Ship Less Bugs: Static Analysis with PHPStan

THRIVE MARKET

Eli White

php[tek] 2026 · eliw.com

Why me? Why PHPStan?

- 30+ years in the industry
- 25+ years exclusively* working in PHP
- Worked in numerous industries

Specifically: I was a disbeliever

I've now implemented PHPStan at 5 different companies

BTW: What about Claude?

What is PHPStan?

PHPStan is static analysis, but goes much deeper than what you might think.

It doesn’t look at single lines of code but parses your entire codebase, so it can detect when a change to a one file, may have ripple effects elsewhere.

From their website:
“PHPStan scans your whole codebase and looks for both obvious & tricky bugs. Even in those rarely executed if statements that certainly aren't covered by tests.”

Wait, you said types?

PHPStan is best known for focus on types:

- It will track the potential type(s) of a variable as it passes throughout the codebase, and alert you if later you attempt to use it as another type *(or most commonly, forget some aspect of a union type)*
- It looks at method params, class properties, constants, inline variables, and more.

But it looks at a lot more than just that:

- Unknown variables and methods
- Wrong numbers of arguments
- Dead code checks
- Nullable type misuse
- Unknown magic methods
- And more...

Basic Concepts

Installation & Execution

0. basic checks, unknown classes/functions/$this->methods, wrong # of args, always undefined variables
1. possibly undefined variables, unknown magic methods/properties
2. unknown methods checked on all expressions (not just $this), validating PHPDocs
3. return types, types assigned to properties
4. dead code checking: always false, guaranteed instanceof, dead else branches, code after return; etc.
5. checking types of arguments passed to methods and functions
6. report missing typehints
7. report partially wrong union types
8. report calling methods and accessing properties on nullable types
9. be strict about explicit mixed type - only allowed operation is to pass it to another mixed
10. be even more strict about the mixed type - reports errors even for implicit mixed (missing type)

Strict-Rules

- A separate mode that adds a number of additional overlays
- Some can be turned on enmass, others are enabled one at a time

Bleeding-Edge

- Often available as a preview of new features before next version of PHPStan is released

Installation & Execution

Install via composer:

```
composer require --dev phpstan/phpstan
```

Execute via command line:

```
vendor/bin/phpstan analyse
```

That's it!

(ok, not fully...)

Levels (11, plus bonus features)

0. basic checks, unknown classes/functions/$this->methods, wrong # of args, always undefined variables
1. possibly undefined variables, unknown magic methods/properties
2. unknown methods checked on all expressions (not just $this), validating PHPDocs
3. return types, types assigned to properties
4. dead code checking: always false, guaranteed instanceof, dead else branches, code after return; etc.
5. checking types of arguments passed to methods and functions
6. report missing typehints
7. report partially wrong union types
8. report calling methods and accessing properties on nullable types
9. be strict about explicit mixed type - only allowed operation is to pass it to another mixed
10. be even more strict about the mixed type - reports errors even for implicit mixed (missing type)

Strict-Rules

- A separate mode that adds a number of additional overlays
- Some can be turned on enmass, others are enabled one at a time

Bleeding-Edge

- Often available as a preview of new features before next version of PHPStan is released

Docblocks vs Native Typehints

Native PHP Typehints work fine!

```php
function isLengthLessThan(string $a, int $b): bool
{
    return strlen($a) < $b;
}
```

PHPStan supports docblocks as well...

```php
/**
 * @param string $a
 * @param int $b
 * @return bool
 */
function isLengthLessThan($a, $b)
{
    return strlen($a) < $b;
}
```

...and you can use both (as long as they match, or *refine*)

```php
/**
 * @param string $a
 * @param int $b
 * @return bool
 */
function isLengthLessThan(string $a, int $b): bool
{
    return strlen($a) < $b;
}
```

@var *(for good & evil)*

```php
class VarPropertyExample extends BaseExample
{
    /** @var string */
    public const CONFIG = 'a|b|c';

    /** @var null|\DateTimeImmutable $when */
    public ?\DateTimeImmutable $when;

    /** @var int|\DateInterval */
    private int|\DateInterval $secondsElapsed;
}
```

@var serves two purposes ...

One is to declare types of properties and constants, not dissimilar to parameters and return types...

