Санкт-Петербургский государственный университет

Программная инженерия

Группа 24.М71-мм

Эволюция PHP-фреймворков как отражение развития экосистемы веб-разработки (2005–2025)

Альшаеб Басель

Отчёт по учебной практике

Преподаватель:
Басков Антон Андреевич

Санкт-Петербург
2025

Оглавление

1. Введение

PHP традиционно играет важную роль в динамичном развитии веб-разработки в течение почти трех десятилетий. Согласно актуальной статистике W3Techs[?] по состоянию на 1 ноября 2025 года, PHP используется в 72.9% всех веб-сайтов с известным серверным языком.

PHP[?] стал основой для многих систем управления контентом и прикладных веб-решений благодаря низкому порогу входа и широкой доступности хостинга.

Тем не менее, устойчивость языка и его экосистемы на протяжении столь длительного периода в значительной степени связана с развитием общих архитектурных практик и стандартов, которые определили индустрию PHP[?]-разработки.

В начале 2000-х годов появились первые PHP[?]-фреймворки. CakePHP (2005)[?], Symfony (2005)[?] и CodeIgniter (2006)[?] поставили основы MVC-подхода и организовали разработку веб-приложений без стандартов.

Тем не менее, по мере роста сложности веб-приложений стало очевидно, что архитектура 2000-х годов сталкивалась с важными ограничениями, такими как непрозрачная архитектура, несовместимость компонентов, отсутствие унифицированных интерфейсов и низкая тестируемость.

В течение следующих десяти лет (2005–2015) произошли значительные изменения в PHP[?]-стеке. Это было связано с разработкой PHP-FIG [?] и ряда стандартов PSR [?], появлением Composer [?], унификацией HTTP-модели[?], распространением принципов DI [?] и архитектурой middleware [?].

Тем не менее, между 2015 и 2025 годами произошли наиболее значительные структурные изменения в PHP[?]-фреймворке. Эти изменения радикально изменили архитектурные решения, принципы проектирования и практики масштабирования.

Эти изменения включали выход PHP 7[?], массовую типизацию и строгую модель ошибок, переход Symfony на компонентный подход,

стремительный рост Laravel, падение Zend Framework и его перерождение в Laminas.

Цель этой работы состоит в том, чтобы проанализировать основные архитектурные и технологические изменения в PHP[?]-фреймворках за 2015–2025 годы, определить причины этих изменений, оценить эффективность принятых решений и определить, какие идеи были отвергнуты или изменились в ходе обсуждений. В анализе используются материалы рассылок PHP-FIG[?], предложения RFC Internals PHP, официальные публикации Symfony[?] и Laravel[?], дискуссии GitHub о стандартах Composer и PSR, а также выступления разработчиков на конференциях, таких как SymfonyCon[?] и Laracon[?].

Таким образом, в статье рассматривается не только развитие определенных фреймворков, но и более широкий процесс создания профессиональной экосистемы PHP, в которой технологические решения являются основной движущей силой прогресса.

2. Предпосылки: фрагментированная экосистема PHP до 2015 года и необходимость стандартизации

К 2015 году промышленная разработка PHP[?] началась с противоречий.

С другой стороны, благодаря своей простоте развёртывания и широкой совместимости PHP[?] продолжал доминировать в веб-разработке.

Напротив, экосистема испытывала значительные архитектурные ограничения из-за отсутствия единых правил и несовместимости основных компонентов фреймворка.

Эти ограничения замедлили язык и фреймворки.

2.1. Отсутствие единых интерфейсов и высокая фрагментация

До того, как в экосистеме PHP появилась PHP-FIG[?], не существовало общепризнанных соглашений по:

- Система каталогов,
- Автоматическая загрузка классов,
- Форматы запросов и ответов HTTP,
- Интерфейсы контейнеров, которые зависят от зависимостей,
- Методы middleware.

Каждый фреймворк, включая Symfony[?], Zend Framework[?], CakePHP[?] и CodeIgniter[?], разработал свои собственные решения, которые часто не пересекались.

Что привело к эффекту «изолированных островов» в PHP-мире, заключалось в том, что библиотеки и инструменты не могли быть повторно использованы между проектами.

