

Dave Liddament (Lamp Bristol):

# Unlock the Full Potential of Static Analysis



Confoo.CA

DEVELOPER CONFERENCE

**Can we write code in such a way to reduce the chance of introducing bugs?**

**Can static analysis help us achieve this goal?**

# HIERARCHY OF CONTROL

Most effective

**ELIMINATE**

**SUBSTITUTE**

**ENGINEERING CONTROLS**

**ADMINISTRATIVE CONTROLS**

Least effective

**PPE**

# PREVENTING BUGS IN SOFTWARE

Most effective



**ELIMINATE**



**STATIC ANALYSIS**



**TESTING**

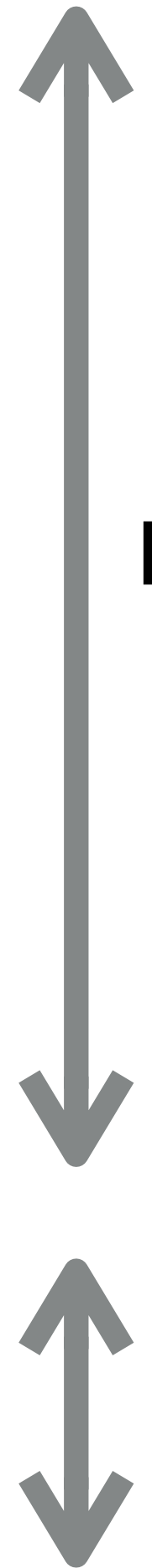


**MANUAL INSPECTION**

Least effective



**ALERT**



Developer

Users



## **The context for this is:**

- application code**
- code or requirements evolve**
- large projects**
- projects with many developers**

**#1: Use value objects**

**#2: Use extended type system**

**#3: Asserts at the system boundaries**

**#4: Prevent objects from being in invalid states**

**#5: Remove default handling**

**#6: Assume impure functions**

**#7: Enforce architectural constraints**

**# Bonus?**

```
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

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.**

# Developers make mistakes.

## You can still have bugs even if:

- Tests have 100% code coverage
- You've used TDD

```
function addDetails(  
    string $name, string $email, string $address  
): void { ... }
```

TESTING

```
addDetails (
```

```
"dave@example.com",
```

```
"dave",
```

```
"123 Some Street, Some City, AB1 2CD",
```

```
);
```

MANUAL INSPECTION

ALERT

# #1: Use value objects

```
final readonly class Name
{
    public function __construct(public string $value) {}
}
```

```
function addDetails(  
    Name $name, Email $email, Address $address  
): void { ... }
```

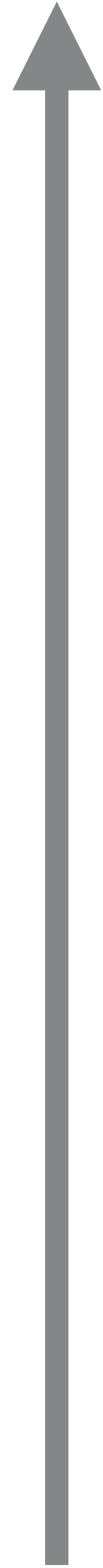
```
$email = new Email("dave@example.com");
```

```
$name = new Name("dave");
```

```
$address = new Address("123 Some Street, Some City, AB1 2CD");
```

ELIMINATE

```
addDetails($email, $name, $address);
```



**ELIMINATE**

**STATIC ANALYSIS**

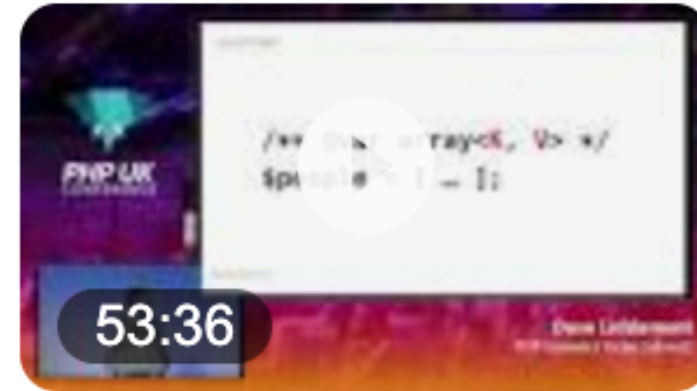
**TESTING**

**MANUAL INSPECTION**

**ALERT**

## #2: Use extended type system

```
/** @template T */  
class Queue  
{  
    /** @param T $item */  
    public function add($item): void {...}  
  
    /** @return T */  
    public function next(): {...}  
}
```



PHP Generics Today (almost) ... Support for generics is high up many PHP developers' wish lists. This talk is a deep dive into generics, their benefits a...

PHP UK Conference · PHP UK Conference · 18 Mar 2020

```
final readonly class Power
{
    /** @param int<0,100> $value */
    public function __construct(public int $value) {}
}
```

## STATIC ANALYSIS

`$validPower = new Power(76);` 

`$invalidPower = new Power(101);` 

```
/** @var int<0,100> */
```

```
/** @var int<min,7> $value */
```

```
/** @param positive-int $value */
```

```
/** @return 4|6|18 */
```

```
/** @param 8|negative-int $value */
```


```
/** @return "red"|"green" $value */
```

```
/** @return non-empty-string $value */
```

```
final class Status
{
    public const STATUS_SUCCESS = 0;
    public const STATUS_FILE_ACCESS_ERROR = 255;
    public const STATUS_INVALID_CONTENTS = 254;
}
```

