
PHP TEK
2026
I'm speaking at PHP Tek 2026
Custom PHPStan Rules: Automate Standards and Save Time
Tue, 5/19/2026 11:00 am — Chicago, Illinois, United States
Dave Liddament
https://phptek.io

Using PHPStan means:

- Higher code quality
- Fewer bugs

Custom rules amplify these benefits

PHPStan

Coding Standards as Code

Easy upgrades

Eliminate bugs

Better than AGENTS.md

PHPStan

Custom Rules

Easy upgrades

Coding standards as code

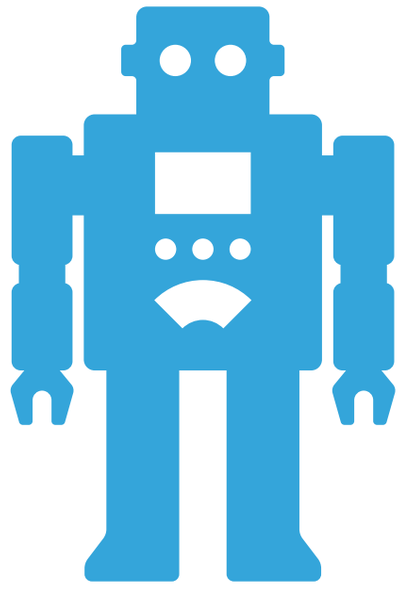
PHPStan rule

Eliminate bugs

AST

@daveliddament

HOW TO DOCUMENT CODING STANDARDS



@daveliddament

Coding Standards as Code

You must follow PSR-12

URLs must be kebab-case,
parameters must be camelCase

Repository **get*** methods must
return a value or throw an exception.
Repository **find*** methods must
return a value or null.

Coding Standards as Code

php-cs-fixer

PHP_CodeSniffer

PHPStan + Custom Rules

URL is kebab-case

```
Route::put('say-hello/{firstName}');
```

Parameters are camelCase

https://spatie.be/guidelines/laravel-php#content-routing

Correct

```
Route::get('hello');
Route::post('hello/{name}');
Route::put('say-hello/{firstName}');
```

Wrong

```
Route::get('sayHello');
Route::get('hello/{first-name}');
```

```
#[Route(path: 'hello')]
```

