Talk

PHP UK
CONFERENCE

Practical Advanced Static Analysis
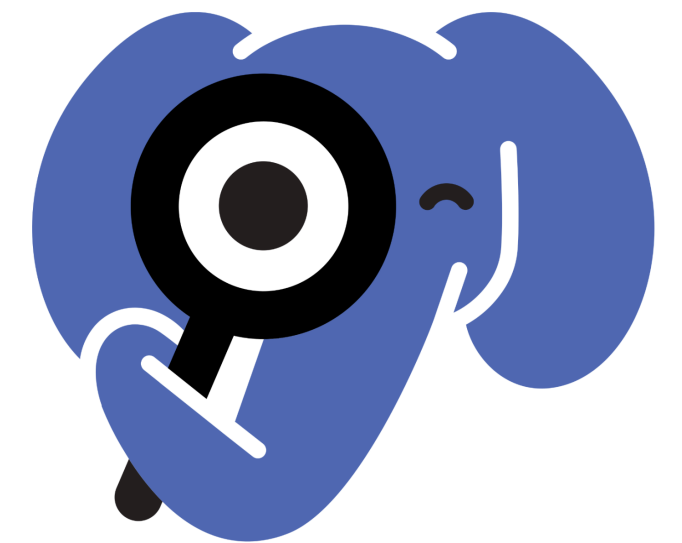
Dave Liddament
@DaveLiddament

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
