XDEBUG

ZooRoyal IT

Sebastian Knott

1/21/2019

BEVOR ES LOS GEHT

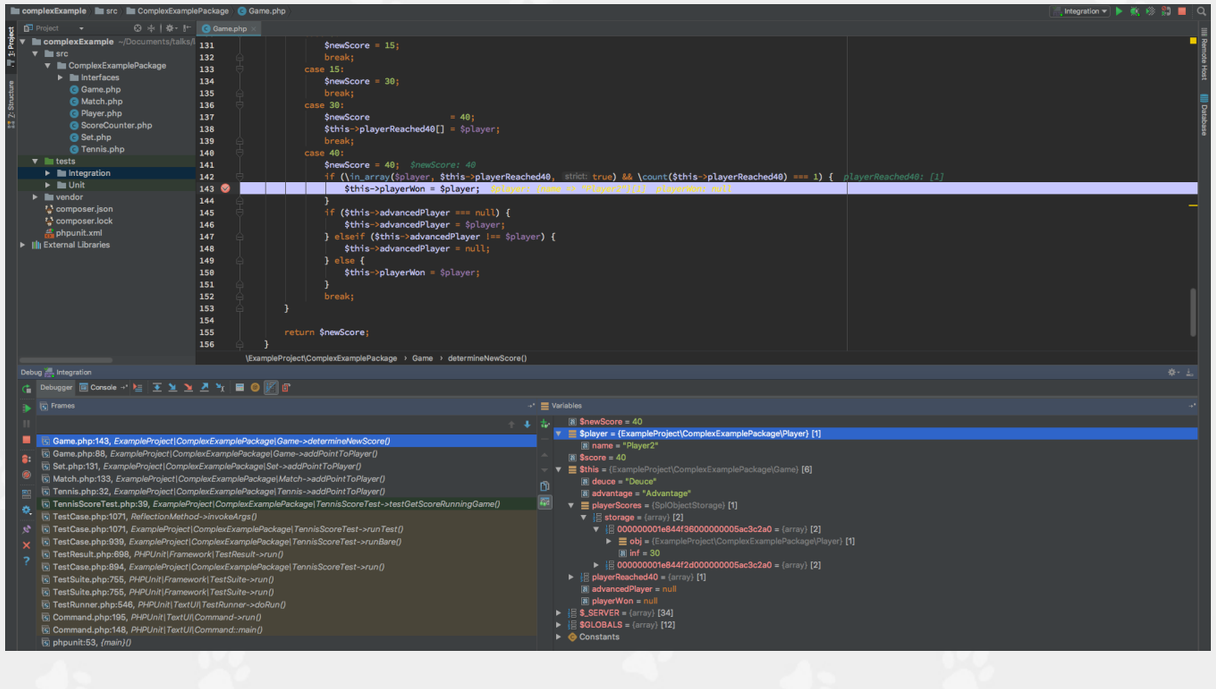
ZooRoyal

WAS IST XDEBUG?

- It contains a single step debugger to use with IDEs
- It upgrades PHP's `var_dump()` function
- It adds stack traces for Notices, Warnings, Errors and Exceptions
- It features functionality for recording every function call and variable assignment to disk
- It contains a profiler
- It provides code coverage functionality for use with PHPUnit