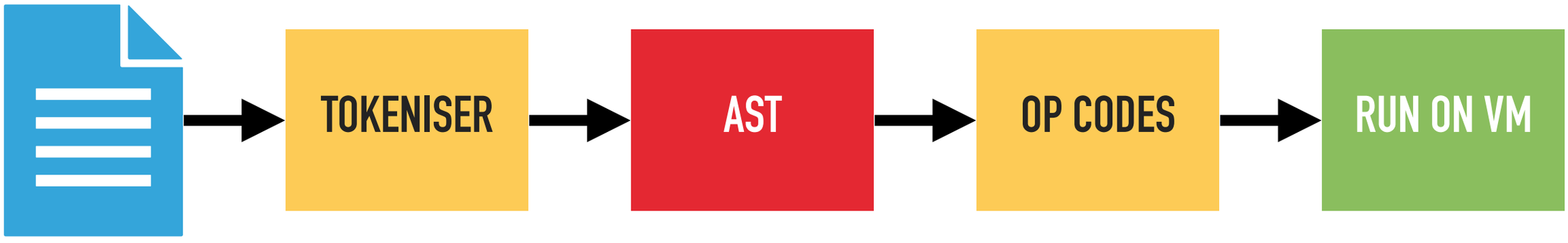
Easy upgrades

Coding standards as code

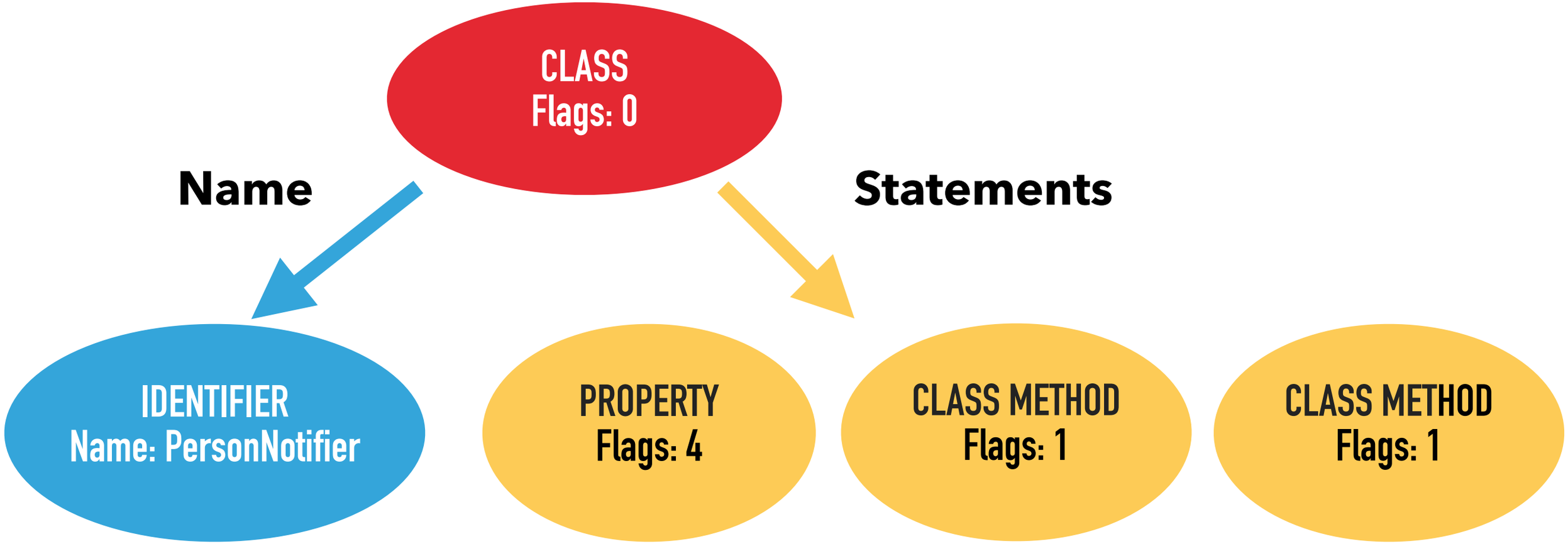
PHPStan rule

Eliminate bugs

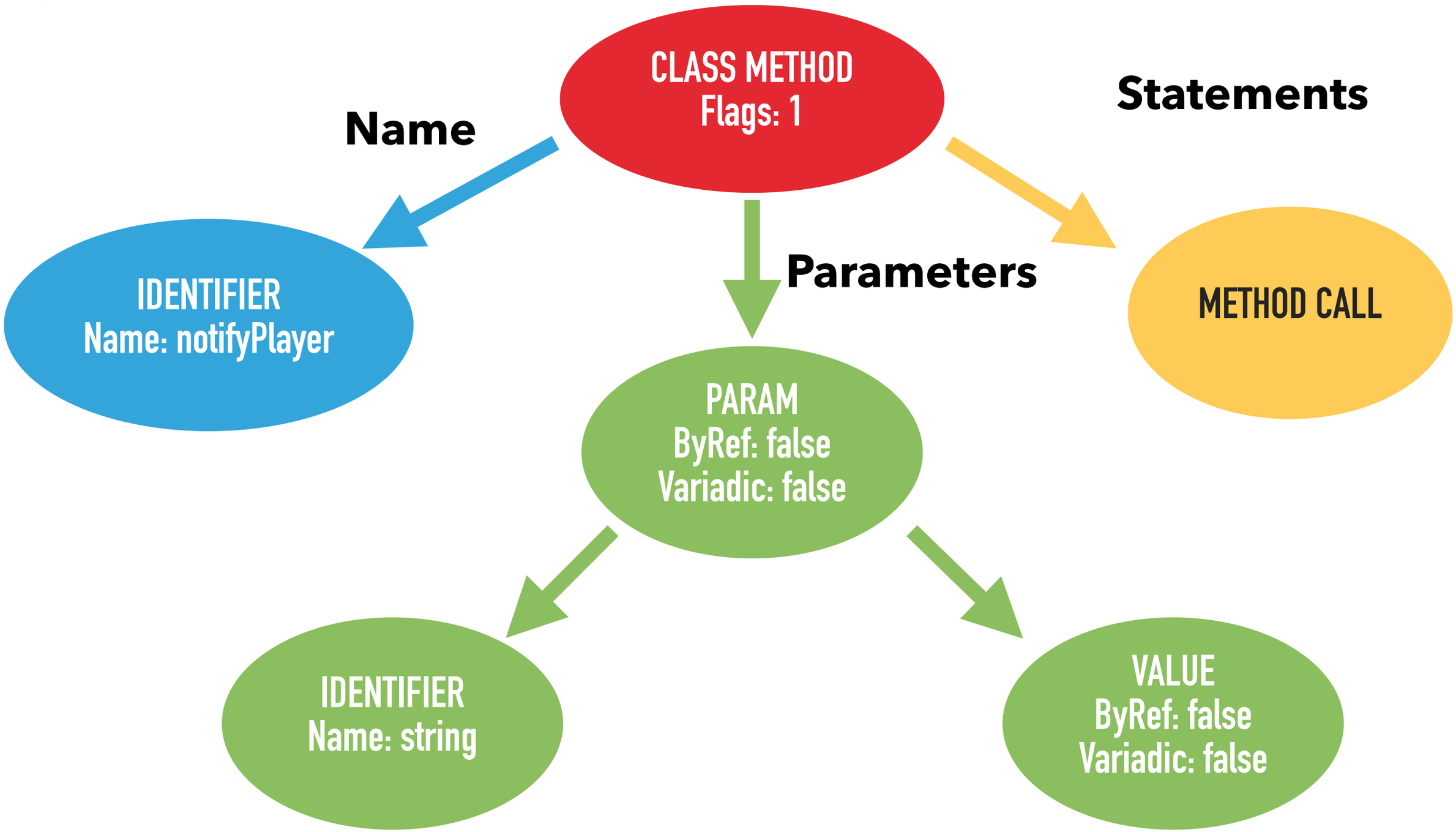
AST


TOKENISER
AST
OP CODES
RUN ON VM

```
class PersonNotifier
{
    private TextMessageSender $sender;
    public function __construct() {…}
    public function notifyPlayer() {…}
}
```



```
public function notifyPlayer (string $msg)
{
    $this->sender->sendMessage($msg);
}
```



https://github.com/nikic/PHP-Parser

nikic / PHP-Parser Public

Watch 232 | Fork 891

Code | Issues 44 | Pull requests 9 | Actions | Wiki | Security | Insights

master | 9 branches | 80 tags | Go to file | Add file | Code

nikic Bail out on PHP tags in removed code ... b0edd4c 2 hours ago 1,526 commits

.github/workflows	Test PHP 8.2 in CI	3 days ago
bin	Add --version flag to php-parse	17 days ago
doc	Fix pretty printing example	17 days ago
grammar	Support readonly before DNF type	3 days ago
lib/PhpParser	Bail out on PHP tags in removed code	2 hours ago
test	Bail out on PHP tags in removed code	2 hours ago
test_old	Avoid repeatedly downloading archive in run-php-src.sh	2 months ago
tools	Add tools/ directory	10 days ago
.editorconfig	[PHP 8.1] Add support for enums (#758)	17 months ago
.gitattributes	Add CONTRIBUTING.md	23 days ago
.gitignore	gitignore: add phpunit test cache	3 years ago
.php-cs-fixer.dist.php	Also format the grammar directory	23 days ago
CHANGELOG.md	Release PHP-Parser 5.0.0-alpha1	17 days ago
CONTRIBUTING.md	Add CONTRIBUTING.md	23 days ago
LICENSE	Corrected license text	2 years ago
README.md	Partial documentation update	17 days ago
UPGRADE-1.0.md	Fix typos	8 years ago