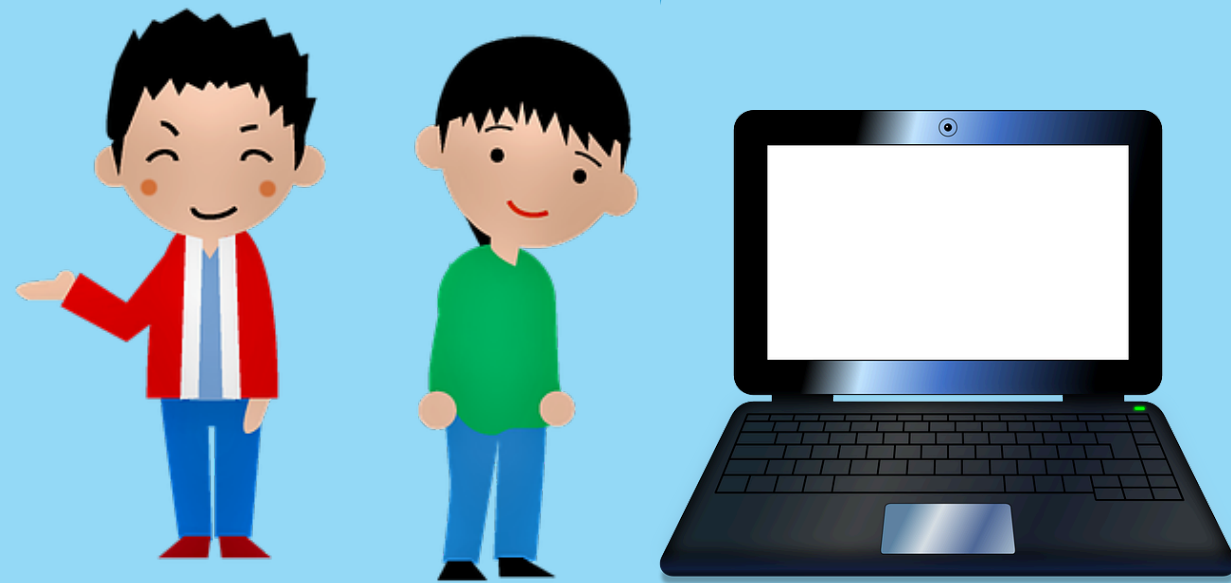
4 Extend PHP to provide new features
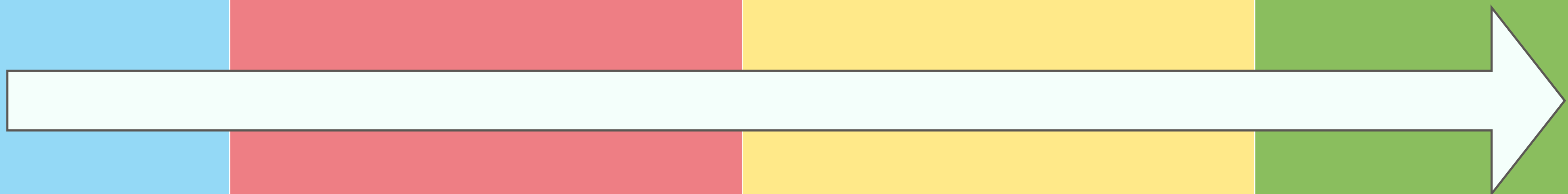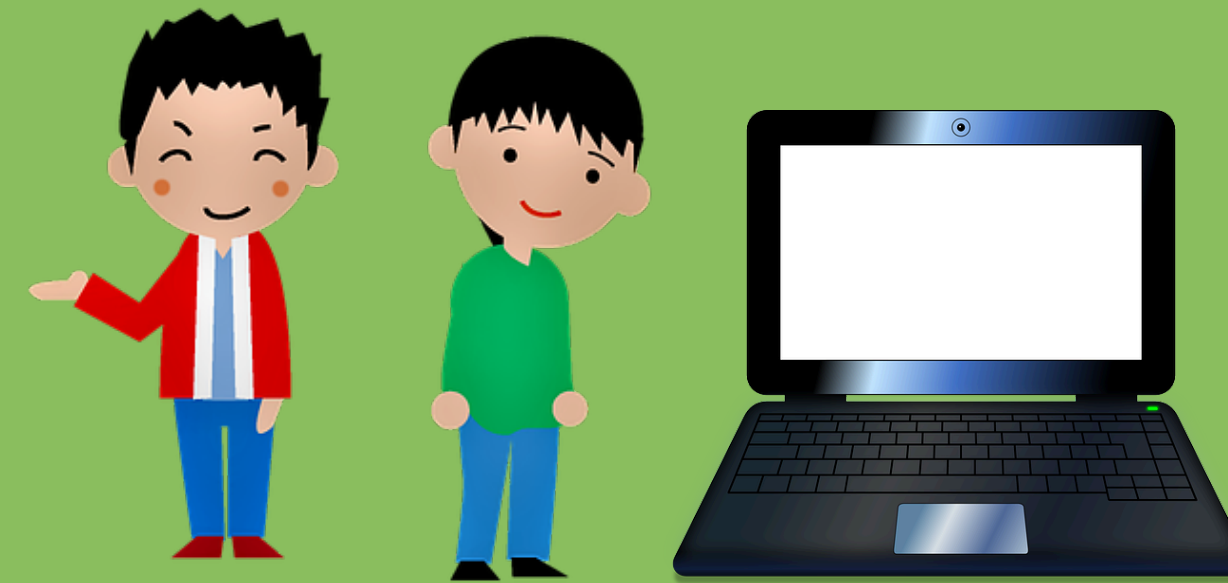
Psalm
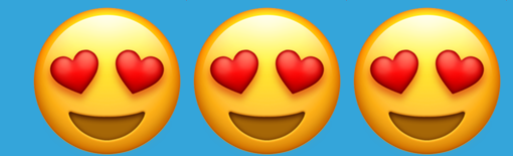
TEAM PERFORMANCE