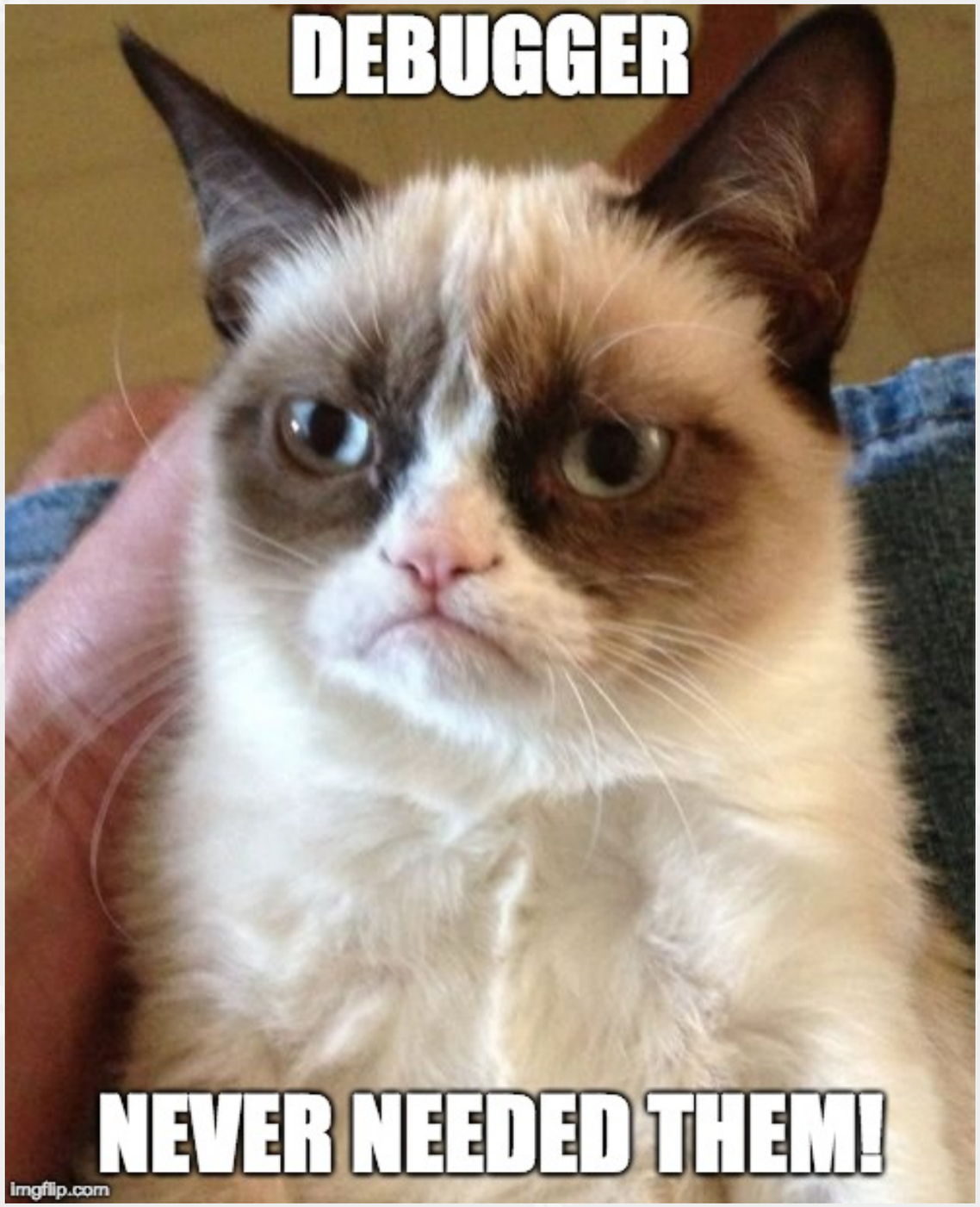
-- xdebug.org

DEBUGGING



- Steuern durch ...
  - Haltepunkte
  - Schrittweises abarbeiten von Anweisungen
- Inspizieren von Daten in flüchtigem Speicher
- Modifizieren von Daten und Programmcode
- "Just in Time"-Ausführung von Programmcode

WARUM XDEBUG?



Brauche ich nicht

... macht das Leben aber einfacher

`var_dump()` reicht mir

... ist aber aufwendig, langsam und statisch

INSTALLATION

PEAR/PECL

VORAUSSETZUNGEN FÜR DIE INSTALLATION ÜBER PEAR

- Xdebug > 0.9.0
- PEAR > 0.9.1
- *nix System. Auf Windows nicht ohne weiteres möglich

INSTALLATION MIT PEAR

1. Befehl in der Unit-Shell ausführen

```
pear install pecl/xdebug
```

2. Folgende Zeile zu `php.ini` hinzufügen

```
zend_extension="/path/to/extension/xdebug.so"
```

INSTALLATION MIT APT UNTER UBUNTU

In vielen *nix-Betriebssystemen kann man Xdebug auch über den Paketmanager beziehen. Hier beispielhaft für Ubuntu.

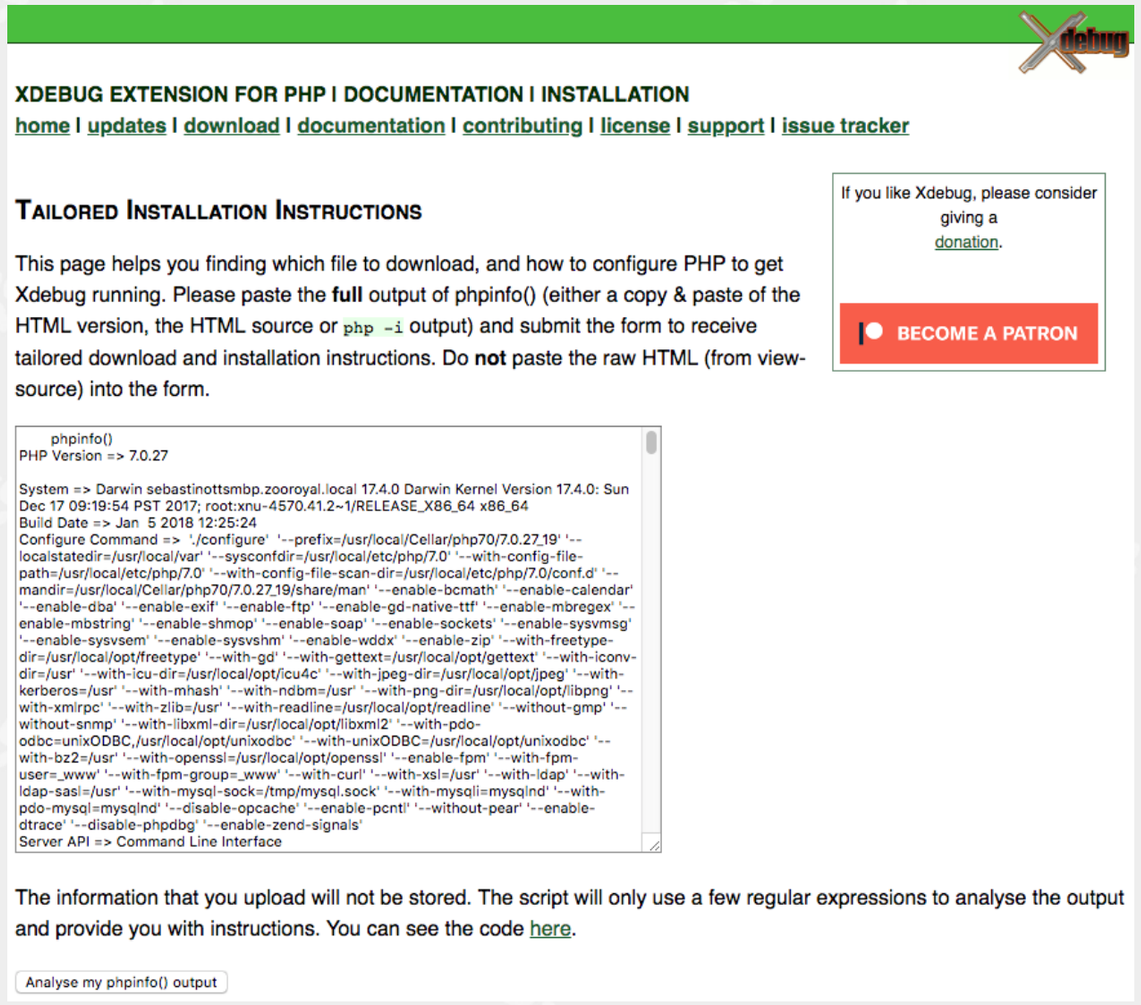
```
sudo apt-get install php-xdebug
```

`php -m` zeigt einem die installierten Erweiterungen an. Nach der Installation also auch Xdebug.

BINARIES INSTALLIEREN MIT WIZZARD

Xdebug bietet auf ihrer Webseite einen wizard, mit dessen Hilfe man Xdebug selbst compilieren kann.

