

# Talk

**PHP UK**  
CONFERENCE



## Practical Advanced Static Analysis

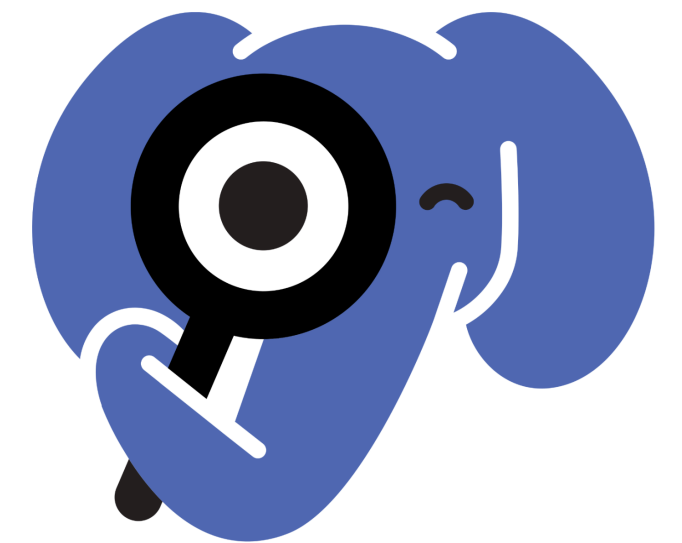
**Dave Liddament**

@DaveLiddament

16th, 17th & 18th February 2022  
[www.phpconference.co.uk](http://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

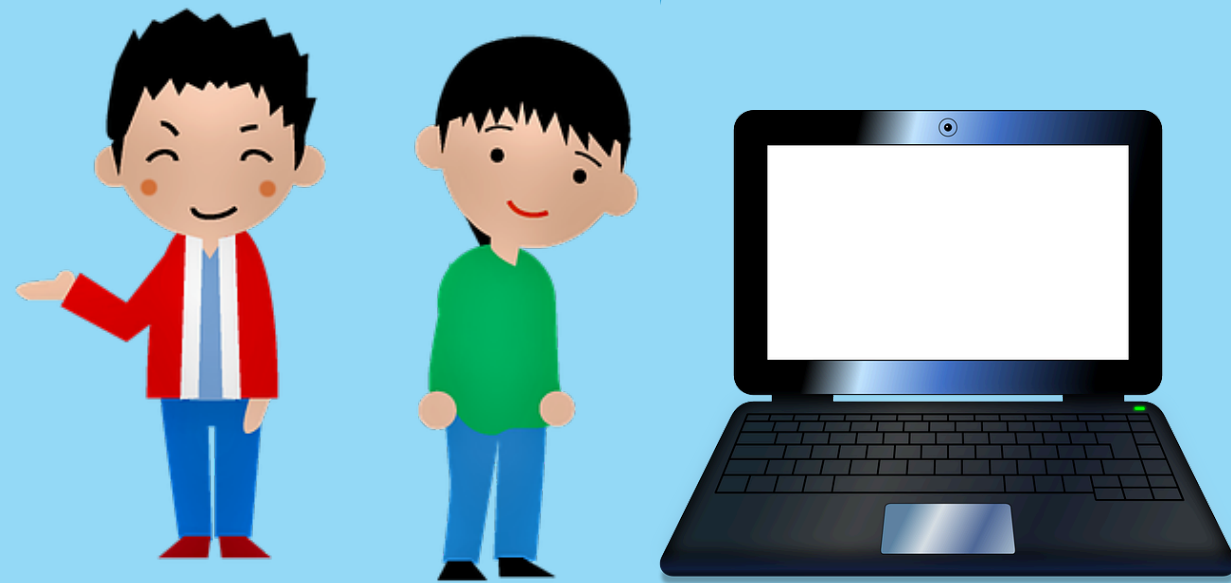


**Psalm**



# Forming

Welcome



# Storming

What the



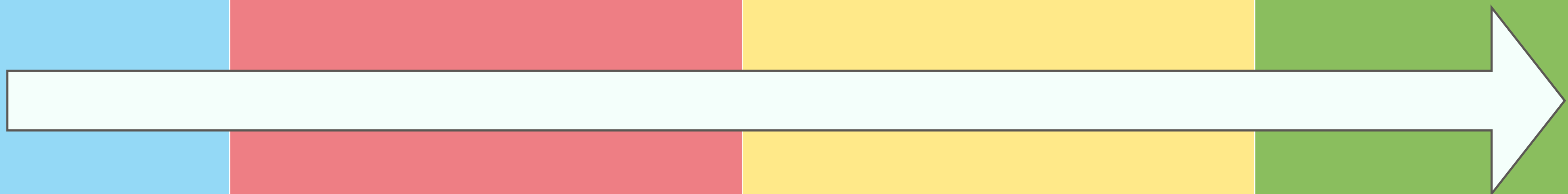
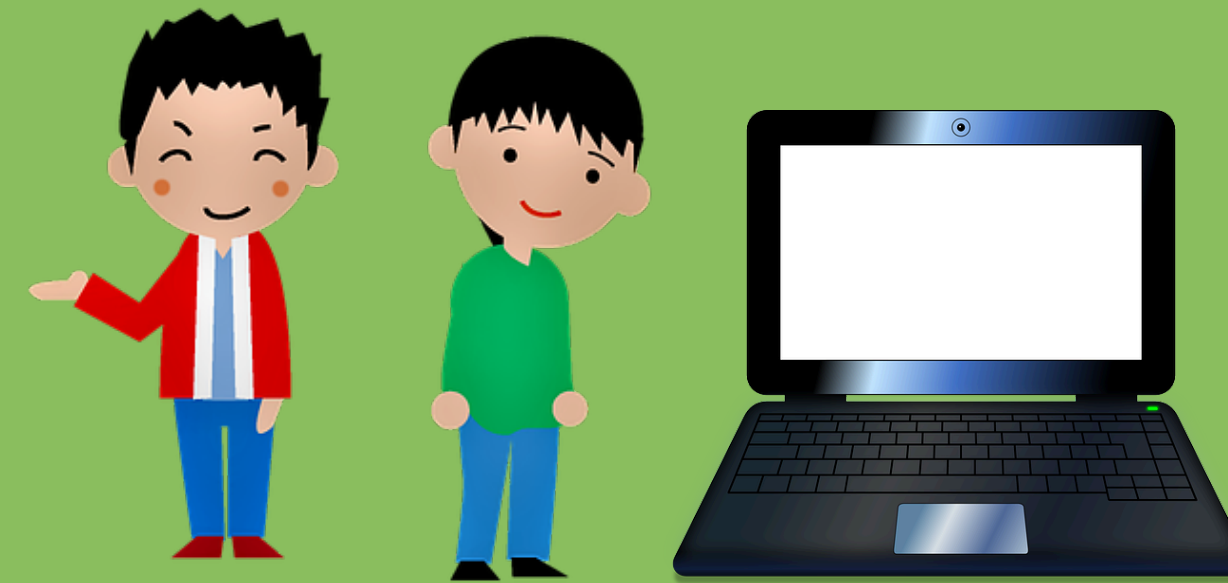
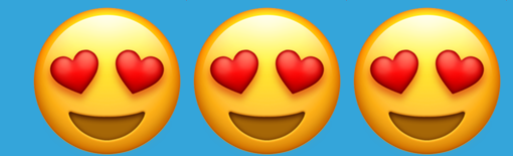
# Norming