Forming

Storming

Norming

Performing

```
function cost(string $type): int
{

    if ($type === "CHILD") {

        $price = 10;

    }

    if ($type === "ADULT") {

        $price = 20;

    }

    return $price;

}
```

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20

✅ All tests pass

💯 Code coverage

Price might not be set

@daveliddament

Static analysis shows you where your code is incorrect.

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

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

```
# Psalm - install

composer require --dev vimeo/psalm

# create config file

vendor/bin/psalm --init src 1

# run

vendor/bin/psalm
```

We write clean code.
I don't think you will find many bugs!

```
# PHPStan - install

composer require --dev phpstan/phpstan

# run

vendor/bin/phpstan analyse --level=8 src tests
```
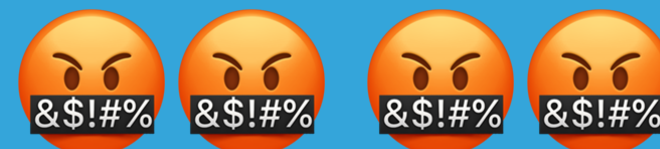
```
interface UserRepository {

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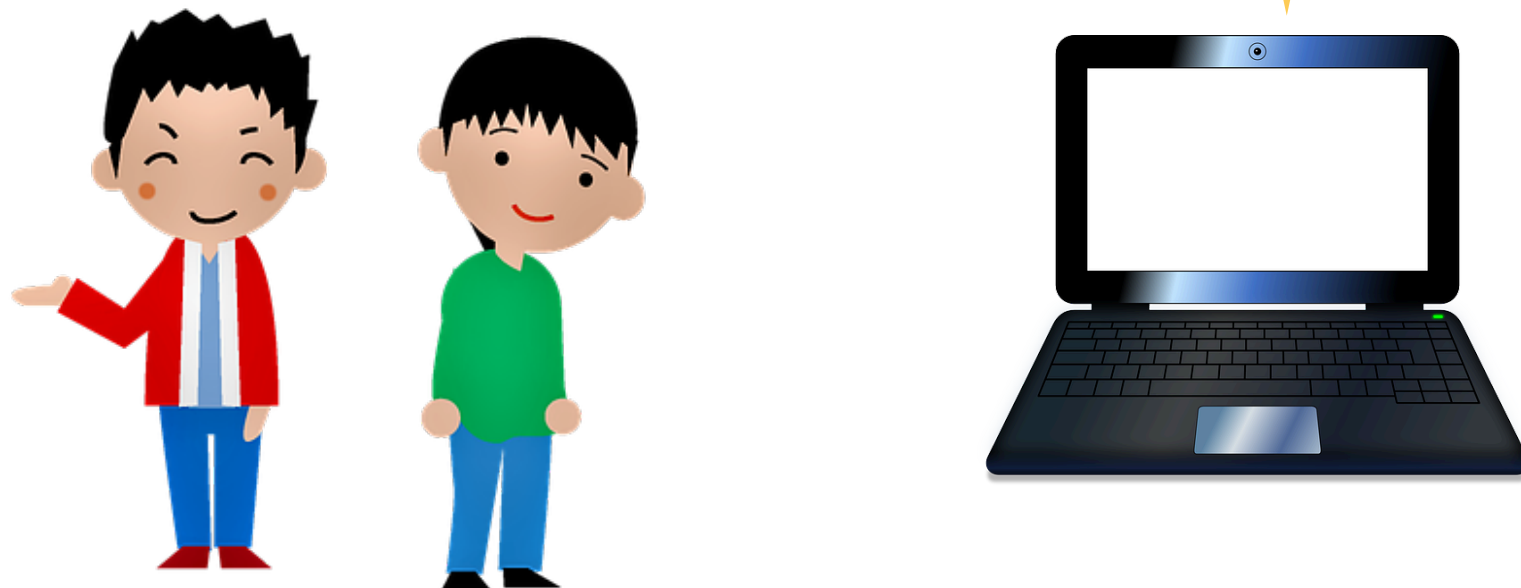
The code works most of the time, but it fails in some scenarios.
E.g. this code will fail if a user is not found.

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

A very brief intro to generics

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


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