About

A PHP parser written in PHP

php | parser | static-analysis | ast

Readme

BSD-3-Clause license

15.7k stars

232 watching

891 forks

Releases 76

PHP-Parser 4.15.1 Latest

18 days ago

+ 75 releases

Packages

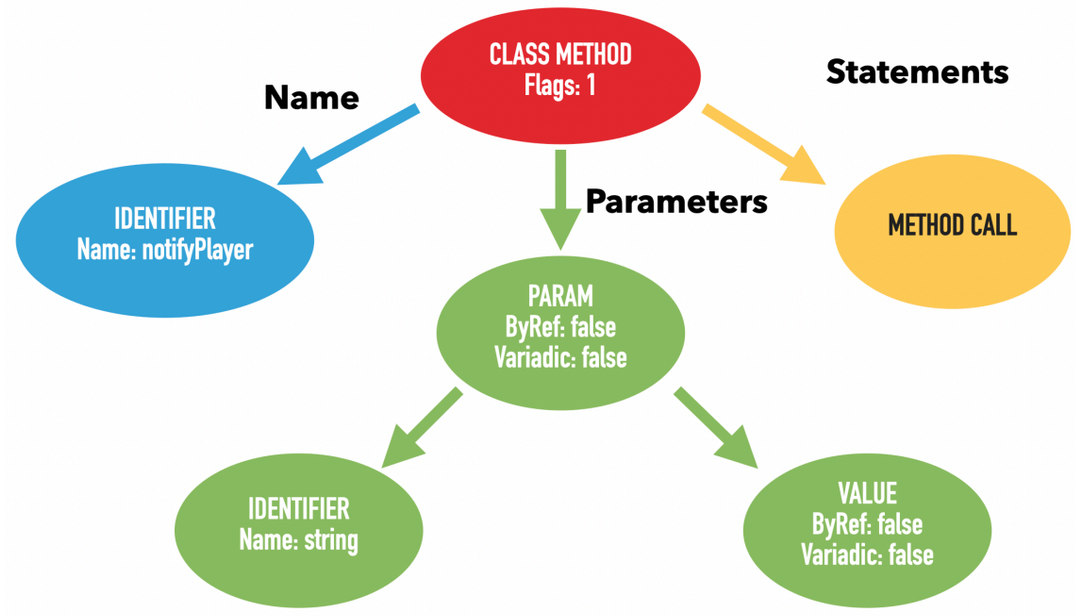
No packages published

Used by 1.5m

+ 1,486,143

Contributors 123

```php
class PersonNotifier
{
  private TextMessageSender $sender;
  public function __construct() {…}
  public function notifyPlayer() {…}
}
```




```php
class MethodCall extends \PhpParser\Node\Expr\CallLike
{
  /** @var Expr Variable holding object */
  public $var;

  /** @var Identifier|Expr Method name */
  public $name;

  /** @var array<Arg|VariadicPlaceholder> Arguments */
  public $args;

  // Rest of class …
```

```php
$this->sender -> sendMessage ($msg) ;
```

- PHP code can be represented by an AST
- Different types of Node
- Nodes contain information
- Each type of node has different information

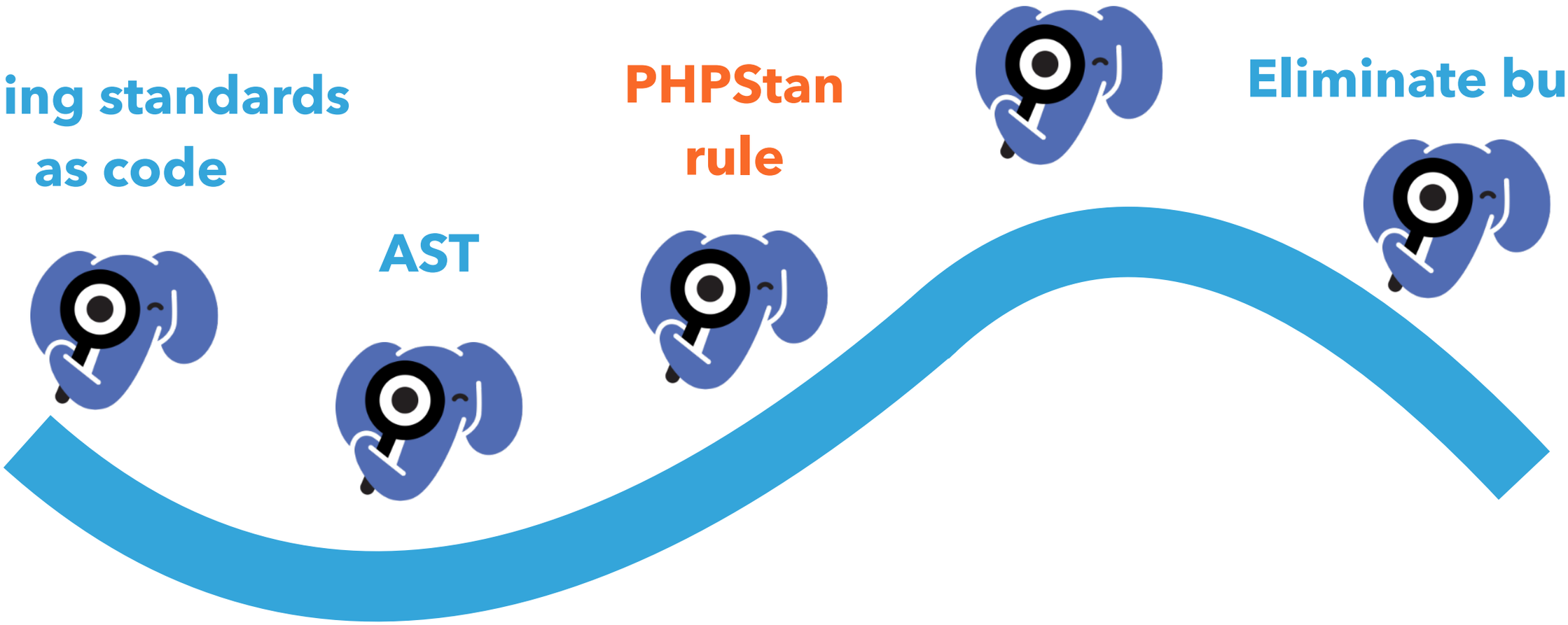
Easy upgrades

Coding standards as code

PHPStan rule

Eliminate bugs

AST



@daveliddament

CLASS
Flags: 0

IDENTIFIER
Name: PersonNotifier

PROPERTY
Flags: 4

CLASS METHOD
Flags: 1

CLASS METHOD
Flags: 1

IDENTIFIER
Name: notifyPlayer

PARAM
ByRef: false
Variadic: false

METHOD CALL

IDENTIFIER
Name: string

VALUE
ByRef: false
Variadic: false

```
interface Rule
{

  public function getNodeType() : string;


  /**
   * @return (string|RuleError)[] errors
   */
  public function processNode(
    \PhpParser\Node $node,
    \PHPStan\Analyser\Scope $scope
  ) : array;

}
```

```
Route::get('/hello');
```

