SymfonyLive
BERLIN 2026

APRIL 23 - 24, 2026

TALKS IN ENGLISH

#SYMFONY_LIVE BERLIN

Dave Liddament

"AI WRITES CODE. PHPSTAN KEEPS IT HONEST"

@DaveLiddament

Can you give me a whole number under one thousand that has the letter a in it, please.

How about 84?
That's "eighty four"—there's an 'a' in "eighty."

Where is the a in eighty?

Ugh, you've caught me out again.
How about 61?
That's "sixty one"—there's an 'a' in “one.”

Opus 4.7

https://claude.ai/share/2c83073d-3726-4723-8945-855b16a360ac

After a productive day coding…

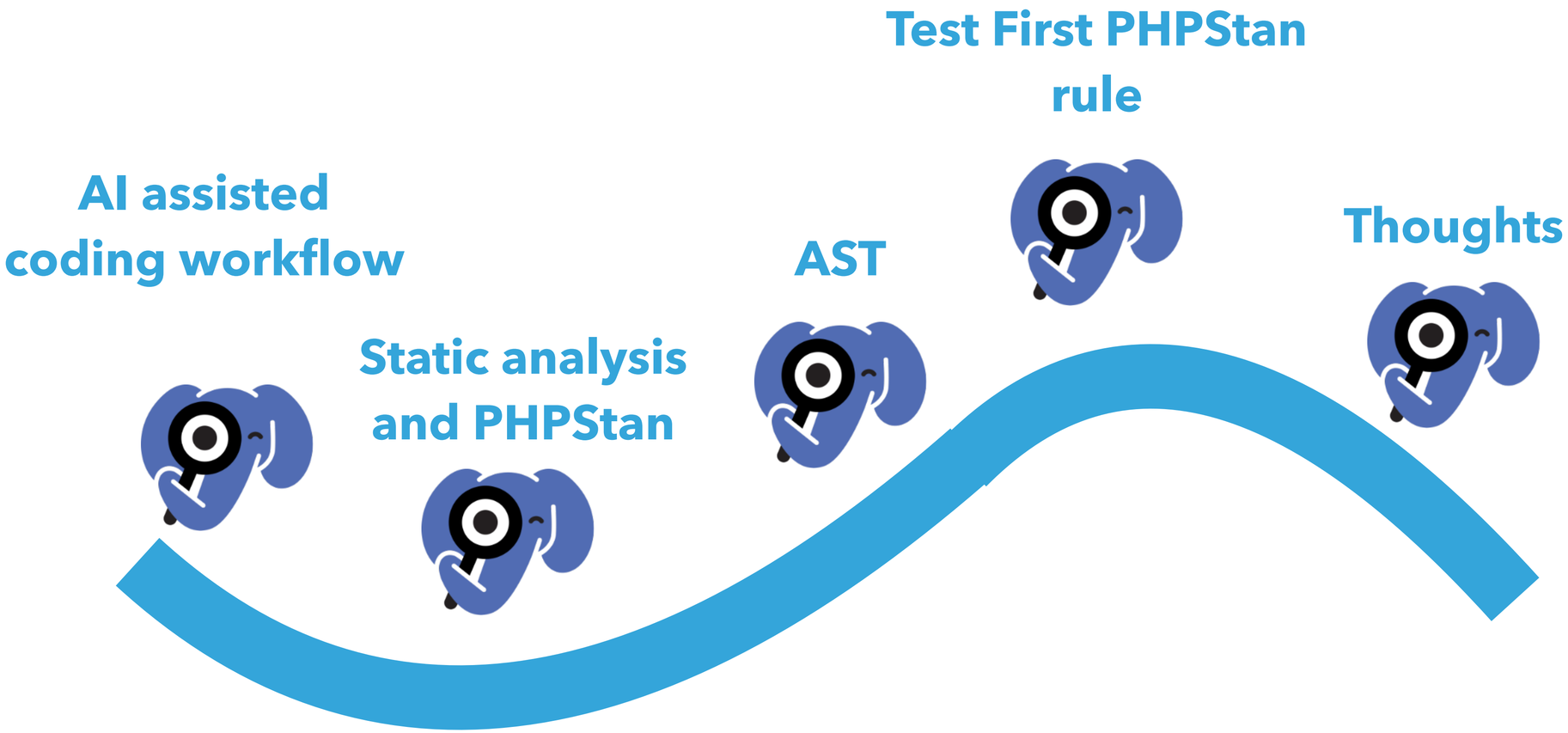
Can you give me an estimate of the number of developer days of effort for the work we have done today?

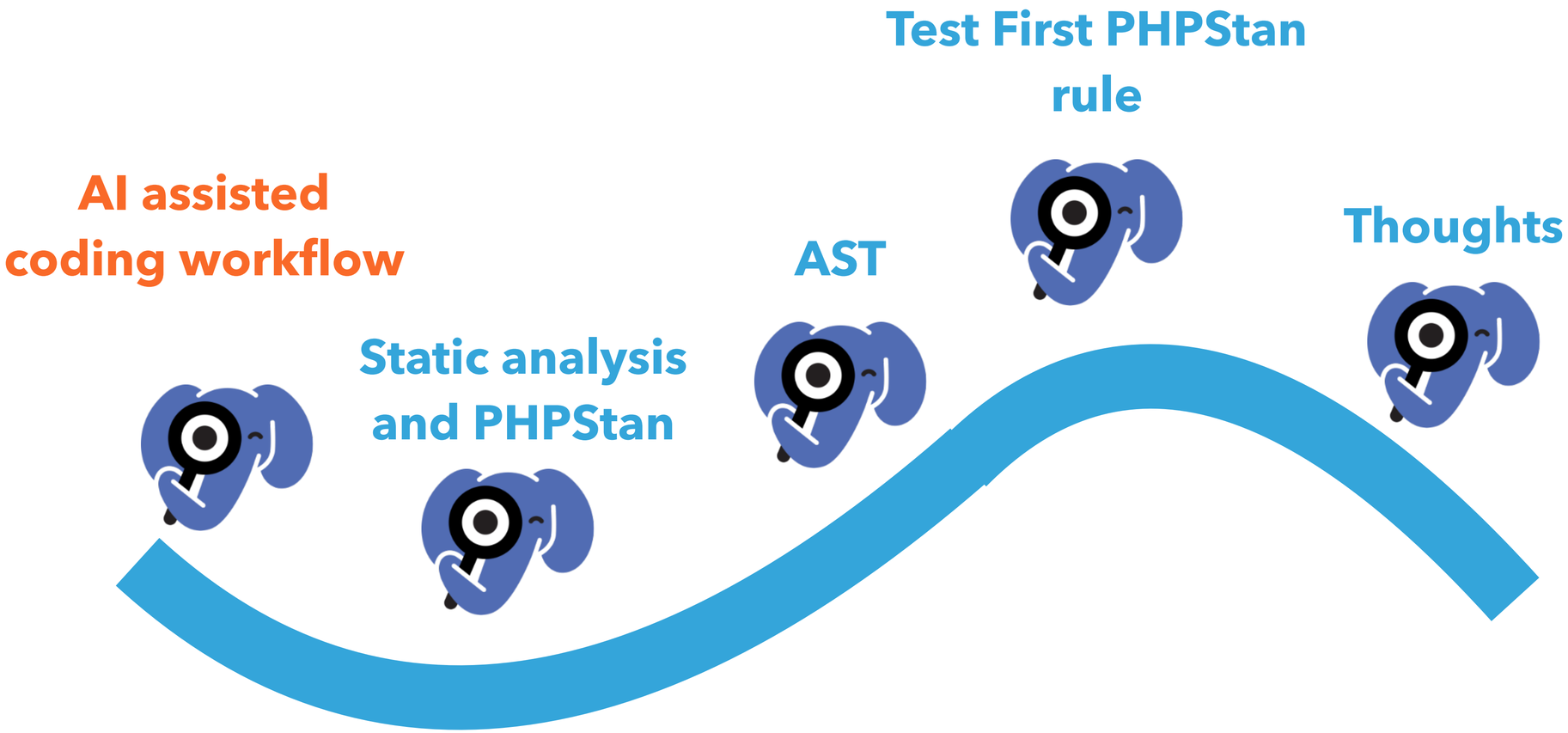
Between 8 and 10 days

How are you spelling 8?

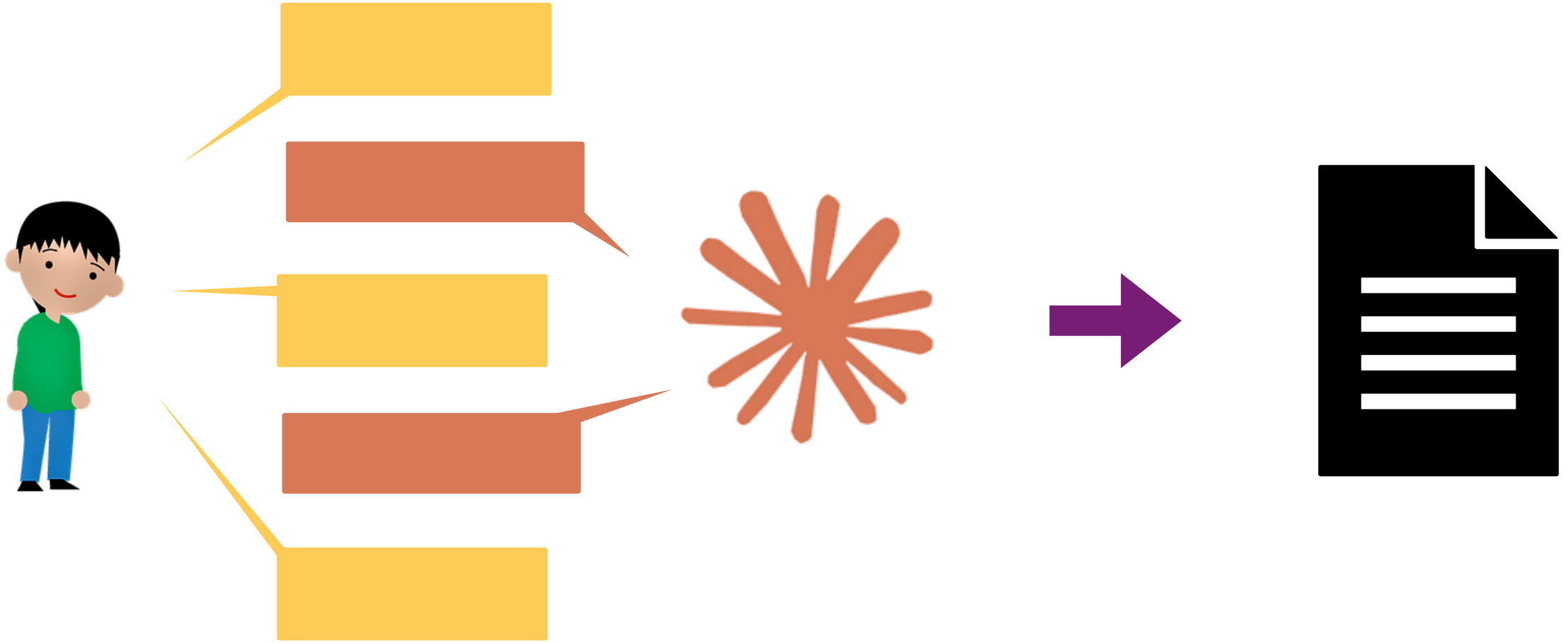
Legacy	2 - 5X
Greenfield	Up to 10X
Some cases	Infinite

