

3. Инструменты экосистемы РНР



Инструменты разработчика

- Интерпретатор ЯП
- Менеджер зависимостей
- Линтер
- Форматтер
- Статический анализатор кода
- Фреймворк для тестирования
- Менеджер задач (task runner)
- . . .



PHP, стандарт PSR,
PHP Standard Library

Git, Docker, make, **Composer**, **PHPLint**,
PHP Code Sniffer, **PHPUnit**, **psalm**,
веб-фреймворки, режимы выполнения

Синтаксис языка
программирования

Прикладные
библиотеки, фреймворки и
инструменты

Итераторы, генераторы

Алгоритмы и
структуры данных

Программирование

Практические приемы

Стек, очередь,
деревья

Архитектура (дизайн)
ПО

Групповая работа

Структурное процедурное программирование,
ООП, функциональные возможности, принципы
SOLID, паттерны проектирования

GitHub

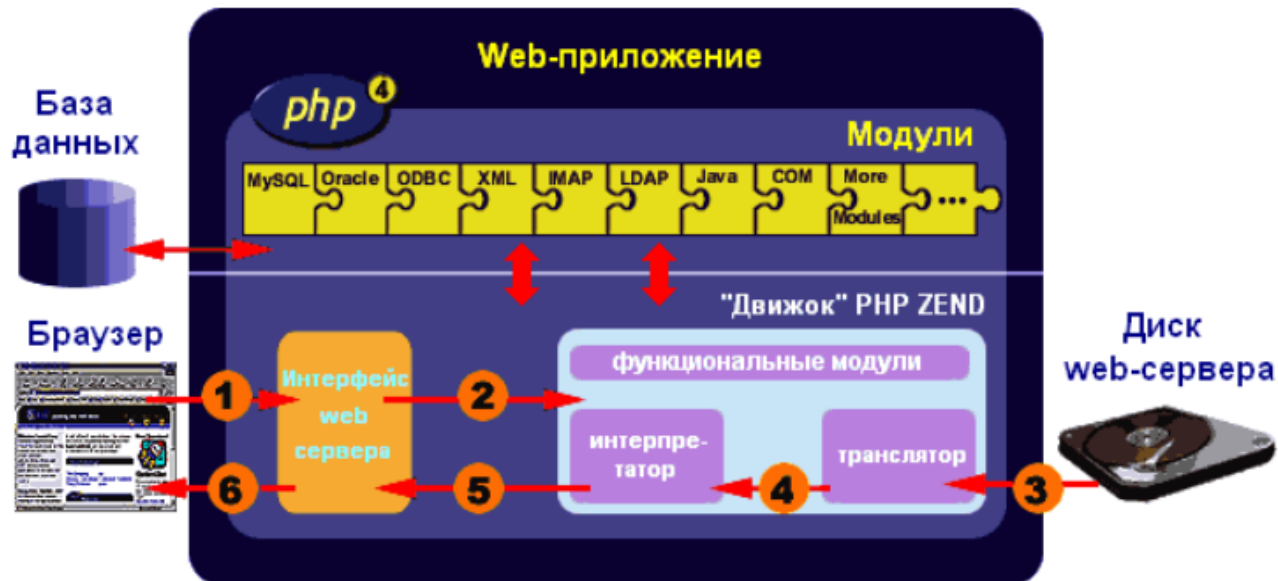
Интерпретатор языка PHP

Интерпретатор PHP

- Языком PHP управляет сообщество разработчиков, а не централизованный комитет или организация. Развитие языка происходит через обсуждения и голосование в рамках группы PHP Internals.
- Спецификации языка PHP в виде единого документа официально не существует.
- Официальная документация на сайте <https://www.php.net/docs.php>
- Изменения в языке PHP обсуждаются и документируются через систему RFC (Request for Comments). Они доступны на сайте PHP Internals Wiki: <https://wiki.php.net/rfc>
- Исходный код интерпретатора PHP (написан на C) доступен на GitHub: <https://github.com/php/php-src>

Режимы запуска интерпретатора PHP

- Консольное приложение (утилита php)
- CGI-скрипт (утилита php-cgi)
- Модуль веб-сервера Apache (модуль mod_php)
- FastCGI-скрипт (утилита php-fpm)
- Альтернативные рантаймы и движки (RoadRunner, ...)
- Server-less



Программа на PHP:

```
$a = 3;  
echo "hello world";  
print $a + 1;
```

Байт-код для этой программы:

Line	#*	E	I	O	op	fetch	ext	return	operands
2	0	E	>		ASSIGN				!0, 3
3	1				ECHO				'hello+world'~
4	2				ADD			~1	!0, 1
	3				PRINT			~2	~1
	4				FREE			~2	
	5			>	RETURN				

- Каждая строка – команда с набором операторов.
- Вместо имен переменных – числа.
- Выполняется быстрее, чем классическая интерпретация

Режимы запуска кода PHP CLI

- Выполнение скрипта из файла

```
php -f script.php  
php script.php
```

- Выполнение кода из командой строки

```
php -r "echo date('Y-m-d H:i:s');"  
php -r "for ($i = 1; $i <= 5; $i++) { echo $i . ' '; }"
```

- Выполнение кода из stdin

```
echo "print_r(get_defined_constants());" | php
```

Режимы запуска кода PHP CLI

- Интерактивный (REPL-оболочка) `php -a`
 - Автодополнение функций, констант, имен классов, переменных, статических методов и констант классов
 - История введенных команд
 - Можно подключить пейджер `#cli.pager=less`
 - Выход - команда `exit`
- Продвинутая оболочка и дебаггер `PsySH`
 - Вычисление значений
 - Подключение внешнего редактора
 - Вывод переменных, функций, классов, объектов
 - Вывод содержимого объектов, методов и функций
 - Встроенная документация по функциям, объектам, классам, методам, константам