```
/** @param Status::* $value */
```

```
function processStatus(int $value): void {...}
```

```
processStatus(Status::STATUS_SUCCESS); 
```

```
processStatus(255); 
```

```
processStatus(8); 
```

```
final class Flags
```

```
{
```

```
    public const FLAG_SORT = 1;
```

```
    public const FLAG_VERBOSE = 2;
```

```
    public const FLAG_ENCODE = 4;
```

```
}
```

```
/** @param int-mask<1,2,4> $flags */
```

```
function takesFlags(int $flags): void {...}
```

```
takesFlags(Flags::FLAG_VERBOSE|Flags::FLAG_ENCODE); 
```

```
takesFlags(7); 
```

```
takesFlags(8); 
```

```
final class Flags
```

```
{
```

```
    public const FLAG_SORT = 1;
```

```
    public const FLAG_VERBOSE = 2;
```

```
    public const FLAG_ENCODE = 4;
```

```
}
```

```
/** @param int-mask-of<Flags::*> $flags */
```

```
function takesFlags(int $flags): void {...}
```

```
takesFlags(Flags::FLAG_VERBOSE|Flags::FLAG_ENCODE); 
```

```
takesFlags(7); 
```

```
takesFlags(8); 
```

```
final class Flags
{
    public const FLAG_SORT = 1;
    public const FLAG_VERBOSE = 2;
    public const FLAG_ENCODE = 4;

    public const STATUS_SUCCESS = 0;
    public const STATUS_FILE_ACCESS_ERROR = 255;
    public const STATUS_INVALID_CONTENTS = 254;
}

/** @param int-mask-of<Flags::FLAG_*> $flags */
function takesFlags(int $flags): void {...}
```

```
function getAddress(): array
{
    return [
        "Street": "1 Some street",
        "City": "Bristol",
        "Postcode": "BS1 1AB",
    ];
}
```

```
/**
 * @return string[]
 */
function getAddress(): array
{
    return [
        "Street": "1 Some street",
        "City": "Bristol",
        "Postcode": "BS1 1AB",
    ];
}
```

```
/**  
 * @return array{string, string, string}  
 */  
function getAddress(): array  
{  
    return [  
        "Street" => "1 Some street",  
        "City" => "Bristol",  
        "Postcode" => "BS1 1AB",  
    ];  
}
```

```
/**
 * @return array{street:string, city:string, postcode:string}
 */
function getAddress(): array
{
    return [
        "street" => "1 Some street",
        "city" => "Bristol",
        "postcode" => "BS1 1AB",
    ];
}
```

```
/**  
 * @return array{name:string, ?age:int, registered:bool}  
 */  
function getPersonDetails(): array  
{  
    return [  
        "name" => "Dave",  
        "registered" => false,  
    ];  
}
```

```
/**
 * @return Person[]
 * @return array<Person>
 * @return array<string, Person>
 */
function getPersonDetails(): array
{
    return [
        "Jane" => new Person("Jane"),
        "Bob" => new Person("Bob"),
        "Charlie" => new Person("Charlie"),
    ];
}
```

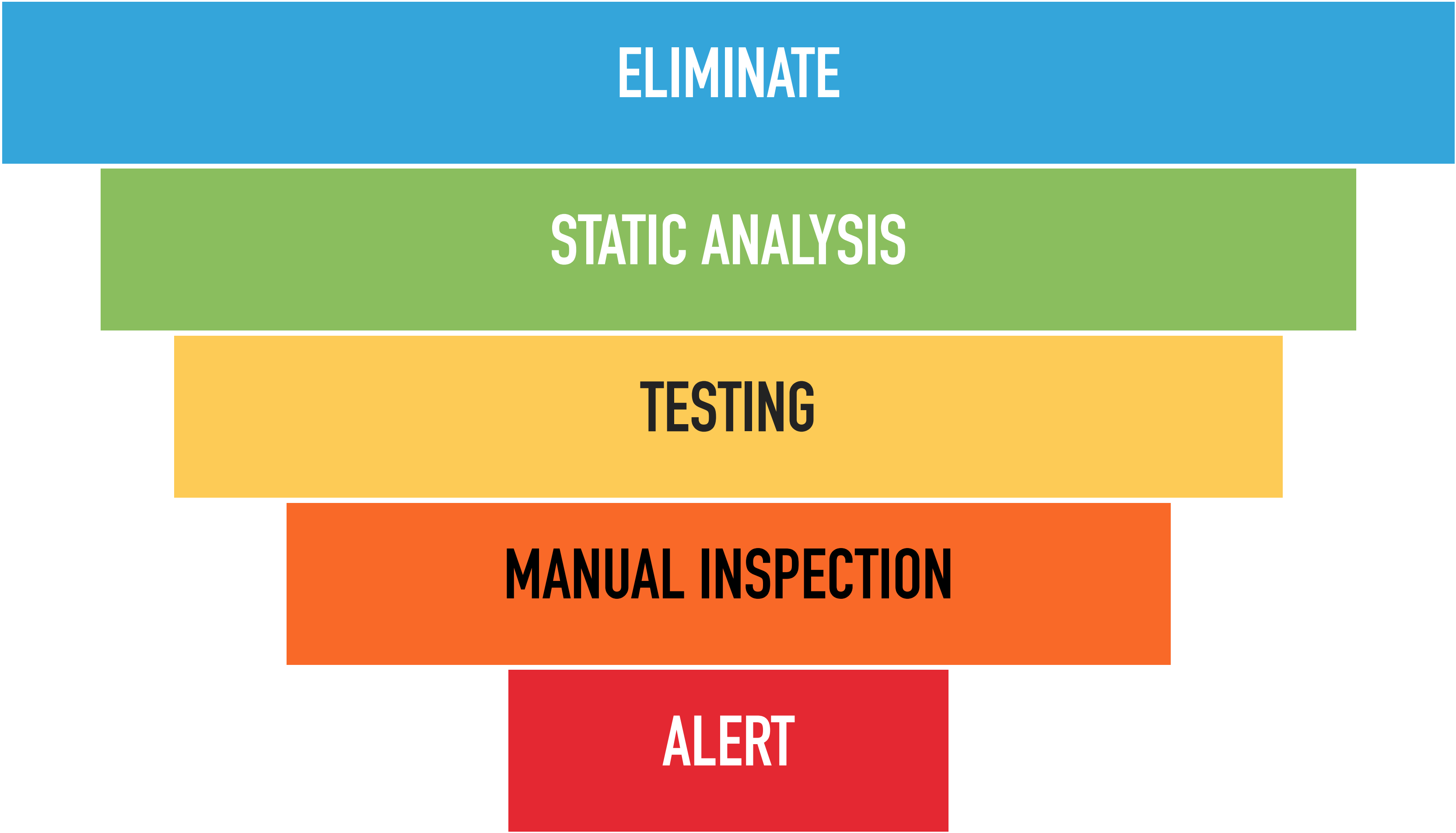
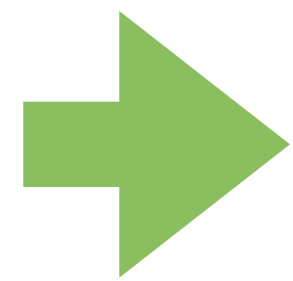
```
/**
 * @return array<int, Person>
 */
function getPersonDetails(): array
{
    return [
        1 => new Person("Jane"),
        10 => new Person("Bob"),
        8 => new Person("Charlie"),
    ];
}
```