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

@daveliddament

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


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


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

@daveliddament

```php
/**
 * @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);

}
```

@daveliddament

```
/**
 * @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();
```

@daveliddament

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.

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

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
Psalm Plugin for Symfony   ★ 163

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

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
```

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

Symfony Framework extensions and rules for PHPStan   ★ 469

nunomaduro/larastan   PHP   ⊕ 7 640 536

Larastan - Discover bugs in your code without running it. A phpstan/phpstan wrapper for Laravel   ★ 3 690

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
```

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

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

@daveliddament

A few quick questions…

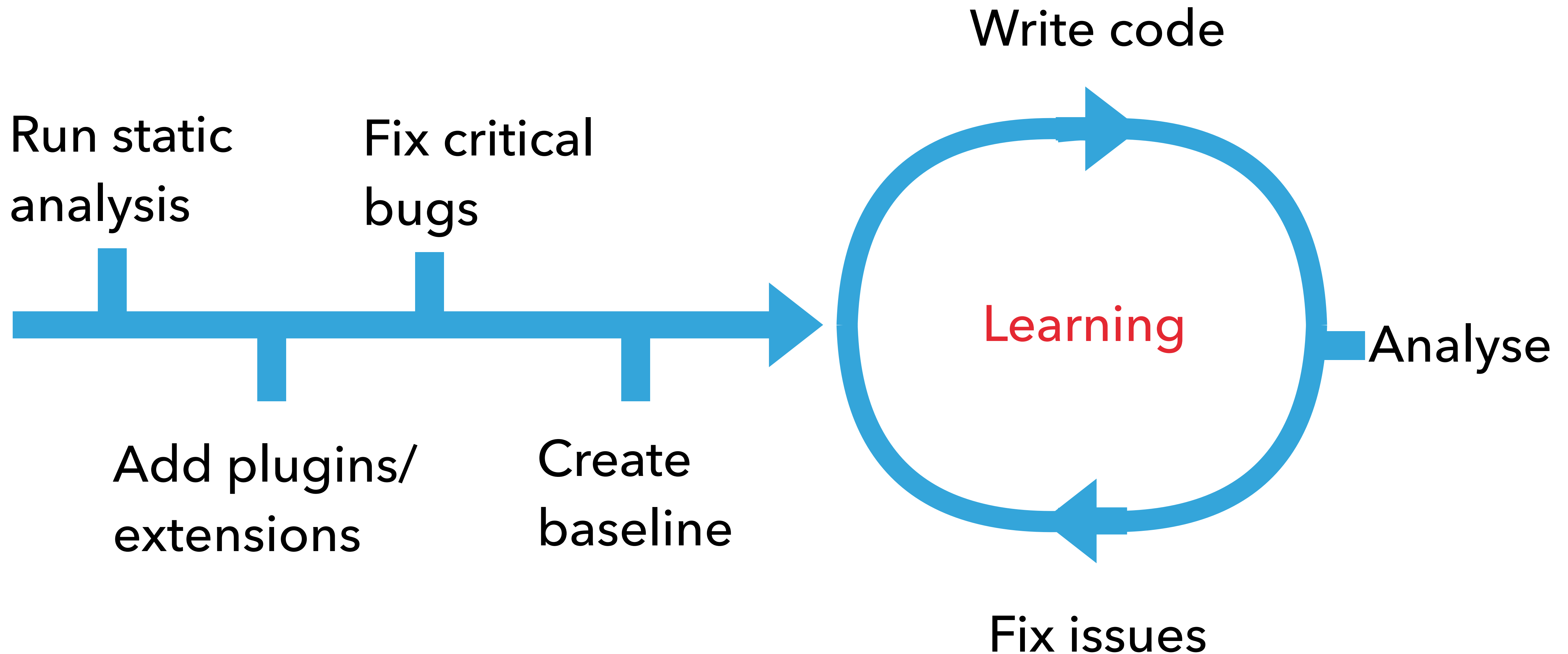Do we run static analysis on our test code?

Yes!

@daveliddament

@daveliddament