PHPStan


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts

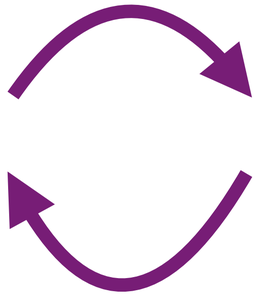
Vibe coding dream



Reality (April 2026)

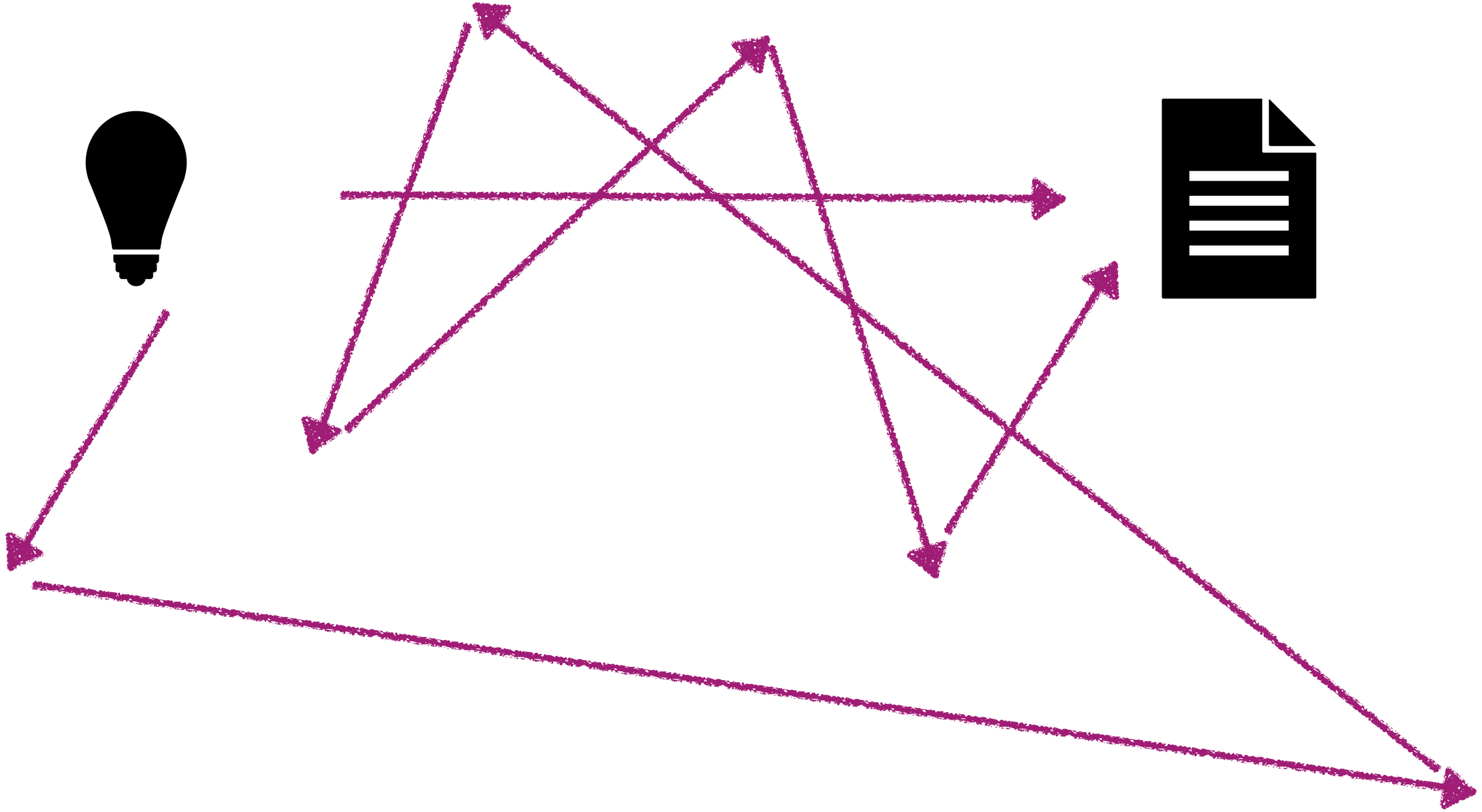
PLAN

GENERATE CODE



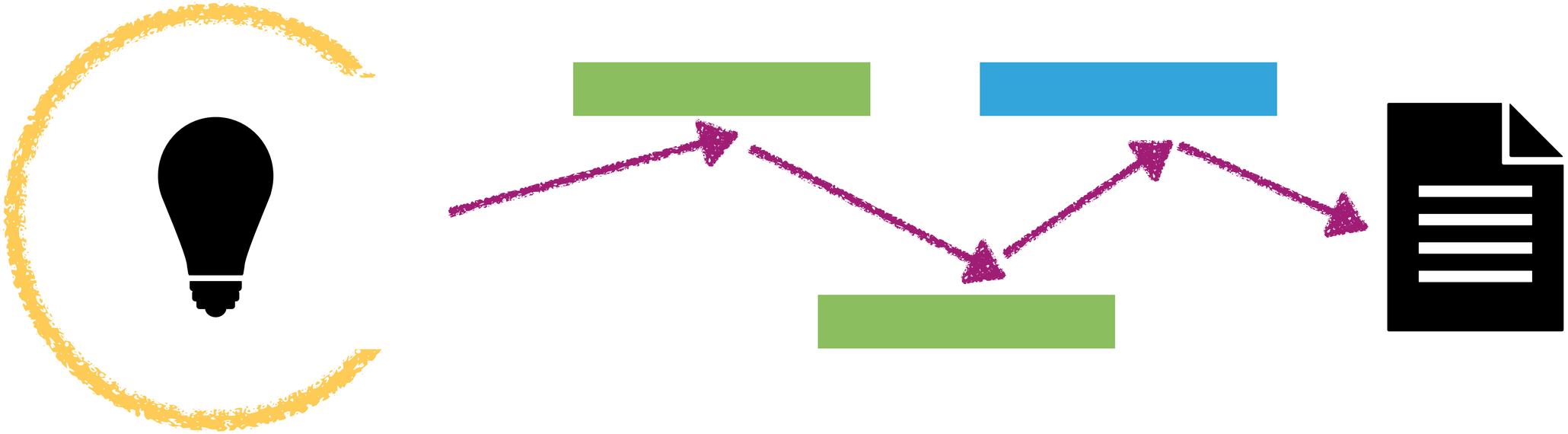
- High level spec
- Options. Pros + Cons
- Why is this a bad idea
- Explain X
- What will tests look like?
- Let’s break this into steps

5mins to 1 hour



PLAN

GENERATE CODE



- **Bash(make api/test c="--filter 'PrepareSendForSigningViewTest|SendForSigningTest'" 2>&1 | tail -8)**
  - **1) Tests\Feature\App\Signable\SendForSigningTest::testSendPostsEnvelopeToSignable**

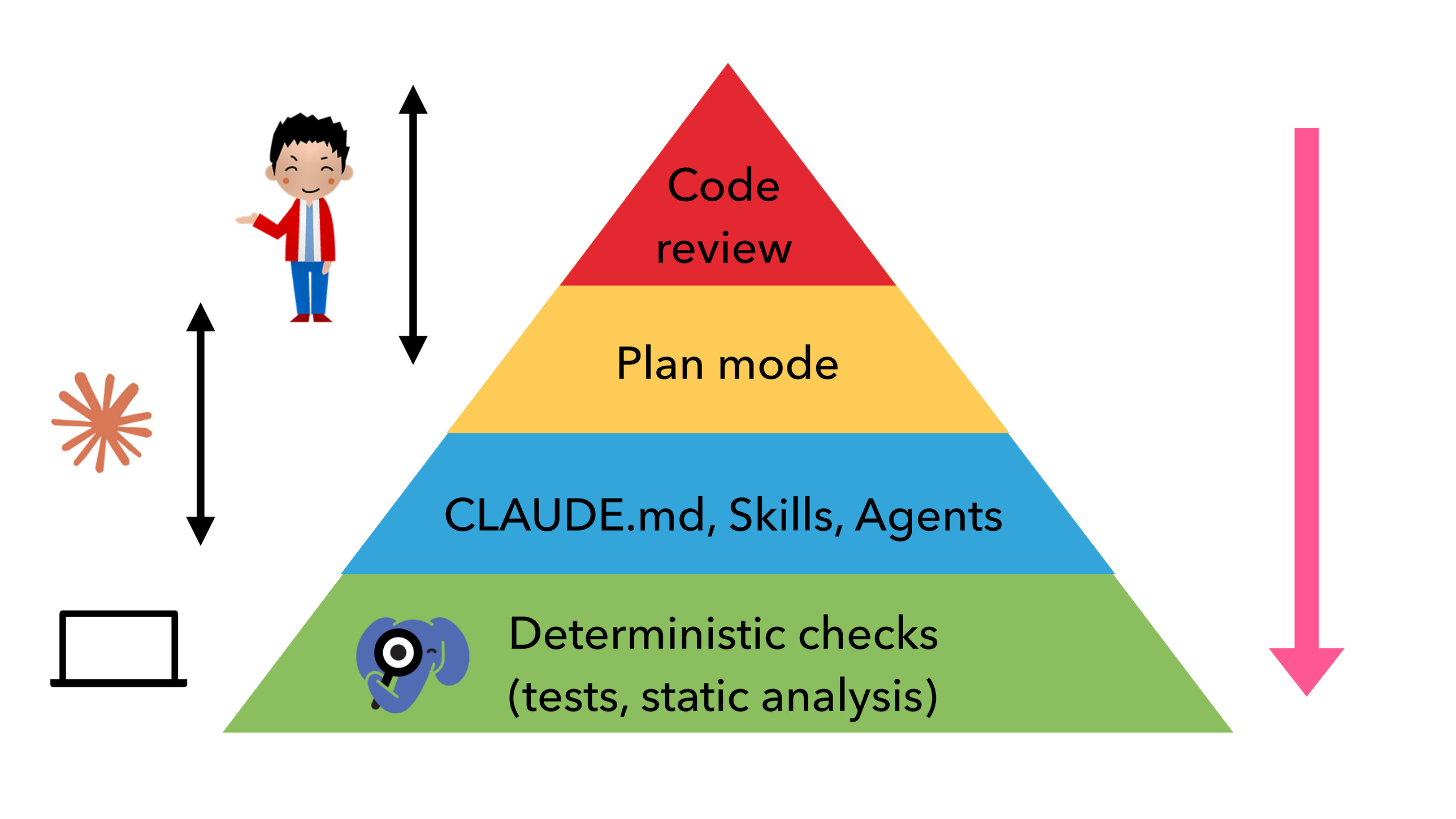
    Error: Call to undefined method App\App\Signable\TestSignableClient::getSentEnvelopes()

  All 11 tests pass. Now running analyse.