```
/**
 * @return list<Person>
 */
function getPersonDetails(): array
{
    return [
        new Person("Jane"),
        new Person("Bob"),
        new Person("Charlie"),
    ];
}
```

- Array keys are ints
- Array keys increment by 1
- Array is zero indexed

$\$i \geq 0 \ \&\& \ < \text{count}(\$array)$

$\$array[\$i]$  must exist



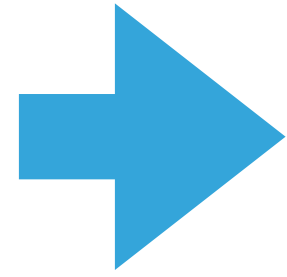
**ELIMINATE**

**STATIC ANALYSIS**

**TESTING**

**MANUAL INSPECTION**

**ALERT**



**ELIMINATE**

**STATIC ANALYSIS**

**TESTING**

**MANUAL INSPECTION**

**ALERT**

```
final class Status
{
    public const STATUS_SUCCESS = 0;
    public const STATUS_FILE_ACCESS_ERROR = 255;
    public const STATUS_INVALID_CONTENTS = 254;
}
/** @param Status::* $value */
function processStatus(int $value): void {...}
```

STATIC ANALYSIS

---

```
enum Status: int
{
    case STATUS_SUCCESS = 0;
    case STATUS_FILE_ACCESS_ERROR = 255;
    case STATUS_INVALID_CONTENTS = 254;
}

function processStatus(Status $value): void {...}
```

ELIMINATE

```
final class Flags
{
    public const FLAG_SORT = 1;
    public const FLAG_VERBOSE = 2;
    public const FLAG_ENCODE = 4;
}
/** @param int-mask-of<Flags::*> $flags */
function process(int $flags): void {...}
```

STATIC ANALYSIS

```
final readonly class Flags
{
    public function __construct(
        public bool $sort = false,
        public bool $verbose = false,
        public bool $encode = false,
    ) {}
}

function process(Flags $flags): void {...}
```

ELIMINATE

```
/** @return array{name:string, age:int} */  
function getPersonDetails(): array {...}
```

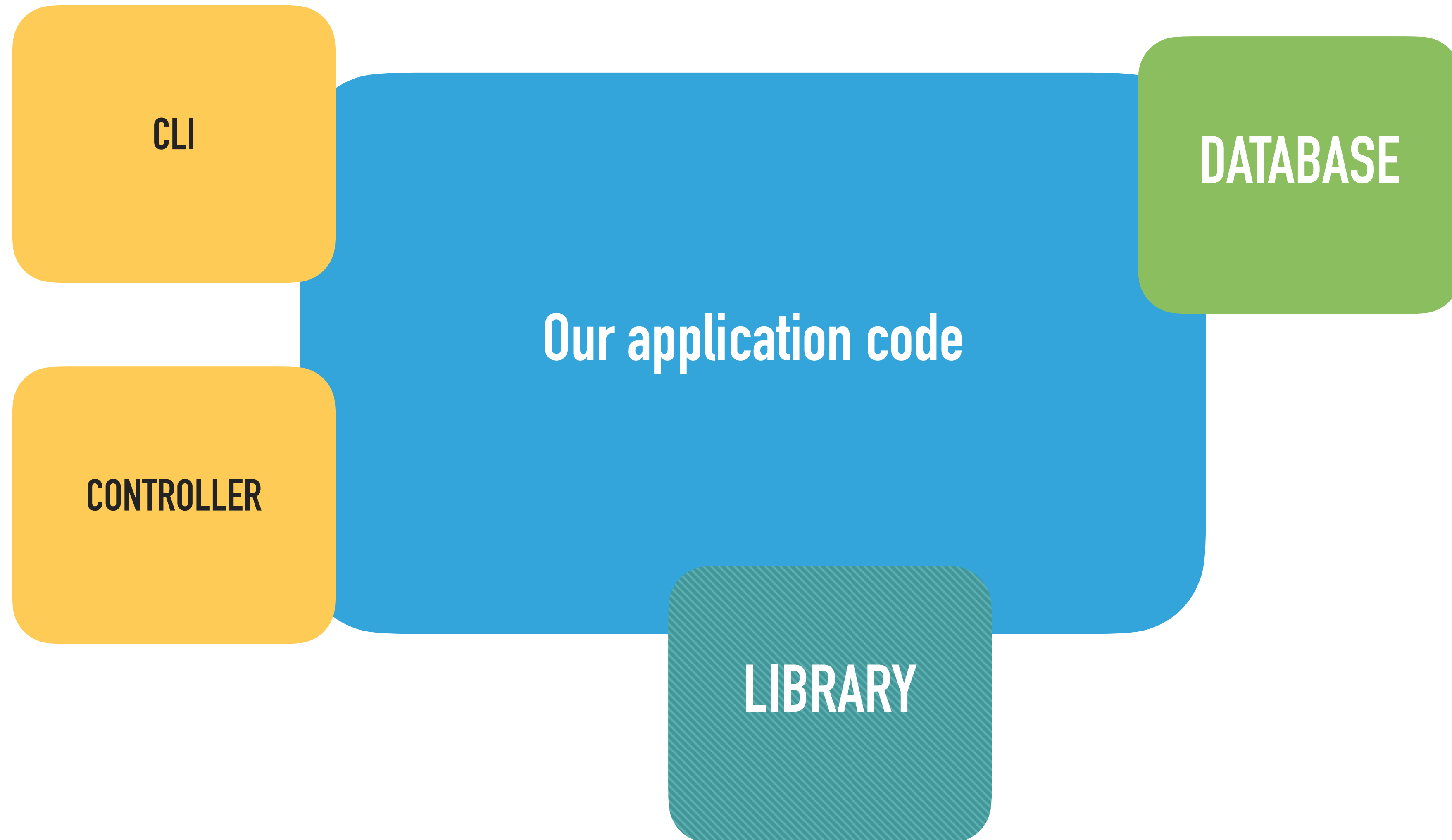
STATIC ANALYSIS

