Talk

PHP UK CONFERENCE

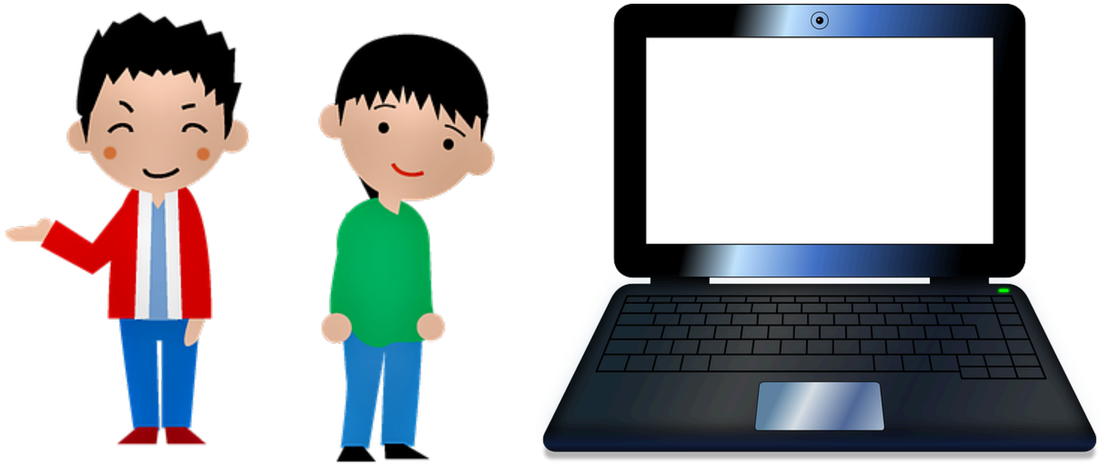
Practical Advanced Static Analysis

Dave Liddament
@DaveLiddament

16th, 17th & 18th February 2022
www.phpconference.co.uk

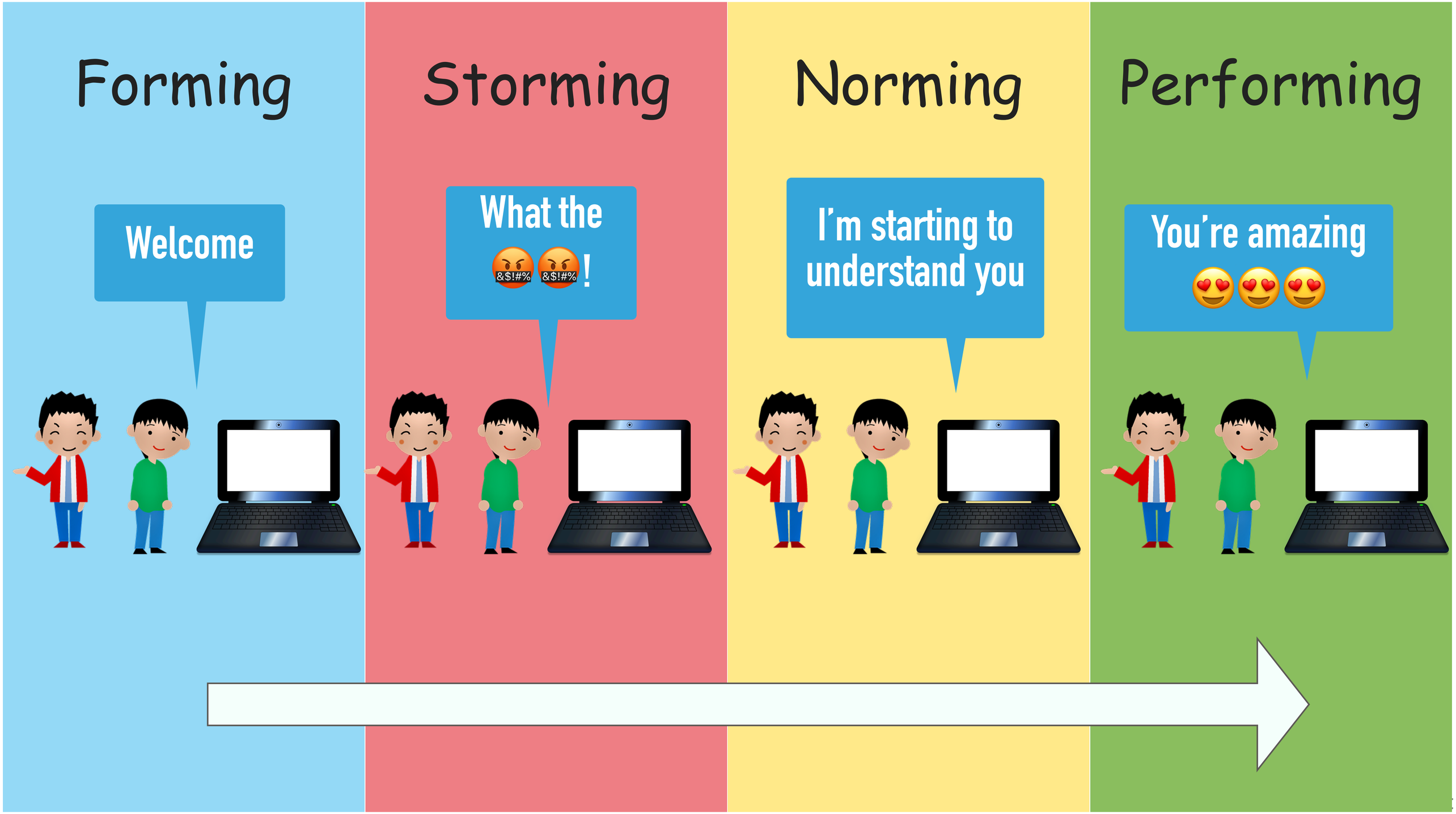
Imagine a new team member

1. Find bugs & prevent bugs
2. Push you to write clearer code
3. Add new features and refactor with confidence
4. Extend PHP to provide new features