1. Write short PHP code you want to understand

```php
<?php

class Foo extends Bar {}
```

Parse

2. Click on any part of the code

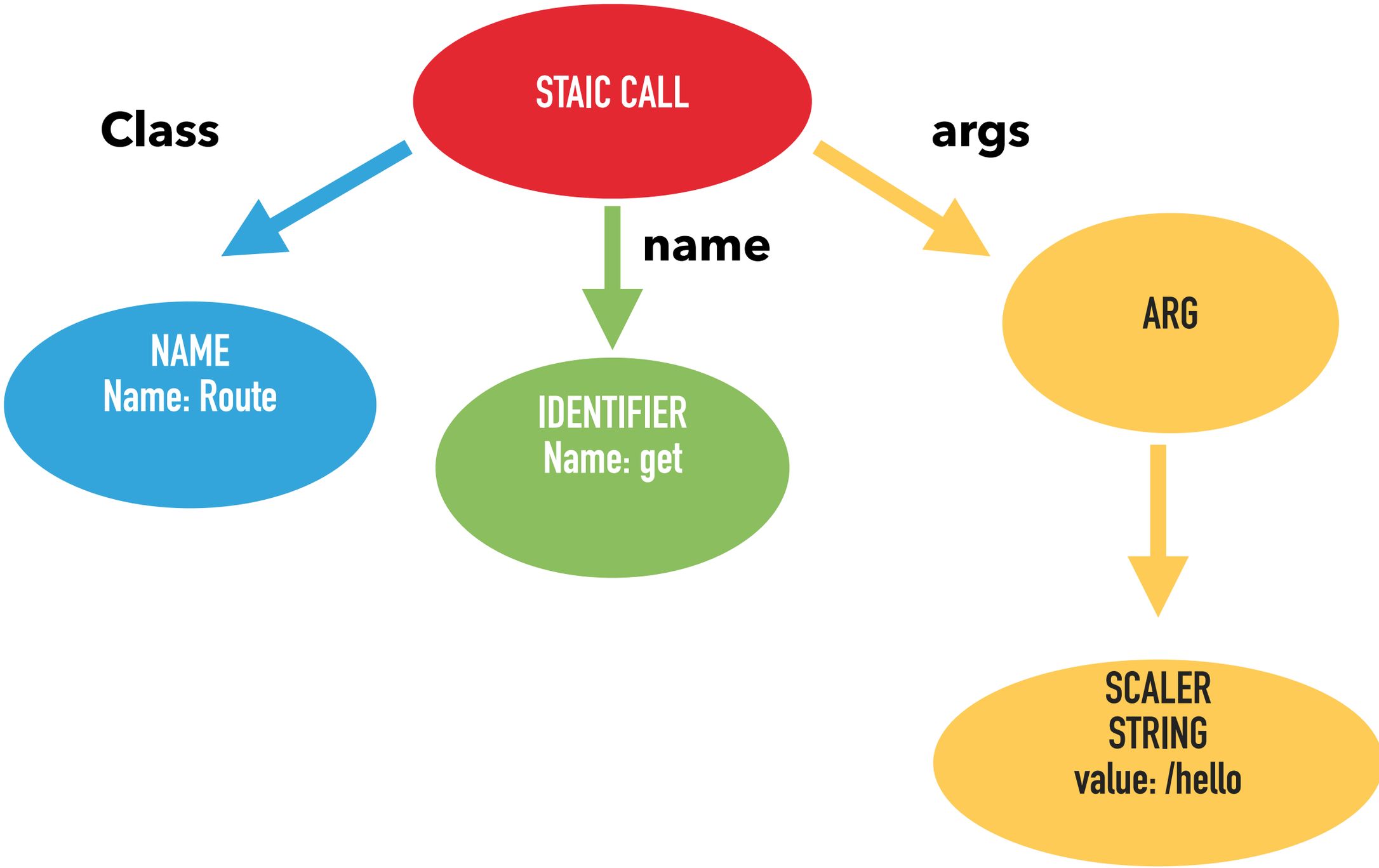
```php
<?php

class Foo extends \Bar
{
}
```

3. See Abstract Syntax Tree created by php-parser for full file

```
PhpParser\Node\Stmt\Class_(
    attrGroups: []
    flags: 0
    name: PhpParser\Node\Identifier( name: "Foo" )
    extends: PhpParser\Node\Name\FullyQualified( parts: ["Bar"] )
    implements: []
    stmts: []
)
```

https://getrector.com/ast

```
Route::get('/hello');
```



```
class StaticCall extends CallLike {

    /** @var Name|Expr Class name */
    public Node $class;

    /** @var Identifier|Expr Method name */
    public Node $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public array $args;
```

```
Route::get('/hello');
```

```
class RouteRule implements Rule

{

    public function getNodeType() : string
    {
        return StaticCall::class;
    }
}
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {
```

```
  // 1. Check if the class we are calling is Route

  if (!$node->class instanceof Node\Name) {
    return [];
  }

  if ($node->class->toString() !== Route::class) {
    return [];
  }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {
  // 1. Check if the class we are calling is Route
  if (!$node->class instanceof Node\Name) {
    return [];
  }

  if ($node->class->toString() !== Route::class) {
    return [];
  }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // 1. Check if the class we are calling is Route

  if (!$node->class instanceof Node\Name) {
    return [];
  }

  if ($node->class->toString() !== Route::class) {
    return [];
  }
```

```
class StaticCall extends CallLike {
    /** @var Name|Expr Class name */
    public Node $class;

    /** @var Identifier|Expr Method name */
    public Node $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public array $args;
```

```
Route::get();    // Name

($className)::get();  // Expr
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // 1. Check if the class we are calling is Route
  if (!$node->class instanceof Node\Name) {
    return [];
  }
```

```
if ($node->class->toString() !== Route::class) {
   return [];
}
```

```php
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // 1. Check if the class we are calling is Route
  if (!$node->class instanceof Node\Name) {
    return [];
  }
```

```php
  if ($node->class->toString() !== Route::class) {
     return [];
  }
```

```
// 2. Check if the method name is get, post, put
// or delete
```

```
if (!($node->name instanceof Identifier)) {
    return [];
}
```

```
$methodName = $node->name->toLowerString();

$methodsToCheck = ['get', 'post', 'put', ‘delete'];

if (!in_array($methodName, $methodsToCheck, true)) {
    return [];
}
```

```
// 2. Check if the method name is get, post, put
// or delete

if (!($node->name instanceof Identifier)) {
    return [];
}
$methodName = $node->name->toLowerString();

$methodsToCheck = ['get', 'post', 'put', ‘delete'];

if (!in_array($methodName, $methodsToCheck, true)) {
    return [];
}
```

```
class StaticCall extends CallLike {

    /** @var Name|Expr Class name */
    public Node $class;

    /** @var Identifier|Expr Method name */
    public Node $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public array $args;
```

```
// 2. Check if the method name is get, post, put
// or delete

if (!($node->name instanceof Identifier)) {
    return [];
}
$methodName = $node->name->toLowerString();

$methodsToCheck = ['get', 'post', 'put', ‘delete'];

if (!in_array($methodName, $methodsToCheck, true)) {
    return [];
}
```

```
// 2. Check if the method name is get, post, put
// or delete

if (!($node->name instanceof Identifier)) {
    return [];
}

$methodName = $node->name->toLowerString();

$methodsToCheck = ['get', 'post', 'put', ‘delete'];

if (!in_array($methodName, $methodsToCheck, true)) {
    return [];
}
```

```
// 3. Get the 1st argument's value if supplied

// Is first argument is an Arg
$arg = $node->args[0] ?? null;
if (!$arg instanceof Arg) {
    return [];
}

// Check if the first argument is a string
$value = $arg->value;
if (!$value instanceof String_) {
    return [];
}

// 4. Check if the value is a valid URL
if (RouteValidator::validate($value->value)) {
    return [];
}
```

```
// 3. Get the 1st arguments value if supplied

// Is first argument is an Arg
$arg = $node->args[0] ?? null;
if (!$arg instanceof Arg) {
    return [];
}

// Check if the first argument is a string
$value = $arg->value;
if (!$value instanceof String_) {
    return [];
}

// 4. Check if the value is a valid URL
if (RouteValidator::validate($value->value)) {
    return [];
}
```

```
// 3. Get the 1st arguments value if supplied

// Is first argument is an Arg
$arg = $node->args[0] ?? null;
if (!$arg instanceof Arg) {
    return [];
}

// Check if the first argument is a string
$value = $arg->value;
if (!$value instanceof String_) {
    return [];
}

// 4. Check if the value is a valid URL
if (RouteValidator::validate($value->value)) {
    return [];
}
```