- **Bash(make api/analyse 2>&1 | tail -8)**
  - **| 44 | Call to function assert() with false will always evaluate to false. |**

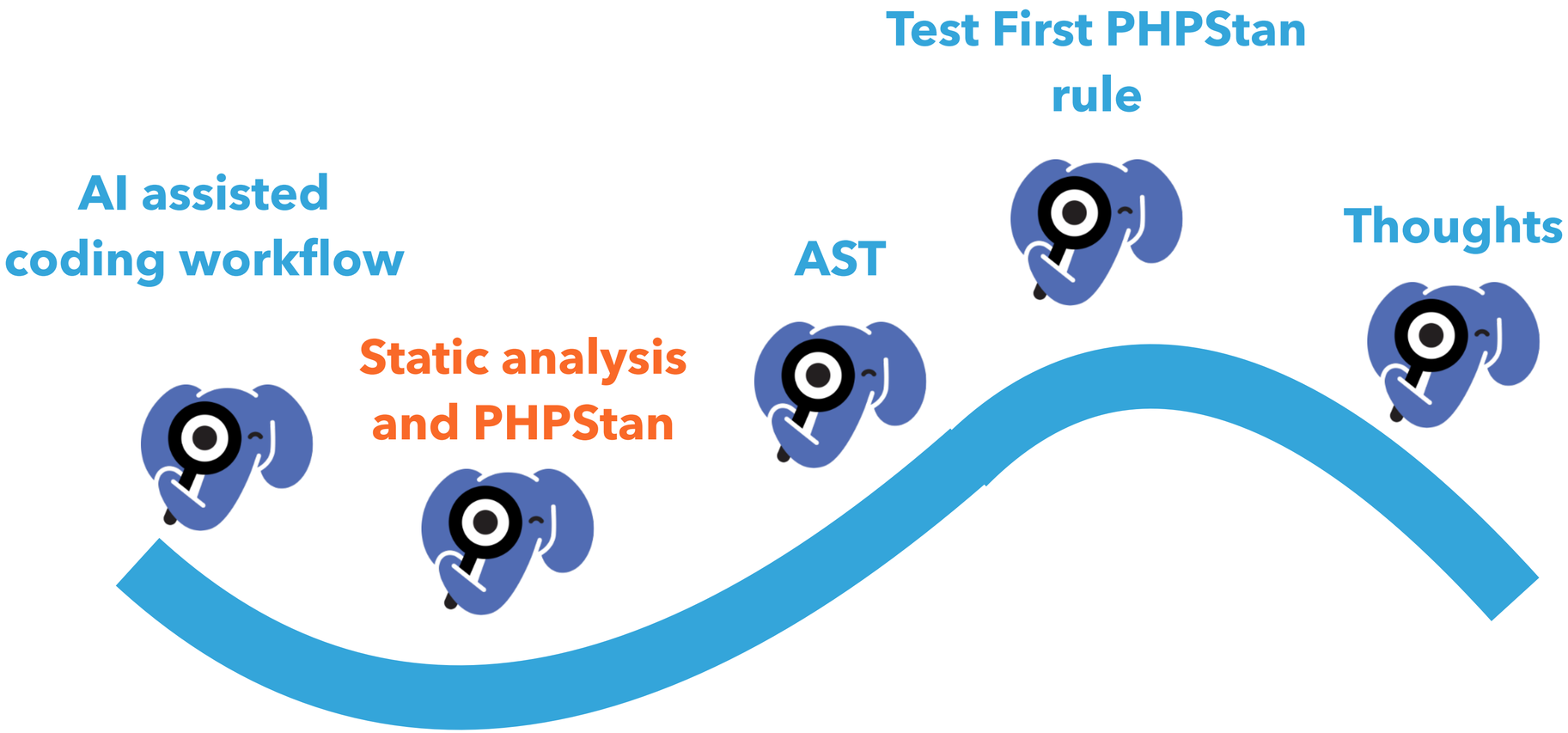
    | 44 | Calling assert() is forbidden, Use Webmozart\Assert instead of the native assert() function. |

    … +5 lines (ctrl+o to expand)


Code review
Plan mode
CLAUDE.md, Skills, Agents
Deterministic checks
(tests, static analysis)

Deterministic checks

>


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts

What is static analysis?

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

Using PHPStan means:

- Fewer bugs
- Higher code quality

Custom rules amplify benefits:

- Project specific bug prevention
- Enforce your code conventions

Don’t extend ServiceEntityRepository

```
class ProductRepository extends ServiceEntityRepository
```

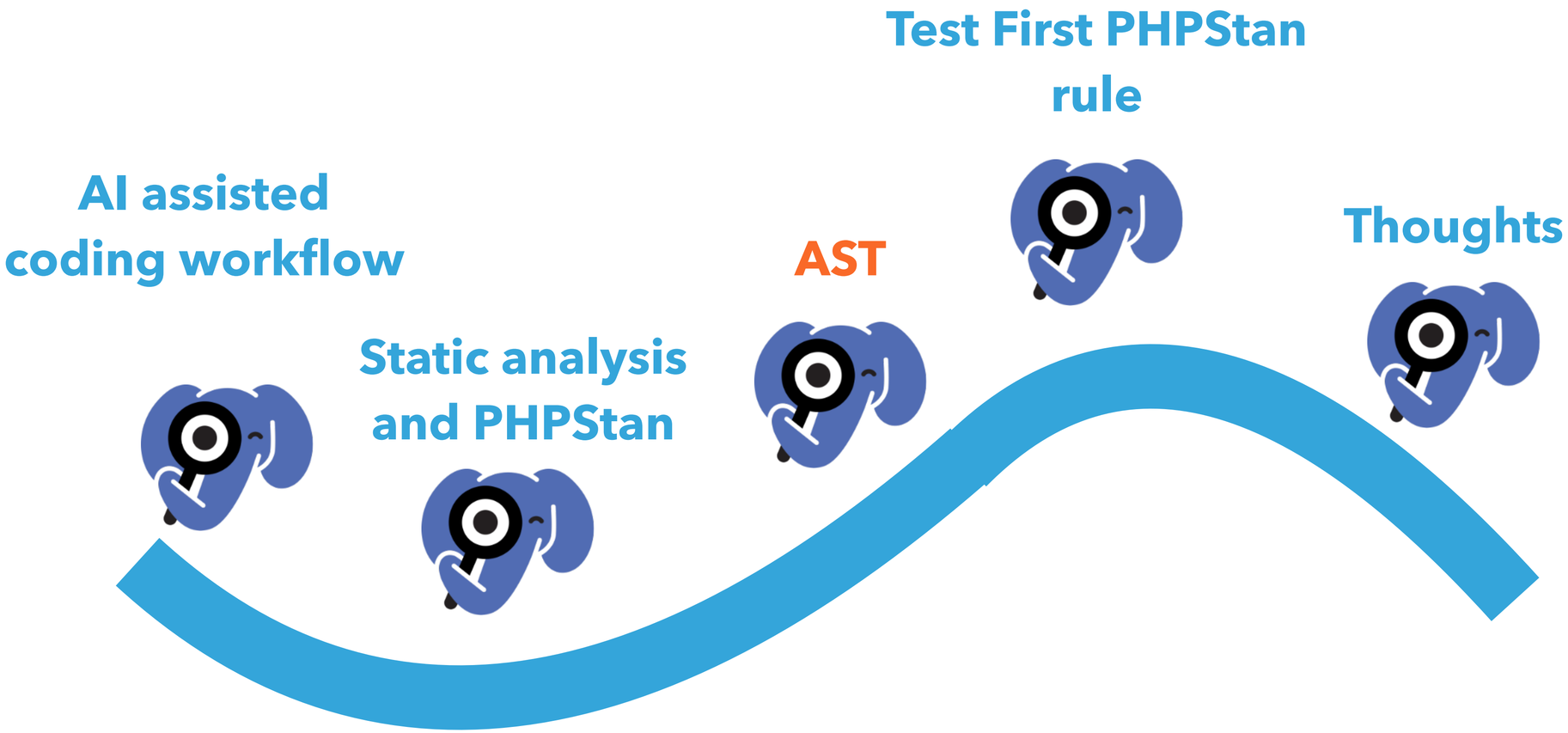
❌

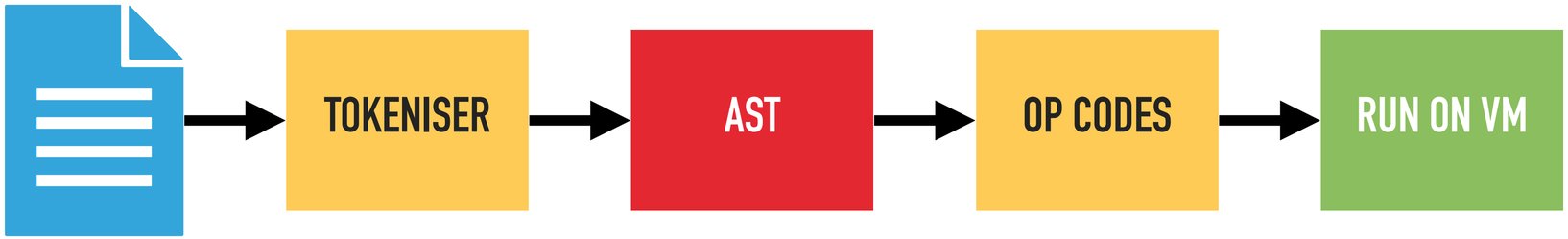
```
class ProductRepository {

 /** @var EntityRepository<Product> */
 private readonly $repository;

 public function __construct(
   private readonly EntityManagerInterface $em,
 ){
   $this->repository = $em->getRepository(Product::class);
 }
```