Повторное использование кода. Пакетный менеджер Composer

Повторное использование кода

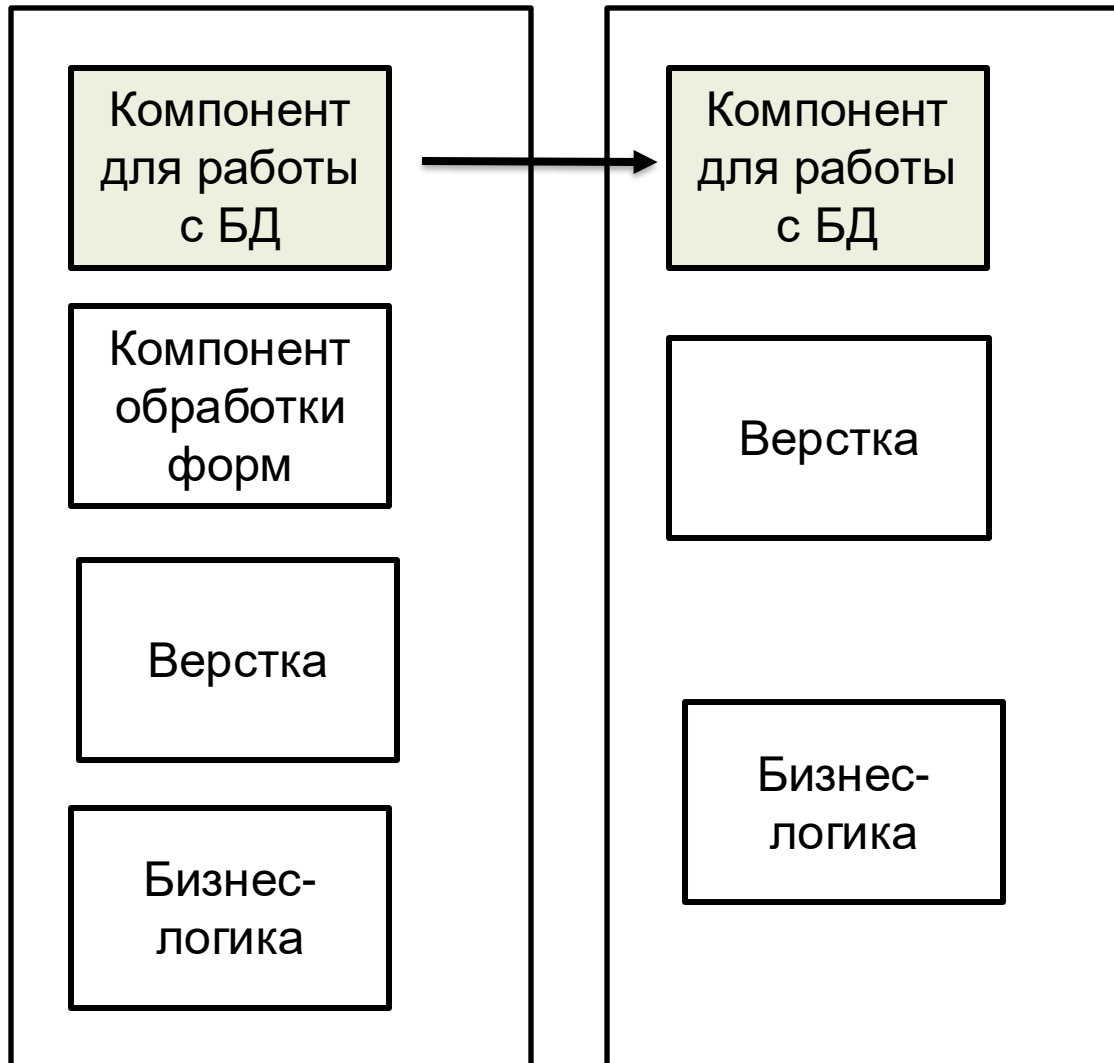
Приложение 1



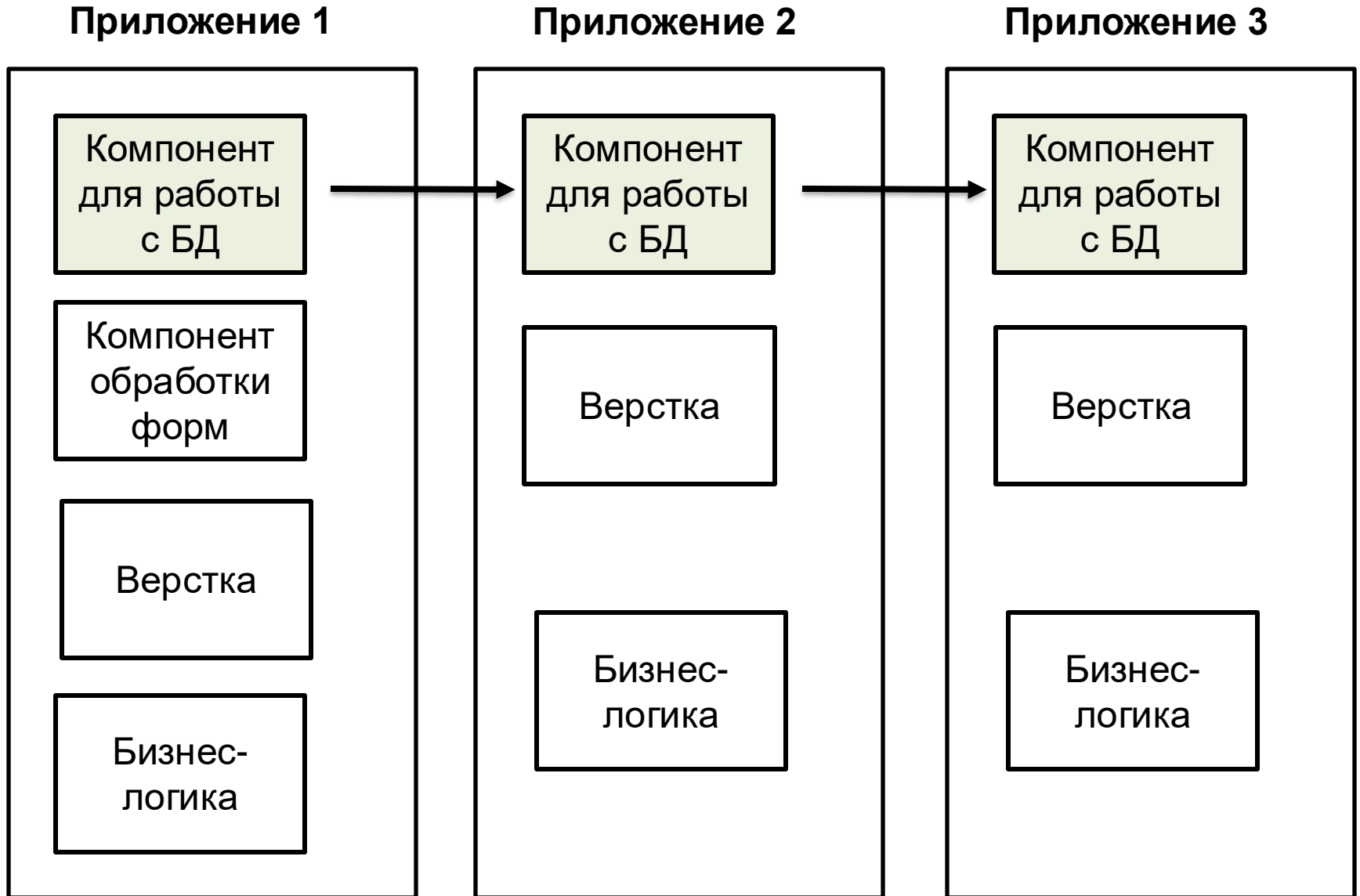
Повторное использование кода