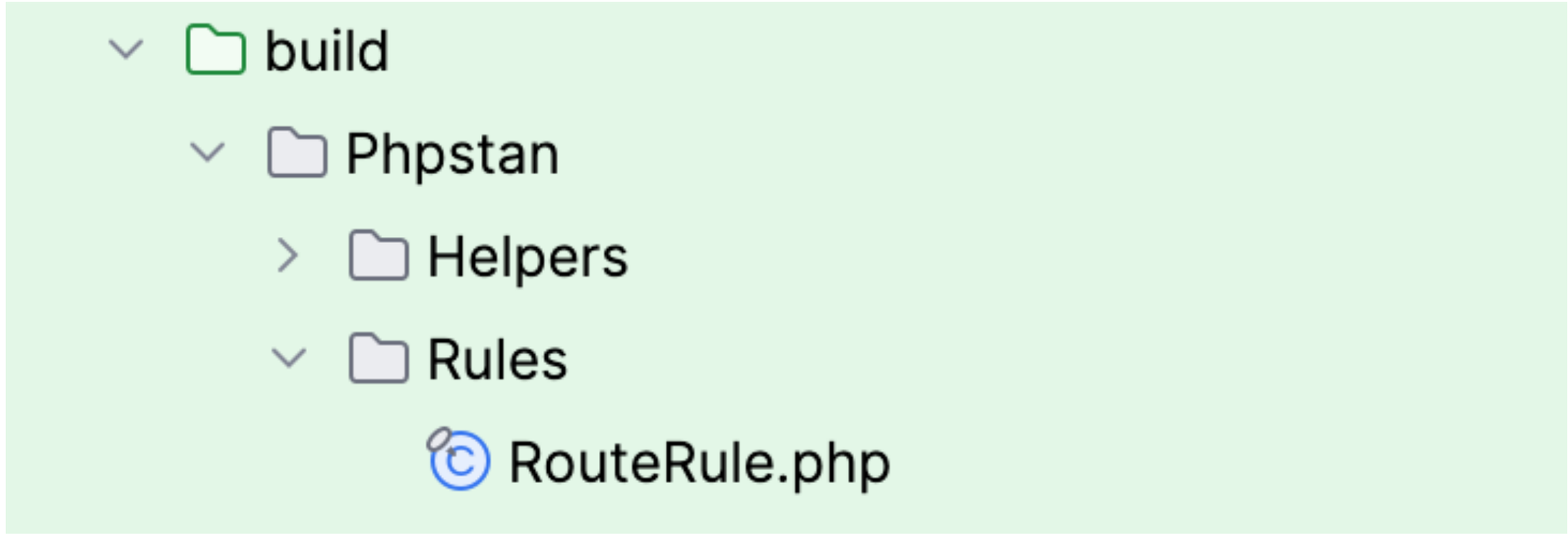
```
// 3. Get the 1st arguments value if supplied

// Is first argument is an Arg
$arg = $node->args[0] ?? null;
if (!$arg instanceof Arg) {
    return [];
}

// Check if the first argument is a string
$value = $arg->value;
if (!$value instanceof String_) {
    return [];
}

// 4. Check if the value is a valid URL
if (RouteValidator::validate($value->value)) {
    return [];
}
```

```
// 5. Return an error as the URL is not
valid
```

```
$msg = 'URL must be in kebab-case and any
parameters in camelCase’;

return [
    RuleErrorBuilder::message($msg)
                    ->identifier('route.url')
                    ->build()
];
```

```
// 5. Return an error as the URL is not
valid
```

```
$msg = 'URL must be in kebab-case and any
parameters in camelCase’;


return [
    RuleErrorBuilder::message($msg)
                    ->identifier('route.url')
                    ->build()
];
```




```
"autoload-dev": {
    "psr-4": {
        "DaveLiddament\\Phpstan\\": "build/Phpstan/"
    }
},
```

```
services:
-
    class: DaveLiddament\Phpstan\Rules\RouteRule
    tags:
    - phpstan.rules.rule
```

DaveLiddament / **custom-phpstan-rules-talk-example**

Type / to search

Code | Issues | Pull requests | Actions | Projects | Wiki | Security | Insights | Settings

custom-phpstan-rules-talk-example

Public

Pin | Unwatch 1 | Fork 0 | Star 0

main | 1 Branch | 0 Tags | Go to file t | Add file | Code

DaveLiddament Initial check in — a084282 · yesterday — 1 Commit

build/Phpstan	Initial check in	yesterday
src	Initial check in	yesterday
.gitignore	Initial check in	yesterday
LICENSE.md	Initial check in	yesterday
README.md	Initial check in	yesterday
composer.json	Initial check in	yesterday
composer.lock	Initial check in	yesterday
phpstan-upgrade.neon	Initial check in	yesterday
phpstan.neon	Initial check in	yesterday
phpunit.xml	Initial check in	yesterday

About

Example custom PHPStan rules used for enforcing coding standards

- Readme
- MIT license
- Activity
- 0 stars
- 1 watching
- 0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

https://github.com/DaveLiddament/custom-phpstan-rules-talk-example

Easy upgrades

Coding standards as code

PHPStan rule

Eliminate bugs

AST

@daveliddament

v1.3.0

```
class AlertService {
  public function alert(
      string $message,
  ){…}
}
```

v1.4.0

```
class AlertService {
  public function alert(
      string $message,
      ?string $type = null,
  ){
      // Log if $type is null.
  }
}
```

v2.0.0

```
class AlertService {
  public function alert(
      string $message,
      string $type,
  ){…}
}
```

RUN TIME VS STATIC ANALYSIS

```
public function alert(
        string $message,
        ?string $type = null,
){
    if ($type === null) {
        @trigger_error(
            'Provide a value for $type',
            \E_USER_DEPRECATED)
    }

    … Rest of method …
```

STATIC ANALYSIS

```
$this->alerter->alert($msg, $type);
```

AlertService

Missing

or null


STOP!
WRITE A TEST

Correct

```
$this->alerter->alert($msg, $type);
```

Wrong

```
$this->alerter->alert($msg);
$this->alerter->alert($msg, null);
```

Similar code that must be ignored

```
$this->alerter->info($msg);
```

```
$this->register->alert($msg);
```

