Strict typing & static analysis

Rob Allen

January 2020

Use Types to Help Focus on What You're Doing, Not How You're Doing It

Anthony Ferrara, 2016

PHP Types

boolean
float
array
callable
resource

integer
string
object
iterable
NULL

PHP is a *dynamically typed* language

Type Juggling

PHP will automatically convert types when it can.

```
var_dump("Two + three = " . 5);
```

Type Juggling

PHP will automatically convert types when it can.

```
var_dump("Two + three = " . 5);
// string(15) "Two + three = 5"
```

Type Juggling

PHP will automatically convert types when it can.

```
var_dump("Two + three = " . 5);
// string(15) "Two + three = 5"

var_dump("10" + 5);
```

Type Juggling

PHP will automatically convert types when it can.

```php
var_dump("Two + three = " . 5);
// string(15) "Two + three = 5"

var_dump("10" + 5);
// int(15)
```

But...

```
$result = 10 + "10,000";
var_dump($result);
```

But...

```
$result = 10 + "10,000";
var_dump($result);

// int(20)
```

But...

```
$result = "false";
if ($result) {
  echo "TRUE";
} else {
  echo "FALSE";
}
```

But...

```php
$result = "false";
if ($result) {
  echo "TRUE";
} else {
  echo "FALSE";
}

// TRUE
```

You can't trust a computer!

Test all the things

Validate your inputs

```
function foo($i, $b) {
  if (! is_int($i)) {
    throw new Exception('$i must be an integer');
  }
  if (! is_bool($b)) {
    throw new Exception('$b must be a boolean');
  }
  // ...
}
```

& write tests for the new code-paths

A type system solves this class of errors

If we filter input by a type — you can think of it as a subcategory of all available input — many of the tests become obsolete.

@brendt_gd

Function Type Declarations

```
function foo(int $i, bool $b) : string {
  // ...
}
```

- `$i` is always an integer
- `$b` is always a boolean
- `foo()` will always return a string

This function is much easier to reason about

Coercion of Typed Declarations

```php
function foo(int $i, bool $b) : string {
  // ...
}

foo("1", 1);

// $i = int(1)
// $b = bool(true)
```

Strictly Typed Declarations

```
declare(strict_types=1);

foo("1", 1);
```

```
Fatal error: Uncaught TypeError: Argument 1 passed to
foo() must be of the type int, string given
```

This reduces cognitive load

Typed Properties in PHP 7.4!

We can now add types to class properties

```
class Person
{
  public int $age;
}
```

Initialisation of Typed Properties

```
$liz = new Person();
var_dump($liz->age);
```

Initialisation of Typed Properties

```
$liz = new Person();
var_dump($liz->age);
```

```
Fatal error: Uncaught Error: Typed property Person::$age
must not be accessed before initialization
```

Coercion of Typed Properties

```php
class Person { /* ... */}

$liz = new Person();
$liz->age = "93";  // int(93)
```

Strictly Typed Properties

```
declare(strict_types=1);

$liz = new Person();
$liz->age = "93";


Fatal error: Uncaught TypeError:
Typed property Person::$age must be int, string used
```

But that's all at runtime…

Static Analysis checks our code before we run it

Static Analysis

Static code analysers simply reads your code and points out errors

They ensure:

- no syntax errors
- classes, methods, functions constants, variables exist
- no arguments or variables unused
- DocBlocks make sense
- data passed to methods are the correct type