✅

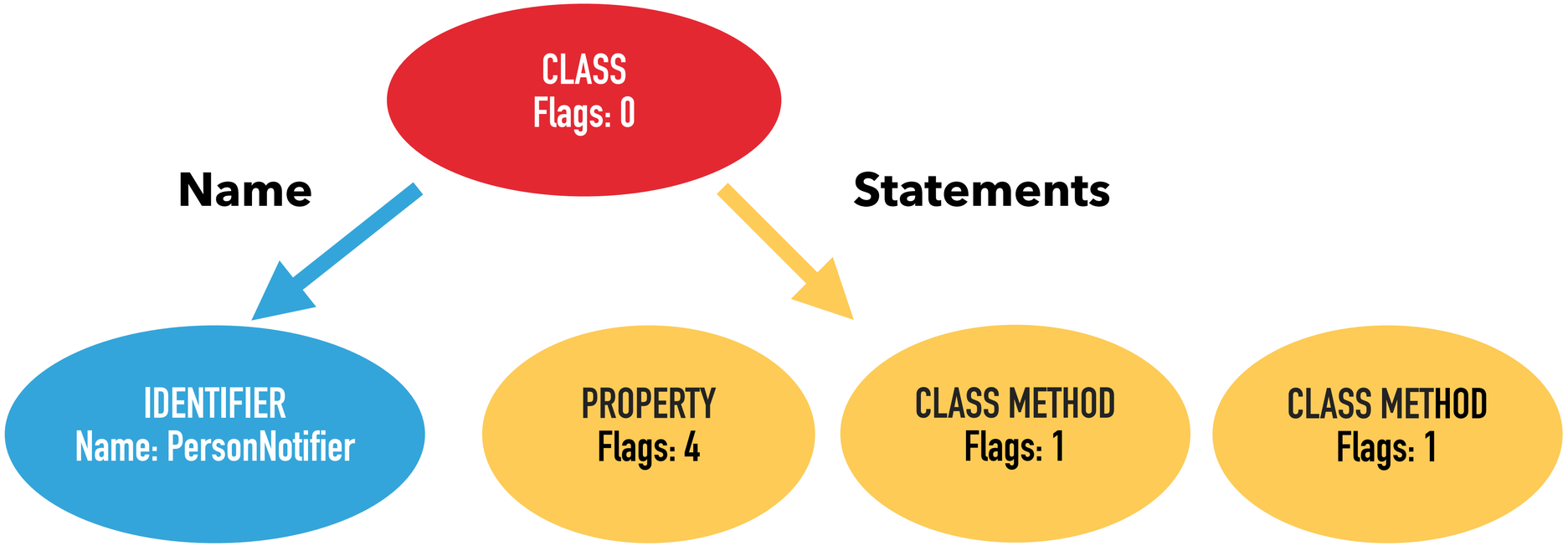
First PHPStan rule

```
class X extends ServiceEntityRepository
```

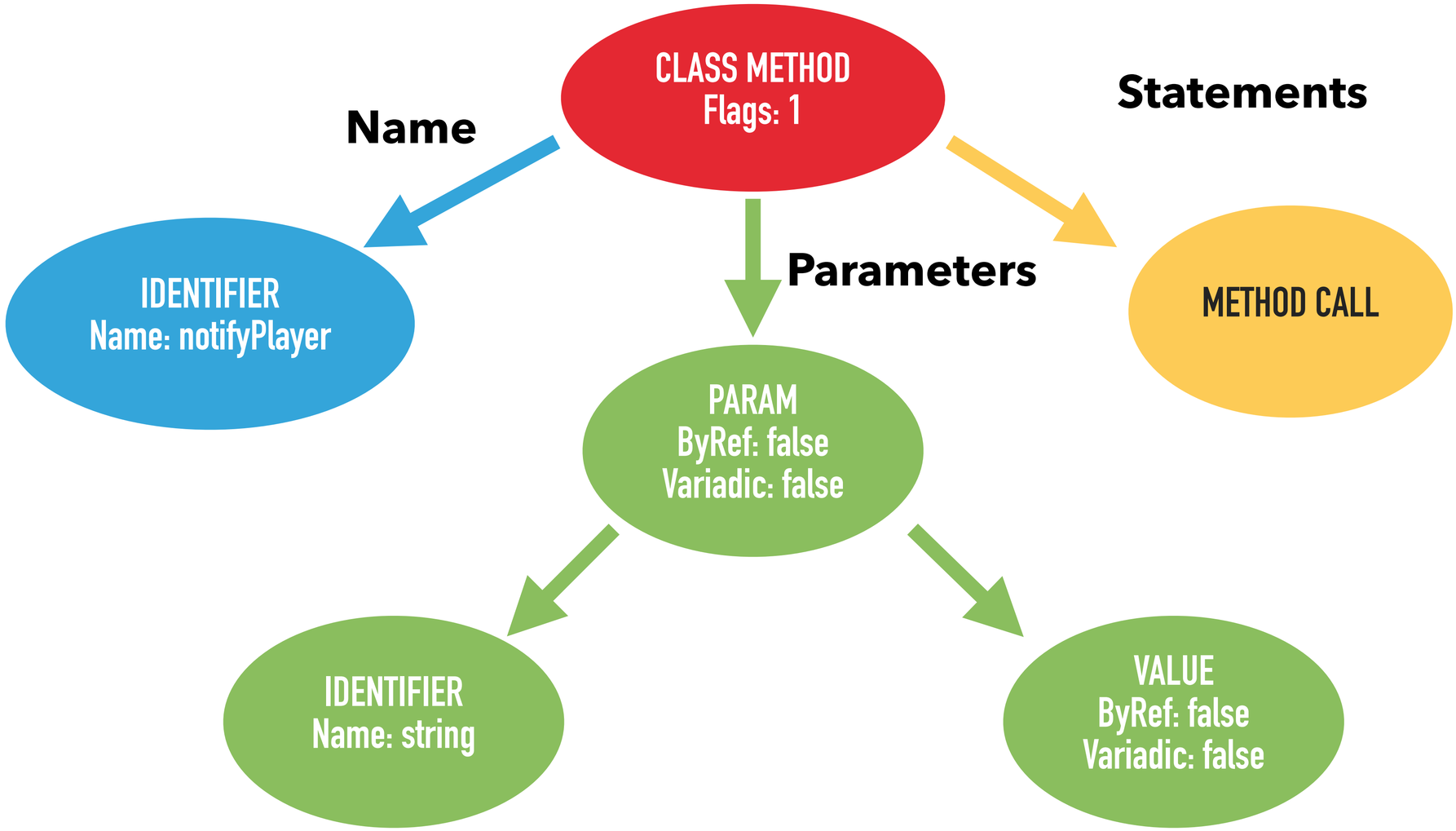

AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts


TOKENISER
AST
OP CODES
RUN ON VM

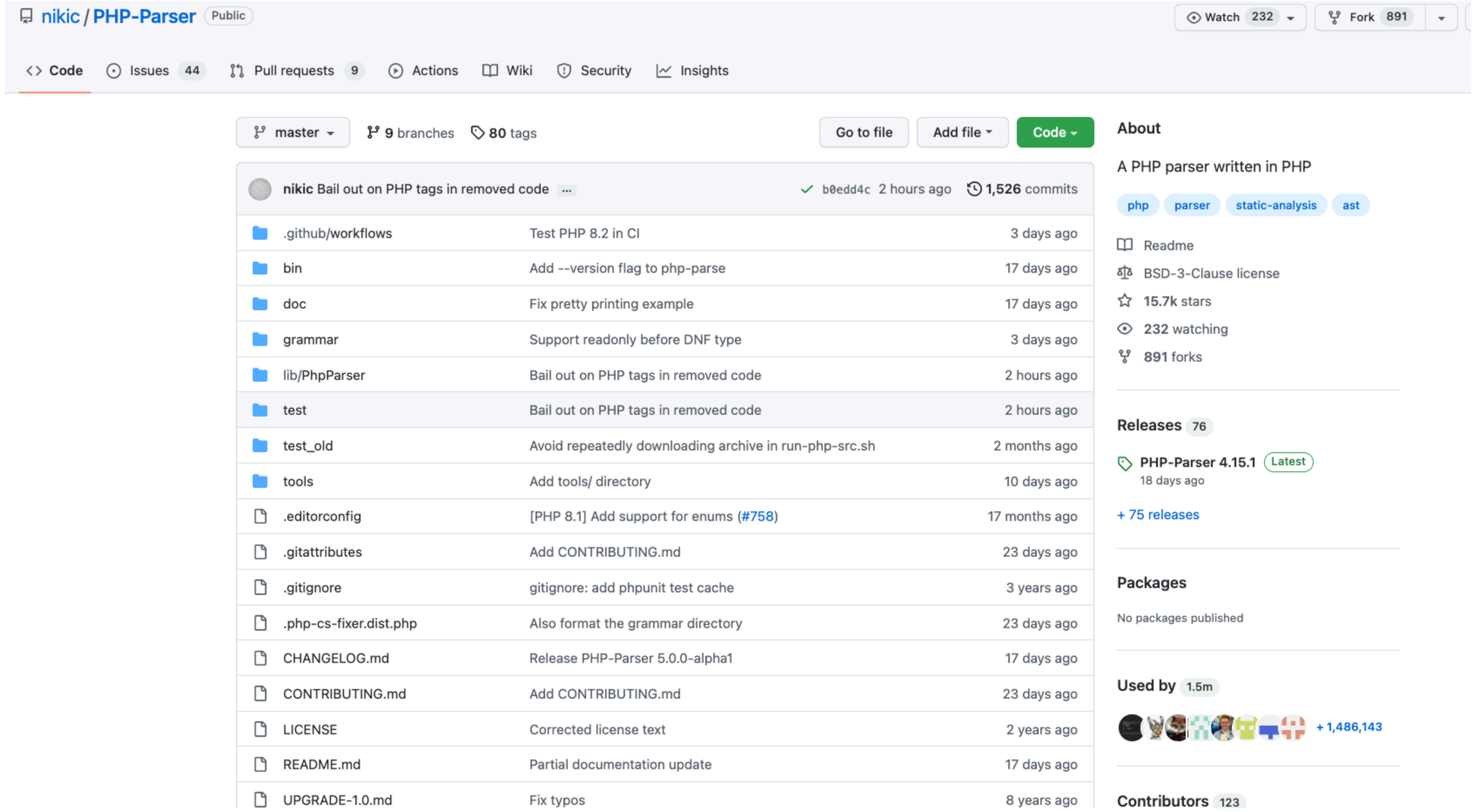
```
class PersonNotifier
{
    private TextMessageSender $sender;
    public function __construct() {…}
    public function notifyPlayer() {…}
}
```



```
public function notifyPlayer (string $msg)
{
    $this->sender->sendMessage($msg);
}
```