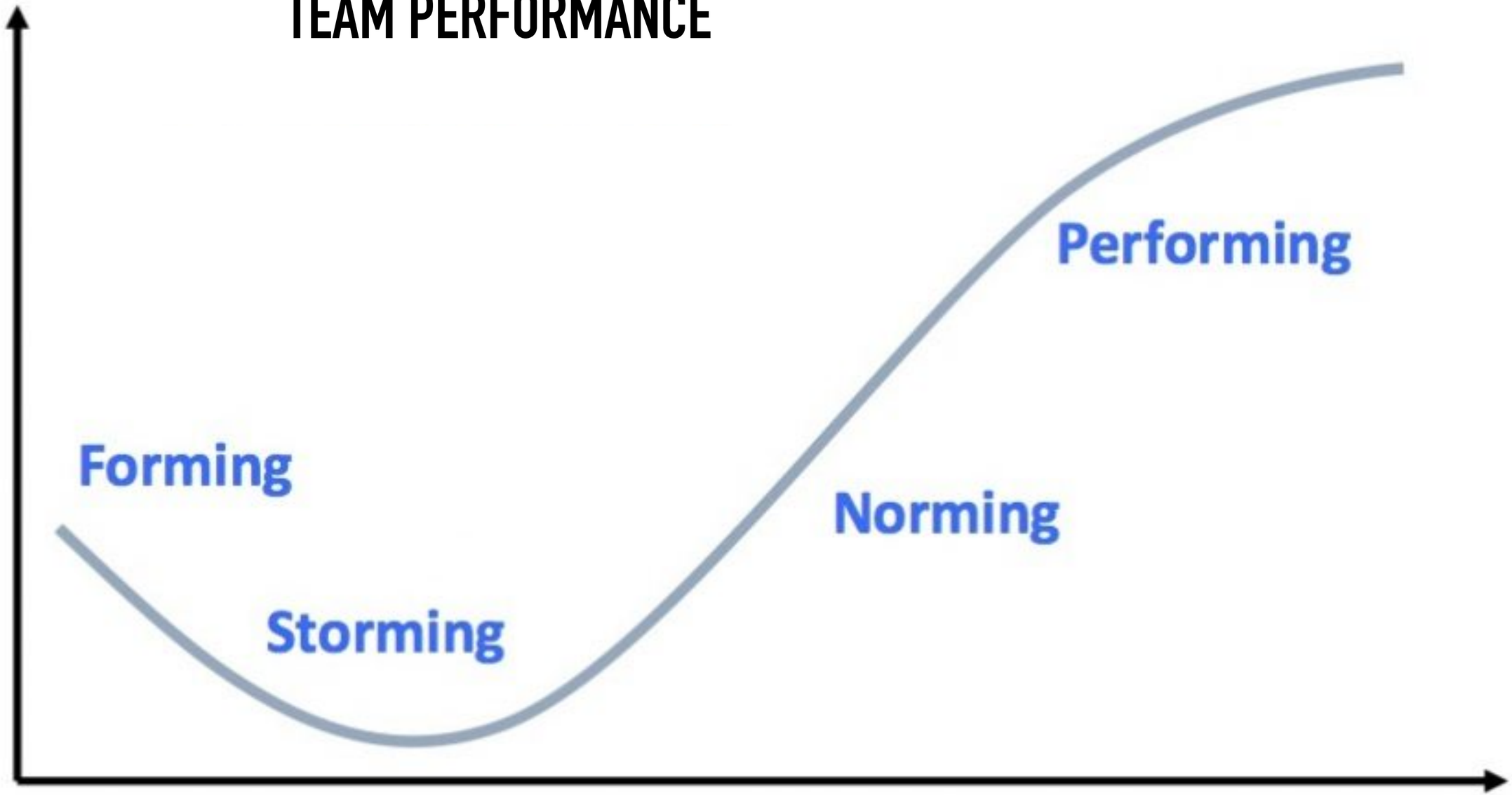


Psalm

@daveliddament


Forming
Welcome
Storming
What the
&$!#%
&$!#%
!
Norming
I'm starting to understand you
Performing
You're amazing

TEAM PERFORMANCE



@daveliddament

Forming

What do you do?

I look at your code and find bugs.

🚨🚨🚨

These examples are deliberately simple.

```
function register(User $user) {
  // some implementation
}
$user = 1;
register($user);
```

This code will fail if it is run

Surely our tests will catch these bugs?

```
function cost(string $type): int
{
    if ($type === "CHILD") {
        $price = 10;
    }
    if ($type === "ADULT") {
        $price = 20;
    }
    return $price;
}
```

Price might not be set

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20

✅ All tests pass

💯 Code coverage

Static analysis shows you where your code is, or could be, incorrect.

Tests tell you that the behaviour is correct, but ONLY for the scenarios tested.

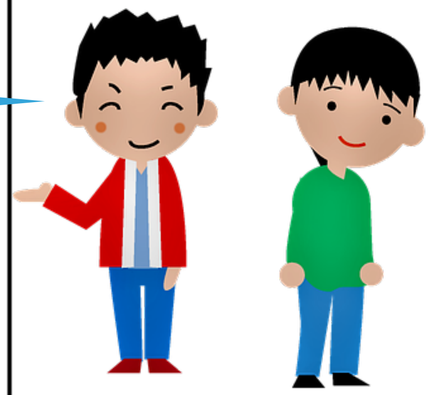
```php
<?php

function foo(string $s) : void {
    return "bar";
}

$a = ["hello", 5];
foo($a[1]);
foo();

if (rand(0, 1)) $b = 5;
echo $b;

$c = rand(0, 5);
if ($c) {} elseif ($c) {}
```

```
Psalm output (using commit add7c14):

ERROR: InvalidReturnStatement - 4:5 - No return values are expected for foo

INFO: UnusedParam - 3:21 - Param $s is never referenced in this method

ERROR: InvalidReturnType - 3:27 - The declared return type 'void' for foo is incorrect, got 'string'
```

↗ Shrink

Get link

Sounds amazing, can we try this out?



Head over to psalm.dev or phpstan.org to try out the virtual playground.

AbstractInstantiation
AbstractMethodCall
ArgumentIssue
ArgumentTypeCoercion
AssignmentToVoid
CircularReference
ClassIssue
CodeIssue
ComplexFunction
ComplexMethod
ConfigIssue
ConflictingReferenceConstraint
ConstructorSignatureMismatch
ContinueOutsideLoop
DeprecatedClass
DeprecatedConstant
DeprecatedFunction
DeprecatedInterface
DeprecatedMethod
DeprecatedProperty
DeprecatedTrait
DocblockTypeContradiction
DuplicateArrayKey
DuplicateClass
DuplicateConstant
DuplicateEnumCase
DuplicateEnumCaseValue
DuplicateFunction
DuplicateMethod
DuplicateParam
EmptyArrayAccess
ExtensionRequirementViolation
FalsableReturnStatement
FalseOperand
ForbiddenCode
ForbiddenEcho
FunctionIssue
IfThisIsMismatch
ImplementationRequirementViolation
ImplementedParamTypeMismatch
ImplementedReturnTypeMismatch
ImplicitToStringCast
ImpureByReferenceAssignment
ImpureFunctionCall
ImpureMethodCall
ImpurePropertyAssignment
ImpurePropertyFetch
ImpureStaticProperty
ImpureStaticVariable
ImpureVariable
InaccessibleClassConstant
InaccessibleMethod
InaccessibleProperty
InterfaceInstantiation
InternalClass
InternalMethod
InternalProperty
InvalidArgument