Приложение 1

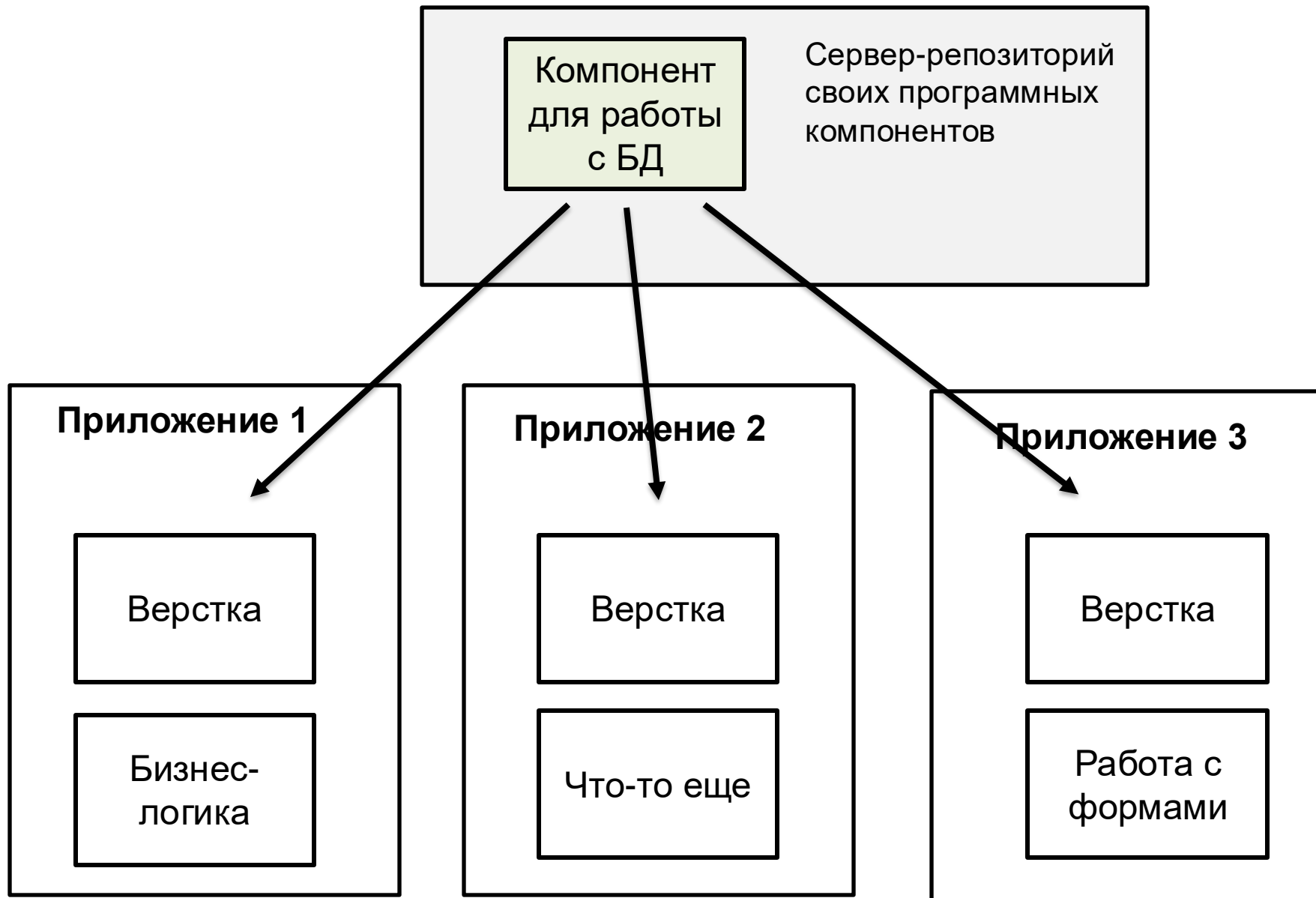
Приложение 2



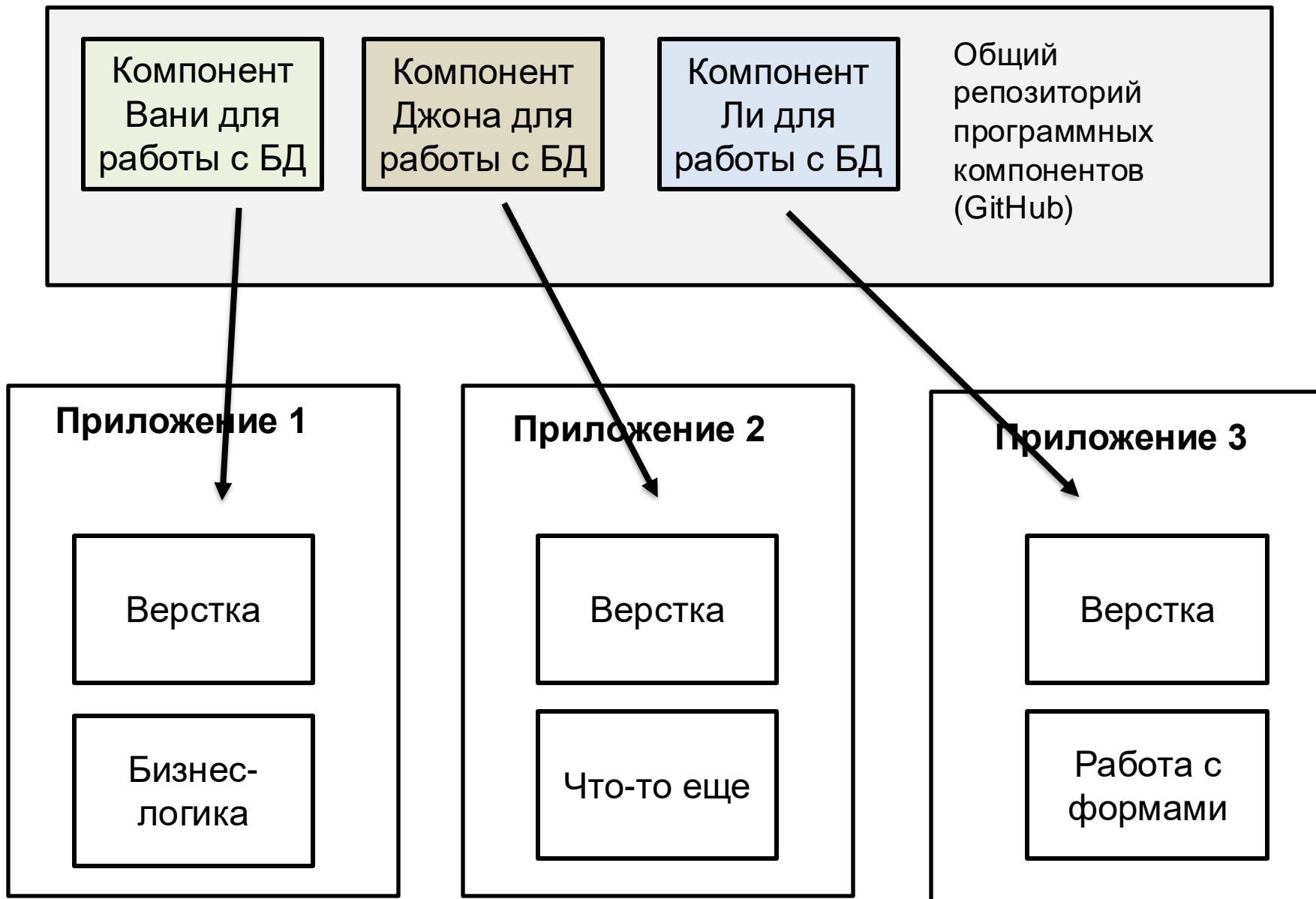
Прямое копирование кода



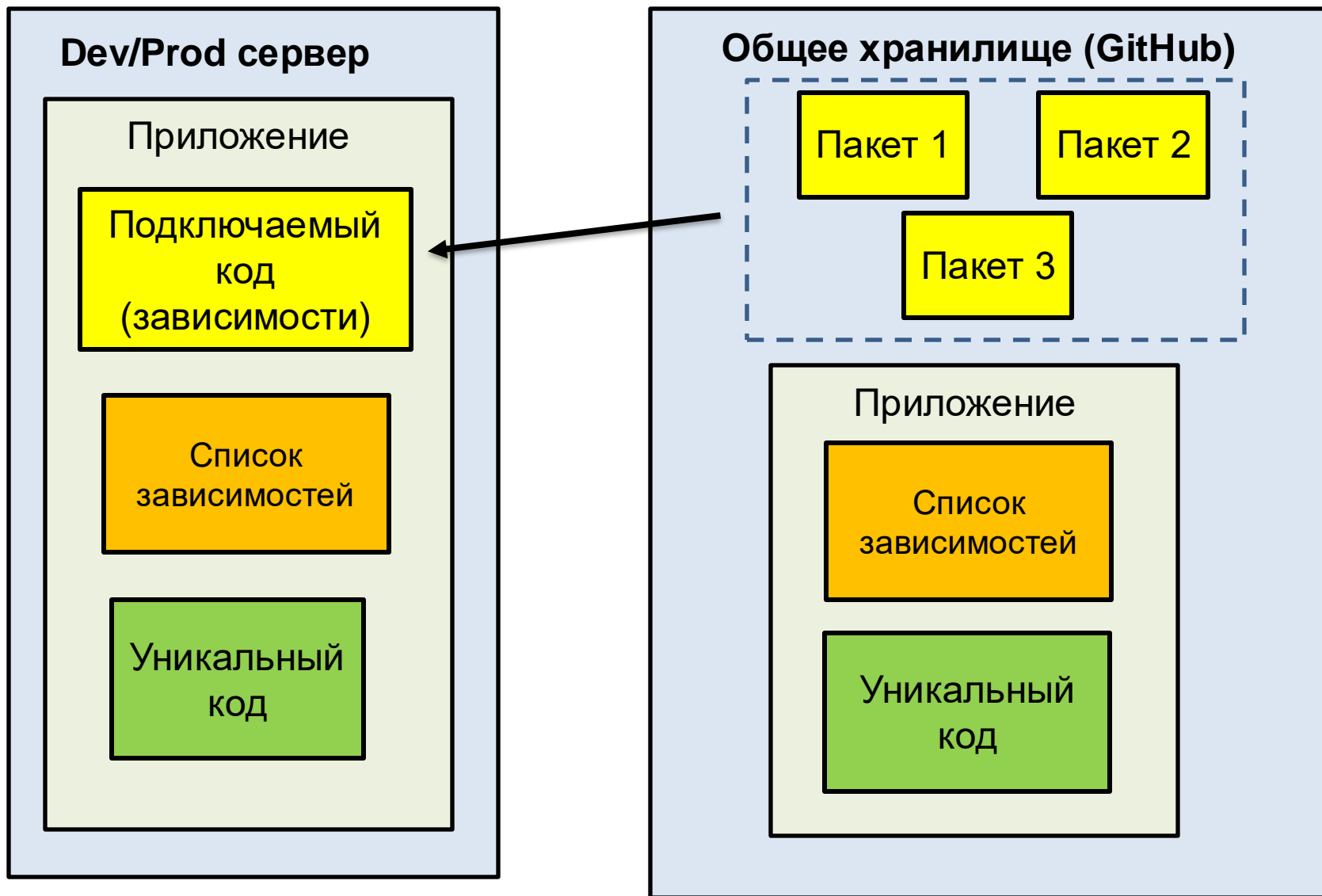
Хранилище своих компонентов