I'm starting to  
understand you

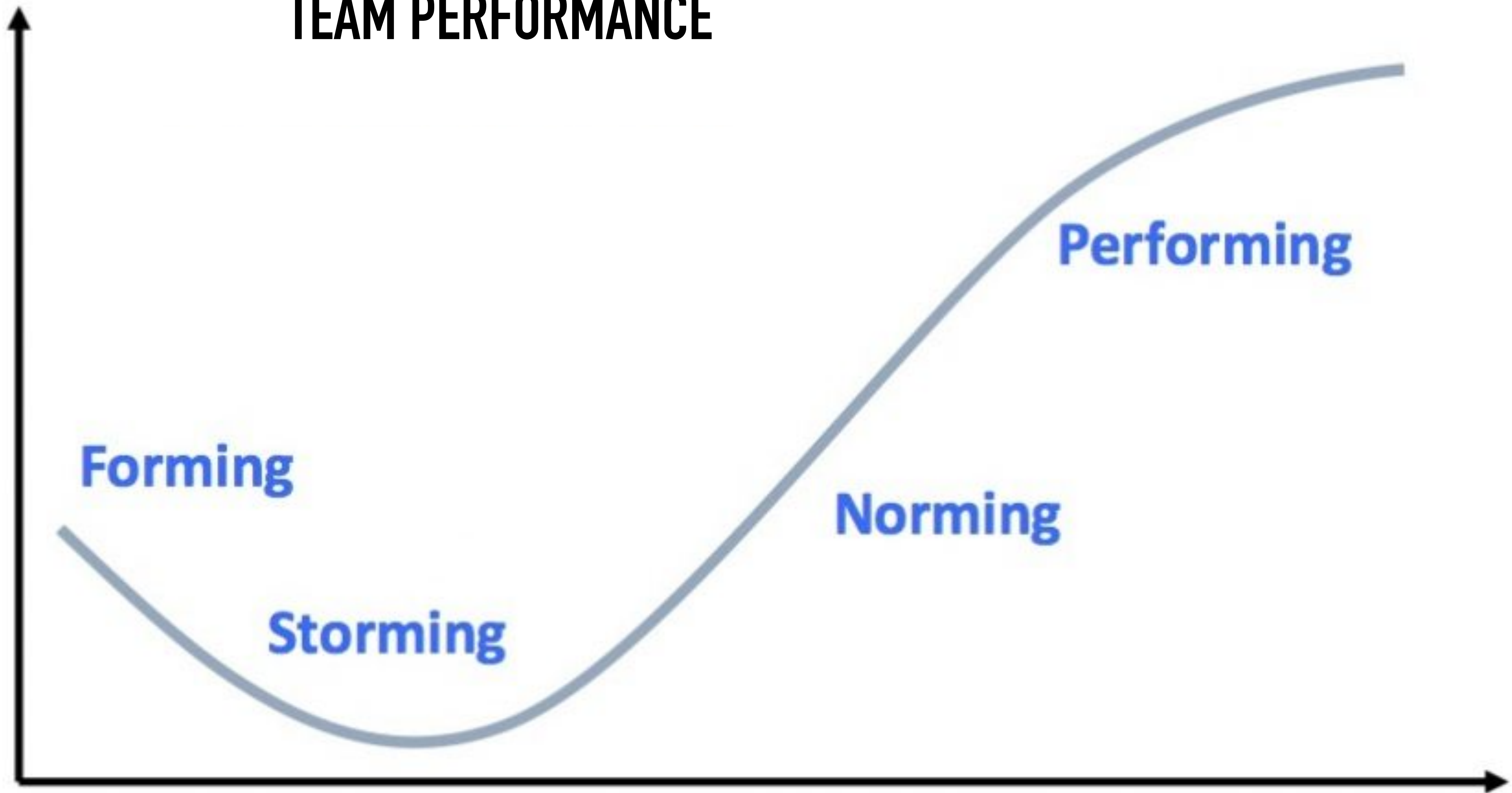


# Performing

You're amazing



# TEAM PERFORMANCE



# Forming



What do you do?

I look at your code and  
find bugs.



```
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;
}
```

return \$price;

Price might not be set

	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20

✓ All tests pass

100 Code coverage



**Static analysis shows you where  
your code is incorrect.**

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



```
1 <?php
2
3 function foo(string $s) : void {
4     return "bar";
5 }
6
7 $a = ["hello", 5];
8 foo($a[1]);
9 foo();
10
11 if (rand(0, 1)) $b = 5;
12 echo $b;
13
14 $c = rand(0, 5);
15 if ($c) {} elseif ($c) {}
16
```

Sounds amazing, can we try this out?

Head over to [psalm.dev](https://psalm.dev) or [phpstan.org](https://phpstan.org) to try out the virtual playground.