```
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;

}
```

```
function cost(string $type): int

{

    switch ($type) {

        case 'CHILD':

            return 10;


        case 'ADULT':

            return 20;

    }

    throw new LogicException(

        "Unknown type [$type]");

}
```

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]");

}
```
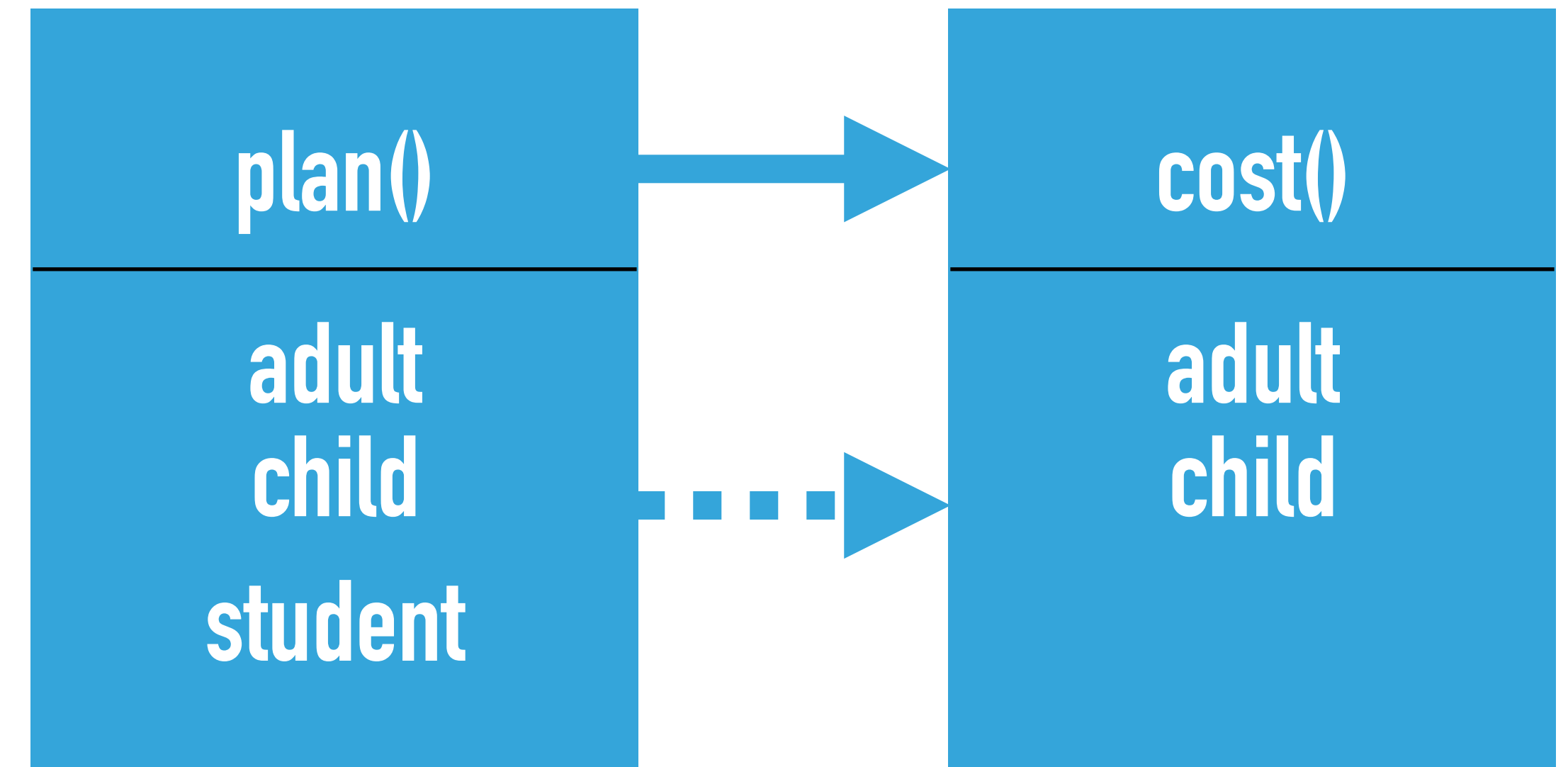