Общее хранилище компонентов

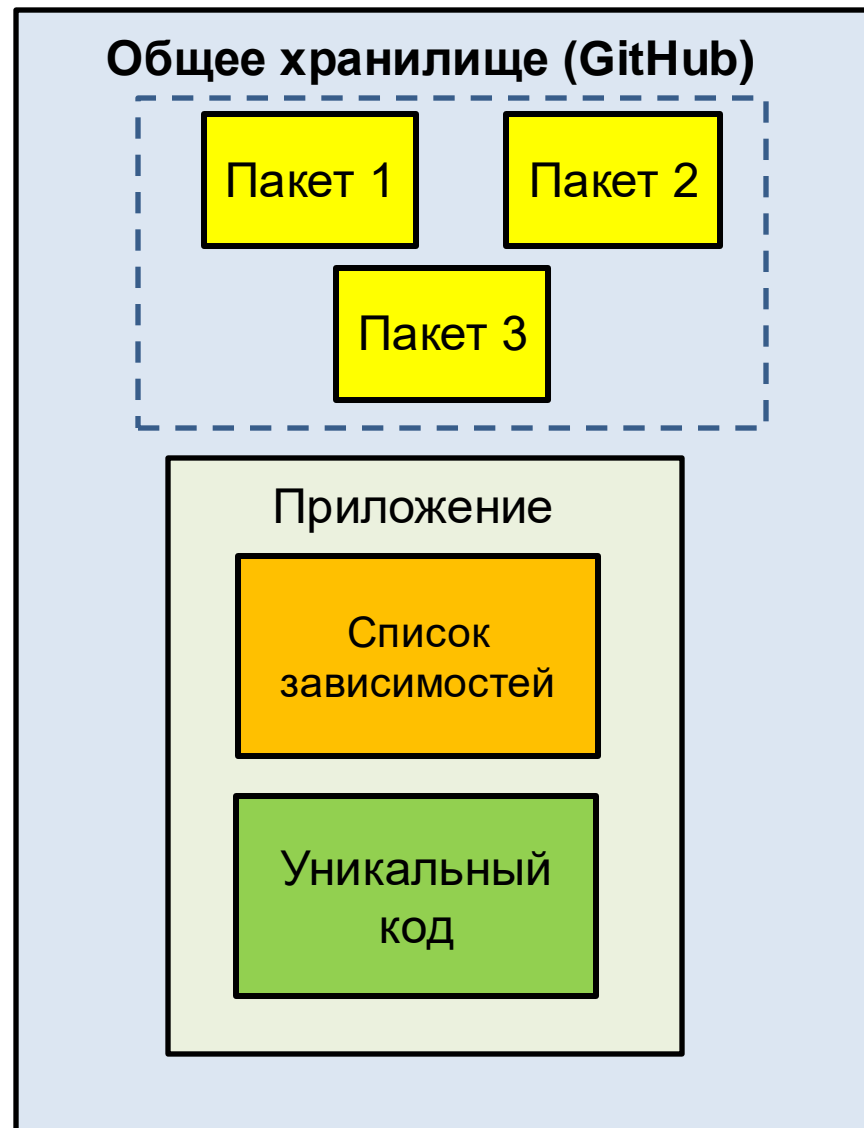


Структура приложения



Проблемы

- Тяжело найти нужный пакет
- Вручную следить за версиями пакетов, за зависимостями зависимостей и т.п. очень сложно.