InvalidArrayAccess
InvalidArrayAssignment
InvalidArrayOffset
InvalidAttribute
InvalidCast
InvalidCatch
InvalidClass
InvalidClone
InvalidDocblock
InvalidDocblockParamName
InvalidEnumBackingType
InvalidEnumCaseValue
InvalidExtendClass
InvalidFalsableReturnType
InvalidFunctionCall
InvalidGlobal
InvalidIterator
InvalidLiteralArgument
InvalidMethodCall
InvalidNamedArgument
InvalidNullableReturnType
InvalidOperand
InvalidParamDefault
InvalidParent
InvalidPassByReference
InvalidPropertyAssignment
InvalidPropertyAssignmentValue
InvalidPropertyFetch
InvalidReturnStatement
InvalidReturnType
InvalidScalarArgument
InvalidScope
InvalidStaticInvocation
InvalidStringClass
InvalidTemplateParam
InvalidThrow
InvalidToString
InvalidTraversableImplementation
InvalidTypeImport
LessSpecificImplementedReturnType
LessSpecificReturnStatement
LessSpecificReturnType
LoopInvalidation
MethodIssue
MethodSignatureMismatch
MethodSignatureMustOmitReturnType
MismatchingDocblockParamType
MismatchingDocblockPropertyType
MismatchingDocblockReturnType
MissingClosureParamType
MissingClosureReturnType
MissingConstructor
MissingDependency
MissingDocblockType
MissingFile
MissingImmutableAnnotation
MissingParamType
MissingPropertyType

MissingReturnType
MissingTemplateParam
MissingThrowsDocblock
MixedArgument
MixedArgumentTypeCoercion
MixedArrayAccess
MixedArrayAssignment
MixedArrayOffset
MixedArrayTypeCoercion
MixedAssignment
MixedClone
MixedFunctionCall
MixedInferredReturnType
MixedIssue
MixedIssueTrait
MixedMethodCall
MixedOperand
MixedPropertyAssignment
MixedPropertyFetch
MixedPropertyTypeCoercion
MixedReturnStatement
MixedReturnTypeCoercion
MixedStringOffsetAssignment
MoreSpecificImplementedParamType
MoreSpecificReturnType
MutableDependency
NamedArgumentNotAllowed
NoEnumProperties
NoInterfaceProperties
NoValue
NonInvariantDocblockPropertyType
NonInvariantPropertyType
NonStaticSelfCall
NullArgument
NullArrayAccess
NullArrayOffset
NullFunctionCall
NullIterator
NullOperand
NullPropertyAssignment
NullPropertyFetch
NullReference
NullableReturnStatement
OverriddenMethodAccess
OverriddenPropertyAccess
ParadoxicalCondition
ParamNameMismatch
ParentNotFound
ParseError
PluginIssue
PossibleRawObjectIteration
PossiblyFalseArgument
PossiblyFalseIterator
PossiblyFalseOperand
PossiblyFalsePropertyAssignmentValue
PossiblyFalseReference
PossiblyInvalidArgument
PossiblyInvalidArrayAccess

PossiblyInvalidArrayAssignment
PossiblyInvalidArrayOffset
PossiblyInvalidCast
PossiblyInvalidClone
PossiblyInvalidDocblockTag
PossiblyInvalidFunctionCall
PossiblyInvalidIterator
PossiblyInvalidMethodCall
PossiblyInvalidOperand
PossiblyInvalidPropertyAssignment
PossiblyInvalidPropertyAssignmentValue
PossiblyInvalidPropertyFetch
PossiblyNullArgument
PossiblyNullArrayAccess
PossiblyNullArrayAssignment
PossiblyNullArrayOffset
PossiblyNullFunctionCall
PossiblyNullIterator
PossiblyNullOperand
PossiblyNullPropertyAssignment
PossiblyNullPropertyAssignmentVa
PossiblyNullPropertyFetch
PossiblyNullReference
PossiblyUndefinedArrayOffset
PossiblyUndefinedGlobalVariable
PossiblyUndefinedIntArrayOffset
PossiblyUndefinedMethod
PossiblyUndefinedStringArrayOffset
PossiblyUndefinedVariable
PossiblyUnusedMethod
PossiblyUnusedParam
PossiblyUnusedProperty
PossiblyUnusedReturnValue
PropertyIssue
PropertyNotSetInConstructor
PropertyTypeCoercion
PsalmInternalError
RawObjectIteration
RedundantCast
RedundantCastGivenDocblockType
RedundantCondition
RedundantConditionGivenDocblockType
RedundantIdentityWithTrue
RedundantPropertyInitializationCheck
ReferenceConstraintViolation
ReservedWord
StringIncrement
TaintedCallable
TaintedCookie
TaintedCustom
TaintedEval
TaintedFile
TaintedHeader
TaintedHtml
TaintedInclude
TaintedInput
TaintedLdap
TaintedSSRF

TaintedShell
TaintedSql
TaintedSystemSecret
TaintedTextWithQuotes
TaintedUnserialize
TaintedUserSecret
TooFewArguments
TooManyArguments
TooManyTemplateParams
Trace
TraitMethodSignatureMismatch

UndefinedMagicPropertyAssignment
UndefinedMagicPropertyFetch
UndefinedMethod
UndefinedPropertyAssignment
UndefinedPropertyFetch
UndefinedThisPropertyAssignment
UndefinedThisPropertyFetch
UndefinedTrace
UndefinedTrait
UndefinedVariable
UnevaluatedCode
UnhandledMatchCondition
UnimplementedAbstractMethod
UnimplementedInterfaceMethod
UninitializedProperty
UnnecessaryVarAnnotation
UnrecognizedExpression
UnrecognizedStatement
UnresolvableInclude
UnsafeGenericInstantiation
UnsafeInstantiation
UnusedClass
UnusedClosureParam
UnusedConstructor
UnusedForeachValue
UnusedFunctionCall
UnusedMethod
UnusedMethodCall
UnusedParam
UnusedProperty
UnusedPsalmSuppress
UnusedReturnValue
UnusedVariable
VariableIssue

Psalm can find over 240 different types of issue

@daveliddament

Amazing, let's add you to our project.

How strict you want me to be?

	Least strict	Strictest
Psalm	8	1
PHPStan	0	9

Should we use PHPStan or Psalm?
Or both?

Just pick one, it doesn't really matter.
Adding 1 to will push up your code quality.

```
# Psalm - install
composer require --dev vimeo/psalm
# create config file
vendor/bin/psalm --init src 1
# run
vendor/bin/psalm
```

```
# PHPStan - install
composer require --dev phpstan/phpstan
# run
vendor/bin/phpstan analyse --level=8 src tests
```

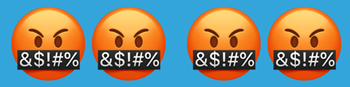
Storming

3246 errors
that's ridiculous!

Why are you reporting that as an error?

That's clearly not a bug!

stick with it.
It will be worth it. trust me!



OK smarty pants!
If you can find thousands of issues, how is it that my code is working?

Let's look at some of this issues I've found to see if your code is always working.

#1: real bugs

```
function register(User $user) {
  // some implementation
}
$user = 1;
register($user);
```

This code will fail.
Maybe this code is only run infrequently.
When did you last check your logs?