Static Analysers

- Phan
- PHPStan
- Psalm

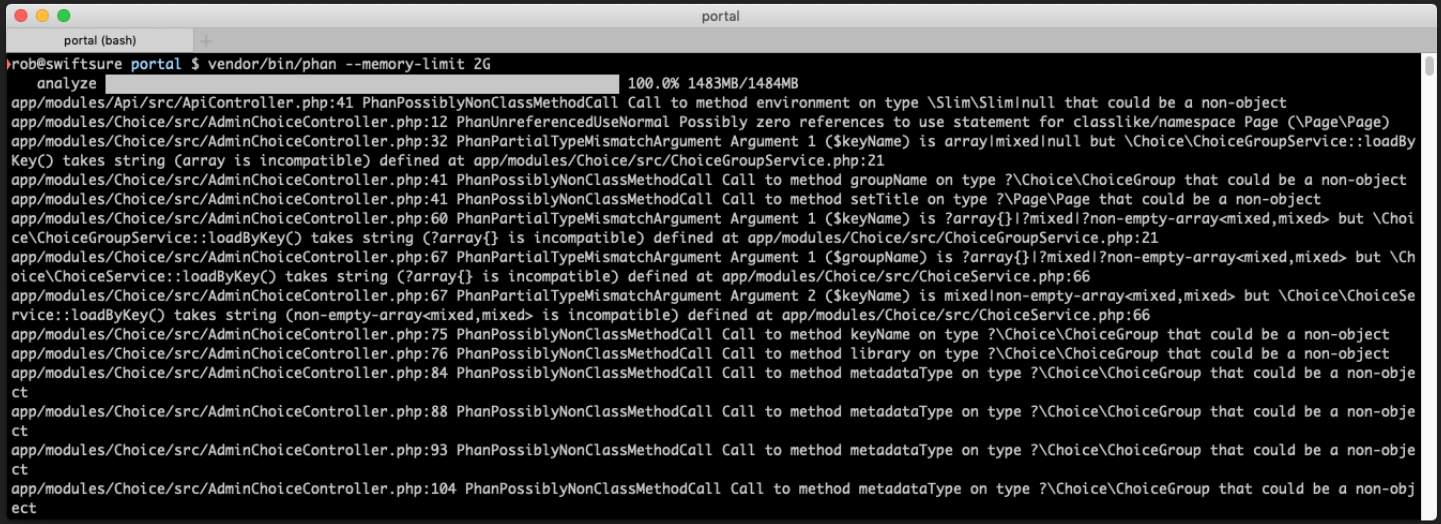
Phan

- Developed by Rasmus Lerdorf & Etsy
- Stable
- Parses entire code base and then analyses
- Requires php-ast extension
- Doesn't parse `/** @var Foo $foo */`; use `assert()` instead

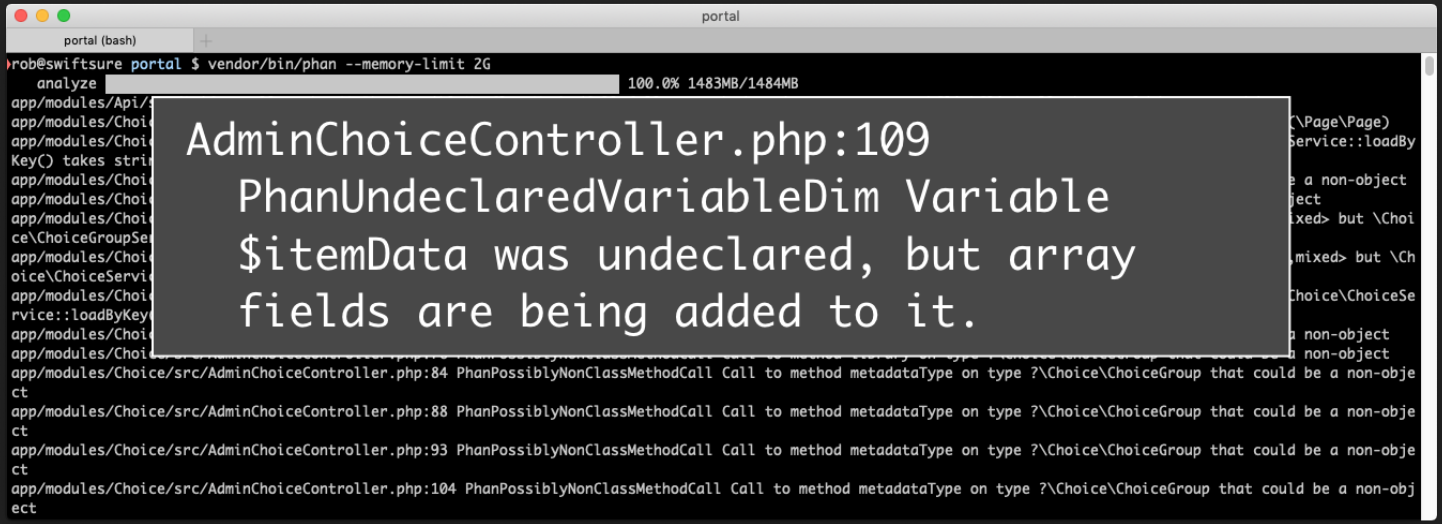
Phan

1. `composer require --dev "phan/phan"`
2. `vendor/bin/phan --init --init-level=1`
3. `vendor/bin/phan --memory-limit 2G`

