 5.6 was released. But in 2014 it was already too late. PHP from the very beginning was very easy to use and learn and was a very forgiving language – which was both it's reason for great success and trouble on the other hand. In the years 2010 to 2016 there were more and more voices from the developers community that PHP failed to follow modern programming language practices and was not strict enough for developers to enforce a good, proper code. This also resulted in more and more backend programmers moving to ASP.NET and Python for web development. Additionally, due to internal conflicts around PHP6 (which aimed to provide Unicode support) ZEND development was behind current standards. But PHP was still the most used web language, and the community heard those voices and reacted. Because of wide PHP usage, it was quite difficult to develop a language up to modern standards while keeping backward compatibility, but the community faced those issues. Thus, after a long time, PHP 7 and then PHP 8 was released, providing answers to many troubles that were a pain for PHP developers.

Reasons for delay PHP6 and PHP7

There were two main reasons for delay in the new PHP version releases. The first one is that there was a branch called PHP6 which aimed to provide Unicode support albeit there were many conflicts around this version and the value that it provided was still very low. It's true that this caused languages such as Python or C# to be better in this aspect, but it's equally true that most programmers code in English anyway. In the end, PHP6 was abandoned, although it did cost the community a lot of time and effort that otherwise could have been used better.

The second reason for delay was backward compatibility. Since 80% of websites still use PHP as their backend language, there are dependencies that had to be taken into consideration during the development of PHP7. Thankfully, the developers did a great job and also this time, they allowed other solutions to appear, which helped to move from PHP 5.6 to PHP 7 – such as PHP FPM (first non-experimental release in 2011).

Main issues with PHP5

The main issue with PHP was that it was too easy to learn – which was both its strong and weak side. While the strong side is obvious, the weak side was that programmers without experience were able to build applications that were working but, due to bad reasoning and bad approach, were either unsecure or prone to errors. Not to mention the huge mess in the codes themselves.

Some specific problems that caused developers to consider PHP5 as obsolete and bad were:

- No ability to enforce function parameter types
- No ability to enforce function returns
- Bad exceptions handling
- No modern features, like spaceship operator, null coalesce, JIT ability
- Complicated operations on strings
- No native JSON handling (only by extension)
- Due to PHP5 was not forcing good code by design, a lot of bad code was created

It’s worth noting that all of the above and much more was addressed and fixed by PHP 7 and 8.

Frameworks and PSR to the rescue

Some of the issues that appeared in PHP5 were addressed by creating standards. The following frameworks were created: Symphony, Laravel, CakePHP, Codeigniter; they were almost immediately loved by the community. Frameworks enforced working within specific frames – like MVC or MVVC – so applications created using those frameworks were immediately better organized.

At some point PSR standards were created – you can read about them here: https://www.php-fig.org/psr/. Those standards included some elements that are enforced (like how to create classes) and some that were not enforced (like how to comment your code). Even though some of them were not enforced by the language itself, many editors (for instance Visual Studio Code) allowed to install plugins that helped developers almost automatically comply with these standards.

Now also AI can be used to improve code. While it still often produces unusable programs, it can be of great help when creating tests, automating comments or pointing out small issues. From our experience I can say, that if set up properly it can improve code and delivery of applications written in PHP and Reactjs.