1. Wizard öffnen
2. Ausgabe von `php -i` in die Textbox kopieren
3. Anweisungen auf dem Bildschirm folgen



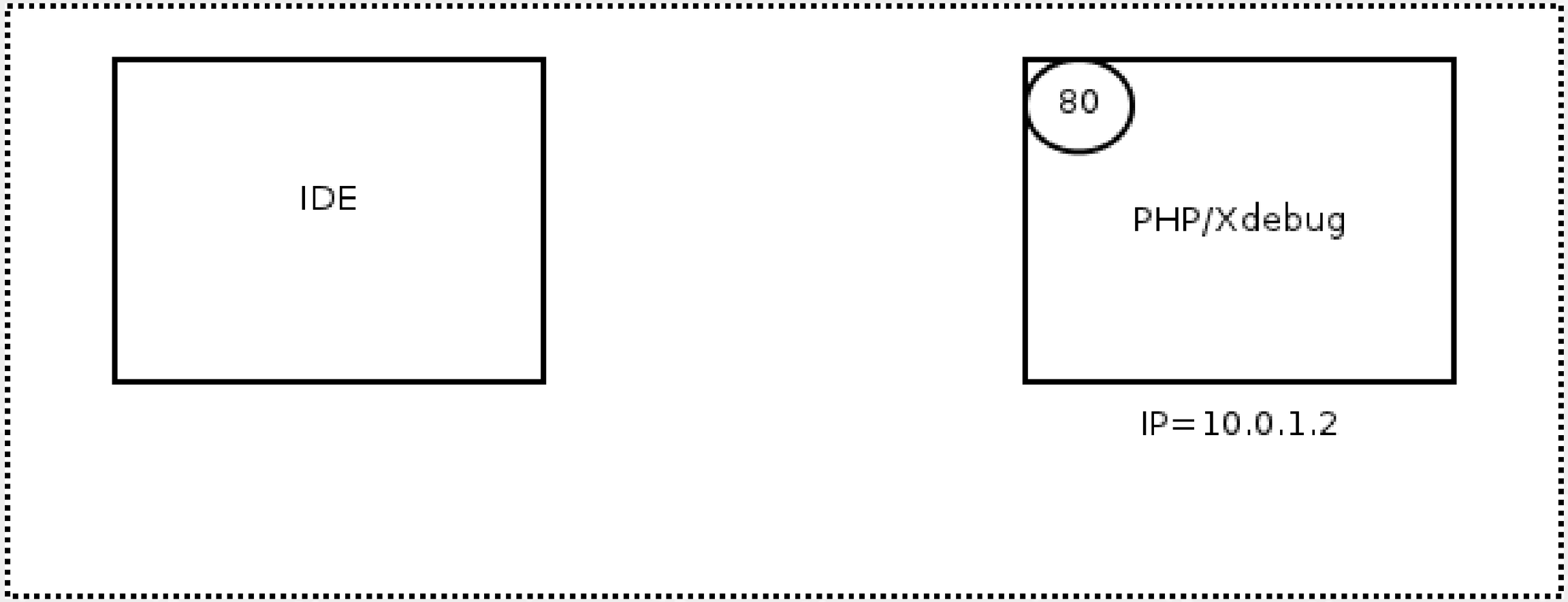
KONFIGURATION

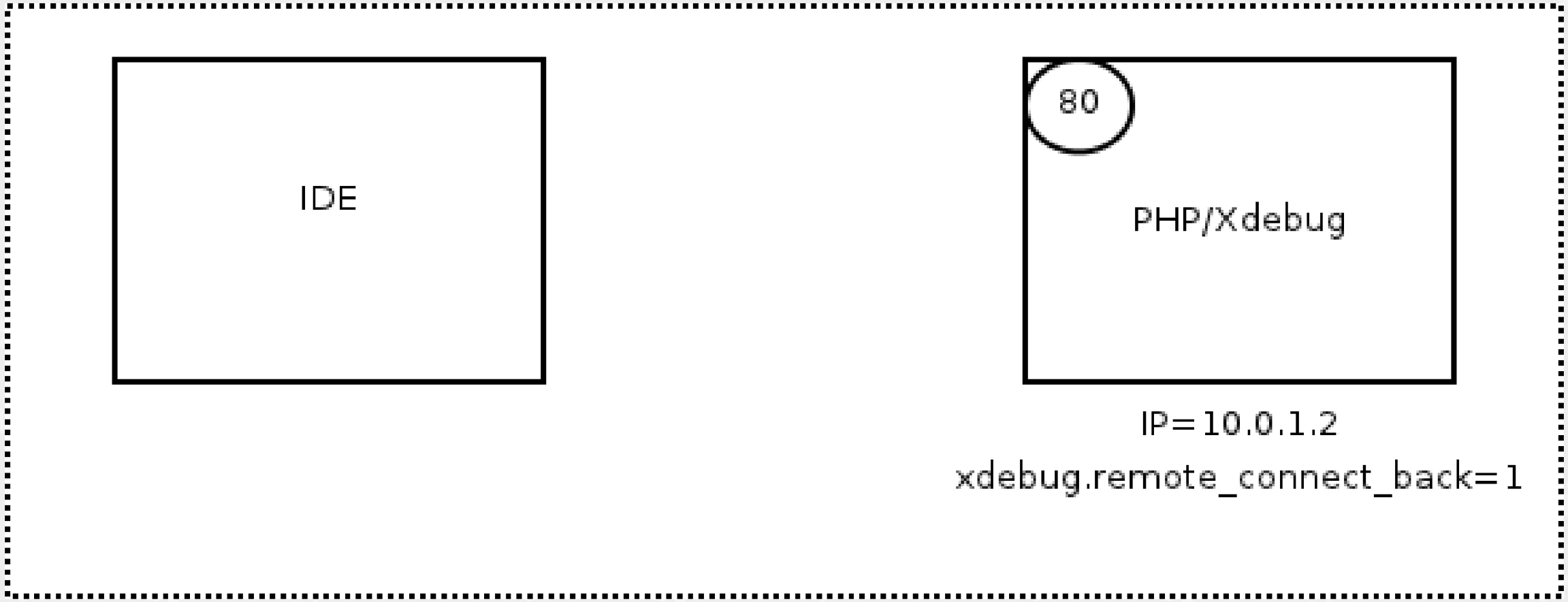
Die Konfiguration für lokale PHP-Interpreter läuft out-of-the-box.