```
final readonly class PersonDetails  
{  
    public function __construct(  
        public string $name,  
        public int $age,  
    ) {}  
}
```

ELIMINATE

```
function getPersonDetails(): PersonDetails {...}
```

# #3: Asserts at the system boundaries



```
final readonly class Power
{
    /** @param int<0,100> $value */
    public function __construct(public int $value) {}
}
```

```
final readonly class Power
{
    /** @param int<0,100> $value */
    public function __construct(public int $value) {}
}
```

```
interface Request
```

```
{
    public function getInt(string $key): int;
}
```

**Expect int<0,100> got int**

```
$powerAsInt = $request->getInt("power");
$power = new Power($powerAsInt);
```

```
function validate(int $power): void
{
    if ($power < 0 || $power > 100) {
        throw new InvalidValue();
    }
}
```

```
$powerAsInt = $request->getInt("power");
```

```
validate($powerAsInt);
```

```
$power = new Power($powerAsInt);
```

Expect int<0,100> got int

```
/** @phpstan-assert int<0,100> $power */  
function validate(int $power): void  
{  
    if ($power < 0 || $power > 100) {  
        throw new InvalidValue();  
    }  
}
```

```
/** @phpstan-assert int<0,100> $power */
```

```
function validate(int $power): void
```

```
{
```

```
    if ($power < 0 || $power > 100) {
```

```
        throw new InvalidValue();
```

```
    }
```

```
}
```

```
$powerAsInt = $request->getInt("power");
```

```
validate($powerAsInt);
```

```
$power = new Power($powerAsInt);
```



```
/** @phpstan-assert int<0,100> $power */
```

```
/** @psalm-assert int<0,100> $power */
```

```
/** @psalm-assert int<1,6> $value */  
function validateDiceValue(int $value): void  
{  
    if ($value <= 1 || $value >= 6) {  
        throw new InvalidArgumentException();  
    }  
}
```

**MANUAL INSPECTION**

```
/** @psalm-assert-if-true int<0,100> $power */  
function isValid(int $power): bool  
{  
    return ($power >= 0 && $power <= 100);  
}
```

```
/** @psalm-assert-if-true int<0,100> $power */  
function isValid(int $power): bool  
{  
    return ($power >= 0 && $power <= 100);  
}  
  
$powerAsInt = $request->getInt("power");  
if (isValid($powerAsInt)) {  
    $power = new Power($powerAsInt);  
} else {  
    // Handle invalid data  
}
```

# ASSERTIONS

```
/** @phpstan-assert !null $value */
```

```
/** @psalm-assert-if-true string $value */
```

```
/** @psalm-assert-if-false string $value */
```

```
final class Person {  
    public function __construct(private ?string $email) {}
```

```
    public function hasEmail(): bool {  
        return $this->email !== null;  
    }
```

```
    public function getEmail(): ?string {  
        return $this->email;  
    }
```

```
}
```

```
function process(Person $person): void {
```

```
    if ($person->hasEmail()) {  
        sendEmail($person->getEmail());  
    }
```

```
}
```

```
function sendEmail(string $email): void {...}
```

```
final class Person {
    public function __construct(private ?string $email) {}

    /** @psalm-assert-if-true string $this->getEmail() */
    public function hasEmail(): bool {
        return $this->email !== null;
    }

    public function getEmail(): ?string {
        return $this->email;
    }
}
```

```
function process(Person $person): void {
    if ($person->hasEmail()) {
        sendEmail($person->getEmail());
    }
}
```

```
function sendEmail(string $email): void {...}
```

## #4: Prevent objects from being in invalid states