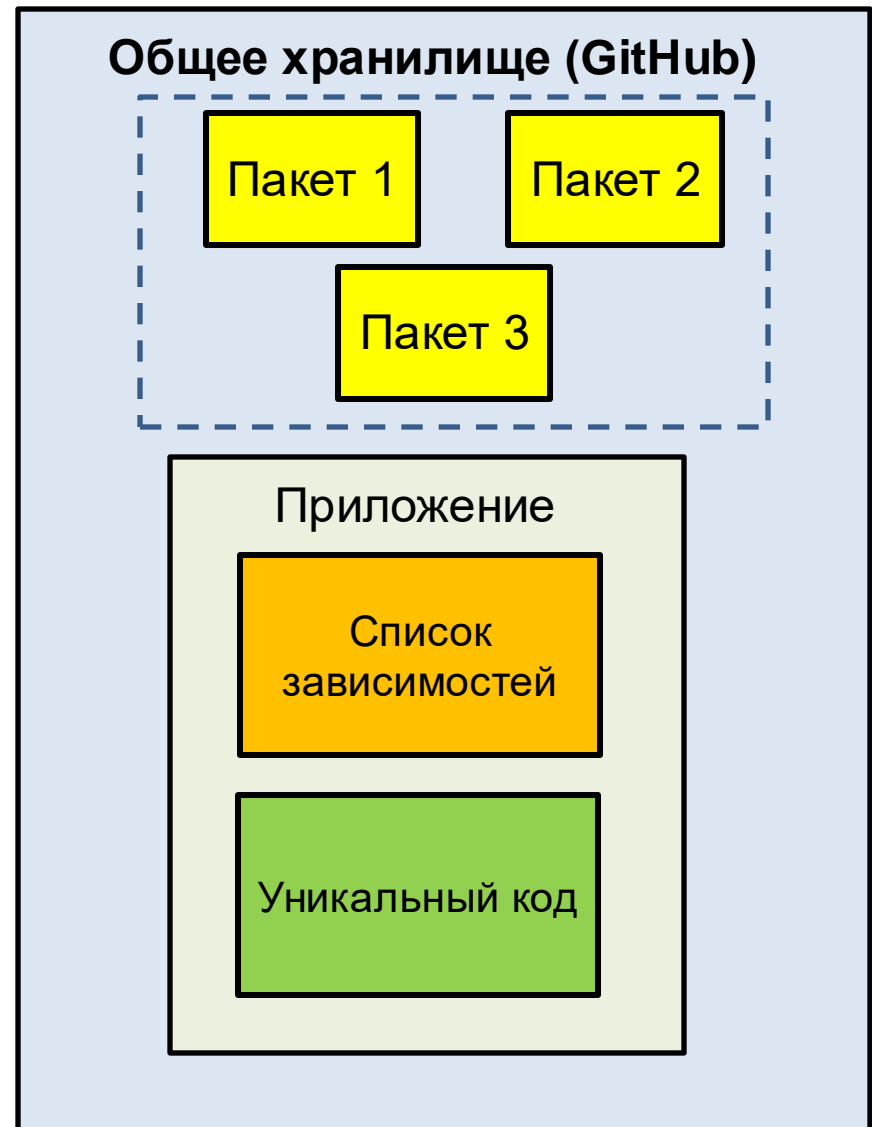
Решения

Тяжело найти нужный пакет

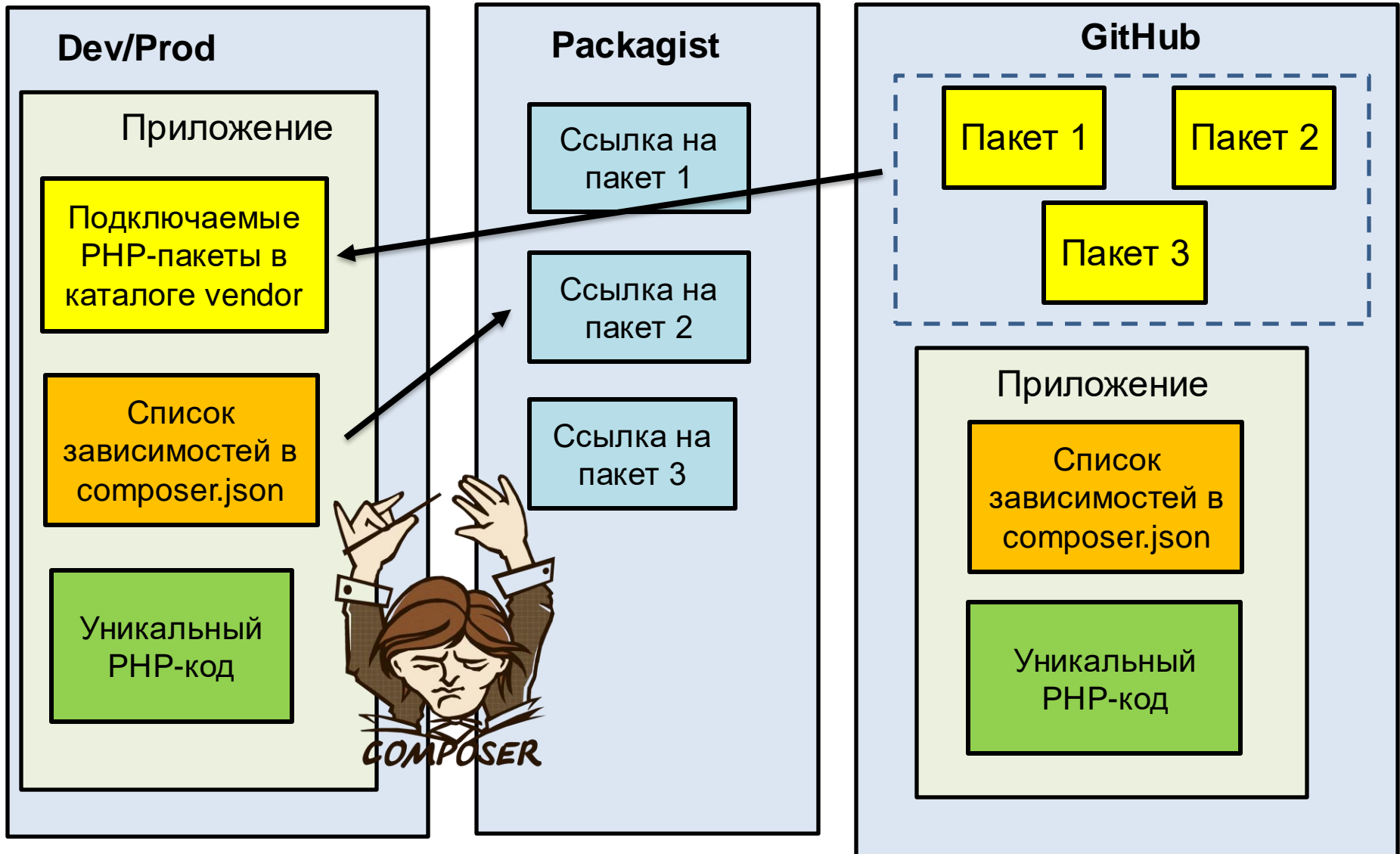
Сделать отдельный сайт для конкретного языка (PHP, JS, Python, ...) с удобным поиском