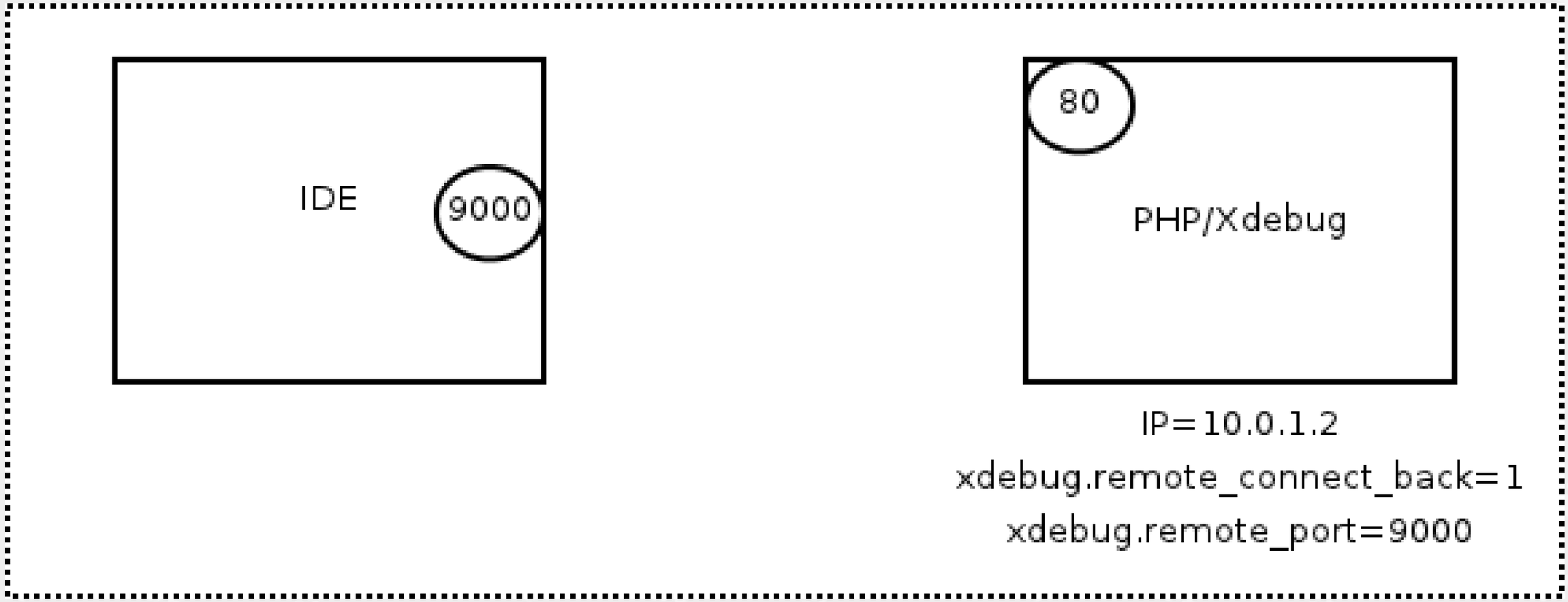
REMOTE DEBUGGING

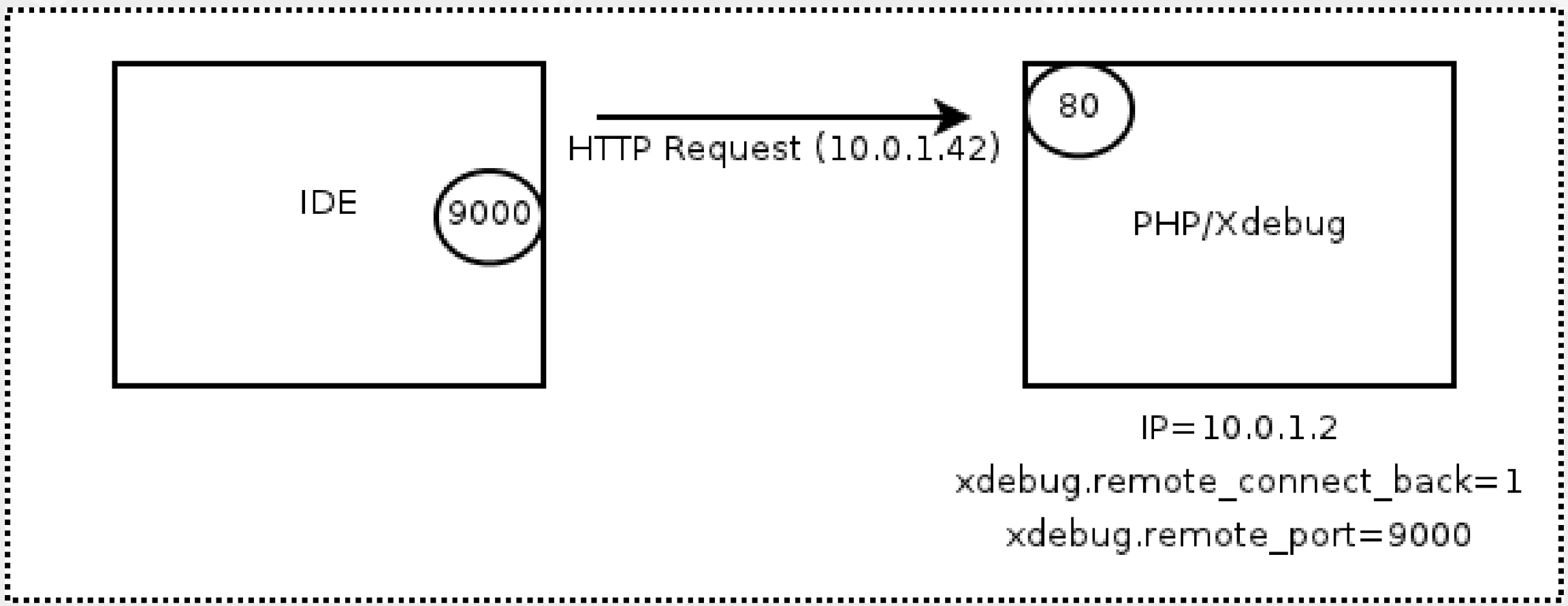
Der Webserver auf dem der PHP-Interpreter läuft ist nicht der selbe Rechner auf dem entwickelt wird.