Ah. Yes. It's been a while.

```
interface UserRepository {
  public function findUser(string $name): ?User {…}
}
interface Emailer {
  public function message(User $user, string $msg): void {…}
}

$user = $userRepository->findUser("bob");
$emailer->message($user, "Hi Bob");
```

The code works most of the time, but it fails in some scenarios.
E.g. this code will fail if a user is not found.

#2: deferred bugs

We've checked. We only have adult and child pricing.

Will that ever change?
If it does change will you remember to update this code?

```
function cost(string $type): int
{
    if ($type === "CHILD") {
        $price = 10;
    }
    if ($type === "ADULT") {
        $price = 20;
    }
    return $price;
}
```

#3: evolvability defects

Code that makes code base less compliant with standards, more error prone, or more difficult to modify, extend or understand.

```
function getPerson() {…}

function register($user) {…}

$person = getPerson();
register($person);
```

Are you definitely passing the right type to register?

We must do. The code works.

But we can't be sure.
It's quite risky code to update.

@daveliddament

#1: real bugs

#2: deferred bugs

#3: evolvability defects

I agree that we must fix real bugs.
Do deferred bugs and evolvability defects really matter?

Yes. Both of these cost money…
They slow down feature development.
They increase bug fix time.

Norming

This starting to make sense.

The more I understand about your codebase the better my analysis will be.

How do we understand each other better?

I learn about your codebase from:
The code itself
Docblocks
Plugins or Extensions
Stub files

I understand everything the PHP interpreter understands.
Please add type declarations everywhere.

```
class Person {
  private int $age;

  public function __construct(string $name, int $age) {…}

  public function updateName(string $name): void {…}

  public function getAge(): int {…}

}
```

I also understand docblocks.
Add type hints where more information is required than can be given in a type declaration.

```
/**
 * @return User[]
 */
function getUsers(): array {…}
```

A very brief intro to generics

```
/**
 * @return User[]
 */
function getUsers(): array {…}

function promote(User $user): void {…}

foreach(getUsers() as $user) {
 promote($user);
}
```

```
/**
 * @return User[]
 */
function getUsers(): array {…}

function welcome(string $name): void {…}
function promote(User $user): void {…}

foreach(getUsers() as $name => $user) {
 welcome($name);
 promote($user);
}
```

```
/**
 * @return array<string,User>
 */
function getUsers(): array {…}

function welcome(string $name): void {…}
function promote(User $user): void {…}

foreach(getUsers() as $name => $user) {
  welcome($name);
  promote($user);
}
```

```
/**
 * @return array<int,string>
 */
function getNames(): array {
  return [
    "Anna" => new Person(),
  ];
}
```

You're returning the wrong type of data

```php
/**
 * @template T
 */
class Queue {

  /** @param T $item */
  public function add($item): void

  /** @return T */
  public function getNext()
}
```

```php
/** @var Queue<Book> */
$books = new Queue();
$books->add(new Book());
```

```php
/** @return Queue<Person> */
function getUserQueue(): Queue {}

$queue = getUserQueue();
$person = $queue->getNext();
```

Psalm website

https://psalm.dev/docs/annotating_code/templated_annotations/

https://psalm.dev/articles/uncovering-php-bugs-with-template

PHPStan website

https://phpstan.org/blog/generics-in-php-using-phpdocs

PHP-UK 2020: PHP Generics Today (Almost)

https://www.youtube.com/watch?v=N2PENQpQVjQ

In summary…

Add type declarations everywhere.

Where additional type information is needed add type hints.

Plugins and extensions have been created to help me better understand libraries and frameworks.

```
ERROR: PropertyNotSetInConstructor

- tests/Unit/Entity/ContactUsTest.php:11:7

Property
Tests\Unit\Entity\ContactUsTest::$runTestInSeparateProcess
is not defined in constructor
```

$runTestInSeparateProcess is related to PHPUnit. I'm guessing we need to use a plugin?

https://psalm.dev/plugins

Psalm plugins

These plugins allow Psalm to work great with popular packages like Laravel, PHPUnit, and Symfony.

Have a look at Psalm's documentation to find out how to install and use them.

psalm/plugin-phpunit
347,191 52
Psalm plugin for PHPUnit

psalm/plugin-symfony
241,958 163
Psalm Plugin for Symfony

weirdan/doctrine-psalm-plugin
121,589 68
Stubs to let Psalm understand Doctrine better

psalm/plugin-laravel
68,137 203
A Laravel plugin for Psalm

psalm/plugin-mockery
39,873 5
Psalm plugin for Mockery

php-standard-library/psalm-plugin
17,808 6
Psalm plugin for the PHP Standard Library

orklah/psalm-insane-comparison
13,485 28

https://packagist.org/?type=psalm-plugin

Packagist *The PHP Package Repository* Browse Submit Create account Sign in

Search packages...

Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.

psalm/plugin-symfony Gherkin 2 482 124 163
Psalm Plugin for Symfony

weirdan/doctrine-psalm-plugin Gherkin 1 289 008 70
Stubs to let Psalm understand Doctrine better

Active filters
type: psalm-plugin Clear all

Package type

1	85
application	309
asgard-module	292
asgard-theme	67

```
composer require --dev psalm/plugin-symfony
vendor/bin/psalm-plugin enable psalm/plugin-symfony
```

https://phpstan.org/user-guide/extension-library https://packagist.org/?type=phpstan-extension

Official extensions

Check out phpstan-strict-rules repository for extra strict and opinionated rules for PHPStan.

Check out as well phpstan-deprecation-rules for rules that detect usage of deprecated classes, methods, properties, constants and traits!

Framework-specific extensions

- Doctrine
- PHPUnit
- Symfony Framework
- beberlei/assert
- webmozart/assert
- Mockery
- Nette Framework
- PHP-Parser
- Dibi - Database Abstraction Library

Unofficial extensions

- Laravel
- Drupal
- WordPress
- Laminas (a.k.a. Zend Framework)
- Phony
- Prophecy
- marc-mabe/php-enum
- myclabs/php-enum
- Yii2
- PhpSpec
- TYPO3
- moneyphp/money
- Nextras ORM
- Sonata
- Magento

3rd party rules

- thecodingmachine / phpstan-strict-rules
- spaze / phpstan-disallowed-calls
- ergebnis / phpstan-rules
- Slamdunk / phpstan-extensions
- ekino / phpstan-banned-code
- taptima / phpstan-custom

Find more on Packagist!

Edit this page on GitHub

Packagist *The PHP Package Repository* Browse Submit Create account Sign in

Search packages...

Packagist is the main Composer repository. It aggregates public PHP packages installable with Composer.

phpstan/phpstan-symfony PHP 12 490 248 469
Symfony Framework extensions and rules for PHPStan

nunomaduro/larastan PHP 7 640 536 3 690
Larastan - Discover bugs in your code without running it. A phpstan/phpstan wrapper for Laravel

Active filters
type: phpstan-extension Clear all

Package type
1 85
application 309
asgard-module 292
asgard-theme 67

```
composer require --dev phpstan/extension-installer
```

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

@daveliddament

We've looked at the issues.