```
class Person {
```

```
    private string $name;
```

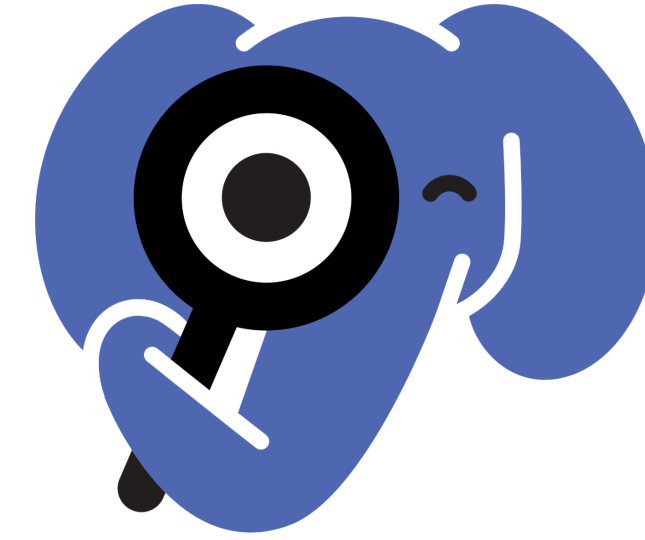
```
    public function setName(string $name): void {  
        $this->name = $name;  
    }
```

```
    public function getName(): string {  
        return $this->name;  
    }
```

```
}
```



# Psalm



`phpstan.neon`

`parameters:`

`checkUninitializedProperties: true`

```
class Person {
```

```
    public function __construct(  
        private string $name,  
    ) {}
```

```
    public function setName(string $name): void {  
        $this->name = $name;  
    }
```

```
    public function getName(): string {  
        return $this->name;  
    }
```

```
}
```

ELIMINATE

STATIC ANALYSIS

```
class Job
{
    public function completedBy(User $user): void {...}

    public function completedAt(int $timestamp): void {...}
}
```

---

```
class Job
{
    public function completed(
        User $user,
        int $timestamp,
    ): void {...}
}
```

**ELIMINATE**

## #5: Remove default handling

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

```
function cost(string $type): int
{
    $price = null;
    if ($type === "CHILD") {
        $price = 10;
    }
    if ($type === "ADULT") {
        $price = 20;
    }
    if ($price === null) {
        throw new LogicException("Invalid type [$type]");
    }
    return $price;
}
```

**ALERT**

```
/** @param Type::* $type */  
function cost(string $type): int  
{  
    $price = null;  
    if ($type === Type::CHILD) {  
        $price = 10;  
    }  
    if ($type === Type::ADULT) {  
        $price = 20;  
    }  
    if ($price === null) {  
        throw new LogicException("Invalid type [$type]");  
    }  
    return $price;  
}
```

```
final class Type  
{  
    public const ADULT = "ADULT";  
    public const CHILD = "CHILD";  
}
```

```
/** @param Type::* $type */
function cost(string $type): int
{
    $price = null;
    if ($type === Type::CHILD) {
        $price = 10;
    }
    if ($type === Type::ADULT) {
        $price = 20;
    }
    if ($price === null) {
        throw new LogicException("Invalid type [$type]");
    }
    return $price;
}
```

```
final class Type
{
    public const ADULT = "ADULT";
    public const CHILD = "CHILD";
    public const OAP = "OAP";
}
```

**ALERT**

```
/** @param Type::* $type */
function cost(string $type): int
{
    $price = null;
    if ($type === Type::CHILD) {
        $price = 10;
    }
    if ($type === Type::ADULT) {
        $price = 20;
    }

    return $price;
}
```

```
final class Type
{
    public const ADULT = "ADULT";
    public const CHILD = "CHILD";
    public const OAP = "OAP";
}
```

STATIC ANALYSIS

**ELIMINATE**

**STATIC ANALYSIS**

**TESTING**

**MANUAL INSPECTION**

**ALERT**



```
final class Type
{
    public const ADULT = "ADULT";
    public const CHILD = "CHILD";
}
```

```
/** @param Type::* $type */
function cost(string $type): int
{
    return match($type) {
        Type::CHILD => 10,
        Type::ADULT => 20,
    };
}
```

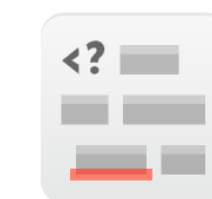
# No default clause

## #6: Assume impure functions

```
class Person {  
    public function getName(): ?string {...}  
}
```



```
function process(Person $person): void {  
    if ($person->getName() !== null) {  
        takesString($person->getName());  
    }  
}
```



**Psalm**

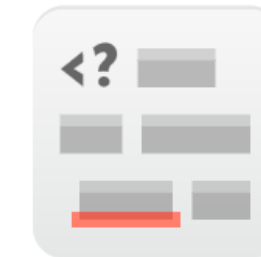


```
function takesString(string $value): void {...}
```

 **Psalm**

```
class Person {  
    /** @psalm-pure */  
    public function getName(): ?string {...}  
}  
  
function process(Person $person): void {  
    if ($person->getName() !== null) {  
        takesString($person->getName());  
    }  
}  
  
function takesString(string $value): void {...}
```

```
final readonly class Person {  
    public function __construct(private ?string $name) {}  
  
    public function getName(): ?string {  
        return $this->name;  
    }  
}
```



# Psalm

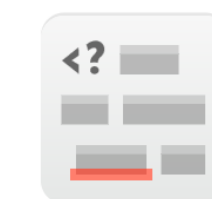
```
function process(Person $person): void {  
    if ($person->getName() !== null) {  
        takesString($person->getName());  
    }  
}
```

```
function takesString(string $value): void {...}
```

```
class Queue {  
    public function getNext(): ?string {...}  
}
```



```
function process(Queue $queue): void {  
    if ($queue->getNext() !== null) {  
        takesString($queue->getNext());  
    }  
}
```



**Psalm**



```
function takesString(string $value): void {...}
```

```
class Queue {  
    /** @phpstan-impure */  
    public function getNext(): ?string {...}  
}
```



```
function process(Queue $queue): void {  
    if ($queue->getNext() !== null) {  
        takesString($person->getNext());  
    }  
}
```