Psalm output (using commit add7c14):

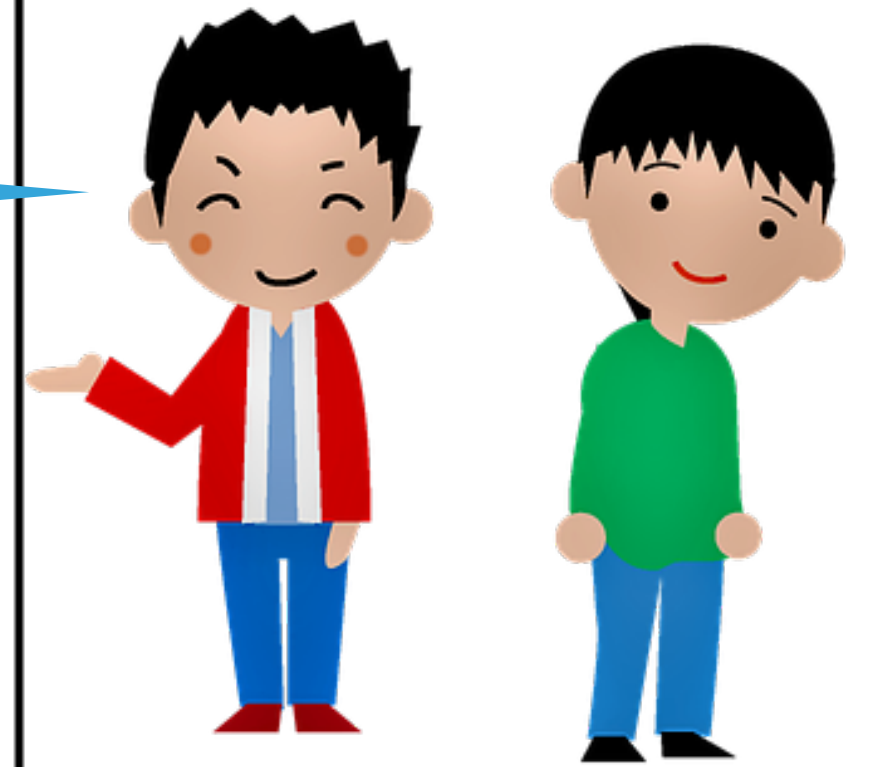
ERROR: InvalidReturnStatement - 4:5 - No return values are expected for foo

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





AbstractInstantiation	InvalidArrayAccess	MissingReturnType	PossiblyInvalidArrayAssignment	TaintedShell
AbstractMethodCall	InvalidArrayAssignment	MissingTemplateParam	PossiblyInvalidArrayOffset	TaintedSql
ArgumentIssue	InvalidArrayOffset	MissingThrowsDocblock	PossiblyInvalidCast	TaintedSystemSecret
ArgumentTypeCoercion	InvalidAttribute	MixedArgument	PossiblyInvalidClone	TaintedTextWithQuotes
AssignmentToVoid	InvalidCast	MixedArgumentTypeCoercion	PossiblyInvalidDocblockTag	TaintedUnserialize
CircularReference	InvalidCatch	MixedArrayAccess	PossiblyInvalidFunctionCall	TaintedUserSecret
ClassIssue	InvalidClass	MixedArrayAssignment	PossiblyInvalidIterator	TooFewArguments
CodeIssue	InvalidClone	MixedArrayOffset	PossiblyInvalidMethodCall	TooManyArguments
ComplexFunction	InvalidDocblock	MixedArrayTypeCoercion	PossiblyInvalidOperand	TooManyTemplateParams
ComplexMethod	InvalidDocblockParamName	MixedAssignment	PossiblyInvalidPropertyAssignment	Trace
ConfigIssue	InvalidEnumBackingType	MixedClone	PossiblyInvalidPropertyAssignmentValue	TraitMethodSignatureMismatch
ConflictingReferenceConstraint	InvalidEnumCaseValue	MixedFunctionCall	PossiblyInvalidPropertyFetch	
ConstructorSignatureMismatch	InvalidExtendClass	MixedInferredReturnType	PossiblyNullArgument	
ContinueOutsideLoop	InvalidFalsyableReturnType	MixedIssue	PossiblyNullArrayAccess	
DeprecatedClass	InvalidFunctionCall	MixedIssueTrait	PossiblyNullArrayAssignment	
DeprecatedConstant	InvalidGlobal	MixedMethodCall	PossiblyNullArrayOffset	
DeprecatedFunction	InvalidIterator	MixedOperand	PossiblyNullFunctionCall	
DeprecatedInterface	InvalidLiteralArgument	MixedPropertyAssignment	PossiblyNullIterator	
DeprecatedMethod	InvalidMethodCall	MixedPropertyFetch	PossiblyNullOperand	
DeprecatedProperty	InvalidNamedArgument	MixedPropertyTypeCoercion	PossiblyNullPropertyAssignment	
DeprecatedTrait	InvalidNullableReturnType	MixedReturnStatement	PossiblyNullPropertyAssignmentValue	
DocblockTypeContradiction	InvalidOperand	MixedReturnTypeCoercion	PossiblyNullPropertyFetch	
DuplicateArrayKey	InvalidParamDefault	MixedStringOffsetAssignment	PossiblyNullReference	
DuplicateClass	InvalidParent	MoreSpecificImplementedParamType	PossiblyUndefinedArrayOffset	UndefinedMagicPropertyAssignment
DuplicateConstant	InvalidPassByReference	MoreSpecificReturnType	PossiblyUndefinedGlobalVariable	UndefinedMagicPropertyFetch
DuplicateEnumCase	InvalidPropertyAssignment	MutableDependency	PossiblyUndefinedIntArrayOffset	UndefinedMethod
DuplicateEnumCaseValue	InvalidPropertyAssignmentValue	NamedArgumentNotAllowed	PossiblyUndefinedMethod	UndefinedPropertyAssignment
DuplicateFunction	InvalidPropertyFetch	NoEnumProperties	PossiblyUndefinedStringArrayOffset	UndefinedPropertyFetch
DuplicateMethod	InvalidReturnStatement	NoInterfaceProperties	PossiblyUndefinedVariable	UndefinedThisPropertyAssignment
DuplicateParam	InvalidReturnType	NoValue	PossiblyUnusedMethod	UndefinedThisPropertyFetch
EmptyArrayAccess	InvalidScalarArgument	NonInvariantDocblockPropertyType	PossiblyUnusedParam	UndefinedTrace
ExtensionRequirementViolation	InvalidScope	NonInvariantPropertyType	PossiblyUnusedProperty	UndefinedTrait
FalsableReturnStatement	InvalidStaticInvocation	NonStaticSelfCall	PossiblyUnusedReturnValue	UndefinedVariable
FalseOperand	InvalidStringClass	NullArgument	PropertyIssue	UnevaluatedCode
ForbiddenCode	InvalidTemplateParam	NullArrayAccess	PropertyNotSetInConstructor	UnhandledMatchCondition
ForbiddenEcho	InvalidThrow	NullArrayOffset	PropertyTypeCoercion	UnimplementedAbstractMethod
FunctionIssue	InvalidToString	NullFunctionCall	PsalmInternalError	UnimplementedInterfaceMethod
IfThisIsMismatch	InvalidTraversableImplementation	NullIterator	RawObjectIteration	UninitializedProperty
ImplementationRequirementViolation	InvalidTypeImport	NullOperand	RedundantCast	UnnecessaryVarAnnotation
ImplementedParamTypeMismatch	LessSpecificImplementedReturnType	NullPropertyAssignment	RedundantCastGivenDocblockType	UnrecognizedExpression
ImplementedReturnTypeMismatch	LessSpecificReturnStatement	NullPropertyFetch	RedundantCondition	UnrecognizedStatement
ImplicitToStringCast	LessSpecificReturnType	NullReference	RedundantConditionGivenDocblockType	UnresolvableInclude
ImpureByReferenceAssignment	LoopInvalidation	NullableReturnStatement	RedundantIdentityWithTrue	UnsafeGenericInstantiation
ImpureFunctionCall	MethodIssue	OverriddenMethodAccess	RedundantPropertyInitializationCheck	UnsafeInstantiation
ImpureMethodCall	MethodSignatureMismatch	OverriddenPropertyAccess	ReferenceConstraintViolation	UnusedClass
ImpurePropertyAssignment	MethodSignatureMustOmitReturnType	ParadoxicalCondition	ReservedWord	UnusedClosureParam
ImpurePropertyFetch	MismatchingDocblockParamType	ParamNameMismatch	StringIncrement	UnusedConstructor
ImpureStaticProperty	MismatchingDocblockPropertyType	ParentNotFound	TaintedCallable	UnusedForeachValue
ImpureStaticVariable	MismatchingDocblockReturnType	ParseError	TaintedCookie	UnusedFunctionCall
ImpureVariable	MissingClosureParamType	PluginIssue	TaintedCustom	UnusedMethod
InaccessibleClassConstant	MissingClosureReturnType	PossibleRawObjectIteration	TaintedEval	UnusedMethodCall
InaccessibleMethod	MissingConstructor	PossiblyFalseArgument	TaintedFile	UnusedParam
InaccessibleProperty	MissingDependency	PossiblyFalseIterator	TaintedHeader	UnusedProperty
InterfaceInstantiation	MissingDocblockType	PossiblyFalseOperand	TaintedHtml	UnusedPsalmSuppress
InternalClass	MissingFile	PossiblyFalsePropertyAssignmentValue	TaintedInclude	UnusedReturnValue
InternalMethod	MissingImmutableAnnotation	PossiblyFalseReference	TaintedInput	UnusedVariable
InternalProperty	MissingParamType	PossiblyInvalidArgument	TaintedLdap	VariableIssue
InvalidArgument	MissingPropertyType	PossiblyInvalidArrayAccess	TaintedSSRF	