Вручную следить за версиями пакетов, за зависимостями зависимостей и т.п. очень сложно

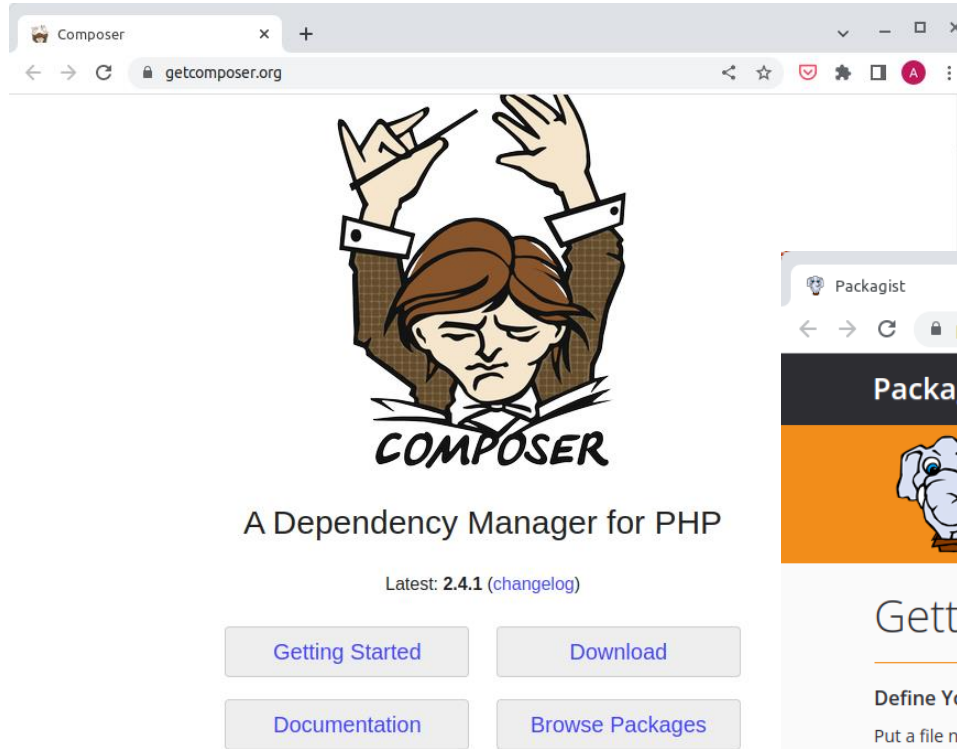
Автоматизировать процесс, создать специальную утилиту - **менеджер зависимостей**



Структура PHP-проекта




Менеджер зависимостей и репозиторий пакетов PHP



Composer

getcomposer.org

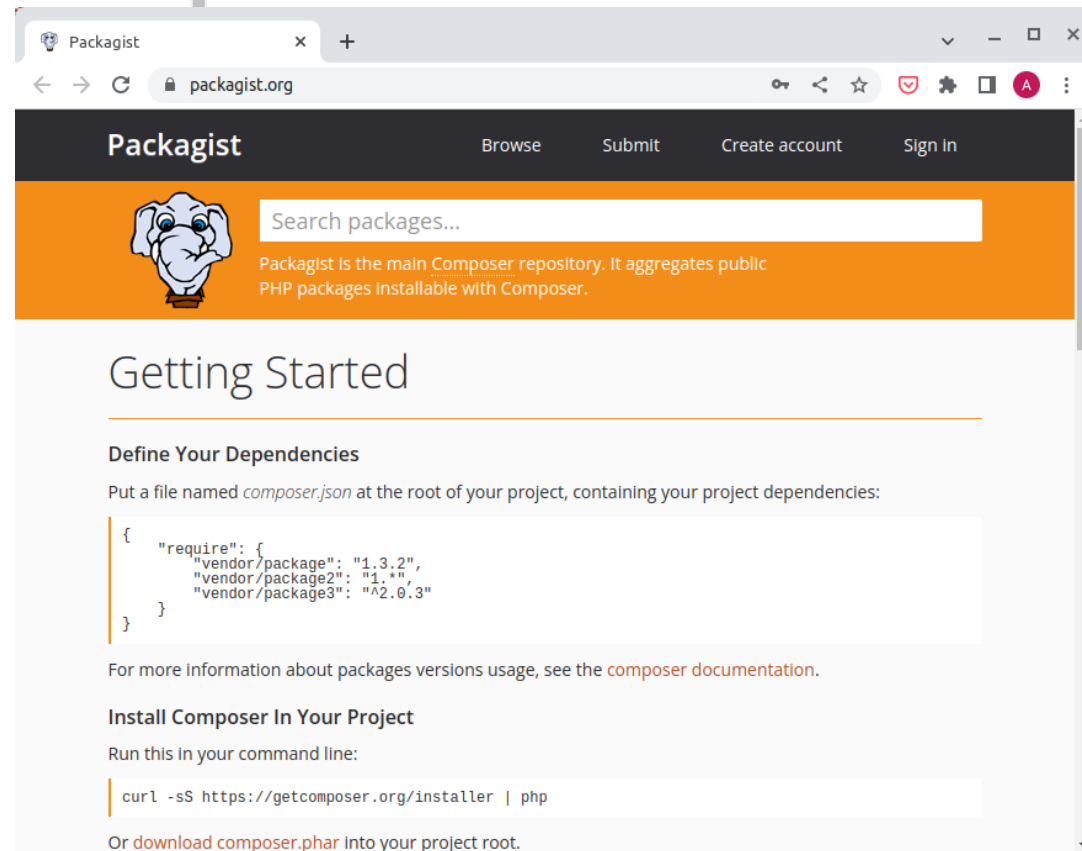


A Dependency Manager for PHP

Latest: 2.4.1 ([changelog](#))


[Getting Started](#) [Download](#)

[Documentation](#) [Browse Packages](#)



Packagist

Browse Submit Create account Sign in



Search packages...

Packagist is the main [Composer](#) repository. It aggregates public PHP packages installable with Composer.

Getting Started

Define Your Dependencies

Put a file named `composer.json` at the root of your project, containing your project dependencies:

```
{
  "require": {
    "vendor/package": "1.3.2",
    "vendor/package2": "1.*",
    "vendor/package3": "^2.0.3"
  }
}
```

For more information about packages versions usage, see the [composer documentation](#).

Install Composer In Your Project

Run this in your command line:

```
curl -sS https://getcomposer.org/installer | php
```

Or [download composer.phar](#) into your project root.

Функции Composer

Авторы: Nils Adermann (Германия) и Jordi Boggiano (Швейцария)

Первый релиз: 2012 год

В версии 2.8.4 (декабрь 2024) доступны 34 команды

- Поиск пакетов на Packagist.
- Инициализация проекта. Задание имен и версий нужных пакетов.
- Установка (скачивание) пакетов в проект. Ставятся в каталог vendor (в репозиторий git его добавлять не нужно).
- Разрешение конфликтов зависимостей
- Обновление установленных пакетов.
- Настройка автозагрузки классов из установленных пакетов в проект.