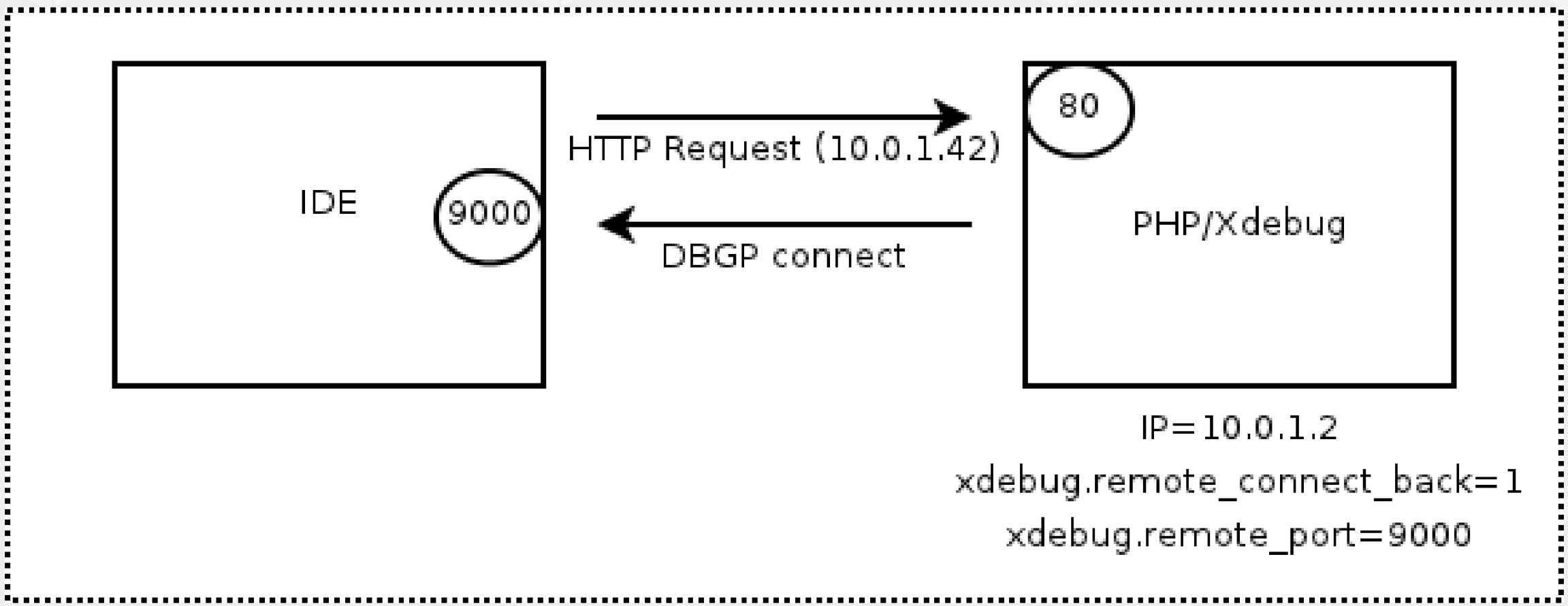
- Lösung per Remote Debugging
- Konfiguration in php.ini

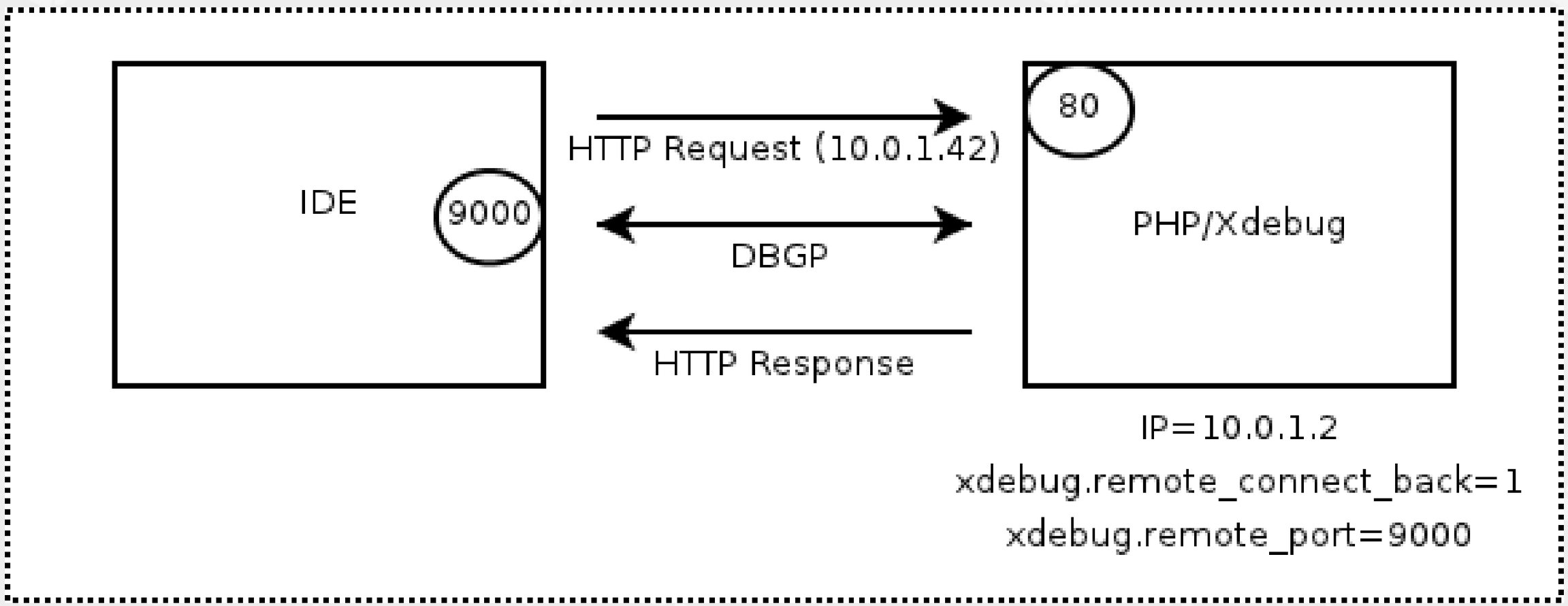

IDE
80
PHP/Xdebug
IP=10.0.1.2


IDE
80
PHP/Xdebug
IP=10.0.1.2
xdebug.remote_connect_back=1


IDE
9000
80
PHP/Xdebug
IP=10.0.1.2
xdebug.remote_connect_back=1
xdebug.remote_port=9000