Phan output


portal
portal (bash)
rob@swiftsure portal $ vendor/bin/phan --memory-limit 2G
analyze 100.0% 1483MB/1484MB
app/modules/Api/src/ApiController.php:41 PhanPossiblyNonClassMethodCall Call to method environment on type \Slim\Slim|null that could be a non-object
app/modules/Choice/src/AdminChoiceController.php:12 PhanUnreferencedUseNormal Possibly zero references to use statement for classlike/namespace Page (\Page\Page)
app/modules/Choice/src/AdminChoiceController.php:32 PhanPartialTypeMismatchArgument Argument 1 ($keyName) is array|mixed|null but \Choice\ChoiceGroupService::loadBy
Key() takes string (array is incompatible) defined at app/modules/Choice/src/ChoiceGroupService.php:21
app/modules/Choice/src/AdminChoiceController.php:41 PhanPossiblyNonClassMethodCall Call to method groupName on type ?\Choice\ChoiceGroup that could be a non-object
app/modules/Choice/src/AdminChoiceController.php:41 PhanPossiblyNonClassMethodCall Call to method setTitle on type ?\Page\Page that could be a non-object
app/modules/Choice/src/AdminChoiceController.php:60 PhanPartialTypeMismatchArgument Argument 1 ($keyName) is ?array{}|?mixed|?non-empty-array<mixed,mixed> but \Choi
ce\ChoiceGroupService::loadByKey() takes string (?array{} is incompatible) defined at app/modules/Choice/src/ChoiceGroupService.php:21
app/modules/Choice/src/AdminChoiceController.php:67 PhanPartialTypeMismatchArgument Argument 1 ($groupName) is ?array{}|?mixed|?non-empty-array<mixed,mixed> but \Ch
oice\ChoiceService::loadByKey() takes string (?array{} is incompatible) defined at app/modules/Choice/src/ChoiceService.php:66
app/modules/Choice/src/AdminChoiceController.php:67 PhanPartialTypeMismatchArgument Argument 2 ($keyName) is mixed|non-empty-array<mixed,mixed> but \Choice\ChoiceSe
rvice::loadByKey() takes string (non-empty-array<mixed,mixed> is incompatible) defined at app/modules/Choice/src/ChoiceService.php:66
app/modules/Choice/src/AdminChoiceController.php:75 PhanPossiblyNonClassMethodCall Call to method keyName on type ?\Choice\ChoiceGroup that could be a non-object
app/modules/Choice/src/AdminChoiceController.php:76 PhanPossiblyNonClassMethodCall Call to method library on type ?\Choice\ChoiceGroup that could be a non-object
app/modules/Choice/src/AdminChoiceController.php:84 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:88 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:93 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:104 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obj
ect

Phan output


portal
portal (bash)
rob@swiftsure portal $ vendor/bin/phan --memory-limit 2G
analyze
100.0% 1483MB/1484MB
AdminChoiceController.php:109
PhanUndeclaredVariableDim Variable
$itemData was undeclared, but array
fields are being added to it.
app/modules/Choice/src/AdminChoiceController.php:84 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:88 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:93 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obje
ct
app/modules/Choice/src/AdminChoiceController.php:104 PhanPossiblyNonClassMethodCall Call to method metadataType on type ?\Choice\ChoiceGroup that could be a non-obj
ect