Added relevant plugins/extensions.

There are still 1000s of issues with no time to fix.
Can you help?

You need the baseline feature.

Don't forget to check the baseline in to your project.

https://phpstan.org/user-guide/baseline

https://psalm.dev/docs/running_psalm/dealing_with_code_issues/#using-a-baseline-file

A few quick questions…

Do we run static analysis on our test code?

Yes!

There is this one rule that I really disagree with. What do I do?

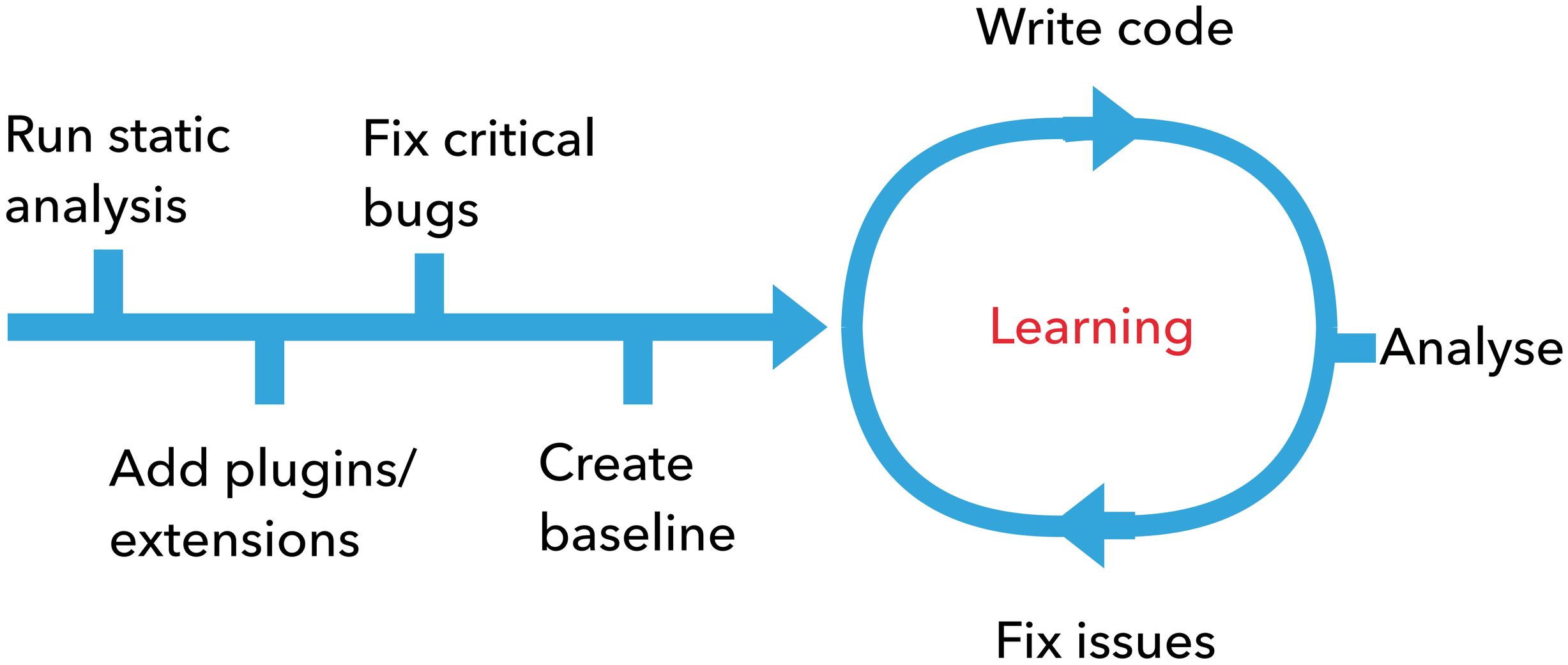
You can update config file to exclude reporting particular rules.

You can also ignore a single problem using a docblock. E.g.:

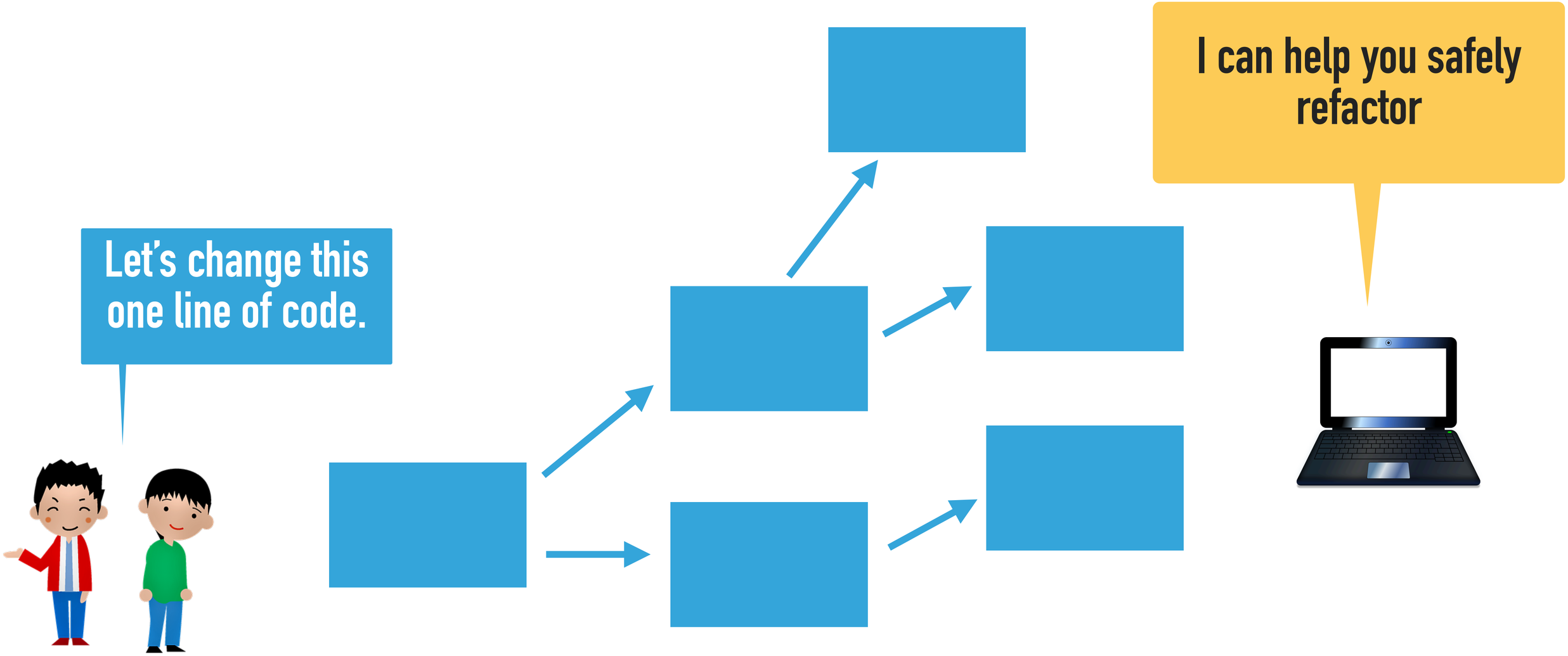
/** @phpstan-ignore-next-line */

I'd love to work with a team that uses PHPStan, do you know any?