IDE
9000
HTTP Request (10.0.1.42)
80
PHP/Xdebug
IP=10.0.1.2
xdebug.remote_connect_back=1
xdebug.remote_port=9000


IDE
9000
HTTP Request (10.0.1.42)
DBGP connect
80
PHP/Xdebug
IP=10.0.1.2
xdebug.remote_connect_back=1
xdebug.remote_port=9000


IDE
9000
HTTP Request (10.0.1.42)
DBGP
HTTP Response
80
PHP/Xdebug
IP=10.0.1.2
xdebug.remote_connect_back=1
xdebug.remote_port=9000

Was müsst ihr jetzt tun wenn ihr in der ZooRoyal-Devbox debuggen wollt?

- Einfach Alexander Fitzkes Anleitung im Wiki folgen.

ZooRoyal

INTERACTIVE DEBUGGING

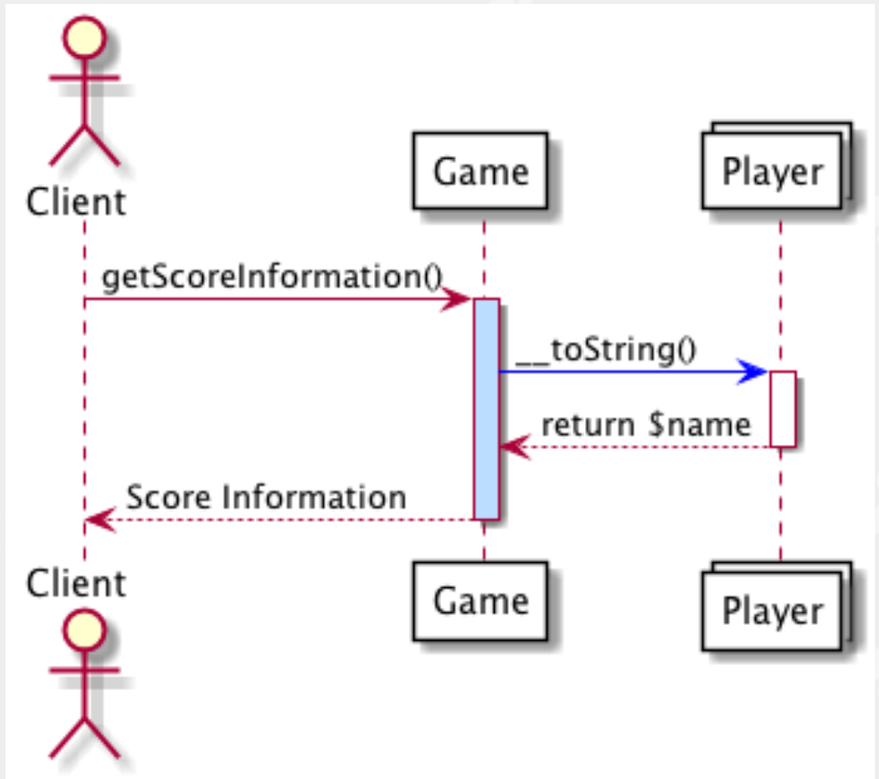
BREAKPOINTS

Breakpoint halten die Application an einer beliebigen Operation an

- Analyse des Speichers
- Analyse des Call Stacks
- Ausführen von PHP-Code
- Schrittweises ausführen

STEP INTO

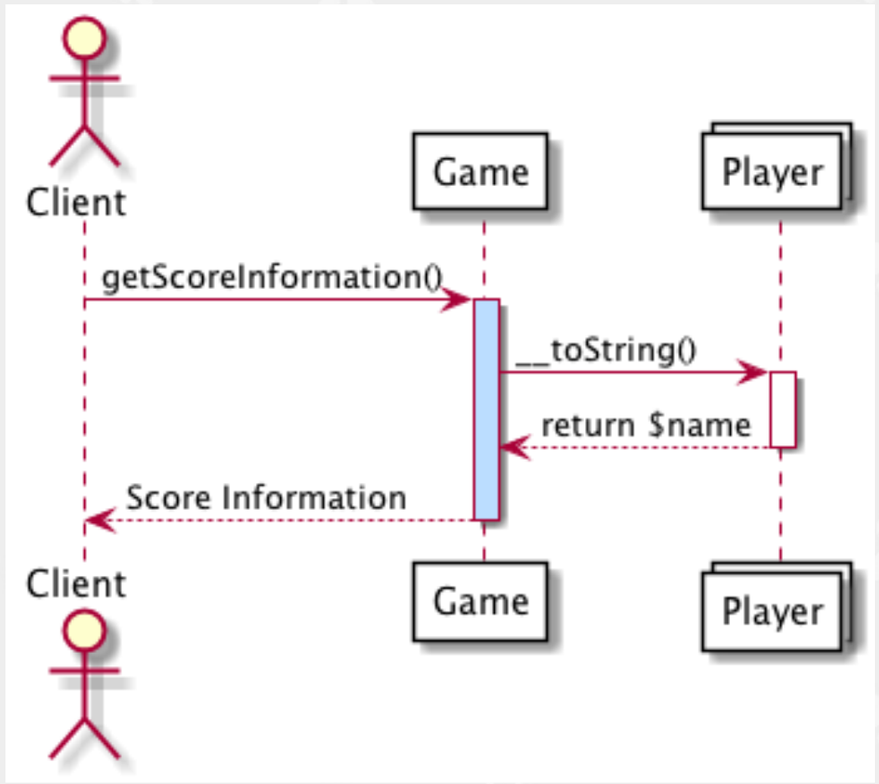
```
$resultingInformation[] = $player . ' : ' . $this->playerScores[$player];
```



- Die Applikation hält in einem Breakpoint in Game
- Mit step into steigt der Debugger den Callstack eine Ebene herab
- Applikation hält in der ersten Zeile von