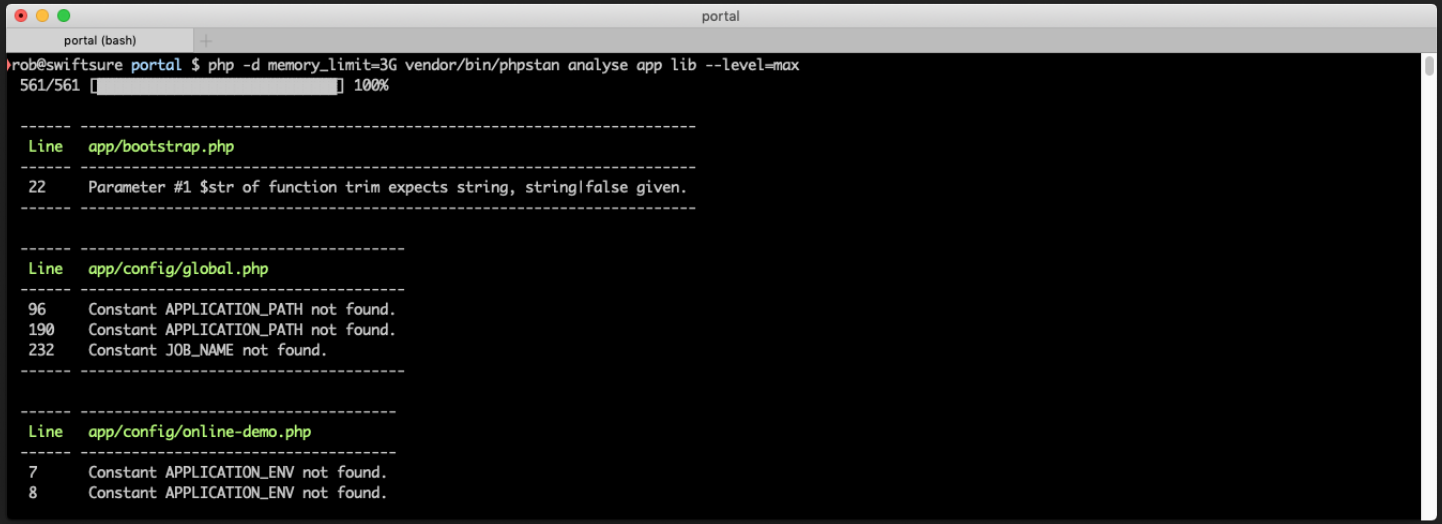
PHPStan

- Developed by Ondřej Mirtes
- Fast
- Can analyse subset of code base
- Autoloads classes
- Support for framework specific magic methods