Psalm can find over 240 different types of issue



Amazing, let's add you to our project.

How strict do 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 your workflow 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
```

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



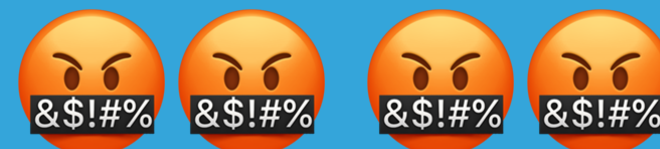
# Storming

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");
```

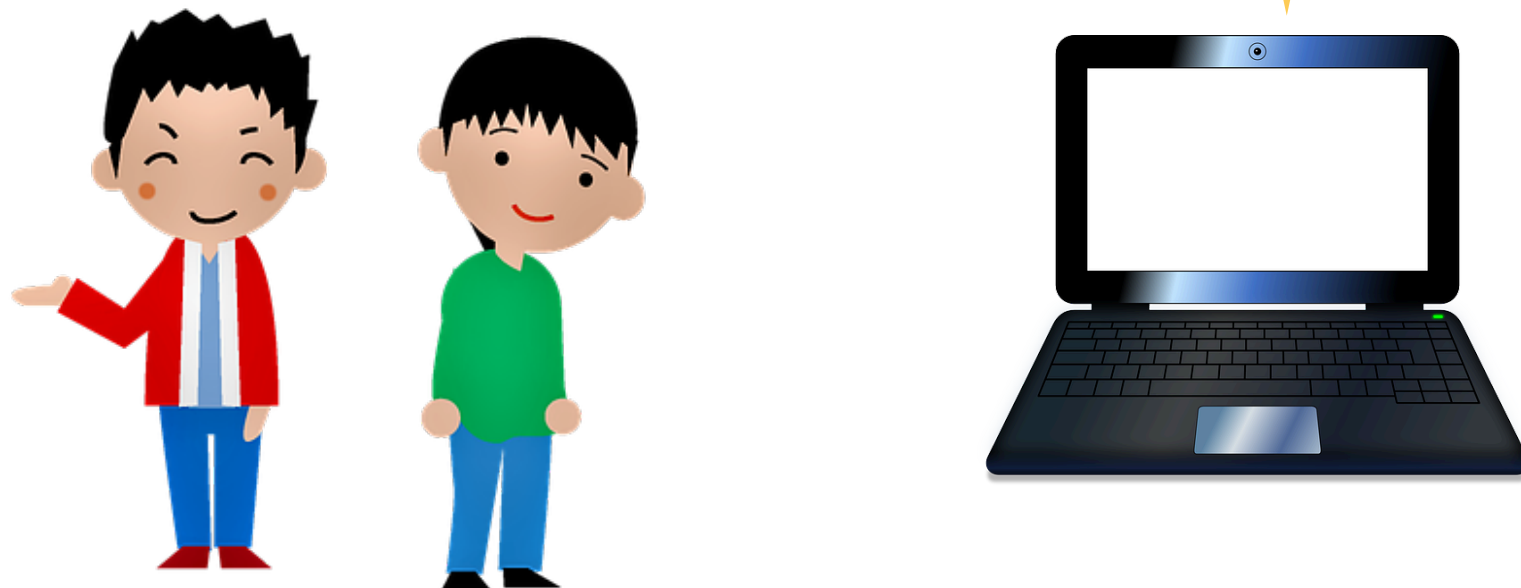
The code works most of the time, but it fails in some scenarios.  
E.g. this code will fail if a user is not found.



## #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.



**#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



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

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

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

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

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

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

Psalm website

[https://psalm.dev/docs/annotating\\_code/templated\\_annotations/](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>

PackagistThe PHP Package Repository

BrowseSubmitCreate accountSign in



Search packages...

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

<b>psalm/plugin-symfony</b>	Gherkin	📄 2 482 124	★ 163
Psalm Plugin for Symfony			
<b>weirdan/doctrine-psalm-plugin</b>	Gherkin	📄 1 289 008	★ 70
Stubs to let Psalm understand Doctrine better			

Active filters

type: psalm-pluginClear 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](#)
- [Nextas 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.