Ask Dave, he knows someone.



@daveliddament



@daveliddament

Performing

You're one of the best things that has ever happened to us.

How can we thank you?

Sponsor on Github?

Work with me a bit more and we can be even better together.

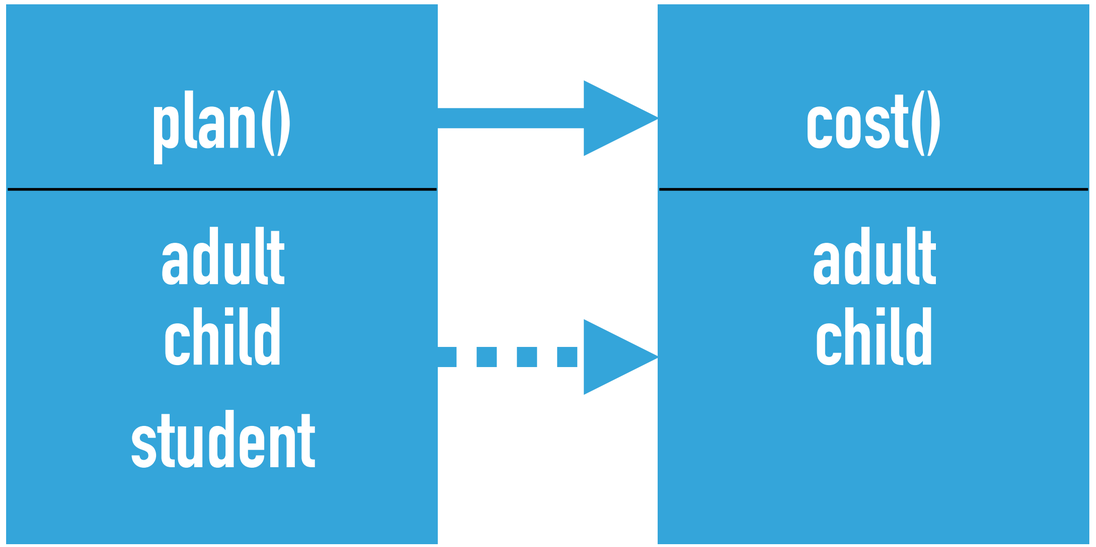
```
function cost(string $type): int
{
   if ($type === “CHILD") {
       $price = 10;
   }
   if ($type === “ADULT”) {
       $price = 20;
   }
   return $price;
}
```

```
function cost(string $type): int
{
    if ($type === "CHILD") {
        $price = 10;
    }
    if ($type === "ADULT") {
        $price = 20;
    }
    return $price;
}
```

This is a slightly better error message.

```
function cost(string $type): int
{
  switch ($type) {
    case 'CHILD':
      return 10;

    case 'ADULT':
      return 20;
  }
  throw new LogicException(
    "Unknown type [$type]");
}
```

```
function cost(string $type): int
{
  switch ($type) {
    case 'CHILD':
      return 10;

    case 'ADULT':
      return 20;
  }
  throw new LogicException(
    "Unknown type [$type]");
}
```



Will our tests find the bug?

Tests can only tell you the code is correct, for the scenarios that have test cases.

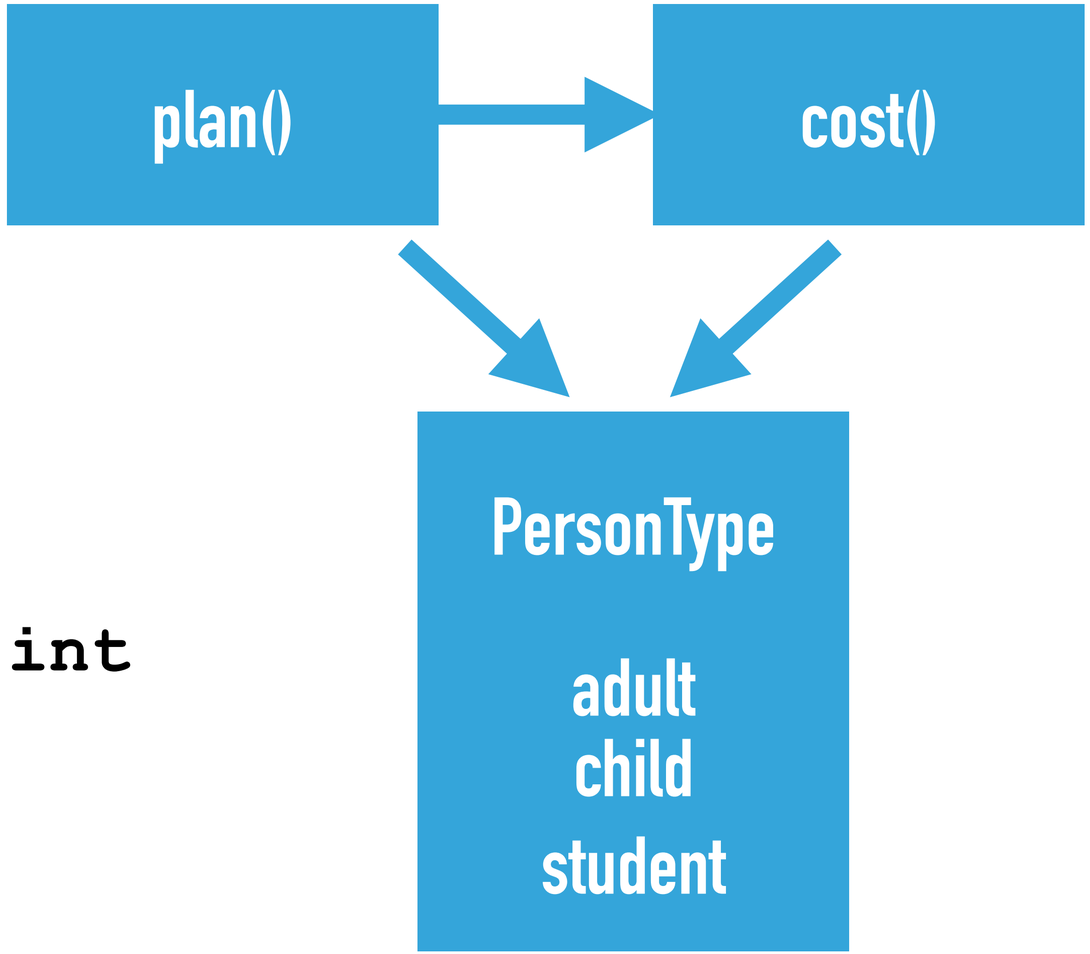
```
enum PersonType
{
  case ADULT;
  case CHILD;

}

function cost(PersonType $type): int
{
  return match($type) {
    PersonType::CHILD => 10,
    PersonType::ADULT => 20,
  };
}
```