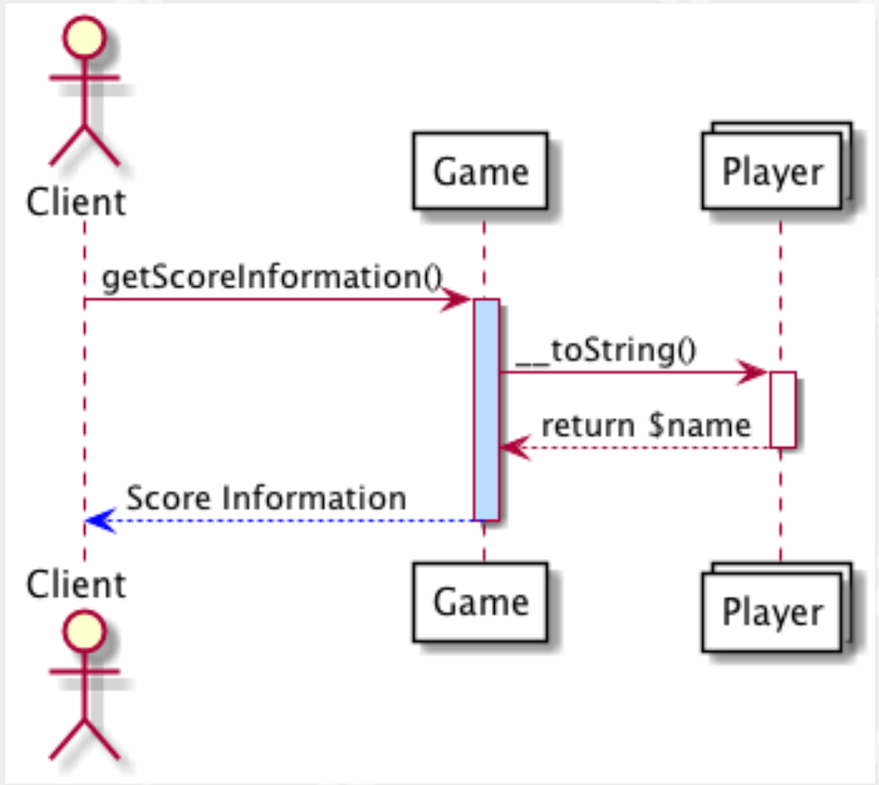
```
player->__toString
```

STEP OVER



- Die Applikation hält in einem Breakpoint in Game
- Mit step over bleibt der Debugger auf der selben Ebene des Callstack
- Applikation führt Operation am Breakpoint aus und hält in der nächsten Zeile in Game

STEP OUT



- Die Applikation hält in einem Breakpoint in Game
- Mit step out steigt der Debugger eine Ebene im Callstack auf
- Applikation führt alle Operationen auf der aktuellen Ebene des Callstacks aus und hält an der nächsten Operation der nächst höheren Ebene.

```
60
61      $resultingInformation = [];  $resultingInformation: [0]
62
63      foreach ($this->playerScores as $player) {  $player: {_mockery_methods => [49], _mockery_expectatior
64          $resultingInformation[] = $player . ' : ' . $this->playerScores[$player];  playerScores: SplObje
65      }
66
67      return implode( glue: ', ', $resultingInformation);
68  }
69
```

\ExampleProject\ComplexExamplePackage › Game › getScoreInformation()