Package	Type	Downloads	Stars
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`



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\\_psalms/dealing\\_with\\_code\\_issues/#using-a-baseline-file](https://psalm.dev/docs/running_psalms/dealing_with_code_issues/#using-a-baseline-file)



A few quick questions...

Do we run static analysis on our test code?

**Yes!**



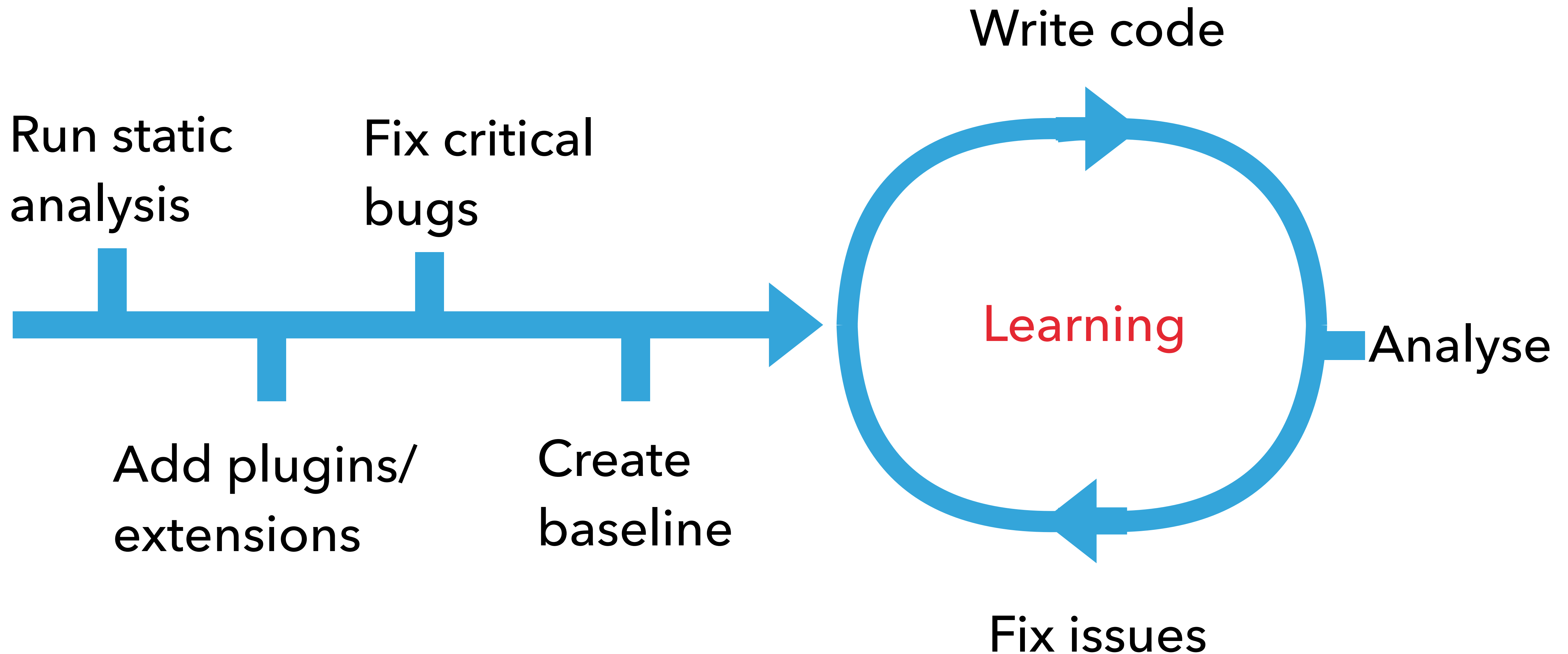
**There is this one rule that I really disagree with.  
What do I do?**