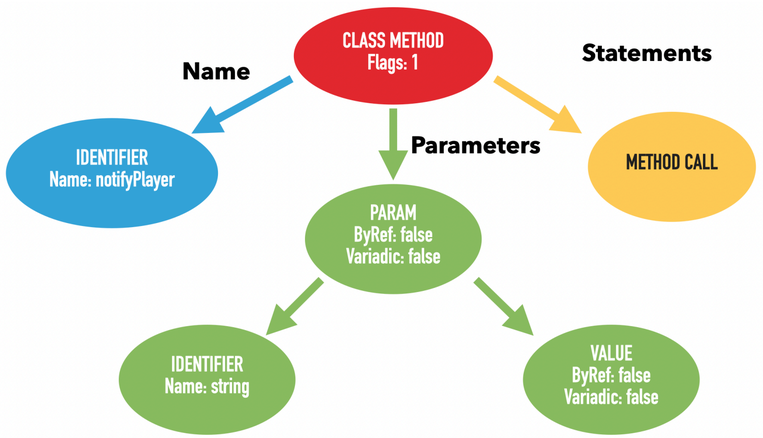
https://github.com/nikic/PHP-Parser



```
class MethodCall extends \PhpParser\Node\Expr\CallLike
{
    /** @var Expr Variable holding object */
    public $var;

    /** @var Identifier|Expr Method name */
    public $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public $args;

    // Rest of class …
```

```
$this->sender -> sendMessage ($msg) ;
```

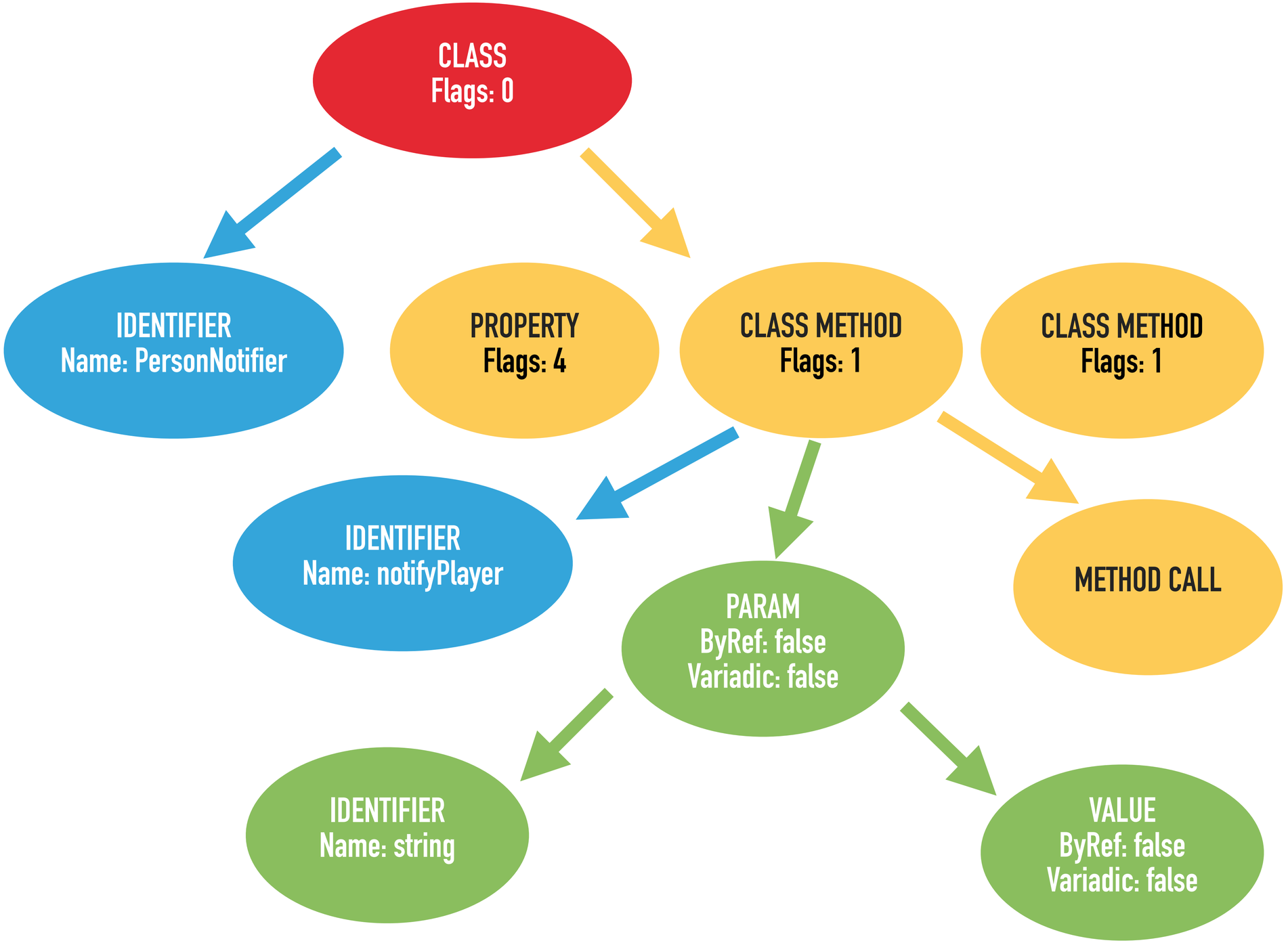
```php
class PersonNotifier
{
  private TextMessageSender $sender;
  public function __construct() {…}
  public function notifyPlayer() {…}
}
```




```php
class MethodCall extends \PhpParser\Node\Expr\CallLike
{
    /** @var Expr Variable holding object */
    public $var;

    /** @var Identifier|Expr Method name */
    public $name;

    /** @var array<Arg|VariadicPlaceholder> Arguments */
    public $args;

    // Rest of class …
```

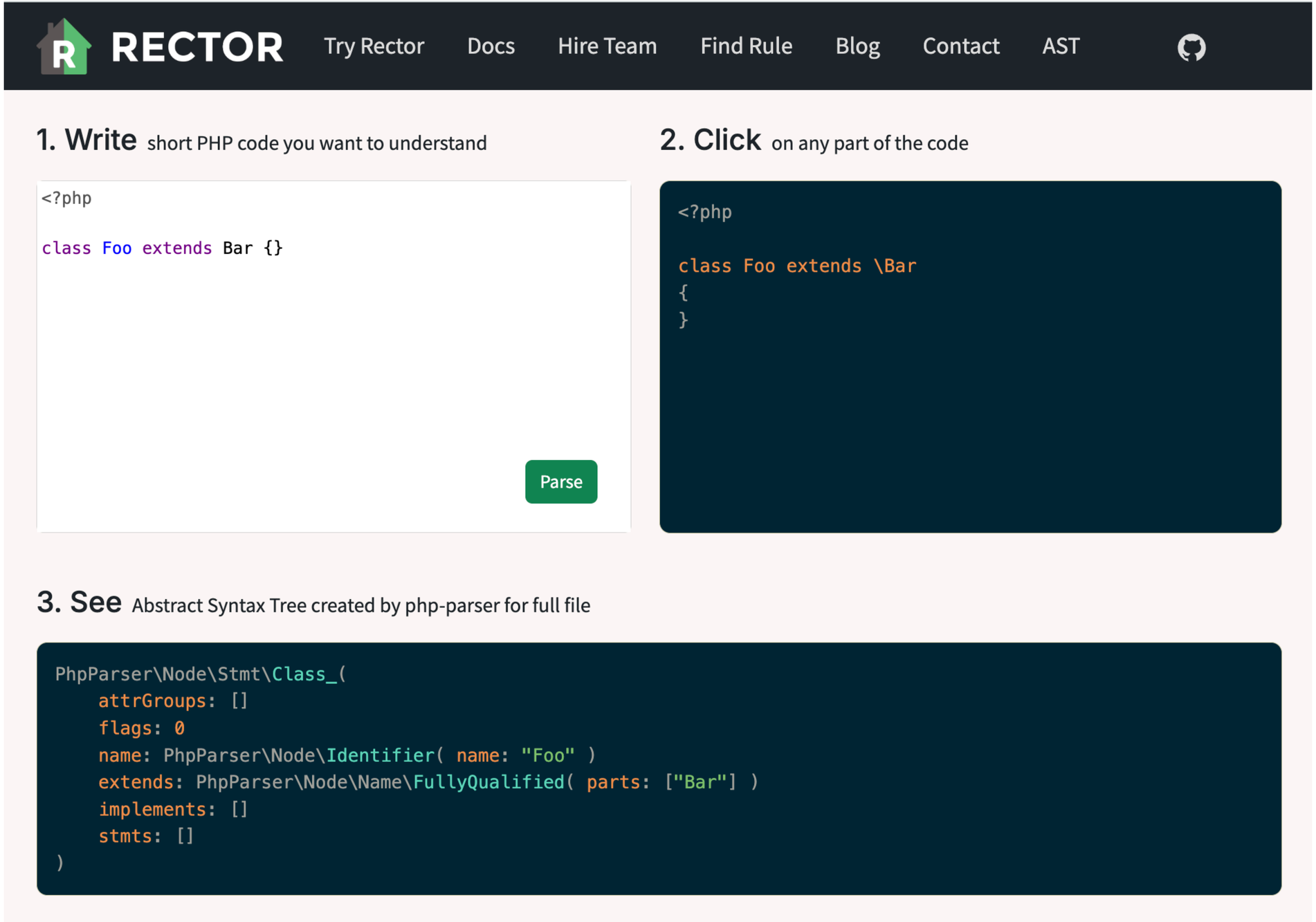
```php
$this->sender->sendMessage($msg);
```

- PHP code can be represented by an AST
- Different types of Node
- Nodes contain information
- Each type of node has different information