The other is to declare inline variable types in your code...

```php
protected function generateInterval(?\DateTimeInterface $since = null): void
{
    /** @var \DateTime */
    $base = $since ?? new \DateTimeImmutable('now');
    $this->secondsElapsed = $base->diff($this->when);
}
```

In general this is seen as an antipattern, as all methods should be properly declaring their return types that you trust, though there are a few cases when you need to use it

Configuration Files

```
includes:
    - vendor/macopedia/phpstan-magento1/extension.neon
    - vendor/phpstan/phpstan-mockery/extension.neon

parameters:
    phpVersion: 80229 # 8.2.29
    tmpDir: .phpstan

    level: 5
    reportWrongPhpDocTypeInVarTag: true
    reportAnyTypeWideningInVarTag: true

    magentoRootPath: /var/www/html/magento
    paths:
        - ./
    excludePaths:
        analyse:
            - vendor
        analyseAndScan:
            # none
    scanDirectories:
        - ../vendor/braintree
```

PHPStan stores configuration in a `phpstan.dist.neon` or `phpstan.neon` file using the NEON file format *(similar to yaml)*

Most of the configuration comes down to **discovery**: Explaining to PHPStan how your codebase is structured.

So that it knows what files it needs to report on, as well as where auxiliary code lives that it needs to understand (such as vendor libraries) to properly parse your codebase.

https://phpstan.org/config-reference

How to Start

Two paths forward

Assuming that you aren't starting from scratch... If so, just set level to 10 and go!

Baselines

- Set your config where you want to be in the end
- Run PHPStan to see all errors & save that as a *baseline*. PHPStan won't report those errors again

Pro: Let's you start enforcing high level immediately

Con: In my experience, leaves you constantly fighting the baseline.

Incremental Fixes

- Configure to level:0
- Run, find errors, fix, commit, push
- Configure to level:1
- *Rinse, Repeat*

Pro: Let's you move codebase forward in a controlled manner

Con: You live at a lower level for a while

HIGHLY SUGGESTED OPTION!

Let’s See Examples

```php
<?php
class DeckBlueprint {
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}

class BadExample extends DeckBlueprint {

    public function addAgenda(\stdClass $title): void {
        (new ImaginaryHttpClient())->sendPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase(), 'agenda');
        } catch (\Throwable) {}
        $this->slideCount += count($pages);
    }

    private function logSlide(string $title): void {
        echo $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }

```

Level 0

Oops, PHPStan can't even run because of a syntax error.

We are missing a closing brace, let's fix that.

```
 ------ ----------------------------------------
  Line   ProgressivelyBadExample0.php
 ------ ----------------------------------------
  32     Syntax error, unexpected EOF on line 32
 ------ ----------------------------------------

 [ERROR] Found 1 error

 ⚠ Result is incomplete because of severe errors. ⚠
  Fix these errors first and then re-run PHPStan
  to get all reported errors.
```

```php
<?php
class DeckBlueprint {
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}

class BadExample extends DeckBlueprint {

    public function addAgenda(\stdClass $title): void {
        (new ImaginaryHttpClient())->sendPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase(), 'agenda');
        } catch (\Throwable) {}
        $this->slideCount += count($pages);
    }

    private function logSlide(string $title): void {
        echo $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 0 - Take 2

3 errors now:

- We have an incorrect class name
- We are using a ‘void’ return
- And we have an undefined property

```
------ ------------------------------------------------------------------------------
 Line   ProgressivelyBadExample0a.php
------ ------------------------------------------------------------------------------
 13     Instantiated class ImaginaryHttpClient not found.
        🪪  class.notFound
        💡  Learn more at phpstan.org/user-guide/discovering-symbols
 15     Result of method BadExample::logSlide() (void) is used.
        🪪  method.void
 17     Access to an undefined property BadExample::$slideCount.
        🪪  property.notFound
        💡  Learn more:
phpstan.org/blog/solving-phpstan-access-to-undefined-property
------ ------------------------------------------------------------------------------

[ERROR] Found 3 errors
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(\stdClass $title): void {
        (new HttpClient())->sendPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase(), 'agenda');
        } catch (\Throwable) {}
        $this->slideCount += count($pages);
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 0 - 3rd time?