**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 */
```

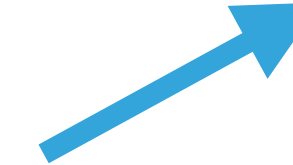
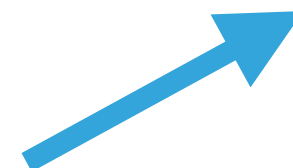
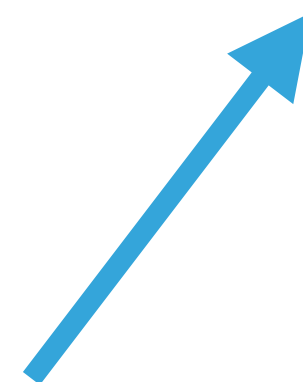




This is great.  
We're introducing fewer bugs  
AND writing clearer code.



Let's change this  
one line of code.



I can help you safely  
refactor



# 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;

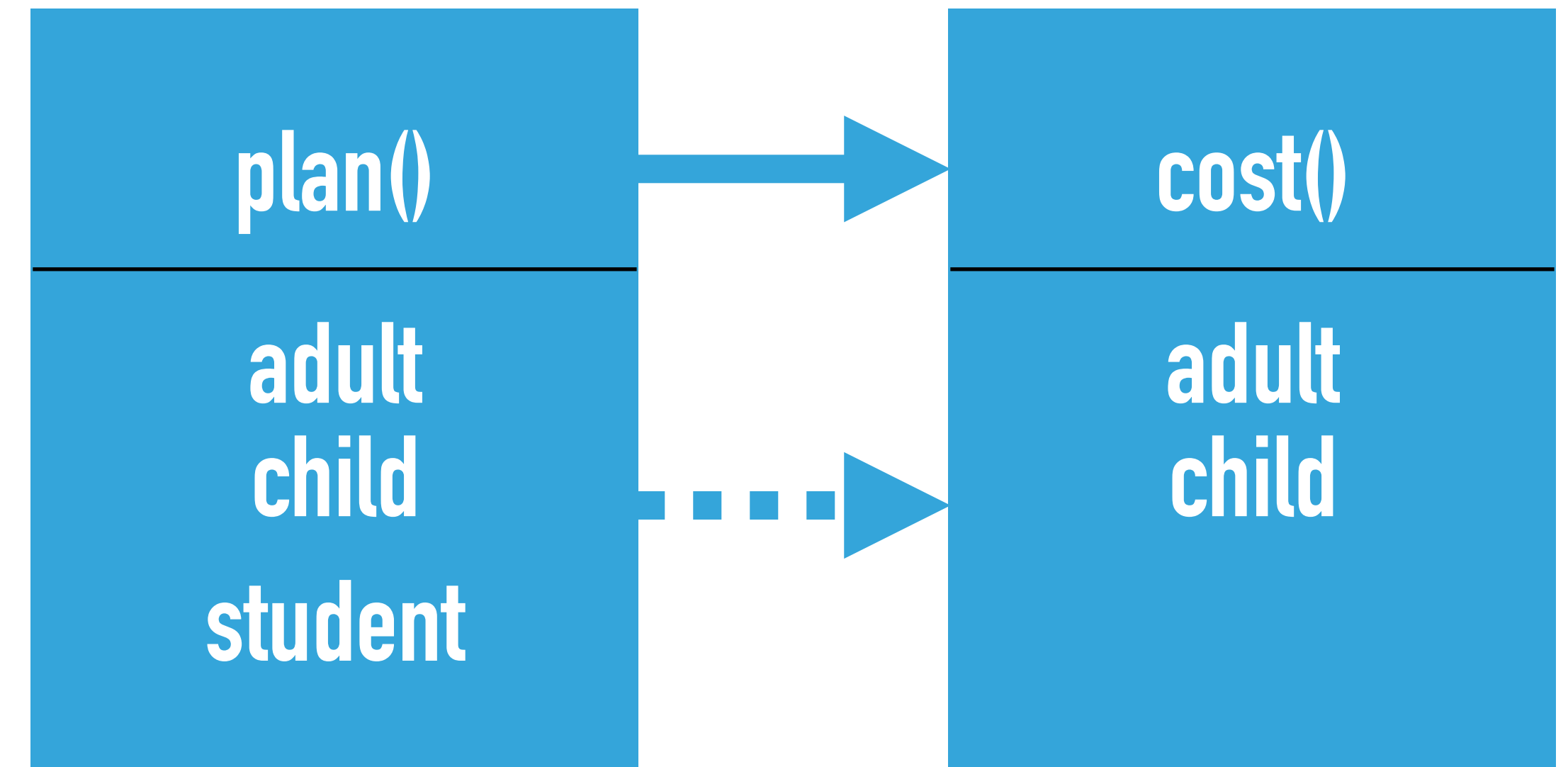
        case 'ADULT':
            return 20;
    }

    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 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



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

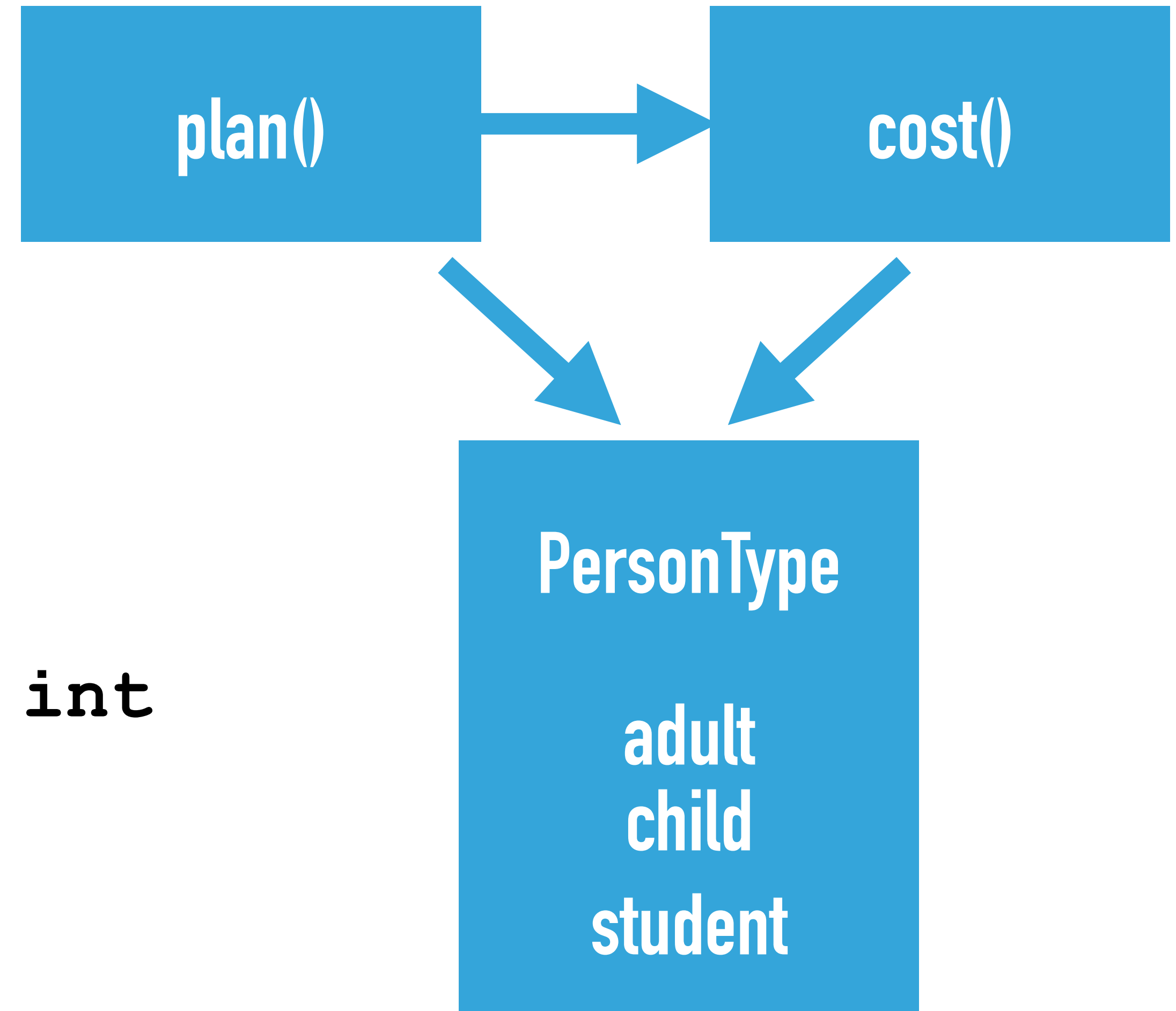
```
function cost(PersonType $type) : int
{
    return match($type) {
        PersonType::CHILD => 10,
        PersonType::ADULT => 20,
    };
}
```

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,
    };
}
```