Например:

- Symfony[?] использовал собственный автозагрузчик и собственные соглашения об именовании классов.
- Zend Framework[?] применял иной подход к структуре каталогов.
- Многие библиотеки требовали ручного подключения файлов и не имели механизма декларирования зависимостей.

Это создавало тесную связность между фреймворком и компонентами. Интеграции часто строились на неформализованных паттернах, а меняющиеся детали реализации ломали совместимость.

2.2. Проблема совместимости и отсутствие переиспользуемых компонентов

Ключевой проблемой периода до 2012–2014 годов была невозможность использовать одну и ту же библиотеку в разных фреймворках.

Например, HTTP-клиенты, логгеры и шаблонизаторы были жестко «привязаны» к конкретному фреймворку, а попытки переносить решения приводили к конфликтам стилей, соглашений и точек расширения.

Symfony[?] первым попытался решить эту проблему, выделив «компонентный подход[?]» (начиная с Symfony 2[?]), но без стандартизации на уровне индустрии это был лишь частичный шаг.

Разработчики других фреймворков могли использовать компоненты Symfony, но не было гарантий совместимости.

2.3. Отсутствие стандартизированной модели HTTP

Одним из главных технологических ограничений было отсутствие стандартизированного представления:

- HTTP-запроса.
- HTTP-ответа.

- атрибутов, заголовков, стримов.

Каждый фреймворк определял собственные классы:

- Symfony HttpFoundation [?],
- Zend
  Http [?] (Request / Response).
- Slim
  Http [?].
- Собственные реализации в CakePHP [?].

Это делало невозможным:

- Взаимозаменяемые middleware.
- Стандартные фильтры и обработчики.
- Переносимость кода между фреймворками.
- Унифицированные HTTP-клиенты.

Проблема была настолько глубокой, что в рассылке FIG обсуждение PSR-7[?] длилось почти 3 года. Это ещё один признак того, насколько фундаментальной была задача стандартизации.

2.4. Отсутствие единых правил автозагрузки и зависимостей

До релиза Composer [?] (2012) и PSR-4 [?] (2013–2014):

- Большинство библиотек подключались вручную через `require` или `include`.
- Не было декларативного управления зависимостями.

- Поставка кода происходила через ZIP-архивы или PEAR (который плохо поддерживался и был неинтуитивен).

`Autoloading` был одной из самых болезненных частей PHP-разработки:

- Фреймворки требовали строгих соглашений по структуре каталогов.
- Библиотеки не могли легко объявлять свои зависимости.
- Конфликты версий были обычным делом.

Composer [**?**] радикально изменил эту ситуацию, но его широкое принятие началось только после 2014–2015 — и именно этот период стал отправной точкой изменений, анализируемых далее.

2.5. Проблемы тестируемости и внедрения зависимостей

До принятия PSR-11 [**?**] (Container Interface) каждый фреймворк реализовывал собственный DI-контейнер, и ни один из них не был совместим с другим:

- Symfony DependencyInjection [**?**].
- Laravel Container [**?**].
- Zend
  ServiceManager [**?**].
- Pimple [**?**].

Отсутствие общего интерфейса:

- Усложняло создание многоразовых пакетов.
- Делало невозможным перенос middleware-компонентов.

- Тормозило развитие архитектур, основанных на инверсии управления.

Архитектура DI[1] была одной из ключевых болевых точек, которую индустрия смогла решить только в последние 10 лет (2015–2025), в ходе стандартизации PSR-11 [**?**].

[1]**DI — Dependency Injection**. Внедрение зависимостей: объект получает свои зависимости извне, а не создаёт их самостоятельно.

Список литературы

[1] CakePHP Team. CakePHP Request and Response Objects. — URL: https://book.cakephp.org/2/en/controllers/request-response.html (дата обращения: 2025-12-07).

[2] CakePHP Team. CakePHP 1.0 Release. — 2006. — May. — URL: https://bakery.cakephp.org/articles/view/cakephp-1-0-released (дата обращения: 2025-12-07). Archived announcement of CakePHP 1.0 release. Development started in 2005.

[3] EllisLab. CodeIgniter 1.0 Release. — URL: https://codeigniter.com/ (дата обращения: 2024-11-01). Initial release of CodeIgniter framework. Archived information available.