Will our tests find the bug?

Tests can only help you for the scenarios that are tested.

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


}

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

Looks good to me

@daveliddament

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


function cost(PersonType $type): int
{
    return match($type) {
        PersonType::CHILD => 10,
        PersonType::ADULT => 20,
    };
}
```
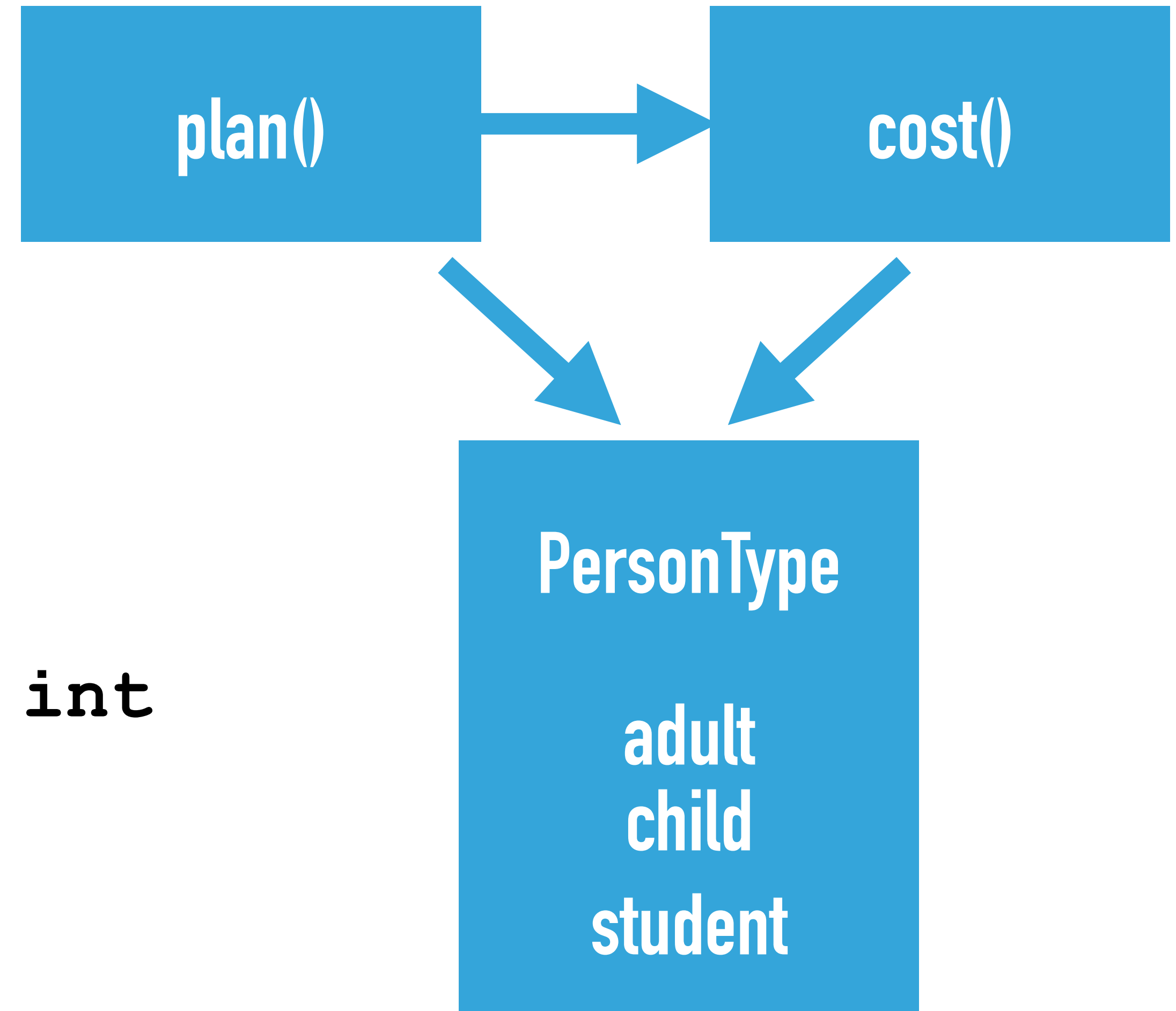
Match expression does not handle remaining value:
PersonType::STUDENT

@daveliddament

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


}

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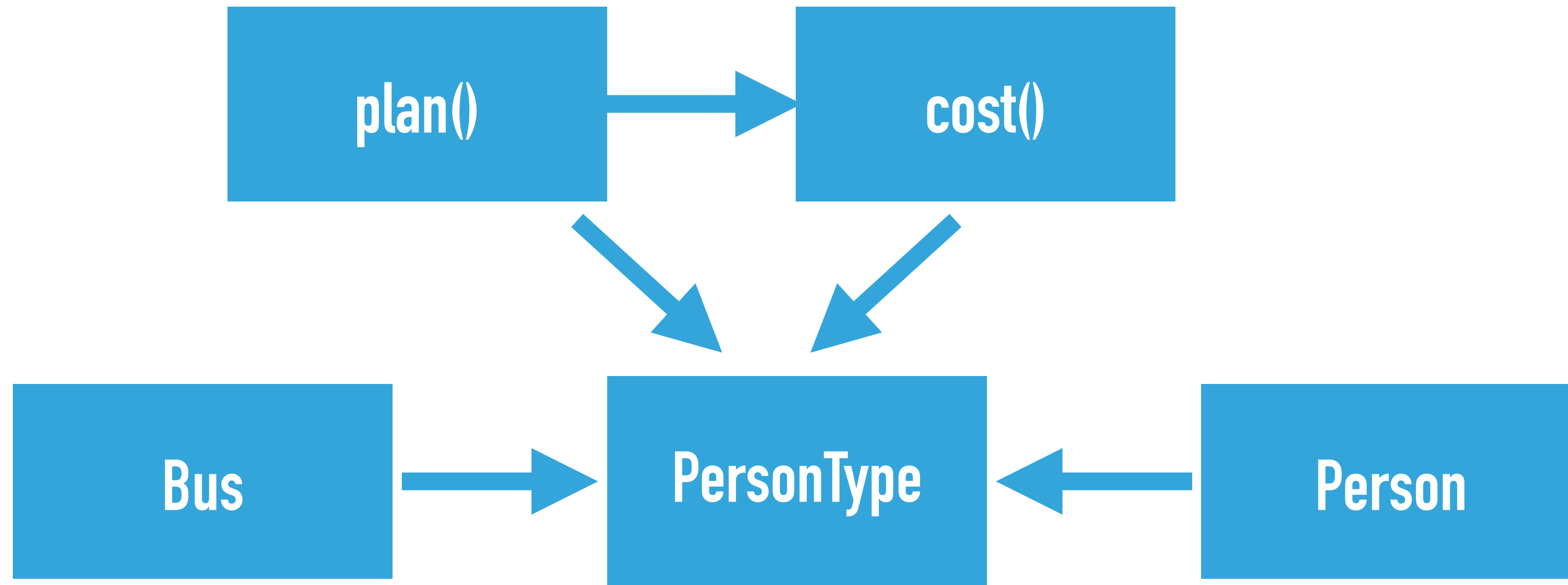

@daveliddament