Create one or more fixtures

```
function foo(AlertService $alertService)
{
    $alertService->alert('hello', 'OK');

    $alertService->alert('hello');

    $alertService->alert('hello', null);
}
```

Errors on lines 6 and 8

```
class AlertServiceAlertUpgradeRuleTest
extends RuleTestCase
{
    protected function getRule(): Rule
    {
        return new AlertServiceAlertUpgradeRule();
    }

    public function testRule(): void
    {
        $this->analyse(
            [__DIR__ . '/Fixtures/routes.php'],
            [
                ['$type must be a string', 6],

                ['$type must be a string', 8],
            ],
        );
    }
}
```

Another fixture file

```
function foo(AlertService $alertService)
{
    $alertService->info('hello');

}
```

STATIC ANALYSIS

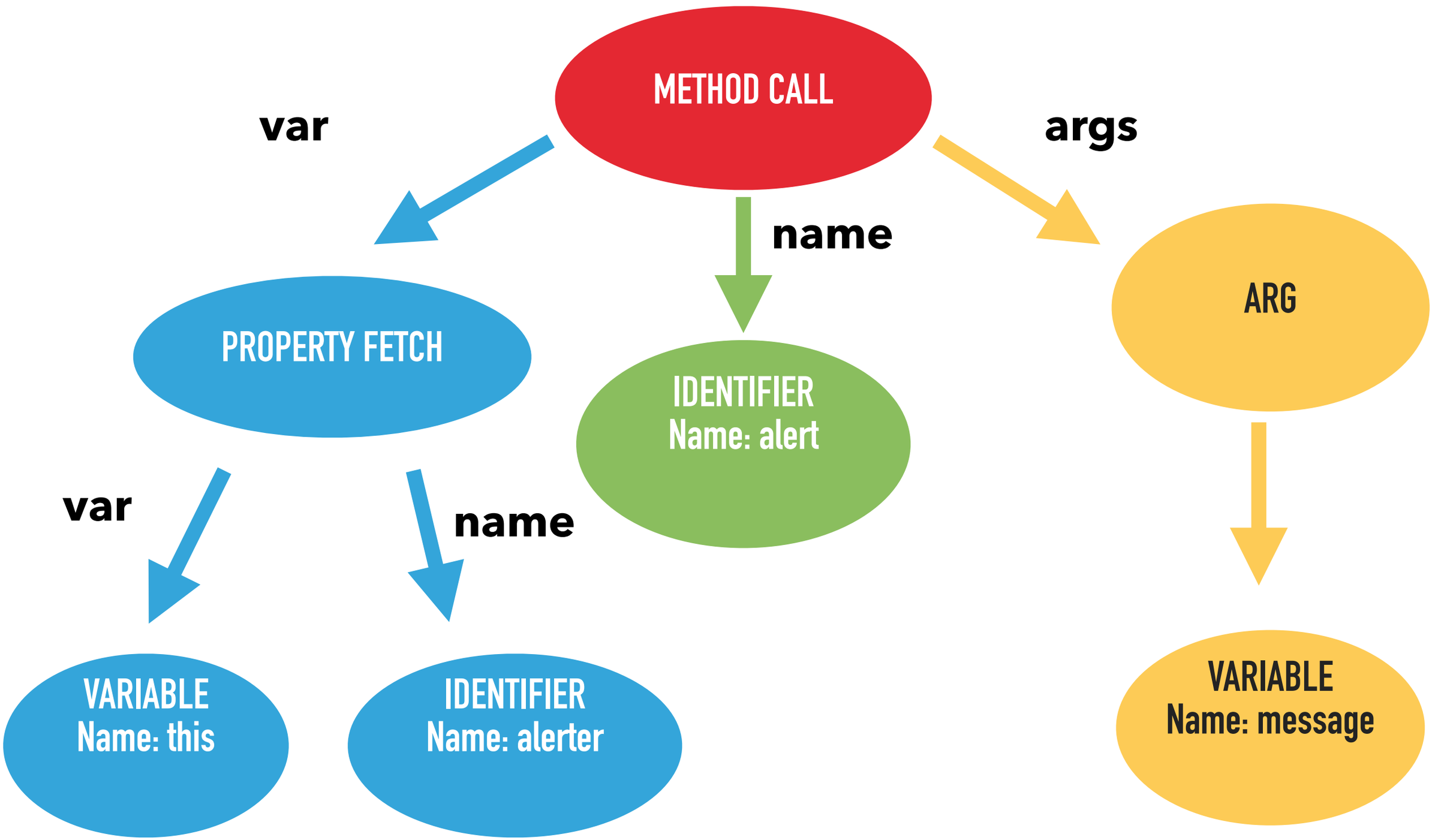
```
$this->alerter->alert($msg, $type);
```

AlertService

Missing

or null

```
$this->alerter->alert($message);
```



```
class MethodCall extends \PhpParser\Node\Expr\CallLike
{
    /** @var Expr Variable holding object */
    public $var;

    /** @var Identifier|Expr Method name */
    public $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public $args;

    // Rest of class …
```

```
$this->alerter -> alert ($message) ;
```

```
final class AlertServiceAlertUpgradeRule
    implements Rule
{

    public function getNodeType(): string
    {
        return MethodCall::class;
    }
```

```
final class AlertServiceAlertUpgradeRule
    implements Rule
{
    public function getNodeType(): string
    {
        return MethodCall::class;
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {
    // Is the call on an AlertService object?

    $objectType = new ObjectType(AlertService::class);

    $var = $scope->getType($node->var);

    if (!$objectType->isSuperTypeOf($var)->yes()) {
        return [];
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

    // Is the call on an AlertService object?

    $objectType = new ObjectType(AlertService::class);

    $var = $scope->getType($node->var);

    if (!$objectType->isSuperTypeOf($var)->yes()) {
        return [];
    }
```

```php
$this->alerter->alert($msg, $type);
```

AlertService

alert

Missing or null

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // Is the call on an AlertService object?

  $objectType = new ObjectType(AlertService::class);

  $var = $scope->getType($node->var);

  if (!$objectType->isSuperTypeOf($var)->yes()) {
      return [];
  }
```

```php
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // Is the call on an AlertService object?

  $objectType = new ObjectType(AlertService::class);
```

```php
  $var = $scope->getType($node->var);
```

```php
  if (!$objectType->isSuperTypeOf($var)->yes()) {
      return [];
  }
```

```php
public function processNode(
    Node $node,
    Scope $scope,
): array {

  // Is the call on an AlertService object?

  $objectType = new ObjectType(AlertService::class);
  $var = $scope->getType($node->var);

  if (!$objectType->isSuperTypeOf($var)->yes()) {
      return [];
  }
```

```
// Is the method name ‘alert'?