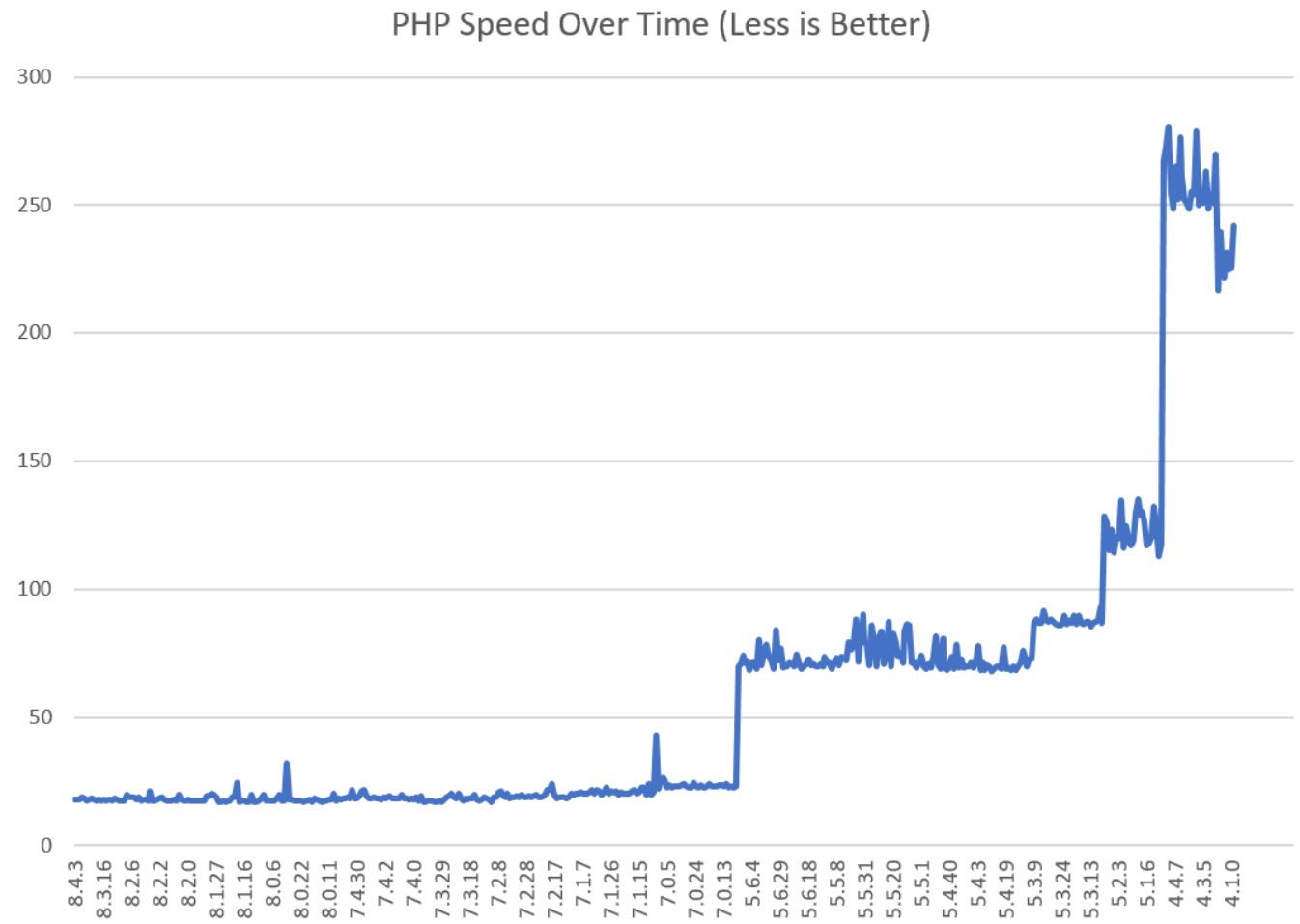
All of this combined resulted in a much better code overall. It is worth noting that in Sailing Byte we are using all four mentioned aspects (PSR, frameworks, editors with proper plugins and AI), which combined with in-house procedures and CI/CD makes it more than suitable for developing high quality online enterprise SaaS services.

PHP 7 and PHP 8 Speed and Memory Improvements

Moving to PHP8 allows developers to use much more functionality that resembles other languages. Some elements are very similar to C# or C++ or even scripting languages like Python. The full list of improvements can be found on Wikipedia, some of them I already mentioned in this article, but there's one more aspect.