Looks good to me

```
enum PersonType
{
  case ADULT;
  case CHILD;
  case STUDENT;
}

function cost(PersonType $type): int
{
  return match($type) {
    PersonType::CHILD => 10,
    PersonType::ADULT => 20,
  };
}
```

Match expression does not handle remaining value: PersonType::STUDENT

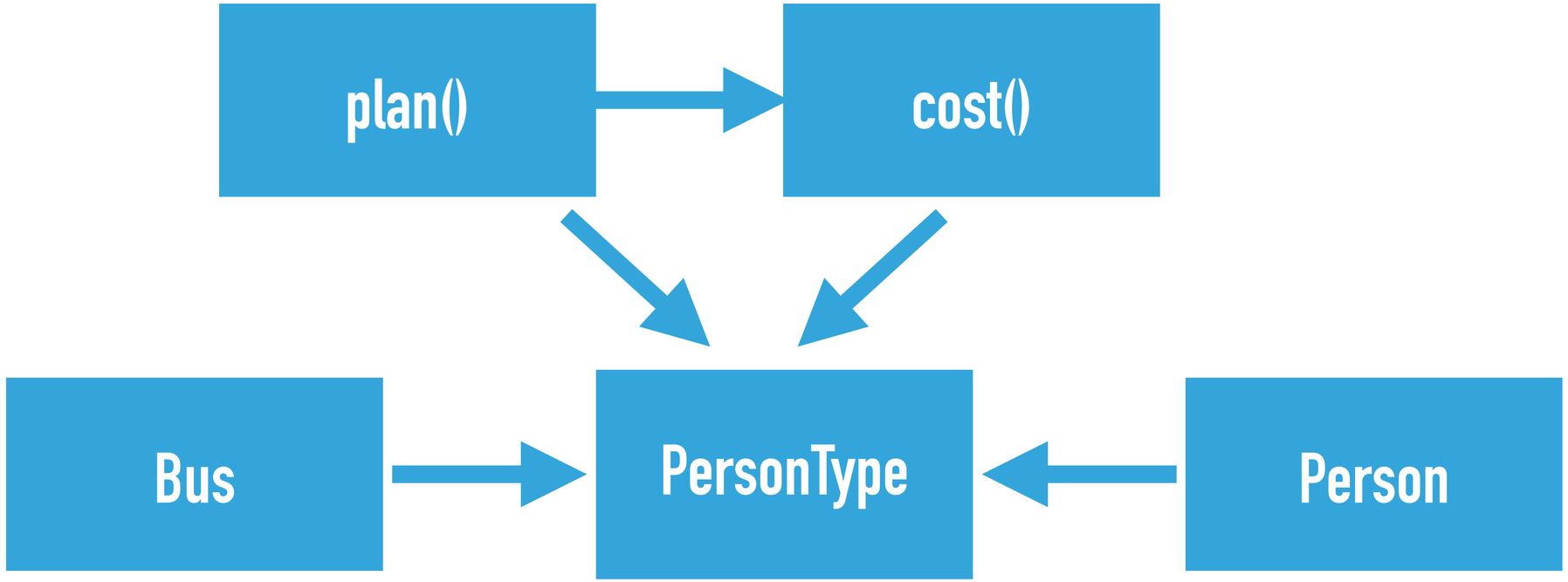
```
enum PersonType
{
  case ADULT;
  case CHILD;

}

function cost(PersonType $type): int
{
  return match($type) {
    PersonType::CHILD => 10,
    PersonType::ADULT => 20,
  };
}
```



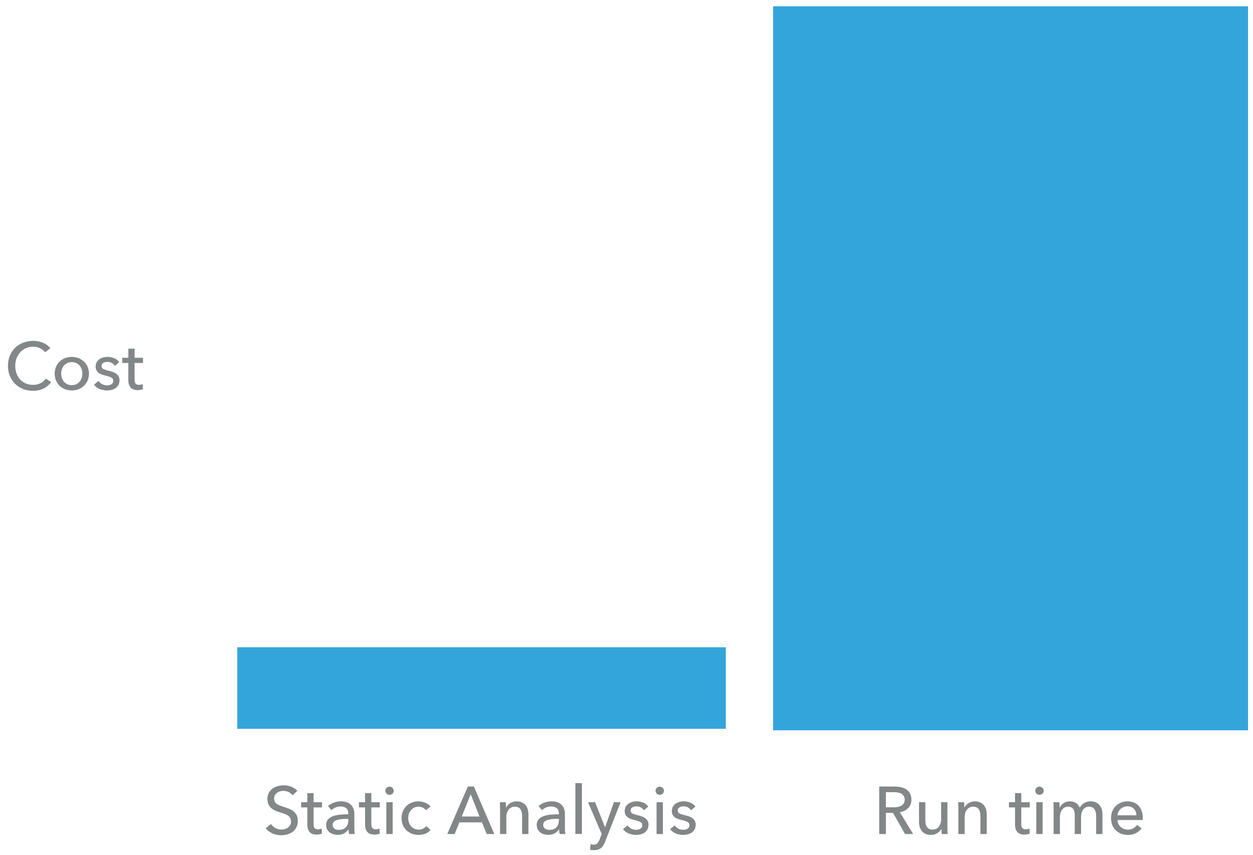
```
enum PersonType
{
  case ADULT;
  case CHILD;
  case STUDENT;
}
```

```
function cost(PersonType $type)
: int {
  return match($type) {
    PersonType::CHILD => 10,
    PersonType::ADULT => 20,
    PersonType::STUDENT => 15,
  };
}
```

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20
Test 3	STUDENT	15


plan()
cost()
Bus
PersonType
Person

COST OF A BUG



Write your code in such a way that static analysis can catch bugs.