Конфигурационные файлы Composer

```
{
  "name": "yourusername/yourproject",
  "description": "Описание вашего проекта",
  "type": "project",
  "license": "MIT",
  "authors": [
    {
      "name": "Ваше Имя",
      "email": "youremail@example.com"
    }
  ],
  "require": {
    "monolog/monolog": "^2.0",
    "symfony/http-client": "^5.4"
  },
  "require-dev": {
    "phpunit/phpunit": "^9.5"
  },
  "autoload": {
    "psr-4": {
      "App\\": "src/"
    }
  },
  "scripts": {
    "test": ["phpunit"]
  },
  "minimum-stability": "stable",
  "prefer-stable": true
}
```

composer.json

Выполнение скриптов

- При локальной установке бинарники или ссылки на них Composer устанавливает в `./vendor/bin`
- Скрипты запускаются командой `composer run-script ИМЯ`. При этом в контекст выполнения команды Composer добавляется к переменной `$PATH` путь `./vendor/bin`.

Функции Composer

Авторы: Nils Adermann (Германия) и Jordi Boggiano (Швейцария)

Первый релиз: 2012 год

В версии 2.8.4 (декабрь 2024) доступны 34 команды

- Поиск пакетов на Packagist.
- Инициализация проекта. Задание имен и версий нужных пакетов.
- Установка (скачивание) пакетов в проект. Ставятся в каталог vendor (в репозиторий git его добавлять не нужно).
- Разрешение конфликтов зависимостей
- Обновление установленных пакетов.
- Настройка автозагрузки классов из установленных пакетов в проект.

Проверка корректности кода

Линтер для PHP файлов

- Проверка синтаксиса в одном файле:

```
php -l файл
```

- Скрипт в `composer.json` для проверки всех файлов в проекте:

```
"scripts": {  
    "lint": "find src public config -name \"*.php\" -print0 |  
xargs -0 -n1 php -l"  
}
```

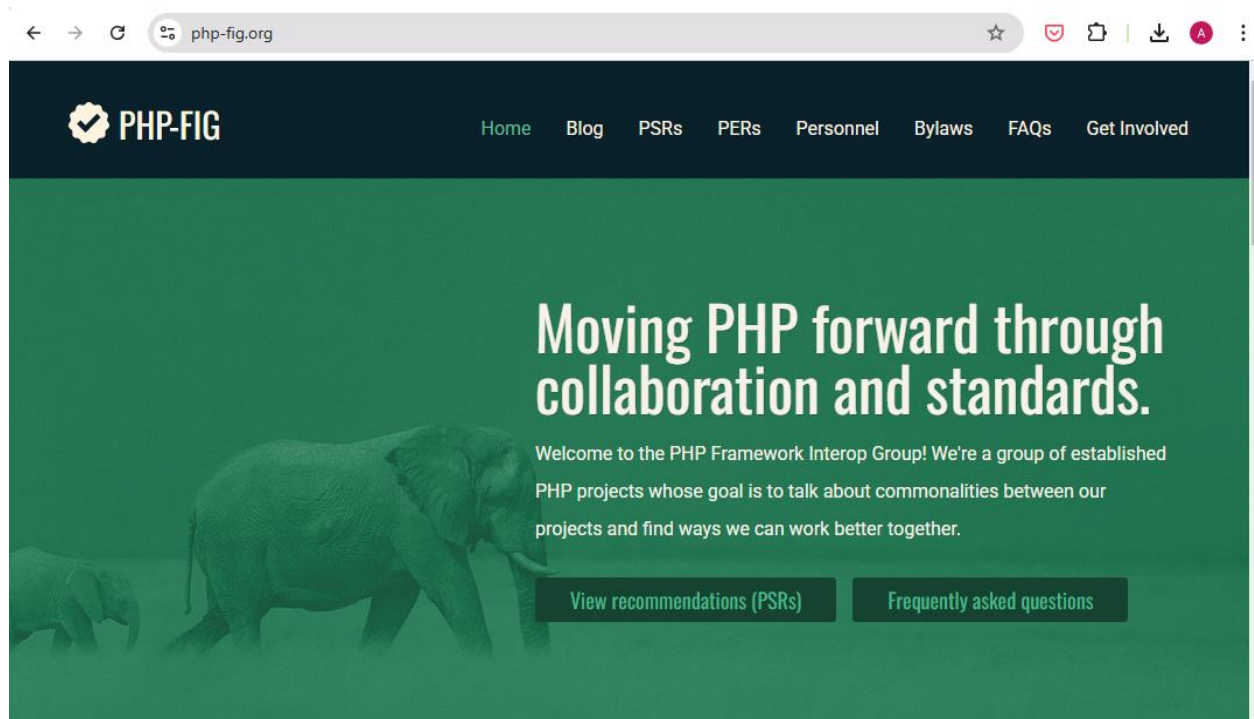
Запуск через `composer run-script lint`

- Утилита **overtrue/phplint**. Обертка над `php -l`, запускает сразу несколько процессов проверки, поддерживает кеш для неизменных файлов, выводит красивые отчеты. Настройка через `yaml`-файл.

Стандарты PSR

PSR

PSR (PHP Standards Recommendations) — набор стандартов, разработанных группой PHP-FIG (Framework Interop Group), которая объединяет разработчиков различных фреймворков и библиотек на PHP.



Стандарты PSR помогают улучшить читаемость, поддерживаемость и переносимость кода, делая его более предсказуемым и понятным для других разработчиков.

История PSR

- С начала 2000-х было написано множество фреймворков и библиотек, имеющих собственные соглашения о структуре проектов, стиле кодирования и интерфейсах. Проблемы с интеграцией решений.
- В 2009 году была создана группа PHP-FIG (разработчики фреймворков CakePHP, Symfony, Zend Framework, Doctrine, ...) для создания общего набора стандартов.
- Первый стандарт PSR-0 (соглашение об автозагрузке классов, основанное на пространствах имен) опубликован в 2012 году.

Разделы PSR

PSR-1, PSR-12
стиль и правила
форматирования
кода

NUM	TITLE	EDITOR(S) / MAINTAINERS	STATUS
0	Autoloading Standard		Deprecated
1	Basic Coding Standard	<i>vacant</i>	Accepted
2	Coding Style Guide		Deprecated
3	Logger Interface	Jordi Boggiano	Accepted
4	Autoloading Standard	<i>vacant</i>	Accepted
5	PHPDoc Standard	Chuck Burgess	Draft
6	Caching Interface	Larry Garfield	Accepted
7	HTTP Message Interface	Matthew Weier O'Phinney	Accepted
8	Huggable Interface	Larry Garfield	Abandoned
9	Security Advisories	Michael Hess	Abandoned
10	Security Reporting Process	Michael Hess	Abandoned
11	Container Interface	Matthieu Napoli, David Négrier	Accepted
12	Extended Coding Style Guide	Korvin Szanto	Accepted
13	Hypermedia Links	Larry Garfield	Accepted
14	Event Dispatcher	Larry Garfield	Accepted
15	HTTP Handlers	Woody Gilk	Accepted
16	Simple Cache	Paul Dragoonis	Accepted
17	HTTP Factories	Woody Gilk	Accepted
18	HTTP Client	Tobias Nyholm	Accepted
19	PHPDoc tags	Chuck Burgess	Draft
20	Clock	Chris Seufert	Accepted
21	Internationalization	Navarr Barnier	Draft
22	Application Tracing	Adam Allport	Draft

Проверка соблюдения PSR

PHP_CodeSniffer (phpcs) — утилита для статического анализа кода на языке PHP и проверки соответствия кода стандартам кодирования.