```
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,
  };
}
```
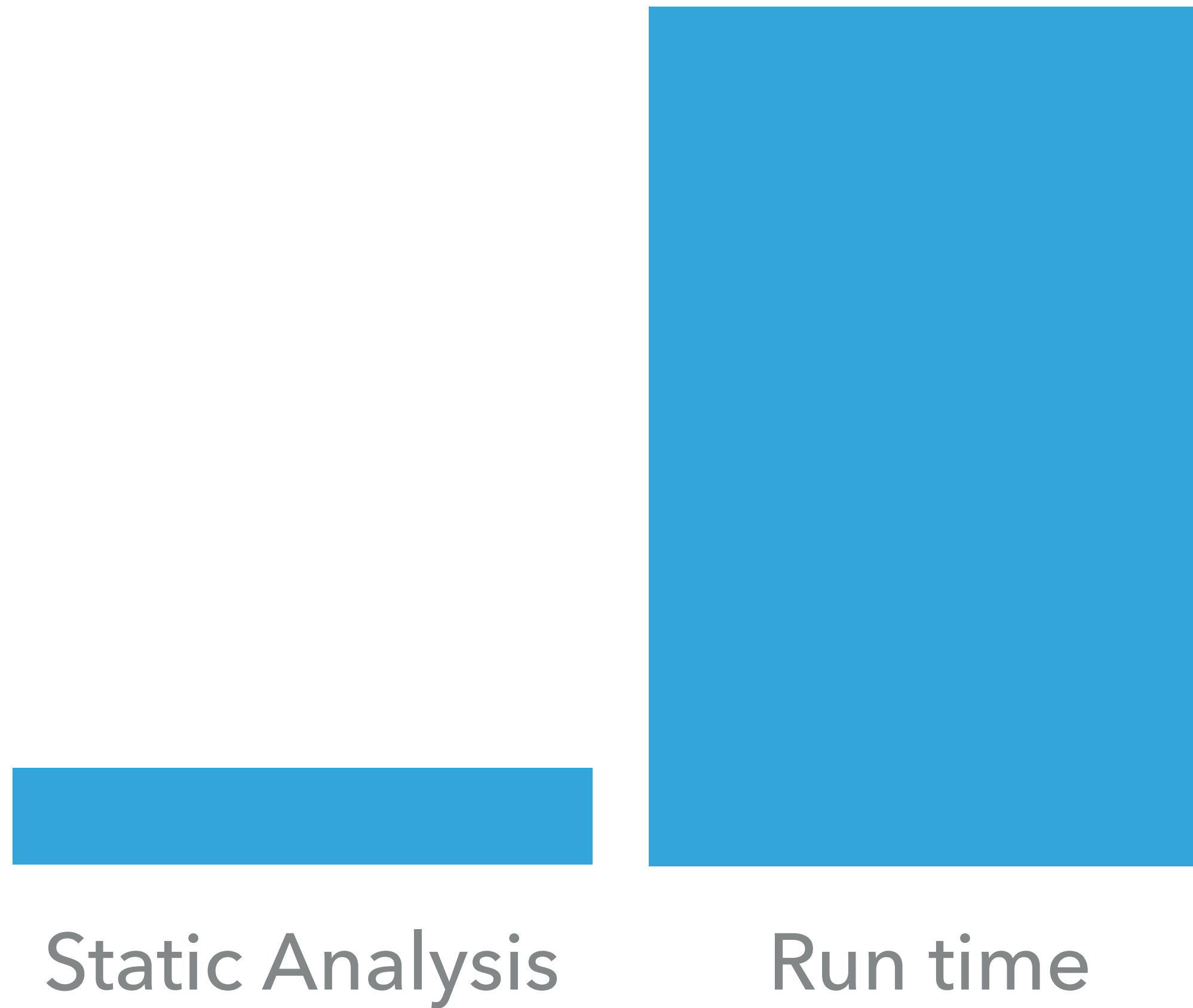
There is a problem with cost()

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20
Test 3	STUDENT	15
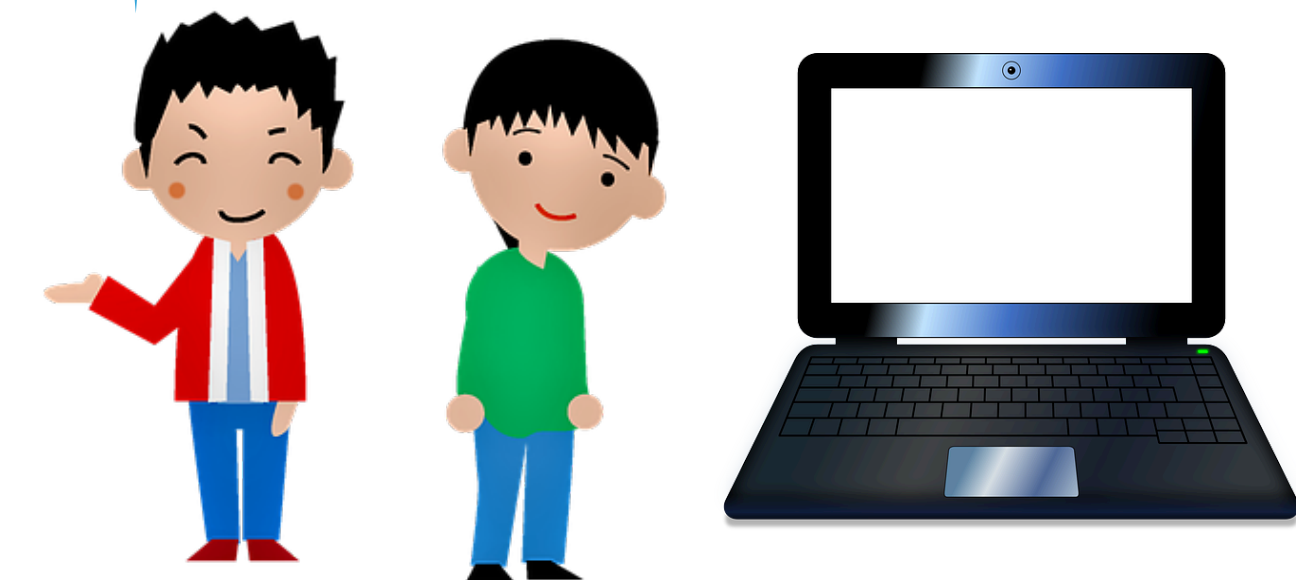
```php
class PersonType {
  public const TYPE_ADULT = 'ADULT';
  public const TYPE_CHILD = 'CHILD';