Well shoot. Now that we are calling HttpClient correctly, turns out we need weren't using the right number of parameters.

Let's add in the URL for the client

```
------ ------------------------------------------------------------------
 Line   ProgressivelyBadExample0b.php
------ ------------------------------------------------------------------
 13     Class HttpClient constructor invoked with 0 parameters, 1
required.
        arguments.count
------ ------------------------------------------------------------------

[ERROR] Found 1 error
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(\stdClass $title): void {
        (new HttpClient('api.tm.com'))->sendPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase(), 'agenda');
        } catch (\Throwable) {}
        $this->slideCount += count($pages);
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 1

2 new issues now at level 1

- Calling logSlide with 2 params
- $pages might not be defined due to an exception

```
------ ------------------------------------------------------------------
 Line   ProgressivelyBadExample1.php
------ ------------------------------------------------------------------
 15     Method BadExample::logSlide() invoked with 2 parameters,
          1 required.
        🪪 arguments.count
 17     Variable $pages might not be defined.
        🪪 variable.undefined
------ ------------------------------------------------------------------

[ERROR] Found 2 errors
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(\stdClass $title): void {
        (new HttpClient('api.tm.com'))->sendPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase());
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 3

Now level 3 highlights different issues for us:

- We've been calling 'sendPayload()' this whole time when the method is named 'setPayload'
- We are trying to call a method on stdClass which has none!

```
------ ------------------------------------------------------
 Line   ProgressivelyBadExample2.php
------ ------------------------------------------------------
 13     Call to an undefined method HttpClient::sendPayload().
        method.notFound
 15     Cannot call method toUpperCase() on stdClass.
        method.nonObject
------ ------------------------------------------------------

 [ERROR] Found 2 errors
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(string $title): void {
        (new HttpClient('api.tm.com'))->setPayload('intro slide');
        try {
            $pages = $this->logSlide($title->toUpperCase());
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 3 - Again?

We tried, but now level 3 shows a new issue, this isn't JavaScript or Java. You can't call methods on scalar types like string!

```
------ ------------------------------------------
 Line   ProgressivelyBadExample3.php
------ ------------------------------------------
 15     Cannot call method toUpperCase() on string.
        🪪 method.nonObject
------ ------------------------------------------

 [ERROR] Found 1 error
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(string $title): void {
        (new HttpClient('api.tm.com'))->setPayload('intro slide');
        try {
            $pages = $this->logSlide(strtoupper($title));
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['details']['name']);
    }
}
```

Level 4

Level 3 passes, but level 4 points out that we are accessing an illegal array offset, based upon the complex typehint.

```
------ ------------------------------------------------------
 Line   ProgressivelyBadExample4.php
------ ------------------------------------------------------
 30     Offset 'name' on string in isset() does not exist.
        🪪 isset.offset
------ ------------------------------------------------------

 [ERROR] Found 1 error
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(string $title): void {
        (new HttpClient('api.tm.com'))->setPayload('intro slide');
        try {
            $pages = $this->logSlide(strtoupper($title));
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }

    private function logSlide(string $title): string {
        return $title;
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['name']);
    }
}
```

Level 5

We've passed 4 ... but 5 points out to us that we've been trying to call 'count' on a string.

```
------ ------------------------------------------------------------------------
 Line   ProgressivelyBadExample4.php
------ ------------------------------------------------------------------------
 16     Parameter #1 $value of function count expects array|Countable,
          string given.
        argument.type
------ ------------------------------------------------------------------------

[ERROR] Found 1 error
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(string $title): void {
        (new HttpClient('api.tm.com'))->setPayload('intro slide');
        try {
            $pages = $this->logSlide(strtoupper($title));
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }

    private function logSlide(string $title): array {
        return range(0, mt_rand(0,9));
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['name']);
    }
}
```

Level 6

At level 6 you start to get much more pedantic requirements (in a good way!) You now must declare value types for any arrays/objects.