```
function takesString(string $value): void {...}
```



`phpstan.neon`

`parameters:`

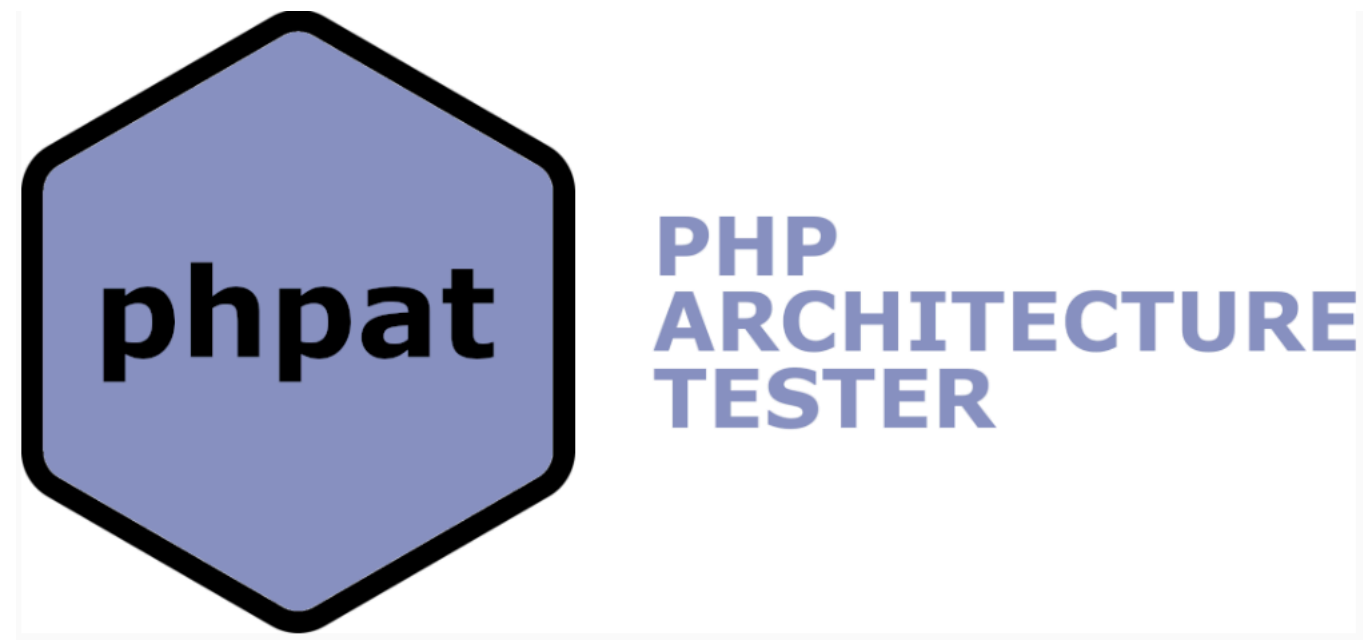
`rememberPossiblyImpureFunctionValues: false`

```
class Foo {  
    public function bar(): ?string {...}  
}
```

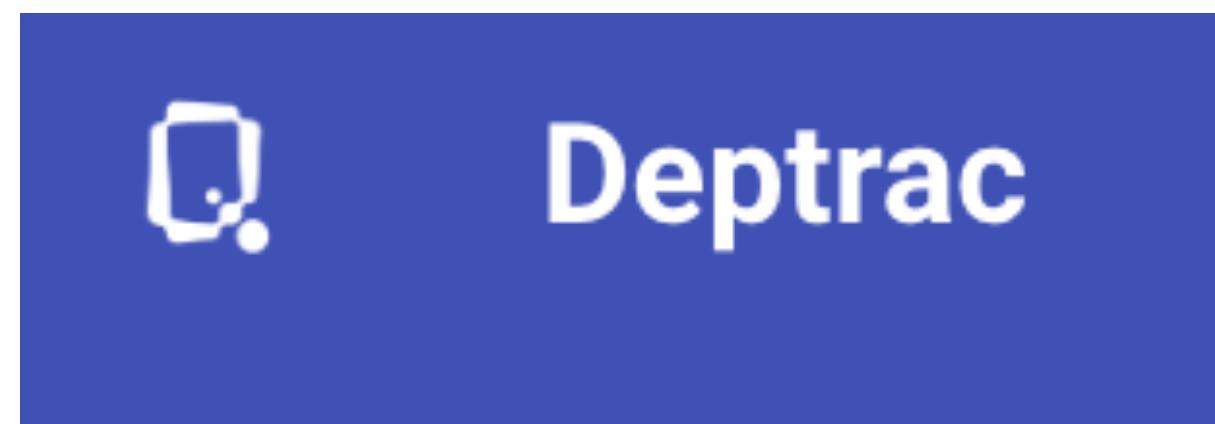
```
function process(Foo $foo): void {  
    $value = $foo->bar();  
    if ($value !== null) {  
        takesString($value);  
    }  
}
```