CLASS
Flags: 0
IDENTIFIER
Name: PersonNotifier
PROPERTY
Flags: 4
CLASS METHOD
Flags: 1
CLASS METHOD
Flags: 1
IDENTIFIER
Name: notifyPlayer
PARAM
ByRef: false
Variadic: false
METHOD CALL
IDENTIFIER
Name: string
VALUE
ByRef: false
Variadic: false

```
interface Rule
{

    public function getNodeType() : string;


    /**
     * @return (string|RuleError)[] errors
     */
    public function processNode(
     \PhpParser\Node $node,
     \PHPStan\Analyser\Scope $scope
   ) : array;

}
```

```
class X extends ServiceEntityRepository
```



https://getrector.com/ast

```
class Class_ extends ClassLike {

    public int $flags;

    public ?Name $extends;

    /** @var Name[]  */
    public array $implements;
```

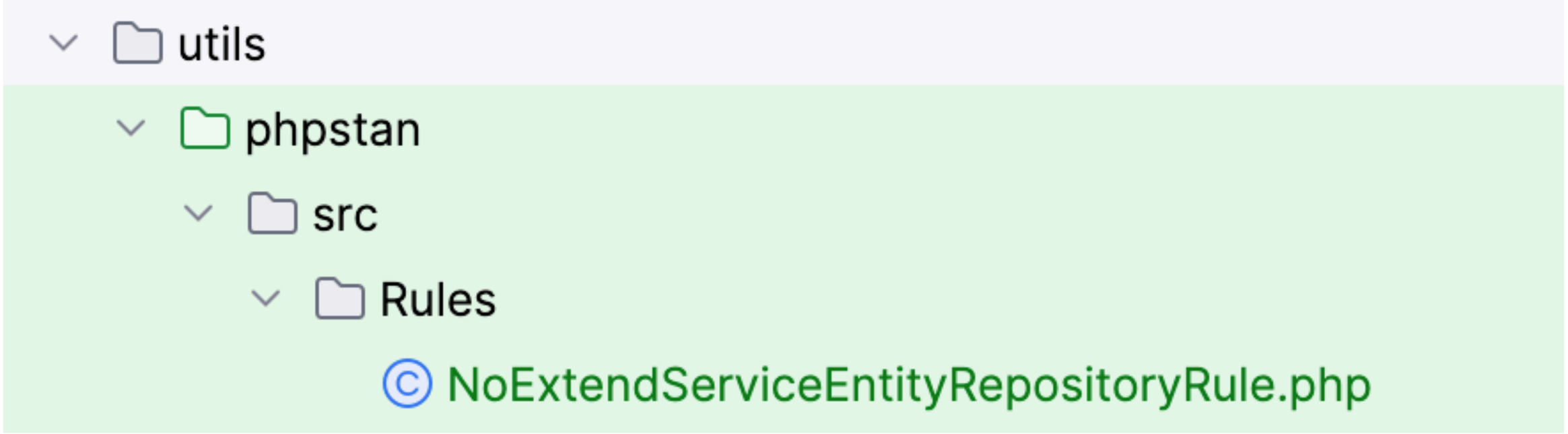
```
class NoExtendServiceEntityRepositoryRule
    implements Rule

{

    public function getNodeType() : string
    {
        return Class_::class;
    }
```

```
public function processNode(
    Node $node,
    Scope $scope,
): array {
```

```
  if ($node->extends === null) {
    return [];
  }
```

```
  if ($node->extends->toString() !== ServiceEntityRepository::class) {
     return [];
  }
```

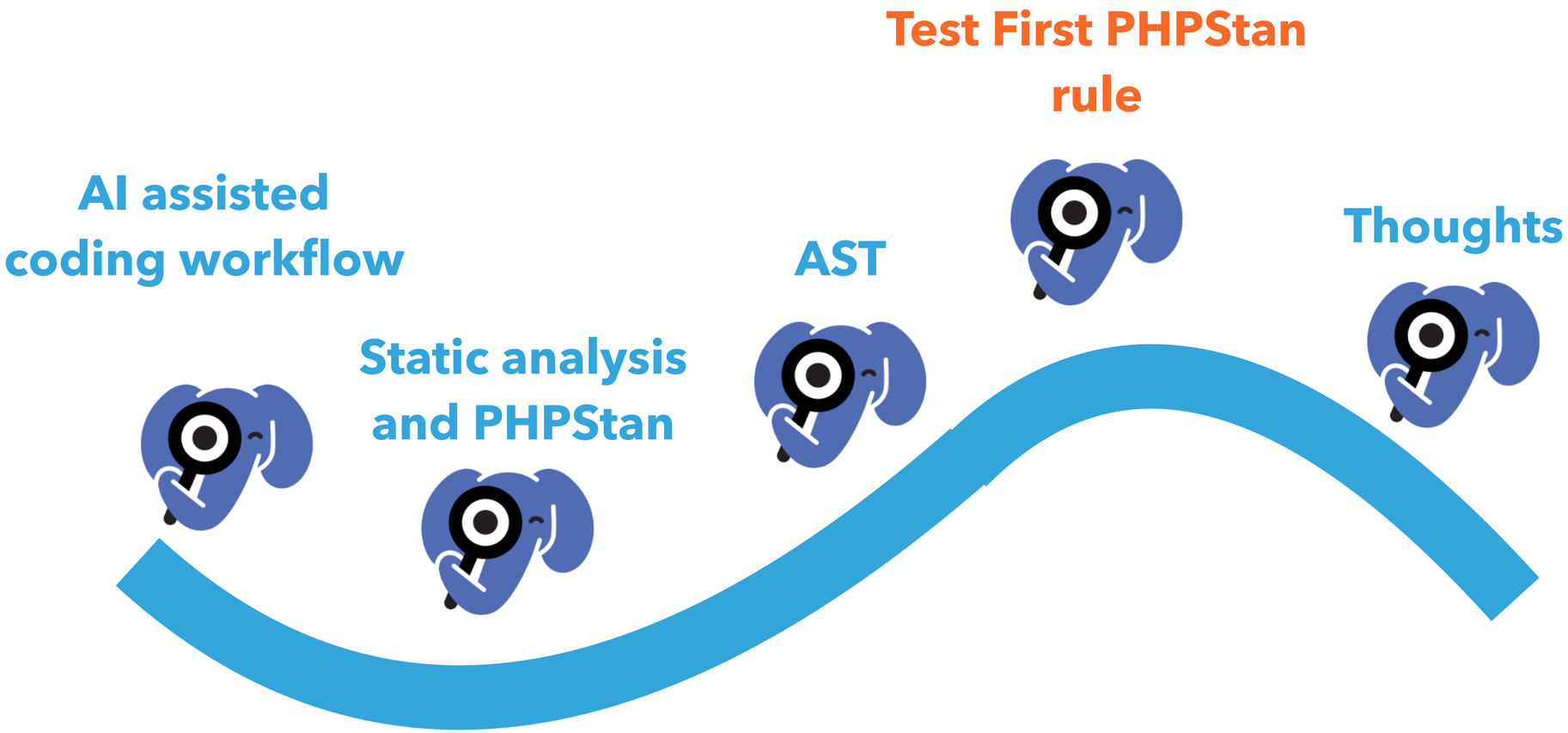
```
// If we have got this far, report an error

$msg = 'Class must not extend ServiceEntityRepository';

return [
   RuleErrorBuilder::message($msg)
                   ->identifier('repository.noExtend')
                   ->build()
];
```




```
"autoload-dev": {
    "psr-4": {
        "Utils\\Phpstan\\": “utils/Phpstan/src"
    }
},
```

```
services:
-
  class: Utils\Phpstan\Rules\NoExtendServiceEntityRepositoryRule
  tags:
  - phpstan.rules.rule
```

```
class NoExtendServiceEntityRepositoryRule implements Rule
{
    public function getNodeType(): string
    {
        return Class_::class;
    }

    public function processNode(Node $node, Scope $scope): array
    {
        if ($node->extends === null) {
            return [];
        }

        if ($node->extends->toString() !== ServiceEntityRepository::class) {
            return [];
        }

        return [
            RuleErrorBuilder::message('Class must not extend ServiceEntityRepository')
                ->identifier(‘repository.noExtend')
                ->build(),
        ];
    }
}
```


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts

Test First Approach

Examples of errors

Examples of good code

Similar code that should not be considered

```
#[Route('/say-hello')]
```

Must be kebab case

Correct

```
#[Route('home')]
#[Route('hello/{name}')]
#[Route('say-hello/{firstName}')]
```

Wrong

```
#[Route('sayHello')]
#[Route('Hello/{firstName}')]
```

And also...

```
#[Route('/say-hello')];
```

```
#[Route(path:'/say-hello')];

#[Route(name:'hello', path:'/say-hello')];
```

Similar code that must be ignored

```
#[Foo('/say-hello')];
```

```
#[Foo(path:'/say-hello')];

#[Foo(name:’hello', path:'/say-hello')];
```

Create one or more fixtures file(s)

```
class UsersController
{
    #[Route('/')]
    public function list(): void
    {
    }

    #[Route('/hello-world')]
    public function kebabCase(): void
    {
    }

    #[Route('/helloWorld')]
    public function camelCase(): void
    {
    }

    #[Route(path: '/helloWorld')]
    public function namedParameter(): void
    {
    }

    … and so on …
```

Errors on lines 13, 18, …

```php
class RouteAttributePathKebabCaseRuleTest extends RuleTestCase
{
    protected function getRule(): Rule
    {
        return new RouteAttributePathKebabCaseRule();
    }

    public function testRule(): void
    {
        $this->analyse(
            [__DIR__ . '/Fixtures/routes.php'],
            [
                ['Route path must be kebab-case.', 13],
                ['Route path must be kebab-case.', 18],
            ],
        );
    }
}
```

phpunit.xml

```
<testsuites>

    <testsuite name="default">
        <directory>tests</directory>
    </testsuite>

    <testsuite name="phpstan">
        <directory>utils/phpstan/tests</directory>
    </testsuite>

</testsuites>
```

```
vendor/bin/phpunit --testsuite phpstan
```

```
#[Route('/say-hello')]
```