```
------ ------------------------------------------------------------------------------
 Line   ProgressivelyBadExample5.php
------ ------------------------------------------------------------------------------
 20     Method BadExample::logSlide() return type has no value type
        specified in iterable type array.
        🪪  missingType.iterableValue
        💡  See:
phpstan.org/solving-phpstan-no-value-type-specified-in-iterable-type
------ ------------------------------------------------------------------------------

[ERROR] Found 1 error
```

```php
<?php
class DeckBlueprint {
    protected int $slideCount = 0;
    public function render(string $slideId): void {}
}
class HttpClient {
    public function __construct(protected string $url) {}
    public function setPayload(string $name): void {}
}
class BadExample extends DeckBlueprint {

    public function addAgenda(string $title): void {
        (new HttpClient('api.tm.com'))->setPayload('intro slide');
        try {
            $pages = $this->logSlide(strtoupper($title));
            $this->slideCount += count($pages);
        } catch (\Throwable) {}
    }
    /** @return int[] */
    private function logSlide(string $title): array {
        return range(0, mt_rand(0,9));
    }

    public function render(string $slideId): void {
        parent::render($slideId);
    }

    /** @param array<string, string> $speakerMeta */
    public function isNamedSpeaker(array $speakerMeta): bool {
        return isset($speakerMeta['name']);
    }
}
```

Passing level 6!

Stopping this exercise here because things get even more complex going forward.

But **speaking of complex...**

```
[OK] No errors                                                    .
```

Complex Type System

PHPStan can refine types more finely than PHP

Integer Ranges
- positive-int
- negative-int
- non-positive-int
- non-negative-int
- non-zero-int
- int<0, 100>
- int<min, 100>
- int<50, max>

Self Referencial
- @return static
- @return self
- @return $this

Union/Intersection
- Type1|Type2
- Type1&Type2

Keys/Values
- key-of<TYPE::CONST>
- value-of<TYPE::CONST>
- value-of<BackedEnum>
- self::PREFIX_*
- Foo::*

General Array Types
- Type[]
- array<Type>
- array<Type, Type>
- non-empty-array<Type>
- non-empty-array<Type, Type>
- list<Type>
- non-empty-list<Type>

Advanced Strings
- callable
- numeric-string
- non-empty-string
- non-falsy-string
- literal-string
- lowercase-string
- class-string

Literals
- 3.14
- 'yes'|'no'
- scalar

... and more!

Conditional Return Types

```php
/**
 * @return ($size is positive-int ? non-empty-array<int> : array<int>)
 */
public function fillArray(int $size): array
{
    ...
}
```

```php
/**
 * @return ($lookup is null ? null : ($lookup is int<min, 100> ? 'n/a' : value-of<self::SKUS>))
 */
public function findSkuByID(?int $lookup): ?string
{
    ...
}
```

Array Shapes

```php
/**
 * A 3-tuple (array of exactly 3 ints)
 * @param array{int, int, int} $coord
 */
function xyz(array $coord): void {}
```

```php
/**
 * A detailed settings array (such as for curl).  The ? makes headers optional
 *
 * @param array{'url': string, 'return': bool, 'headers'?: array<string,string>} $settings
 */
function setSettings(array $coord): void {}
```

Custom Types

```php
/**
 * Define anything as a custom type to reuse later:
 *
 * @phpstan-type ApiSettings array{'url': string, 'return': bool, 'headers'?: array<string,string>}
 */
class ApiManager
{
    /** @param ApiSettings $settings */
    public function setSettings(array $settings): void {}
}

/**
 * Import a type from another class
 *
 * @phpstan-import-type ApiSettings from \ApiManager as ThirdPartySettings
 */
class ApiCaller
{
    /** @return ThirdPartySettings */
    public function generateSettings(): array {}
}
```

Generics

```php
/**
 * @template T
 */
class Collection
{
    /** @var T[] */
    protected array $storage = [];

    /** @param T $item */
    public function add($item): void
    {
        $this->storage[] = $item;
    }

    /** @return T */
    public function get(int $index)
    {
        return $this->storage[$index];
    }
}
```

```php
/**
 * @extends Collection<Cat>
 */
class CatCollection extends Collection
{
    /* Read in a list of cat objects */
}
```

```php
/**
 * @template TK
 * @extends Collection<TK>
 */
class BackedCollection extends Collection
{
    /* Transitive Generics */
}
```

… and the rabbit hole goes deeper

Rate the talk:
joind.in/talk/21086

The End?

Eli White

eliw.com

Questions?

https://phpstan.org/writing-php-code/phpdocs-basics