- Проверка кода на соответствие стандартам PSR-1, PSR-12 или пользовательским стандартам.
- Генерация отчетов о нарушениях стандартов в различных форматах (текст, XML, JSON, ...).
- Интеграция с инструментами CI/CD. Это позволяет автоматизировать процесс проверки кода перед каждым коммитом или пул-реквестом.
- Поддержка пользовательских правил.

Утилита PHP_CodeBeautifier and Fixer (phpcbf) — автоматическое исправление большинства обнаруженных нарушений.

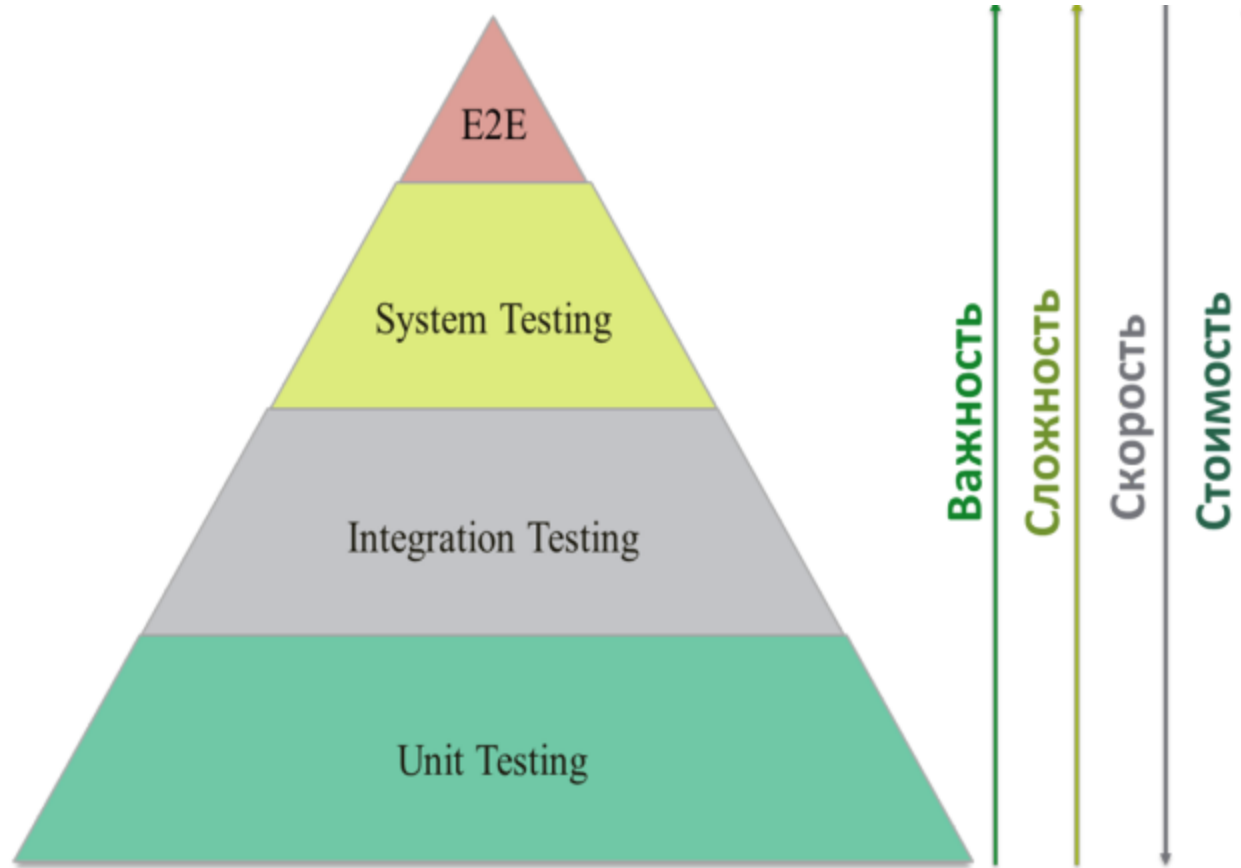
Статический анализ кода

Статический анализатор кода PHP

Psalm — статический анализатор кода для PHP от компании Vimeo.

Помогает находить ошибки (неверные типы данных, недопустимые операции над объектами, ...) и уязвимости в коде ещё до запуска приложения.

Тестирование кода



Пирамида тестирования приложения

Уровни тестирования

- **Unit/component/program/module testing** - тестируется минимально-атомарный модуль программы, чаще всего это одна функция или метод. Таких тестов должно быть больше всего.
- **Integration testing** - несколько модулей программы тестируются вместе.
- **System testing** - вся программа тестируется полностью.
- **Acceptance (E2E) testing** - программа принимается заказчиком на соответствие заявленным требованиям либо тестировщики проходят end-to-end сценарии с точки зрения пользователя.

Фреймворк для тестирования PHPUnit

Автор - Sebastian Bergmann

Первый релиз - 2001 год

Актуальная версия - PHPUnit 11

Утилита make

Для чего нужен task runner

При развертывании, настройке и разработке проекта нужно выполнять разные вспомогательные консольные скрипты/утилиты

- Установка зависимостей
- Запуск сервера
- Запуск линтеров
- Запуск тестов
- Запуск миграций БД
- Сборка проекта
- Манипуляции с контейнерами Docker
- Генерация документации

Утилиты могут запускаться разными интерпретаторами, иметь параметры командной строки.

Нужен инструмент (task runner) для автоматизации выполнения таких повторяющихся задач.

"Склейка" разных утилит: унификация запуска + упрощение задач + самодокументация проекта.

Утилита make

Утилита make была написана в 1976 году для автоматизации сборки исполняемых программ и библиотек из исходного кода на С.

Является стандартной POSIX-утилитой, есть во всех дистрибутивах UNIX-подобных систем.

Выполняет инструкции согласно правилам, заданным в конфигурационном файле Makefile.

Структура Makefile

Makefile состоит из набора правил (rules), каждое из которых определяет цель (target) и набор действий (commands), необходимых для достижения этой цели

```
цель1: зависимости
    команда1 # для отступа используется табуляция
    команда2

цель2: зависимости
    команда3
```

- **Цель** - имя файла, который нужно получить, или задачи, которую нужно выполнить.
- **Зависимости** - список файлов или целей, от которых зависит данная цель.
- **Команды** выполняются для достижения цели.

Выполнение целей из Makefile

```
# Makefile  
setup: env-prepare sqlite-prepare install key db-prepare  
  
env-prepare:  
    cp -n .env.example .env  
  
sqlite-prepare:  
    touch database/database.sqlite  
  
install:  
    composer install  
    npm install  
  
key:  
    php artisan key:generate  
  
db-prepare:  
    php artisan migrate --seed  
  
start:  
    heroku local -f Procfile.dev
```

Bash

```
make setup # выполнит последовательно: env-prepare sqlite-prepare install key db-  
make start
```