$name = $node->name;

if (!$name instanceof Identifier) {
    return [];
}

if ($name->toLowerString() !== 'alert') {
    return [];
}
```

```
// Is the method name ‘alert'?

    $name = $node->name;

    if (!$name instanceof Identifier) {
        return [];
    }

    if ($name->toLowerString() !== 'alert') {
        return [];
    }
```

```
// Is the method name ‘alert'?

 $name = $node->name;

 if (!$name instanceof Identifier) {
     return [];
 }

 if ($name->toLowerString() !== 'alert') {
     return [];
 }
```

```
// Does the call have at least 2 arguments

if (count($node->args) >= 2) {

    // Is the 2nd argument a string?

    $arg2 = $node->args[1];
    if ($arg2 instanceof Arg) {

        $type = $scope->getType($arg2->value);
        if ($type->isString()->yes())
            return [];
        }
    }
}
```

```
// Does the call have at least 2 arguments

if (count($node->args) >= 2) {

    // Is the 2nd argument a string?

    $arg2 = $node->args[1];
    if ($arg2 instanceof Arg) {

        $type = $scope->getType($arg2->value);
        if ($type->isString()->yes())
            return [];
        }
    }
}
```

```
// Does the call have at least 2 arguments

if (count($node->args) >= 2) {

    // Is the 2nd argument a string?

    $arg2 = $node->args[1];
    if ($arg2 instanceof Arg) {
        $type = $scope->getType($arg2->value);
        if ($type->isString()->yes())
            return [];
        }
    }
}
```

```
// Does the call have at least 2 arguments
if (count($node->args) >= 2) {
    // Is the 2nd argument a string?
    $arg2 = $node->args[1];
    if ($arg2 instanceof Arg) {
        $type = $scope->getType($arg2->value);
        if ($type->isString()->yes())
            return [];
        }
    }
}
```

```
        // If we've got this far, there is a problem.

        $msg = '$type must be a string';

        return [
            RuleErrorBuilder::message($msg)
                ->identifier('demoLibUpgrade.AlertService')
                ->build(),
        ];
    }
}
```

- Deprecated classes
- Deprecated methods
- Removed parameters

HOW DO I SHARE WITH OTHERS?

MY LIBRARY

MY LIBRARY V1 TO V2 UPGRADE RULES

composer.json

```
{
  “name”: “acme/my-library-v1-v2-upgrade-rules”,

  “type”: “phpstan-extension”,

  “require”: {
      “acme/my-library” : “^1.4”,
      … other packages …
  },

  “extra”: {
    “phpstan”: {“includes”: [“extension.neon”]}
  },

  … rest or composer.json …
```

composer.json

```
{
 “name”: “acme/my-library-v1-v2-upgrade-rules”,

 “type”: “phpstan-extension”,

 “require”: {
      “acme/my-library” : “^1.4”,
      … other packages …
},

“extra”: {
  “phpstan”: {“includes”: [“extension.neon”]}
},

… rest or composer.json …
```

composer.json

```
{
 “name”: “acme/my-library-v1-v2-upgrade-rules”,

 “type”: “phpstan-extension”,

 “require”: {
     “acme/my-library” : “^1.4”,
     … other packages …
},

“extra”: {
   “phpstan”: {“includes”: [“extension.neon”]}
},

… rest or composer.json …
```

composer.json

```
{
 “name”: “acme/my-library-v1-v2-upgrade-rules”,

 “type”: “phpstan-extension”,

 “require”: {
      “acme/my-library” : “^1.4”,
      … other packages …
},

“extra”: {
  “phpstan”: {“includes”: [“extension.neon”]}
},

… rest or composer.json …
```

extension.neon

```
services:
-
	class: Acme\MyLibrary\UpgradeRule
	tags:
	- phpstan.rules.rule
```

MULTIPLE PHPSTAN CONFIG FILES

phpstan-upgrade.neon

```
parameters:
  customRulesetUsed: true
  paths:
     - src

includes:
   - vendor/acme/upgrade/extension.neon
```

---

phpstan analyse -c phpstan-upgrade.neon

Steps to create custom rules

Tests

- Problem code
- Similar code that’s OK

AST Type

Rule

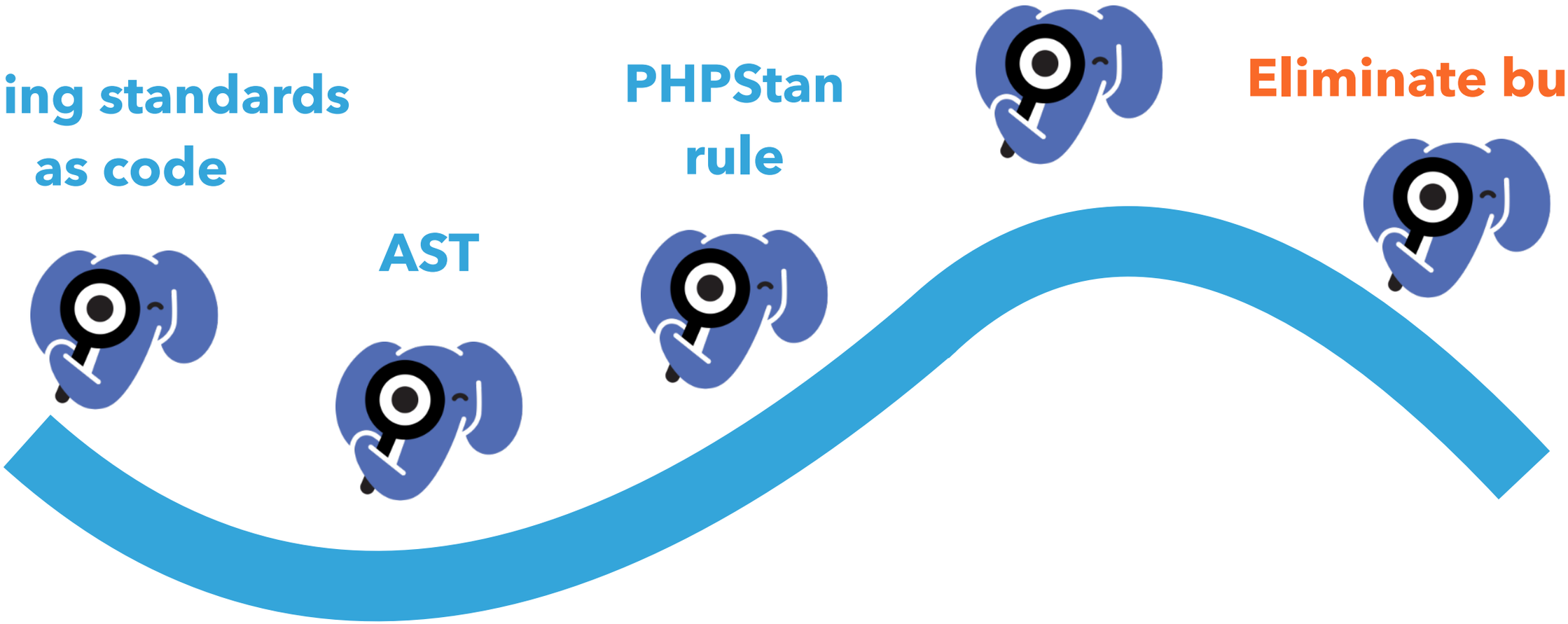
- Early return pattern
- Finish with error

Easy upgrades

Coding standards as code

PHPStan rule

Eliminate bugs