That's amazing.
But it's 2022.
We won't be running PHP 8.1 for years!!

@daveliddament

```
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';


  private string $type;


  public function __construct(string $type) {
    $this->type = $type;
  }


  public function value(): string {
    return $this->type;
  }
}
```

```
function cost(PersonType $type): int
{
  switch ($type->value()) {

    case PersonType::CHILD:
      return 10;

    case PersonType::ADULT:
      return 20;
  }
}
```

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20

✅ All tests pass

InvalidReturnType: Not all code paths of cost end in a return statement, return type int expected.

```
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';


  private string $type;


  public function __construct(string $type) {
    $this->type = $type;
  }

  /** @return 'ADULT'|'CHILD' */
  public function value(): string {
    return $this->type;
  }
}
```

```
function cost(PersonType $type): int
{
  switch ($type->value()) {

    case PersonType::CHILD:
      return 10;

    case PersonType::ADULT:
      return 20;
  }
}
```

I know that value() returns 'CHILD' or 'ADULT' and can infer that cost() always returns an int.

I'm happy here. But...

```
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';


  private string $type;


  public function __construct(string $type) {
    $this->type = $type;
  }

  /** @return 'ADULT'|'CHILD' */
  public function value(): string {
    return $this->type;
  }
}
```

The property $type can be any string,
But value() should return only the strings ADULT of CHILD.

```
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';

  /** @var 'ADULT'|'CHILD' */
  private string $type;

  /** @param 'ADULT'|'CHILD' $type */
  public function __construct(string $type) {
    $this->type = $type;
  }

  /** @return 'ADULT'|'CHILD' */
  public function value(): string {
    return $this->type;
  }
}
```

Another benefit…
If you instantiated PersonType with an invalid type, I'd warn you.

```
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';

  /** @var PersonType::TYPE_* */
  private string $type;

  /** @param PersonType::TYPE_* $type */
  public function __construct(string $type) {
    $this->type = $type;
  }

  /** @return PersonType::TYPE_* */
  public function value(): string {
    return $this->type;
  }
}
```

With Psalm I've replicated the benefits of enum and match in PHP 7

Amazing. What other things can you do?

Functional programming:
@psalm-pure
@psalm-immutable

On the security front I can do taint analysis.

I can offer more fine grained visibility than public, protected and private:
@internal
@psalm-internal

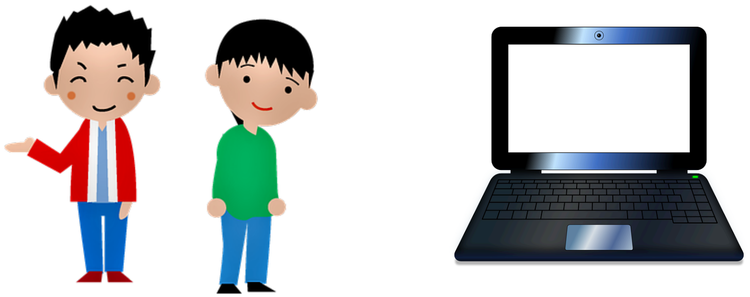
And you can write your own rules.

```
class Job {

  public int $id;

  public function makeLive(): void
  {
     if ($this->status !== 'DRAFT') {
        throw new LogicException(
          "Cant make job [{$this->id}] live");

     }
     … More code …
  }

  public function setId(int $id): void
  {
     $this->id = $id;
  }

  … More code …
}
```

I need the property id set for a unit test.

I've added a setId method. This method should ONLY be called by test code.

You can write a custom rule.



@daveliddament

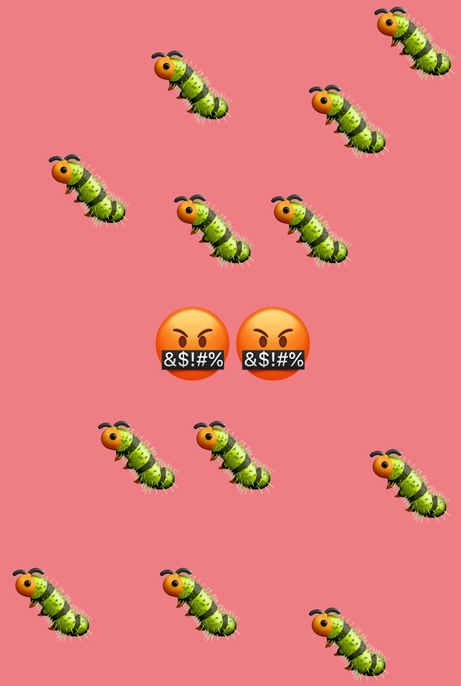
```php
class OnlyCallSetIdMethodOnEntitiesFromTestRule implements Rule
{
    public function getNodeType(): string
    {
        return MethodCall::class;
    }

    public function processNode(Node $node, Scope $scope): array
    {
        if (!$node->name instanceof Node\Identifier) {
            return [];
        }
        if ($node->name->name !== 'setId') {
            return [];
        }
        $type = $scope->getType($node->var);
        $couldBeEntity = false;
        foreach ($type->getReferencedClasses() as $class) {
            if (str_starts_with(haystack: $class, needle: 'App\Entity')) {
                $couldBeEntity = true;
            }
        }
        if (!$couldBeEntity) {
            return [];  // Ignore if target class is not an Entity
        }
        if (str_starts_with(haystack: $scope->getNamespace(), needle: 'App\Test')) {
            return []; // Ignore if in Test namespace
        }
        return [RuleErrorBuilder::message("Calling setId on an Entity is only allowed in Test namespace.")->build()];
    }
}
```

Writing custom rules isn't that scary.

@daveliddament

Forming

Storming



Norming

```
/**
 * @return array<int,Person>
 */
function get(string $name): array
```

Baseline

Performing

@

Dave Liddament

Lamp Bristol

Thank you for listening

Organise PHP-SW

Author of Static Analysis Results Baseliner (SARB)

20 years of writing software (C, Java, Python, PHP)

@daveliddament