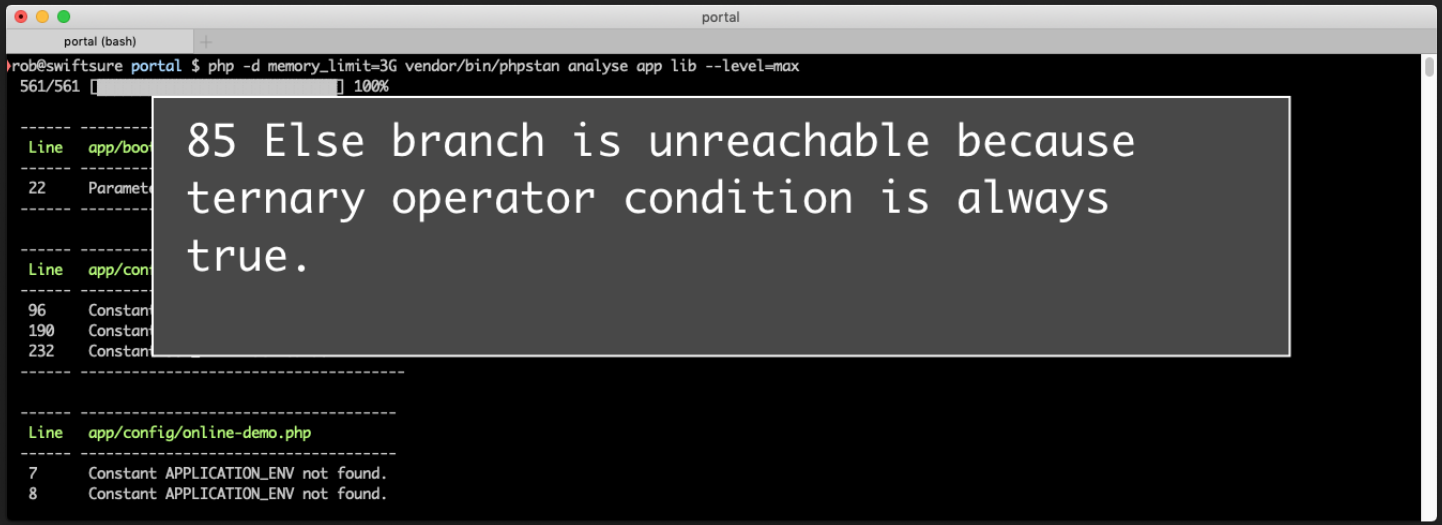
PHPStan

1. `composer require --dev "phpstan/phpstan"`
2. `vendor/bin/phpstan analyse app lib --level=max`

PHPStan output



PHPStan output



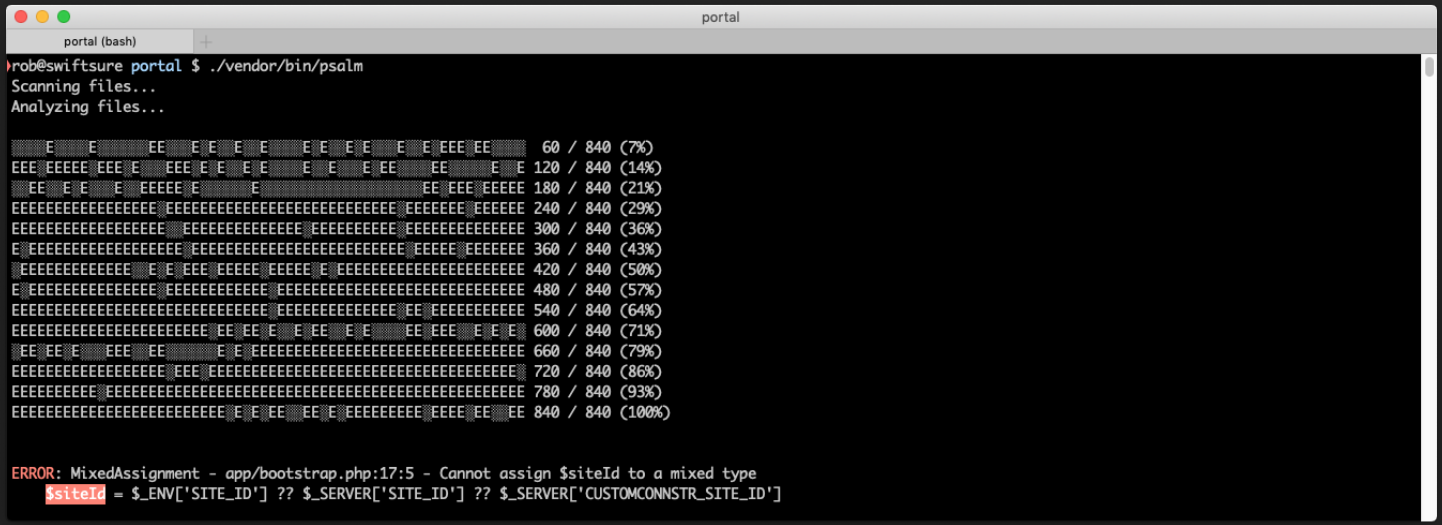
Psalm

- Developed by Vimeo
- Comprehensive
- Can analyse subset of code base
- Autoloads classes
- Support custom PHPDoc tags (e.g. `@psalm-param`)