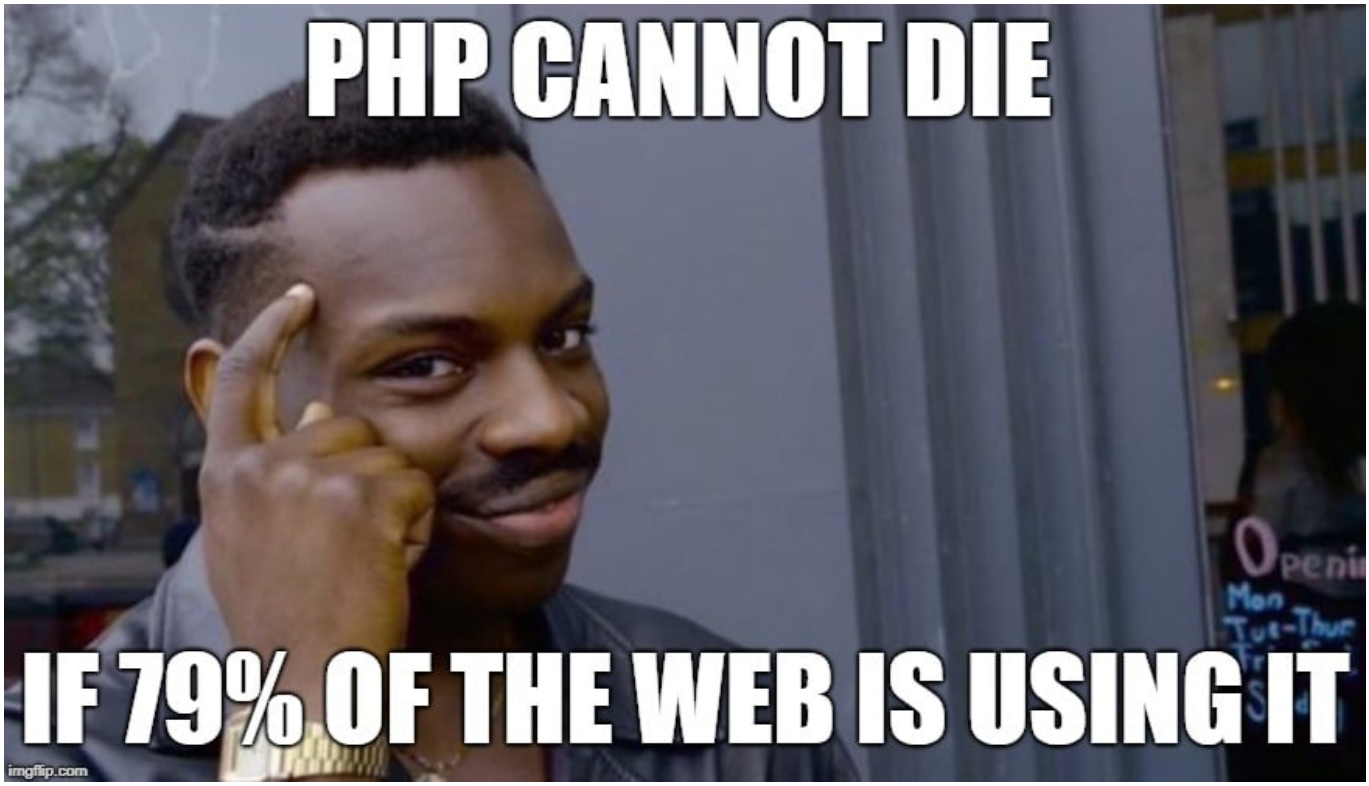
Between PHP5.6 and PHP8.1 there is a significant improvement in speed and memory usage. It was measured that PHP8.1 is 3 to 4 times faster than PHP5.6! Additionally, the amount of memory usage dropped by about 3 times. And PHP 8.4 is even faster, as you can see on chart below.

This is the result of rewriting ZEND – the PHP interpreter engine, written in C. Both code optimizations and better memory handling are currently a much more important factor than in the past, because end users expect websites to open almost immediately. Research results show that if a website is loading too slow, users are more eager to close the page without even viewing it's content. That's why speed improvement is such a big deal.



I have created this chart by using data from onlinephp.io where they are runing multiple tests on every PHP version.

Market Needs PHP Update and You Should Update Too!



Almost 79% of websites use PHP. So that's a big deal and it's currently impossible to imagine the web without PHP. Because of the great success of this language, for many more years we will need PHP developers – both those working on frameworks (Laravel, Symfony), and those working on CMS sites (like Joomla or WordPress).

What Can be Better in New PHP versions?