```
function takesString(string $value): void {...}
```

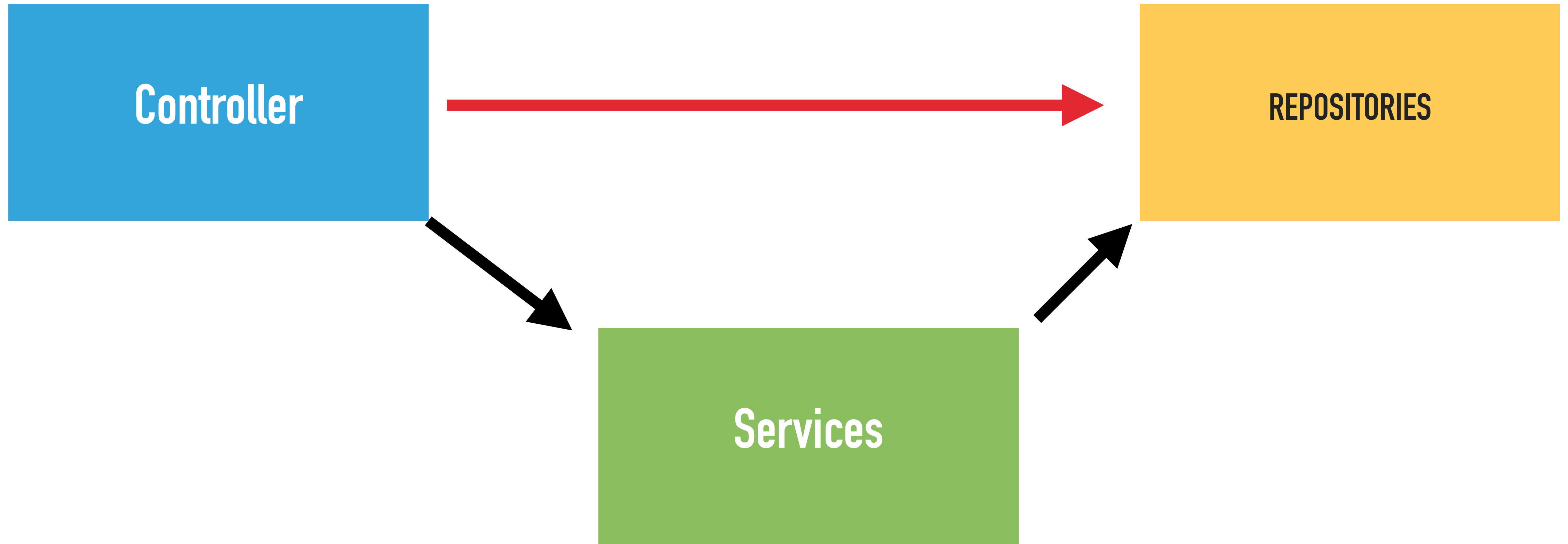
## #7: Enforce architectural constraints



<https://www.phpat.dev/>



<https://github.com/deptrac/deptrac>



```
public function testDomain(): Rule
{
    return PHPat::rule()
        ->classes(Selector::namespace( 'App\Domain' ))
        ->shouldNotDependOn()
        ->classes(
            Selector::namespace( 'App\Application' ),
            Selector::namespace( 'App\Infrastructure' )
        );
}
```

<https://github.com/DaveLiddament/php-language-extensions>

# php language extensions

**#[Friend]**  
**#[MustUseResult]**  
**#[NamespaceVisibility]**  
**#[InjectableVersion]**  
**#[Override]**  
**#[RestrictTraitTo]**  
**#[TestTag]**

<https://github.com/DaveLiddament/php-language-extensions>

```
class Person  
{  
    #[Friend(PersonBuilder::class)]  
    public function __construct() {...}  
  