  private string $type;


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


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

```php
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.

@daveliddament

```php
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;
  }
}
```

```php
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.

@daveliddament

```php
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.

```php
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

@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

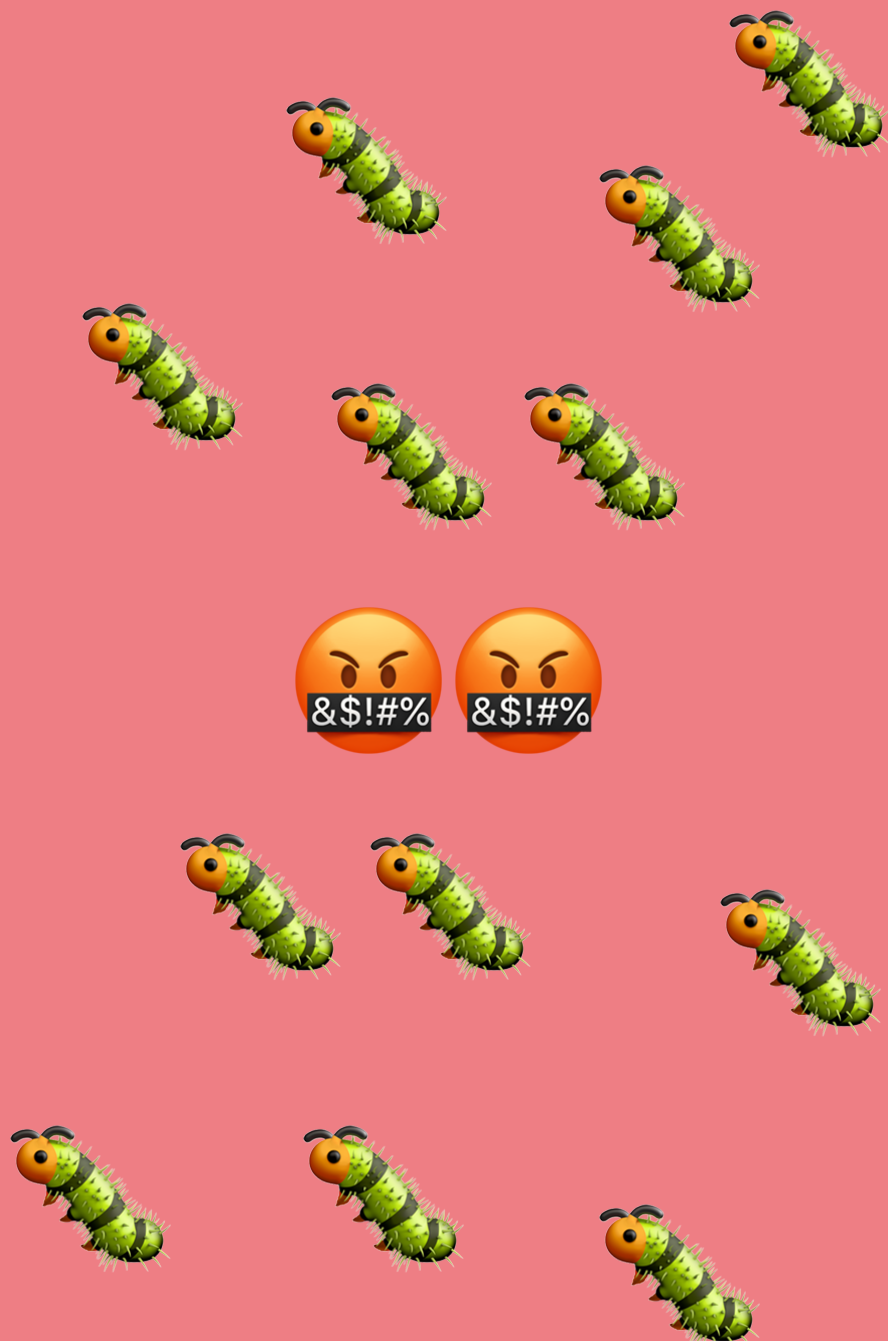# Norming

Performing

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

Baseline

@daveliddament

@daveliddament