Of course, PHP is not a perfect language, it still needs some improvements. Even though there are already some improvements that are implemented, people don't know that they exist (like the ability to handle processes – so asynchronous execution is possible). What definitely could be improved is machine learning for AI – this is a field that is currently best handled by Python. Another area where PHP can be improved is adding native interfaces for IoT devices (since there are more and more of those) and also preparation for handling Web3.0 (smart contracts). What will actually be implemented? Future will reveal.

How to Update Without Breaking Website?

This process must be carefully planned. PHP rarely lives nowadays as stand-alone script: most probably you are using some framework (such as Laravel) or CMS (such as WordPress) and composer packages – to speed up development, be more secure and so on. If you want to keep all up and running, PHP and dependencies shoud be updated in specific order depending on what you use. Along with that you need to remember about components that live around all of this, such as Apache/Nginx, MySQL/MariaDB, PHP extensions and so on. The older your script is, the more complicated process could be – even to the point that it might be more efficient to rewrite everything from scratch than to upgrade it.

Where are we now with PHP 8.4?

If you are still not using PHP 8.4 or at least PHP 8.3, then you should probably read this article about business critical issues when using outdated software. Using outdated software is essentially a great risk to your business. You will also find some help regarding updating Laravel and WordPress. If you have not yet migrated to PHP 8.4, then maybe some of these improvements will help you to make your mind:

- **Property Hooks** One of the most significant additions is property hooks, which allow properties to define custom logic for reading and writing operations without traditional getter/setter methods.
- **Asymmetric Property Visibility** PHP 8.4 introduces asymmetric visibility, allowing properties to have different visibility levels for reading and writing operations.
- **Simplified Class Instantiation** The new without parentheses feature eliminates the need for wrapping new expressions in parentheses when chaining methods or accessing properties.
- **Enhanced Array Functions** PHP 8.4 adds four new array functions that simplify common data processing tasks:
  - array_find(): Returns the first element matching a condition
  - array_find_key(): Returns the key of the first matching element
  - array_any(): Checks if at least one element satisfies a condition
  - array_all(): Verifies that all elements meet a condition
- **Deprecation Attribute** The new #[Deprecated] attribute provides a standardized way to mark functions, methods, and class constants as

deprecated
- **Modern DOM API** PHP 8.4 introduces a new DOM API with HTML5 support through the Dom namespace. The new API provides spec-compliant parsing and modern convenience methods.
- **BCMath Object-Oriented Interface** The new BcMathNumber class enables object-oriented arbitrary precision arithmetic with standard mathematical operators.
- **Additional Enhancements** PHP 8.4 includes numerous other improvements:
  - Enhanced PDO: Driver-specific subclasses (PdoSqlite, PdoMySql, etc.) with tailored methods
  - Multibyte string functions: mb_trim(), mb_ltrim(), mb_rtrim(), mb_ucfirst(), and mb_lcfirst()
  - New BCMath functions: bcceil(), bcdivmod(), bcfloor(), and bcround()
  - DateTime enhancements: New methods for timestamp creation and microsecond handling
  - Improved rounding: New RoundingMode enum with additional rounding modes
  - Request parsing: New request_parse_body() function for parsing non-POST HTTP requests
- **Performance and Security** The release includes performance improvements, updated Unicode Character Database to version 16, increased default Bcrypt cost from 10 to 12 for enhanced security, and various bug fixes and general cleanup

So well.. what are you waiting for?

And What about PHP 9?

There is currently no release date and we might need to wait a bit. Probably before we hear about PHP 9, we will see PHP8.6 and 8.6. Although, we can anticipate some improvements that can be expected in near future. Those would probably be:

- More precise handling of different variable types (such as treating boolean and null as numbers)
- Elimitaiton of odd string increments
- Interpreting empty strings as numbers
- Better serialization handling
- Function signatures

- Stricter array creation rules
- Simplified string interpolation
- Some warnings may be now errors
- Of course removing deprecated features

So what’s the conclusion?

PHP language lives and is in good shape – better than ever. A lot of websites still use it and it does not seem that it will change in any way. PHP is a much more modern language than it was years ago when it had troubles – troubles that had been addressed. Because of that, there is still a lot of work for PHP developers and it is certain that it will not change in the future. New versions are regurarly released and each brings more improvements to current status, so it looks that we can expect even more improvements to something that is already great.