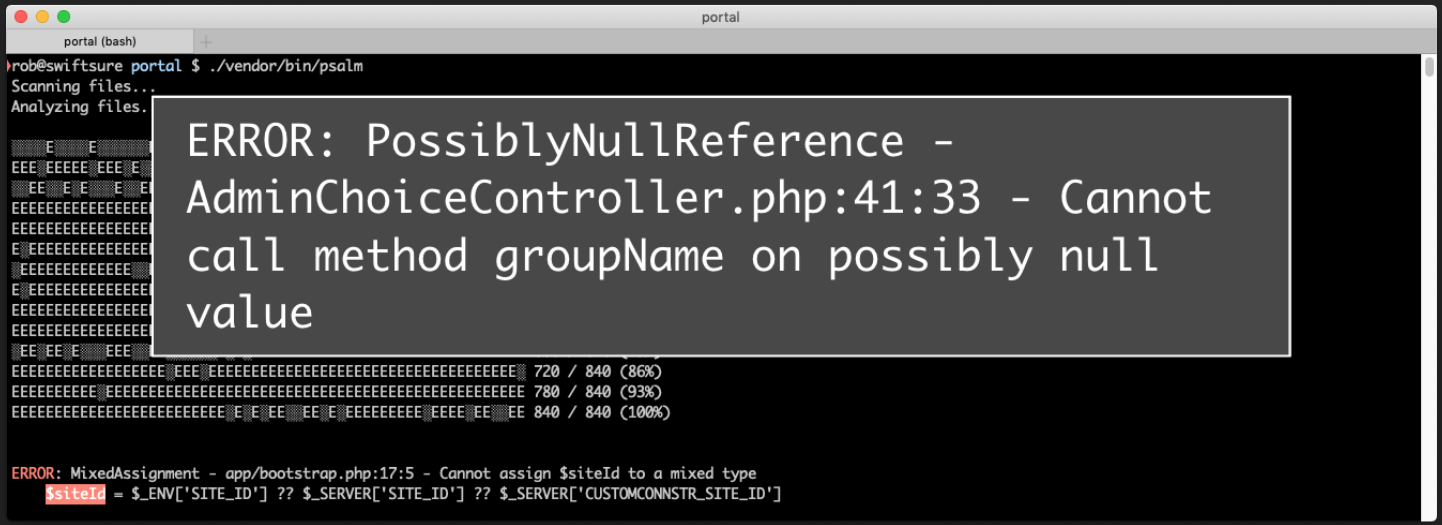
Psalm

1. `composer require --dev "vimeo/psalm"`
2. `vendor/bin/psalm --init`
3. `vendor/bin/psalm`

Psalm output



Psalm output



Observations

- All are good at finding type issues
- All three also found found different issues
  - Phan found unused `use` statements
  - PHPStan found logic errors
  - Psalm is more type-picky, especially with `mixed`

What Problems Do Static Analysers Find?

Report 1

```
class Acl extends BaseAcl
{
    protected $siteKey = '';
    protected $settings;

    public function __construct(array $settings) {
        $this->settings = $settings;
    }
```

Report 1

```
class Acl extends BaseAcl
{
    protected $siteKey = '';
    protected $settings;

    public function __construct(array $settings) {
        $this->settings = $settings;
    }
```

```
INFO: MissingPropertyType - Property Acl::$siteKey
does not have a declared type - consider string
```

Report 1: Fix (PHP 7.4)

```
class Acl extends BaseAcl
{
    protected string $siteKey = '';
    protected $settings;
```

Add property's type

Report 1: Fix (PHP 7.3)

```
class Acl extends BaseAcl
{
    /** @var string */
    protected $siteKey = '';
    protected $settings;
```

Use a docblock to declare property's type

Report 2

```
public function setSiteKey($siteKey)
{
    $this->siteKey = $siteKey;
    return $this;
}
```

Report 2

```
  public function setSiteKey($siteKey)
  {
      $this->siteKey = $siteKey;
      return $this;
  }
```

```
INFO: MissingParamType - Parameter $siteKey has
no provided type
INFO: MissingReturnType - Method setSiteKey does
not have a return type, expecting Acl
```

Report 2: Fix

```
public function setSiteKey(string $siteKey)
{
    $this->siteKey = $siteKey;
    return $this;
}
```

Add parameter's type

Report 2: Fix

```
public function setSiteKey(string $siteKey): Acl
{
    $this->siteKey = $siteKey;
    return $this;
}
```

Add parameter's type & return type

An bona fide potential bug!

```
public function populate(): void
{
    $siteKey = $this->getSiteKey() ?: 1;
    $routes = $this->router->slugRoutes($siteKey);
    foreach ($routes as $route) {
```