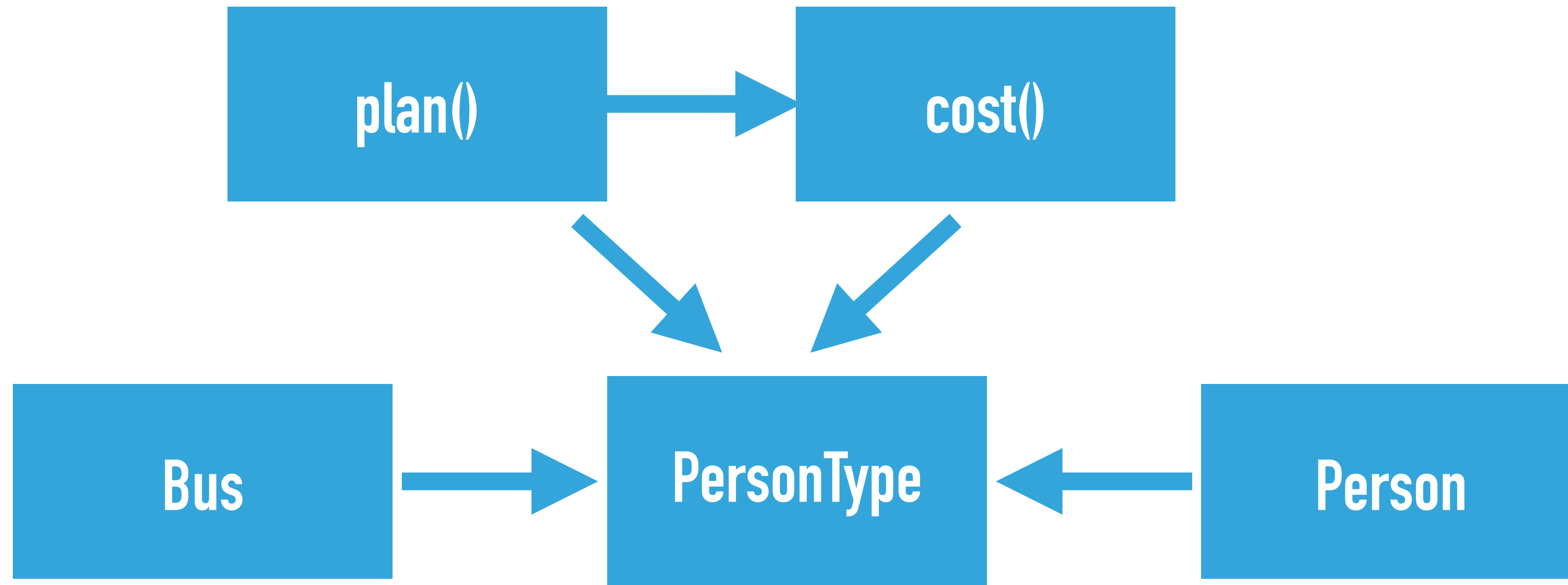
```
enum PersonType
{
    case ADULT;
    case CHILD;
    case STUDENT;
}
```

There is a problem with  
cost()



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

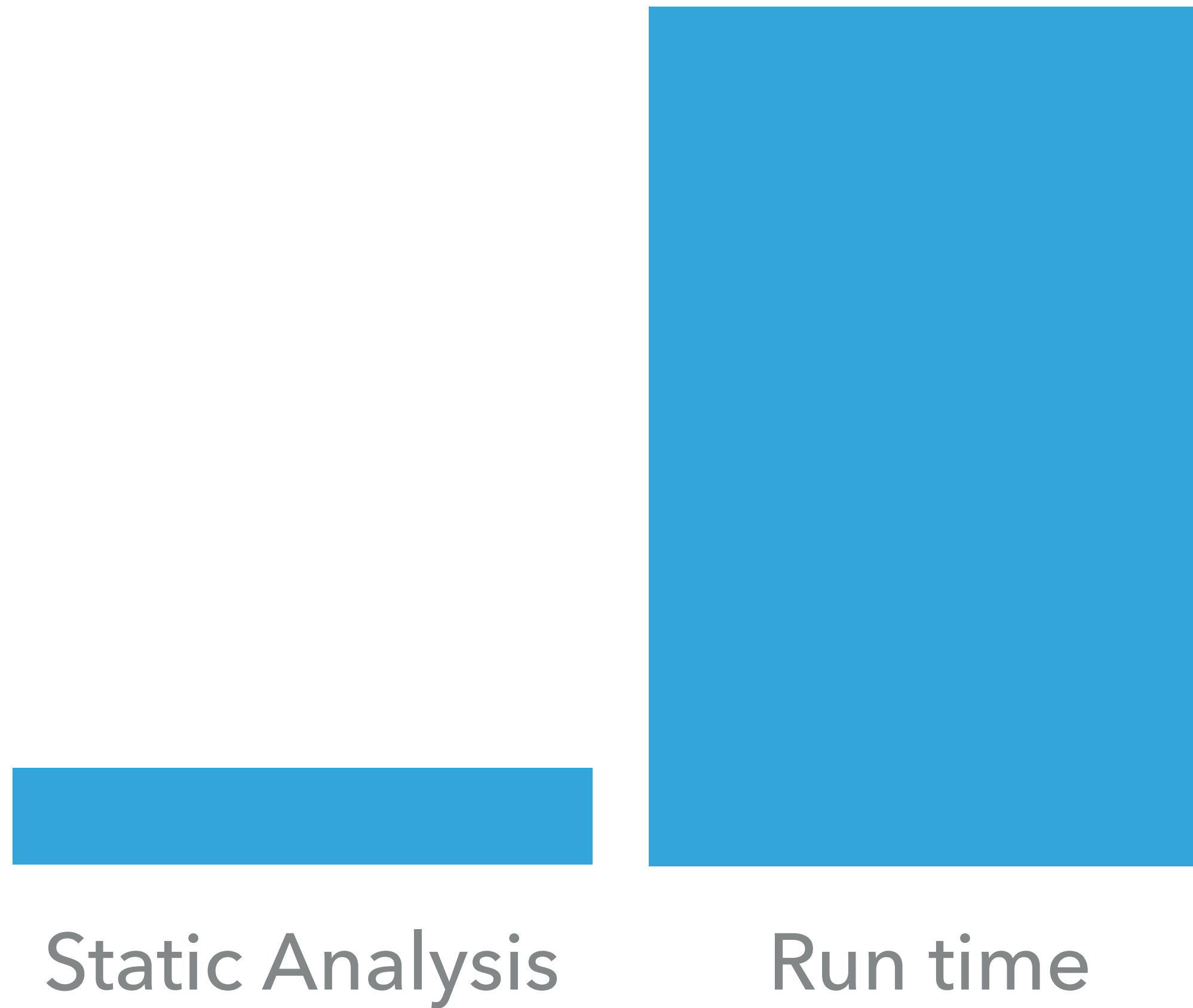
	Input	Output
Test 1	CHILD	10
Test 2	ADULT	20
Test 3	STUDENT	15