```
class MyRule implements Rule
{
}
```

```
final class CheckRuleIsInExtension
    implements Rule

{
    public function getNodeType(): string
    {
        return InClassNode::class;
    }
```

```
final class CheckRuleIsInExtension
    implements Rule

{
    public function getNodeType(): string
    {
        return InClassNode::class;
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {
```

```
    $classReflection = $scope->getClassReflection();

    if (!$classReflection->isSubclassOf(Rule::class)){
        return [];
    }

    if ($classReflection->isAbstract()) {
        return [];
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

    $classReflection = $scope->getClassReflection();
    if (!$classReflection->isSubclassOf(Rule::class)){
        return [];
    }

    if ($classReflection->isAbstract()) {
        return [];
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {


   $classReflection = $scope->getClassReflection();

   if (!$classReflection->isSubclassOf(Rule::class)){
      return [];
   }

   if ($classReflection->isAbstract()) {
      return [];
   }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {

   $classReflection = $scope->getClassReflection();

   if (!$classReflection->isSubclassOf(Rule::class)){
      return [];
   }

   if ($classReflection->isAbstract()) {
      return [];
   }
```

```
if (ExtensionFileChecker::isInFile($className) {
   return [];
}
```

```
$msg = "Rule [$className]not in extension.neon.”;

return [
   RuleErrorBuilder::message($msg)->build(),
];
```

```
if (ExtensionFileChecker::isInFile($className) {
  return [];
}
```

```
$msg = "Rule [$className]not in extension.neon.”;

return [
  RuleErrorBuilder::message($msg)->build(),
];
```

Run time vs static analysis?

Easy upgrades

Coding standards as code

PHPStan rule

Eliminate bugs

AST

@daveliddament

PHPStan

- Prevent legacy way of working
- Services must have readonly properties
- All non abstract classes must be final
- Use named arguments for booleans

Could we reduce review overhead?

```
#[ValueObject]
#[Service]
#[Dto]
#[Repository]
#[AsCommand]
#[Entity]
#[Controller]
…
```

- Assertions about code
- Display high value code first

PHPStan

Better than AGENTS.md

Ask AI to suggest rules

Ask AI to create rules

Prompt

```
Help me create a PHPStan rule to find [x]. Start with test
cases – examples of code that should trigger the rule, and
similar code that shouldn't. Then identify the right AST
node, build the rule using early returns where sensible, and
verify it against the test cases. Stop after each stage and
wait for my feedback before moving on.
```

Determinism

Using PHPStan means:

- Higher code quality
- Fewer bugs

Custom rules amplify these benefits

PHPStan

Coding Standards as Code

Easy upgrades

Eliminate bugs

Better than AGENTS.md

What PHPStan rules will you create on Monday?

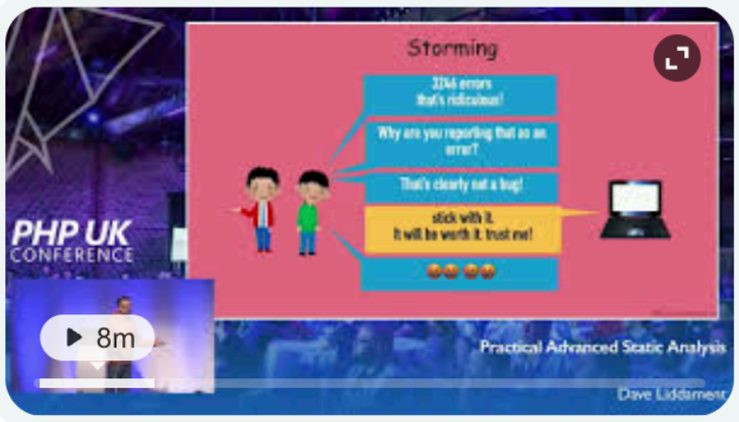
static analysis dave

All Images Videos Shopping Short videos News More Tools

AI Overview

Dave Liddament is a prominent expert in PHP static analysis, known for advocating for tools like PHPStan and Psalm to improve code quality and safety. He created the Static Analysis Results Baseliner (SARB), a tool designed to integrate static analysis into legacy projects by managing existing issues. Dave Liddament +2

This video explains how to use static analysis tools like PHPStan and Psalm:



Practical Advanced Static Analysis - Dave Liddament - PHP ...

PHP UK Conference
YouTube• 30 Mar 2022

Key aspects of Dave Liddament's work include:

- **SARB (Static Analysis Results Baseliner):** A PHP tool used to create a baseline of existing issues in a codebase, allowing developers to focus on new, incoming issues.
- **Practical Static Analysis:** Focuses on using advanced static analysis to find bugs, ensure code quality, and enable safe refactoring.
- **Legacy Code Solutions:** Specializes in introducing modern static analysis tools to old projects without needing to fix thousands of existing issues immediately.
- **Presentations:** Frequent speaker at PHP conferences on topics including static analysis, custom rules, and testing. GitHub +5

DaveLiddament/sarb: Static Analysis Results Baseliner · GitHub

4 Jan 2025 — Static Analysis Baseliner (SARB) is a PHP script that creates a baseline of static analysis…

GitHub

Squash Bugs with Static Analysis | Dave Liddament | IPC 2018

28 Nov 2018 — Squash Bugs with Static Analysis | Dave Liddament | IPC 2018 - YouTube. ... This conte…

YouTube · International PHP Confer…

Introducing Static Analysis Results Baseliner (SARB)

Dave Liddament's technical blog. ... SARB is written in PHP, however can be used to baseline results for any language and any s...

Dave Liddament

Show all

Dave Liddament

www.linkedin.com/in/daveliddament/

@daveliddament@phpc.social

@daveliddament.bsky.social

github.com/DaveLiddament

@daveliddament

Me when I'm not coding!

github.com/DaveLiddament/phpstan-rule-test-helper

github.com/DaveLiddament/sarb

github.com/DaveLiddament/php-language-extensions

php
language
extensions

#[Friend]
#[MustUseResult]
#[NamespaceVisibility]
#[InjectableVersion]
#[Override]
#[RestrictTraitTo]
#[TestTag]

Further information

https://phpstan.org/developing-extensions/rules