}
```

# CREATE YOUR OWN CUSTOM RULES

The diagram illustrates the structure of PHP tokens and their flags:

- CLASS** (Flags: 0) branches into:
  - IDENTIFIER** (Name: PersonNotifier)
  - PROPERTY** (Flags: 4)
  - CLASS METHOD** (Flags: 1)
- CLASS METHOD** (Flags: 1) branches into:
  - IDENTIFIER** (Name: notifyPlayer)
  - PARAM** (ByRef: false, Variadic: false)
  - METHOD CALL**
- PARAM** (ByRef: false, Variadic: false) branches into:
  - IDENTIFIER** (Name: string)
  - VALUE** (ByRef: false, Variadic: false)

Video details: **DAVE LIDDAMENT**, Extending the PHP Language with static analysis, 41:29



**#1: Use value objects**

**#2: Use extended type system**

**#3: Asserts at the system boundaries**

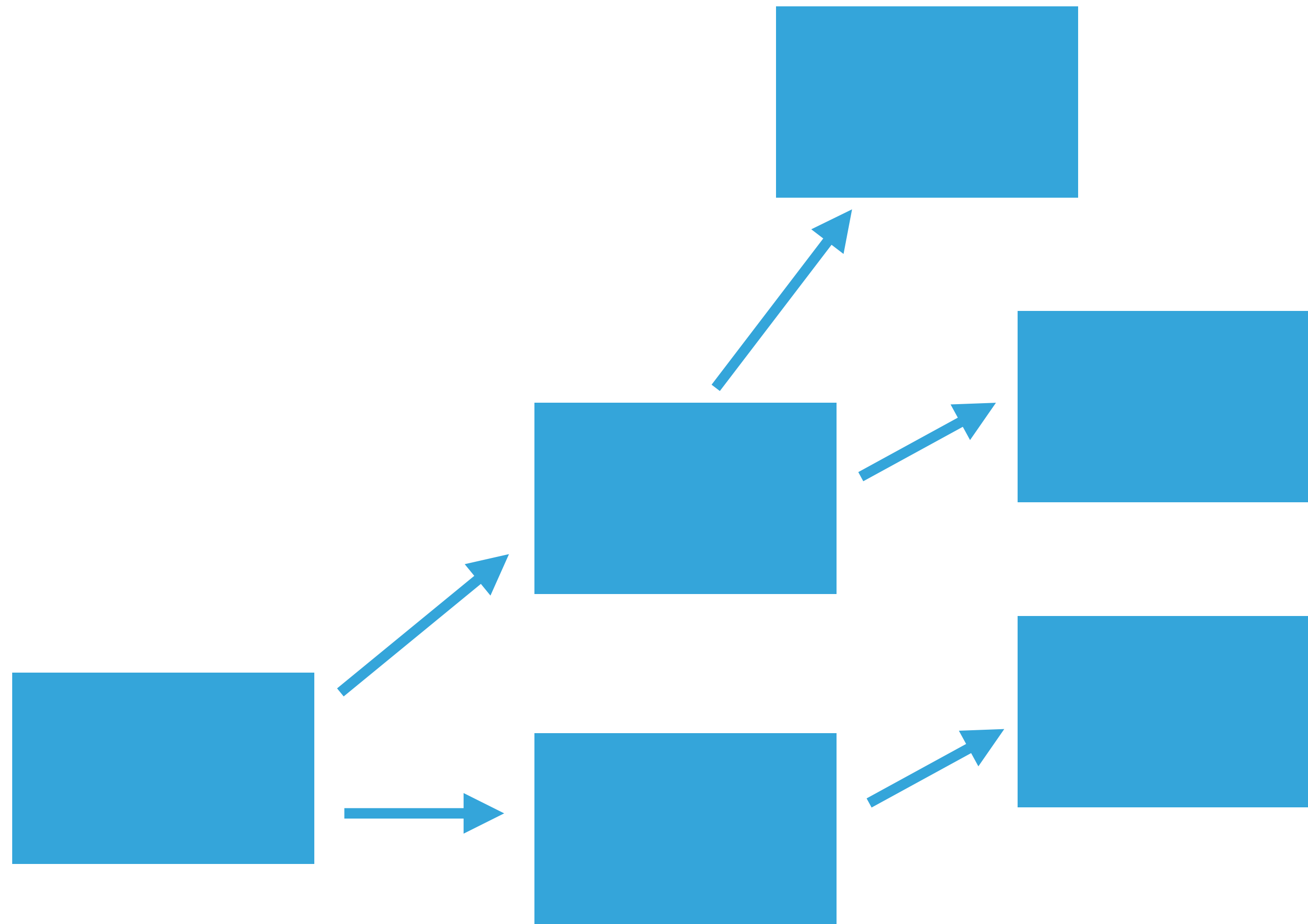
**#4: Prevent objects from being in invalid states**

**#5: Remove default handling**

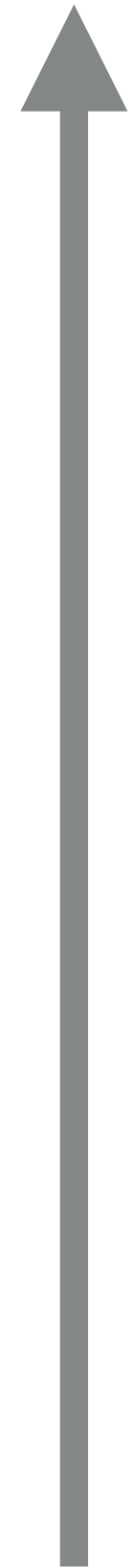
**#6: Assume impure functions**

**#7: Enforce architectural constraints**

# WILL DOING THIS MAKE CHANGE HARD?



- **Use this advice if project needs it**
- **If changes are made developers are shown possible bugs**
- **OK to alter past decisions**





Dave Liddament  
Lamp Bristol

Me when I'm not coding



Author of Static Analysis Results Baseline (SARB)  
and PHP Language Extensions Library

@daveliddament

**#Bonus: Check exceptions are handled**

**Checked**

**Unchecked**

```
function doSomething(): void {
```

```
    // Some code
```

```
    throw new MyException();
```

```
}
```

```
/** @throws MyException */  
function doSomething(): void {  
  
    // Some code  
    throw new MyException();  
}
```

```
function process(): void {
```

```
    doSomething();
```

```
}
```

```
/** @throws MyException */
```

```
function doSomething(): void {
```

```
    // Some code
```

```
    throw new MyException();
```

```
}
```

```
function process(): void {  
    try {  
        doSomething();  
    } catch (MyException) {  
        // process error  
    }  
}
```

```
/** @throws MyException */  
function doSomething(): void {  
    // Some code  
    throw new MyException();  
}
```

psalm.xml



# Psalm

```
<issueHandlers>
```

```
  <MissingThrowsDocblock>
    <errorLevel type="suppress">
      <directory name="tests/" />
    </errorLevel>
  </MissingThrowsDocblock>
```

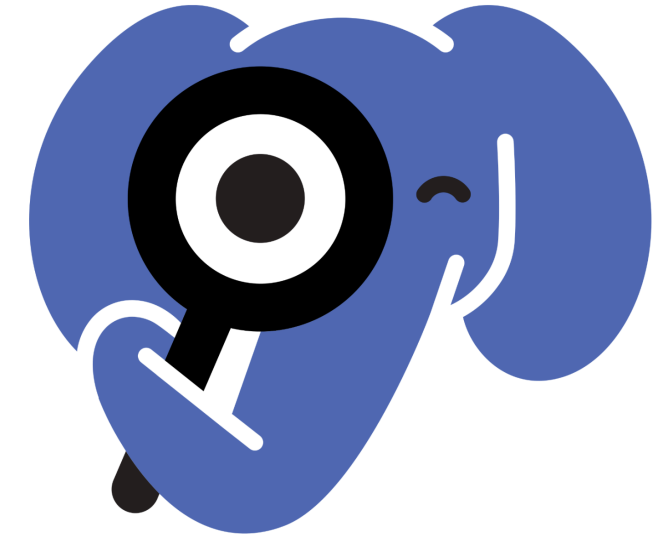
```
</issueHandlers>
```

```
<ignoreExceptions>
```

```
  <class name="Webmozart\Assert\InvalidArgumentException" />
  <class name="InvalidArgumentException" />
  <class name="LogicException" />
```

```
</ignoreExceptions>
```

**phpstan.neon**



**parameters:**

**exceptions:**

**check:**

**missingCheckedExceptionInThrows: true**

**tooWideThrowType: true**

**uncheckedExceptionClasses:**

**- InvalidArgumentException**