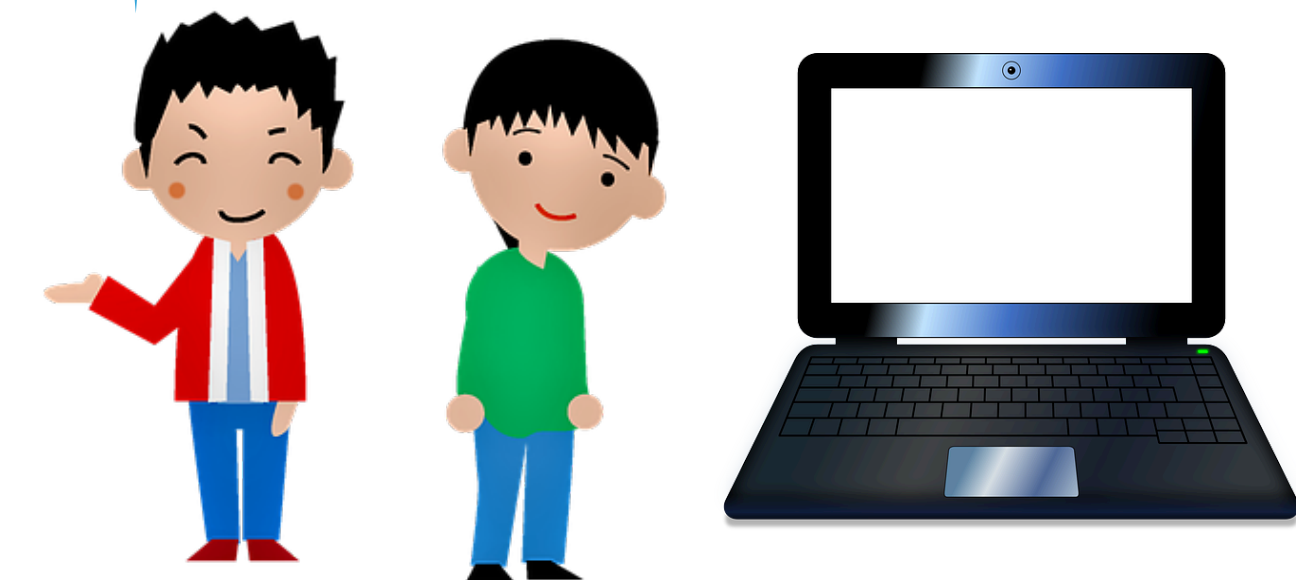
# COST OF A BUG

Cost



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!!



```
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` or `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.





```

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.



It's been quite a journey.

I can't imagine a project without you in the future!

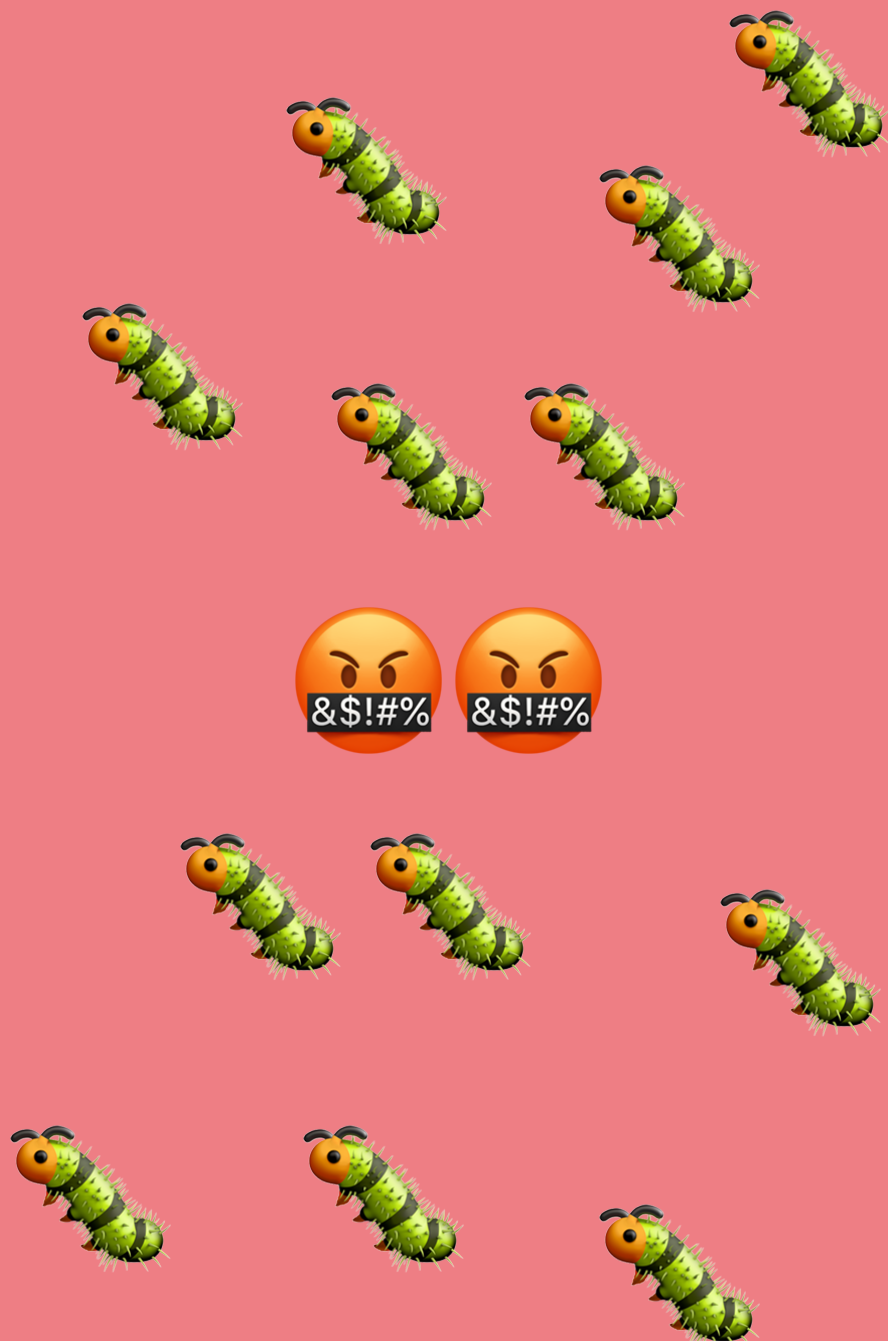
**You're too kind!**



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

@daveliddament

**Any questions?**





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

Ask Dave, he knows someone.