[4] Laravel Team. Laravel Service Container. — URL: https://laravel.com/docs/master/container (дата обращения: 2025-12-07). Official documentation for the Laravel Service Container.

[5] Otwell Taylor. Laracon. — URL: https://laracon.net (дата обращения: 2025-12-07).

[6] PHP Documentation Group. PHP. — URL: https://www.php.net/manual/en/ (дата обращения: 2025-12-07). Version 8.4.

[7] PHP-FIG. PSR-0: Autoloading Standard. — URL: https://www.php-fig.org/psr/psr-0/ (дата обращения: 2024-11-01).

[8] PHP-FIG. PSR-11: Container Interface. — URL: https://www.php-fig.org/psr/psr-11/ (дата обращения: 2025-12-07).

[9] PHP-FIG. PSR-15: HTTP Server Request Handlers. — URL: https://www.php-fig.org/psr/psr-15/ (дата обращения: 2025-12-07).

[10] PHP-FIG. PSR-4: Autoloading Standard. — URL: https://www.php-fig.org/psr/psr-4/ (дата обращения: 2024-11-01). Improved autoloading standard that replaced PSR-0.

[11] PHP-FIG. PSR-7: HTTP Message Interface. — URL: https://www.php-fig.org/psr/psr-7/ (дата обращения: 2025-12-07).

[12] PHP Framework Interop Group. PHP-FIG Charter and Bylaws. — URL: https://www.php-fig.org/bylaws/ (дата обращения: 2025-12-07). Original charter document. Updated versions exist.

[13] PHP Group. PHP 7.0.0 Release Announcement. — URL: https://www.php.net/releases/7_0_0.php (дата обращения: 2024-11-01).

[14] Potencier Fabien. Pimple: A Simple Service Container for PHP. — URL: https://github.com/silexphp/Pimple (дата обращения: 2025-12-07).

[15] Potencier Fabien. Symfony 1.0 Release Announcement. — URL: https://symfony.com/blog/symfony-1-0-released (дата обращения: 2024-11-01). Official announcement of Symfony 1.0. The project started in 2005.

[16] Potencier Fabien. Symfony 2.0 Release Announcement. — URL: https://symfony.com/releases/2.0 (дата обращения: 2025-12-07). Official announcement of Symfony 2.0. The project started in 2007.

[17] Potencier Fabien. Symfony 6: The Future of PHP Frameworks // SymfonyCon. — URL: https://live.symfony.com (дата обращения: 2025-12-07).

[18] Potencier Fabien. Symfony2 Components: Standalone Libraries. — URL: https://symfony.com/blog/symfony2-components-as-standalone-packages (дата обращения: 2025-12-07). Introduction of Symfony's component-based architecture.

[19] Seldaek Jordi B., Contributors. Composer: Dependency Manager for PHP. — URL: https://getcomposer.org/ (дата обращения: 2025-12-07). Initial release announcement and ongoing documentation.

[20] Slim Framework Team. Slim Framework 2.x HTTP Request Object (Slim). — URL: https://github.com/slimphp/Slim-Documentation/blob/master/docs/objects/request.md (дата обращения: 2025-12-07).

[21] Symfony Project. Symfony DependencyInjection Component. — URL: https://symfony.com/doc/current/components/dependency_injection.html (дата обращения: 2025-12-07). Official documentation for the Symfony DependencyInjection component.

[22] Symfony Project. Symfony HttpFoundation Component. — URL: https://symfony.com/doc/current/components/http_foundation.html (дата обращения: 2025-12-07).

[23] Usage statistics of PHP for websites, W3Techs. — URL: https://w3techs.com/technologies/details/pl-php (дата обращения: 2025-11-01). Data for websites with a known server-side programming language. Percentage as of 1 November 2025.

[24] Zend Technologies. Zend Framework: Open Source PHP Framework. — URL: https://framework.zend.com/ (дата обращения: 2025-12-07).

[25] Zend Technologies. ZendComponent (Zend Framework 1.x). — URL: https://framework.zend.com/manual/1.12/en/zend.http.html (дата обращения: 2025-12-07).

[26] Zend Technologies. ZendComponent. — URL: https://packagist.org/packages/zendframework/zend-servicemanager#2.0.3 (дата обращения: 2025-12-07).