An bona fide potential bug!

```
public function populate(): void
{
    $siteKey = $this->getSiteKey() ?: 1;
    $routes = $this->router->slugRoutes($siteKey);
    foreach ($routes as $route) {
```

```
ERROR: InvalidScalarArgument Argument 1 of slugRoutes
expects null|string, int(1)|non-empty-string provided
```

Collections

```
class ChoiceService
{
  public function fetchChoices(): array { … }
  …
```

usage:

```
foreach ($service->fetchChoices() as $choice) { … }
```

Collections

```
/**
 * @return Choice[]
 */
public function fetchChoices(): array { … }
```

Declare type of array returned using a docblock

Collections

```php
/**
 * @return Choice[]
 */
public function fetchChoices(): array { … }
```

```
ERROR: InvalidReturnStatement - The type
'array{0: Choice|null}' does not match the declared
return type 'array<array-key, Choice>' for
ChoiceService::fetchChoices
```

False positive?

```
// loadByKey() returns: ?Choice
$choice = $choiceService->loadByKey($keyName);
if (!$choice) {
  $this->redirectToHome(); // throws an Exception
}
$data = $choice->getArrayCopy();
return $this->format($data);
```

False positive?

```
// loadByKey() returns: ?Choice
$choice = $choiceService->loadByKey($keyName);
if (!$choice) {
  $this->redirectToHome(); // throws an Exception
}
$data = $choice->getArrayCopy();
return $this->format($data);
```

```
Cannot call method getArrayCopy on possibly null
value
```

False positive?: Fix

```php
/**
 * @psalm-return never-return
 */
public function redirectToHome(): void
{
    // ...
}
```

Guarantees that the function exits or throws an exception

Psalm annotations

`@psalm-var, @psalm-param & @psalm-return`:

Psalm versions for types not understood by phpDocumenter

`never-return` adds an implicit `exit` at end of function

Psalm annotations

`@psalm-var, @psalm-param & @psalm-return:`

Psalm versions for types not understood by phpDocumenter
`never-return` adds an implicit `exit` at end of function

`@psalm-seal-properties:`

Prevent __set/__get not declared with `@property`

Psalm annotations

`@psalm-var`, `@psalm-param` & `@psalm-return`:
Psalm versions for types not understood by phpDocumenter
`never-return` adds an implicit `exit` at end of function

`@psalm-seal-properties`:
Prevent __set/__get not declared with `@property`

`@psalm-suppress`:
Suppress specific Psalm issues

Full list: https://psalm.dev/docs/annotating_code/supported_annotations

Applying Static Analysis to Your Project

Rob Allen ~ @akrabat

Applying to Your Project

- Add to CI

Applying to Your Project

- Add to CI
- Use *levels* and fix the "easy" items first
  - Set to lowest level and fix all issues
  - Increase level and repeat

1745 errors found
3229 other issues found

Baselines

1. Log all current errors to a *baseline file*:
   ```
   vendor/bin/psalm --set-baseline=baseline.xml
   ```

Baselines

1. Log all current errors to a *baseline file*:
   `vendor/bin/psalm --set-baseline=baseline.xml`
2. Subsequent `psalm` runs will treat all errors in the *baseline file* as warnings

Baselines

1. Log all current errors to a *baseline file*:
   `vendor/bin/psalm --set-baseline=baseline.xml`
2. Subsequent `psalm` runs will treat all errors in the *baseline file* as warnings
3. Fix warnings as you can (and allow no new errors!)

Baselines

1. Log all current errors to a *baseline file*:
   `vendor/bin/psalm --set-baseline=baseline.xml`
2. Subsequent `psalm` runs will treat all errors in the *baseline file* as warnings
3. Fix warnings as you can (and allow no new errors!)
4. Update baseline as you go:
   `vendor/bin/psalm --update-baseline`

To Sum Up

Takeaways

- PHP 7's type declarations reduce cognitive load
- Upgrade to PHP 7.4 for typed properties!
- Static analysis finds bugs in your code
- Use plugins to give additional info to your static analyser
- Autofixers are great! e.g. psalter, rector
- Add static analysis to your CI


Thank you!