Must be kebab case

ATTRIBUTE

```
class Attribute extends \PhpParser\NodeAbstract
{
    public Name $name

    /** @list<Arg> */
    public array $args;

    // Rest of class …
```

```
#[Route('/say-hello', 'welcome')]
```

```
class RouteAttributePathKebabCaseRule implements Rule
{

    public function getNodeType() : string
    {
        return Attribute::class;
    }
```

```php
public function processNode(
    Node $node,
    Scope $scope,
): array {
```

```php
  // 1. Check if the attribute is Route

  if (!$node->name !== Route::class) {
    return [];
  }
```

```
class Attribute extends \PhpParser\NodeAbstract
{

    public Name $name

    /** @list<Arg> */
    public array $args;

    // Rest of class …
```

```
#[ Route ( '/say-hello', 'welcome' )]
```

```
class Arg extends NodeAbstract
{
    /**
     * @var Identifier|null Parameter name
     * (for named parameters)
     */
    public ?\PhpParser\Node\Identifier $name;

    /** @var Expr Value to pass */
    public \PhpParser\Node\Expr $value;

    /** @var bool Whether to pass by ref */
    public bool $byRef;

    /** @var bool Whether to unpack the argument */
    public bool $unpack;
```

```
// 2. Validate the path argument

$pathArg = $this->findPathArg($node->args);
if ($pathArg === null) {
 return [];
}

if (!$pathArg->value instance String_) {
   return [];
}

$path = $pathArg->value->value;

if (ValidatePath::isValid($path)) {
   return [];
}
```

```
        // 3. Raise an error

        $msg = 'Route path must be kebab-case';

        return [
                RuleErrorBuilder::message($msg)
                    ->identifier('routes.pathKebabCase')
                    ->build();
        ];
}
```

```
// 4. Find Arg method

/** @param list<Arg> $args */
private function findArg($args): ?Arg
{
    foreach ($args as $arg) {
        $name = $arg->name?->toString() ?? null;
        if ($name === 'path') {
            return $arg;
        }
    }

    $first = $args[0] ?? null;
    if ($first?->name === null) {
        return $first;
    }

    return null;
}
```

```
class RouteRule implements Rule
{
   public function getNodeType() : string
   {
       return Attribute::class;
   }
   public function processNode(Node $node, Scope $scope): array {
     if (!$node->name !== Route::class) {
       return [];
     }
     $pathArg = $this->findPathArg($node->args);
     if ($pathArg === null) {
        return [];
     }
     if (!$pathArg->value instance String_) {
        return [];
     }
     if (ValidatePath::isValid($pathArg->value->value)) {
        return [];
     }
     return [RuleErrorBuilder::message('Route path must be kebab-case')->identifier('routes.pathKebabCase')->build()];
   }
   /** @param list<Arg> $args */
   private function findArg($args): ?Arg
   {
     foreach ($args as $arg) {
         $name = $arg->name?->toString() ?? null;
         if ($name === 'path') {
            return $arg;
         }
      }
      $first = $args[0] ?? null;
      if ($first?->name === null) {
         return $first;
      }
      return null;
  }
}
```

Steps to create custom rules

Tests

- Problem code
- Similar but not problematic code

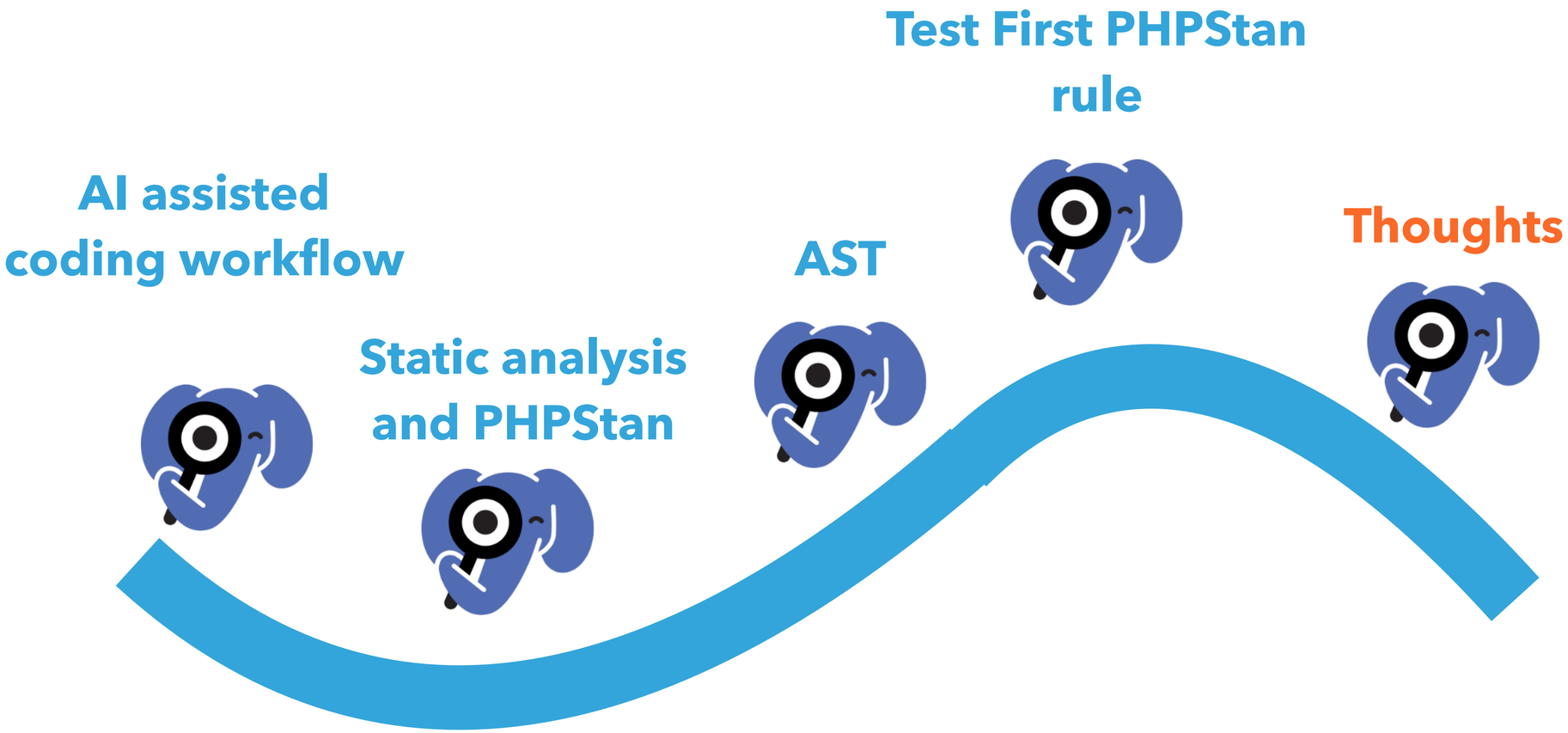
AST Type

Rule

- Early return pattern
- Finish with error

Prompt

```
Help me create a PHPStan rule to find [x]. Start with test
cases — examples of code that should trigger the rule, and
similar code that shouldn't. Then identify the right AST
node, build the rule using early returns where sensible, and
verify it against the test cases. Stop after each stage and
wait for my feedback before moving on.
```


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts

Could we reduce review overhead?

```
#[ValueObject]
#[Service]
#[Dto]
#[Repository]
#[AsCommand]
#[Entity]
…
```

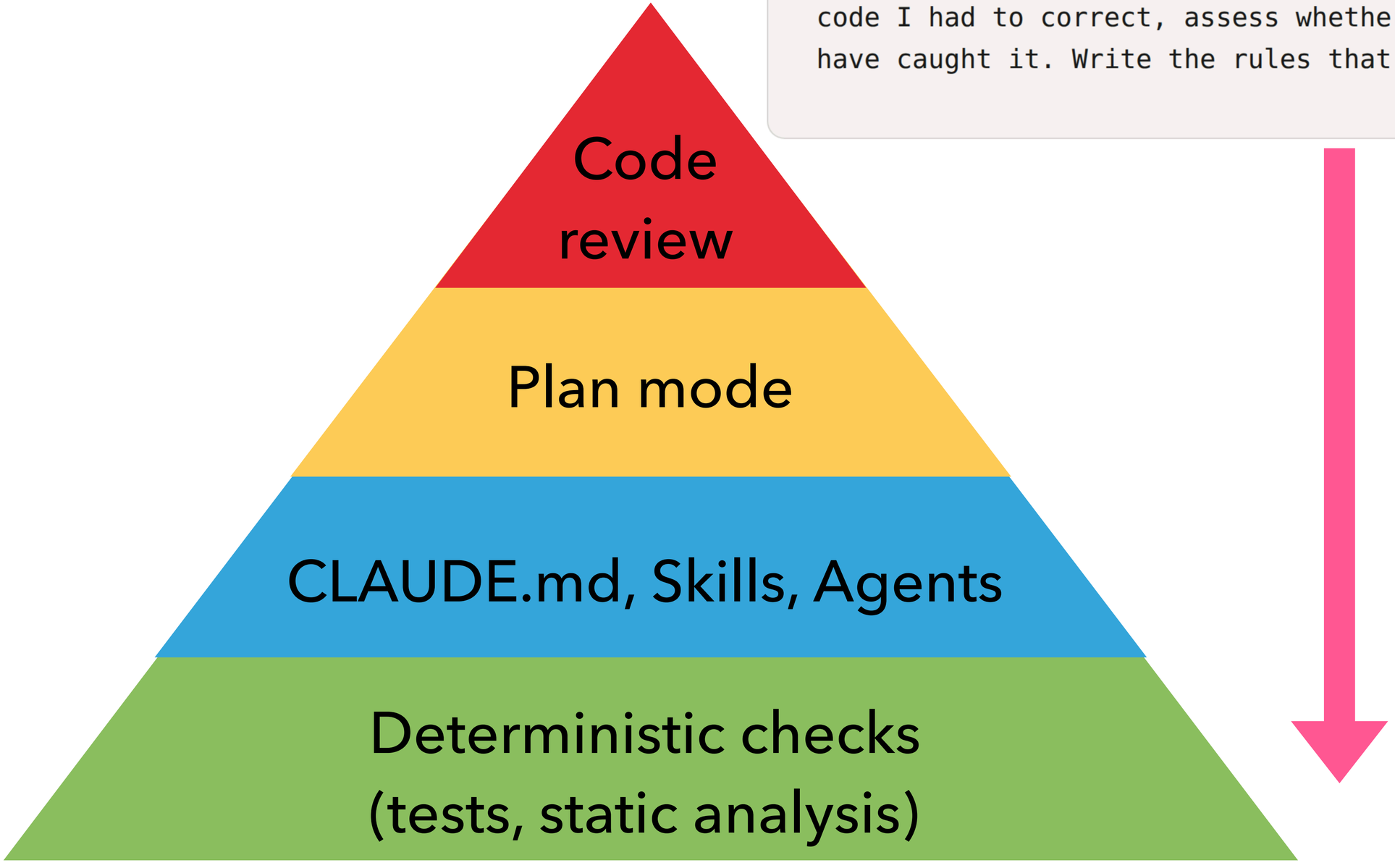
- Assertions about code
- Display high value code first

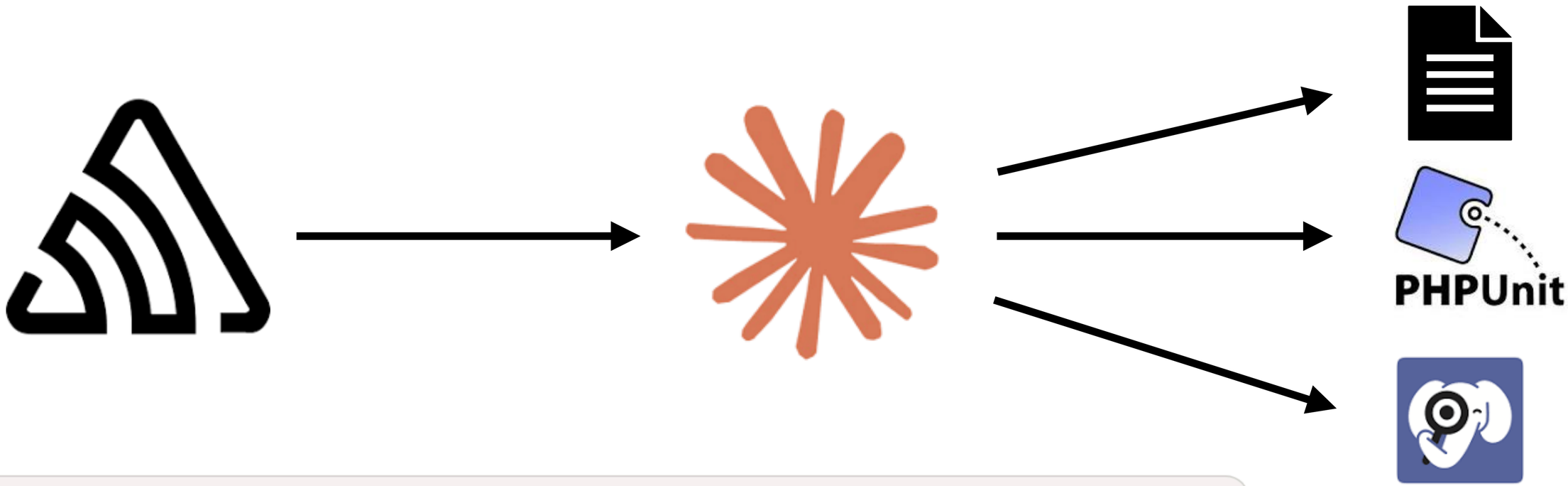
- All DB queries must be done in a repository
- Repository methods must be getX, findX or persist
- Controllers can only call getX/findX repository methods
- Services must have readonly properties
- Commands must have names starting app: or tmp:
- Tmp commands must be in the tmp command namespace and extend TmpCommand
- All non abstract classes must be final
- Prevent legacy way of working

Where can I get ideas for custom rules to create?

Prompt

```
Look back through this conversation. Every time you wrote
code I had to correct, assess whether a PHPStan rule could
have caught it. Write the rules that make sense, with tests.
```





Prompt

```
Pull the latest unresolved Sentry issue. Reproduce it with a
failing PHPUnit test, fix the code, confirm the test passes,
then write a PHPStan rule to prevent that class of bug — if
one makes sense.
```

github.com/spaze/phpstan-disallowed-call

```
disallowedFunctionCalls:
    - function: 'assert'
      message: 'Use Webmozart\Assert instead of the native assert() function.'

disallowedStaticCalls:
  - method: 'Carbon\Carbon::now'
    message: 'Use TimeService'

disallowedMethodCalls:
  - method: 'Carbon\Carbon::__construct'
    message: 'Use TimeService'

  - method: ‘Foo\Bar::baz()’
    message: 'Use hello() instead.'
    errorIdentifier: rule.baz
```

phpstan.neon

github.com/phpstan/phpstan-strict-rules

github.com/carlosas/phpat

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

github.com/DaveLiddament/php-language-extensions

PHP Language Extensions Library

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

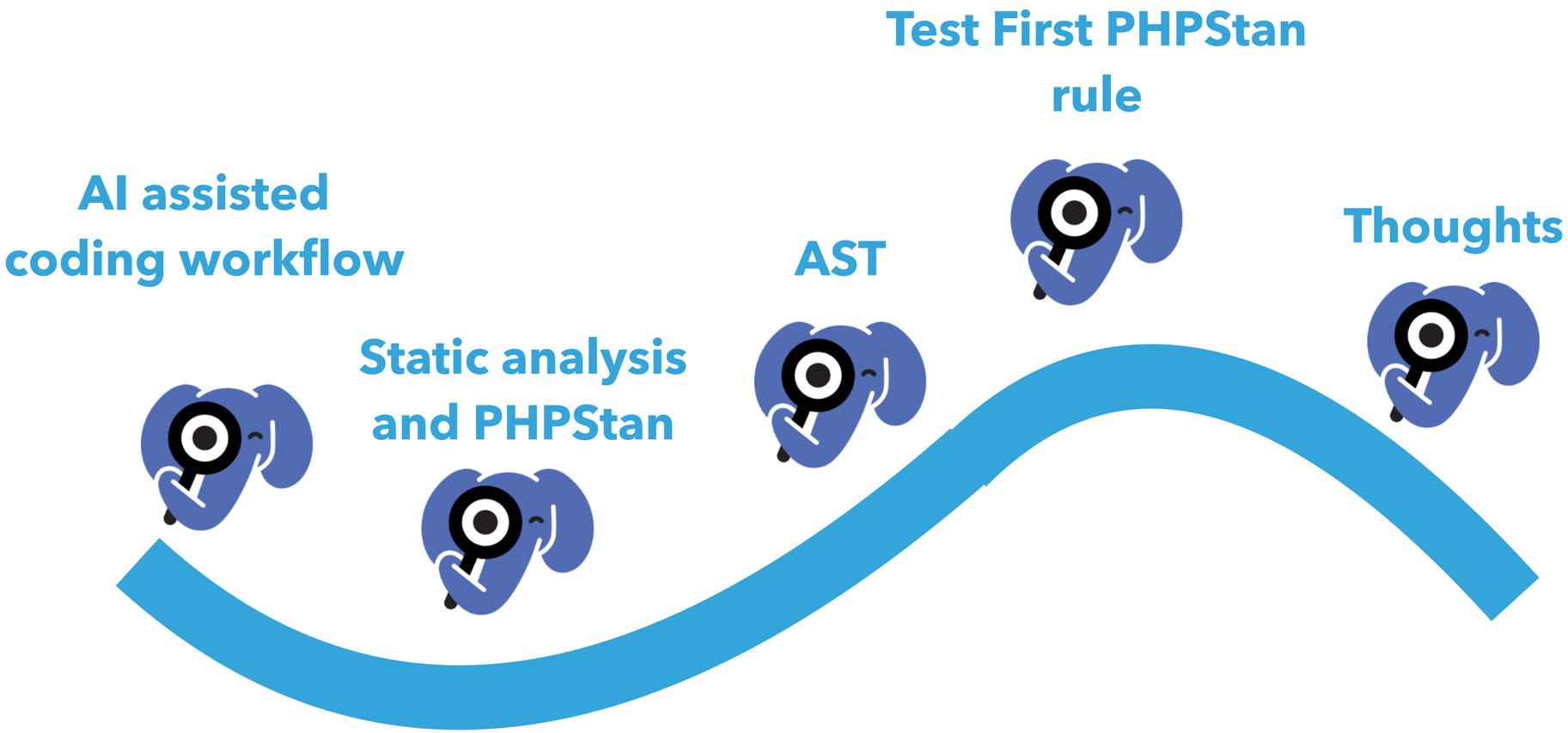
```php
class RouteAttributePathKebabCaseRuleTest
     extends RuleTestCase
{
  protected function getRule(): Rule
  {
    return new RouteAttributePathKebabCaseRule();
  }

  public function testRule(): void
  {
    $this->analyse(
      [__DIR__ . '/Fixtures/routes.php'],
      [
        ['Route path must be kebab-case.', 13],
        ['Route path must be kebab-case.', 18],
      ],
   );
  }
}
```

```php
class UsersController
{
    #[Route('/')]
    public function list(): void
    {
    }

    #[Route('/hello-world')]
    public function kebabCase(): void
    {
    }

    #[Route('/helloWorld')]
    public function camelCase(): void
    {
    }

    #[Route(path: '/helloWorld')]
    public function namedParameter(): void
    {
    }

    … and so on …
```

github.com/DaveLiddament/phpstan-rule-test-helper

```php
use DaveLiddament\PhpstanRuleTestHelper\
                              AbstractRuleTestCase;

class RouteAttributePathKebabCaseRuleTest
      extends AbstractRuleTestCase
{
  protected function getRule(): Rule
  {
    return new RouteAttributePathKebabCaseRule();
  }

  public function testRule(): void
  {
    $this->assertIssuesReported(
      __DIR__ . '/Fixtures/routes.php');
  }

  public function getErrorFormatter(): string
  {
     return 'Route path must be kebab-case.';
  }
}
```

```php
class UsersController
{
    #[Route('/')]
    public function list(): void
    {
    }

    #[Route('/hello-world')]
    public function kebabCase(): void
    {
    }

    #[Route('/helloWorld')] // ERROR
    public function camelCase(): void
    {
    }

    #[Route(path: '/helloWorld')] // ERROR
    public function namedParameter(): void
    {
    }

    … and so on …
```


AI assisted
coding workflow
Static analysis
and PHPStan
AST
Test First PHPStan
rule
Thoughts

Deterministic checks

>

PHPStan

- Better than AGENTS.md
- Custom rules not that difficult
- Ask AI to suggest rules
- Ask AI to help you create rules

What PHPStan rules will you create?

Dave Liddament

github.com/DaveLiddament/php-language-extensions
github.com/DaveLiddament/phpstan-rule-test-helper
github.com/DaveLiddament/test-splitter
github.com/DaveLiddament/sarb

www.linkedin.com/in/daveliddament/
@daveliddament@phpc.social
@daveliddament.bsky.social
@daveliddament

Me when I’m not coding!

Further information

https://phpstan.org/developing-extensions/rules