Debug: GameTest

Debugger | Console

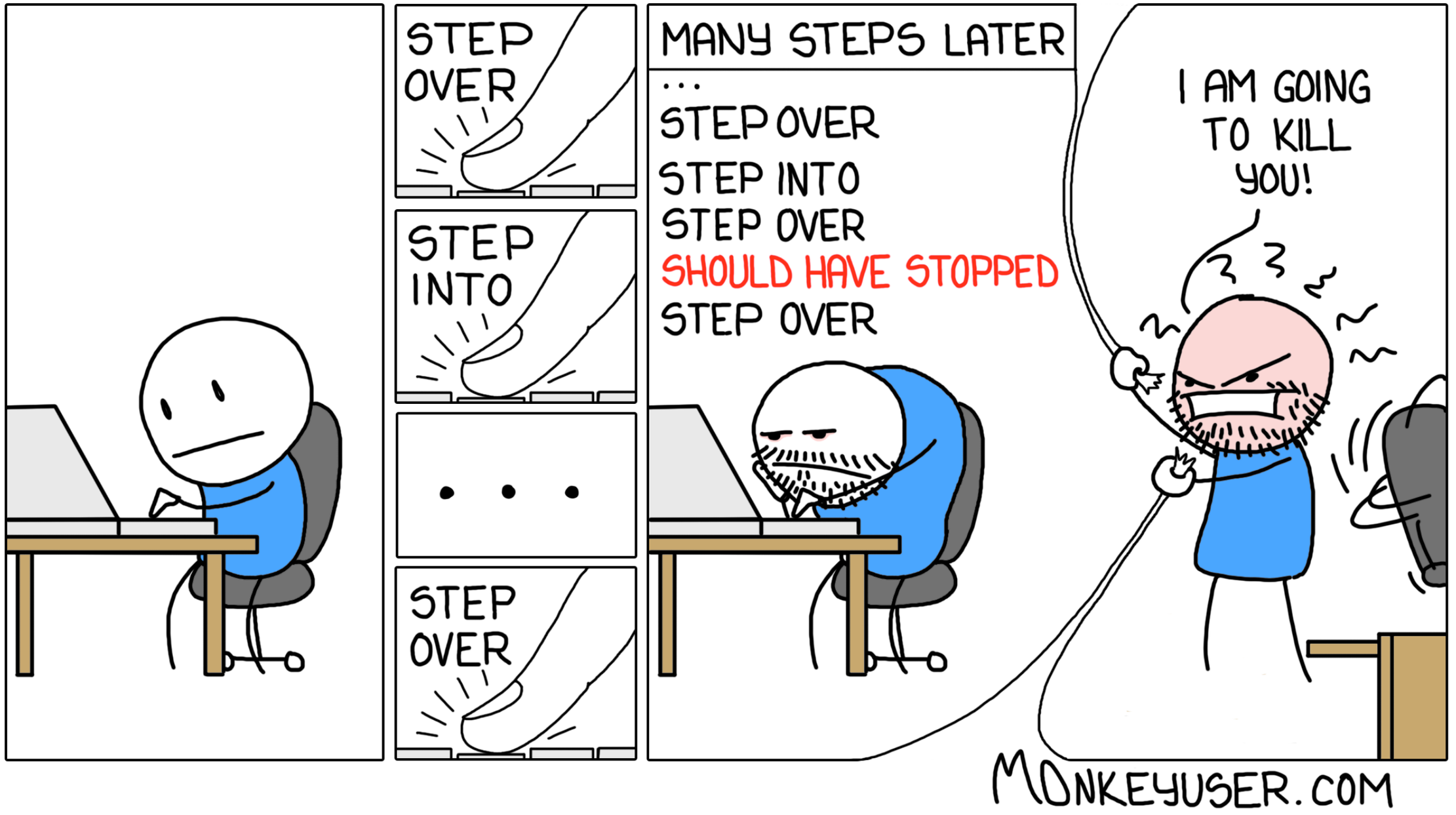
Frames

Game.php:64, *ExampleProject|ComplexExamplePackage|Game->getScoreInformation()*
GameTest.php:69, *ExampleProject|ComplexExamplePackage|GameTest->testGetScoreInformation()*
TestCase.php:1071, *ReflectionMethod->invokeArgs()*
TestCase.php:1071, *ExampleProject|ComplexExamplePackage|GameTest->runTest()*
TestCase.php:939, *ExampleProject|ComplexExamplePackage|GameTest->runBare()*
TestResult.php:698, *PHPUnit|Framework|TestResult->run()*
TestCase.php:894, *ExampleProject|ComplexExamplePackage|GameTest->run()*
TestSuite.php:755, *PHPUnit|Framework|DataProviderTestSuite->run()*
TestSuite.php:755, *PHPUnit|Framework|TestSuite->run()*
TestRunner.php:546, *PHPUnit|TextUI|TestRunner->doRun()*
Command.php:195, *PHPUnit|TextUI|Command->run()*
Command.php:148, *PHPUnit|TextUI|Command::main()*
phpunit:53, *{main}()*

Variables

$player = {Mocke
$resultingInforma
$this = {ExampleF
$_SERVER = {arra
$GLOBALS = {arr
Constants

STEP BY STEP DEBUGGING



CONDITIONS

In bestimmt Situation möchte man die Application an einer Stelle nur unter bestimmten Umständen anhalten

- Rekursionen
- Schleifen
- Exceptions
- In vielen Kontexten genutzter Code

```
60
61      $resultingInformation = [];  $resultingInformation: [0]
62
63      foreach ($this->playerScores as $player) {  $player: {_mockery_methods => [49], _mockery_expectatior
64          $resultingInformation[] = $player . ' : ' . $this->playerScores[$player];  playerScores: SplObje
65      }
66
67      return implode( glue: ', ', $resultingInformation);
68  }
69
```

\ExampleProject\ComplexExamplePackage › Game › getScoreInformation()

Debug: GameTest

Debugger | Console

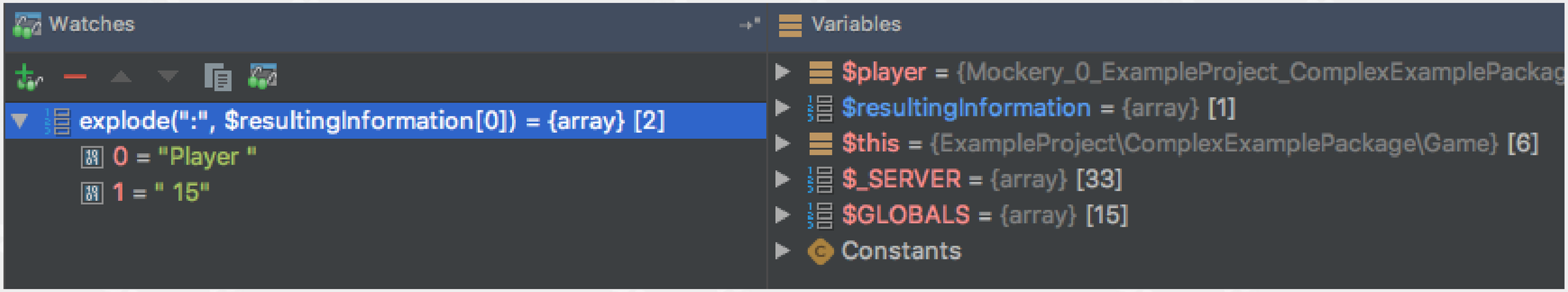
Frames

Game.php:64, *ExampleProject|ComplexExamplePackage|Game->getScoreInformation()*
GameTest.php:69, *ExampleProject|ComplexExamplePackage|GameTest->testGetScoreInformation()*
TestCase.php:1071, *ReflectionMethod->invokeArgs()*
TestCase.php:1071, *ExampleProject|ComplexExamplePackage|GameTest->runTest()*
TestCase.php:939, *ExampleProject|ComplexExamplePackage|GameTest->runBare()*
TestResult.php:698, *PHPUnit|Framework|TestResult->run()*
TestCase.php:894, *ExampleProject|ComplexExamplePackage|GameTest->run()*
TestSuite.php:755, *PHPUnit|Framework|DataProviderTestSuite->run()*
TestSuite.php:755, *PHPUnit|Framework|TestSuite->run()*
TestRunner.php:546, *PHPUnit|TextUI|TestRunner->doRun()*
Command.php:195, *PHPUnit|TextUI|Command->run()*
Command.php:148, *PHPUnit|TextUI|Command::main()*
phpunit:53, *{main}()*

Variables

$player = {Mocke
$resultingInforma
$this = {ExampleF
$_SERVER = {arra
$GLOBALS = {arr
Constants

WATCHES



- Watches sind kleine PHP Snippets
- Werden an jedem Haltepunkt ausgeführt
- Ausgabe wird dem Entwickler dargestellt

```
60
61      $resultingInformation = [];  $resultingInformation: [0]
62
63      foreach ($this->playerScores as $player) {  $player: {_mockery_methods => [49], _mockery_expectatior
64          $resultingInformation[] = $player . ' : ' . $this->playerScores[$player];  playerScores: SplObje
65      }
66
67      return implode( glue: ', ', $resultingInformation);
68  }
69
```

\ExampleProject\ComplexExamplePackage › Game › getScoreInformation()

Debug: GameTest

Debugger | Console

Frames

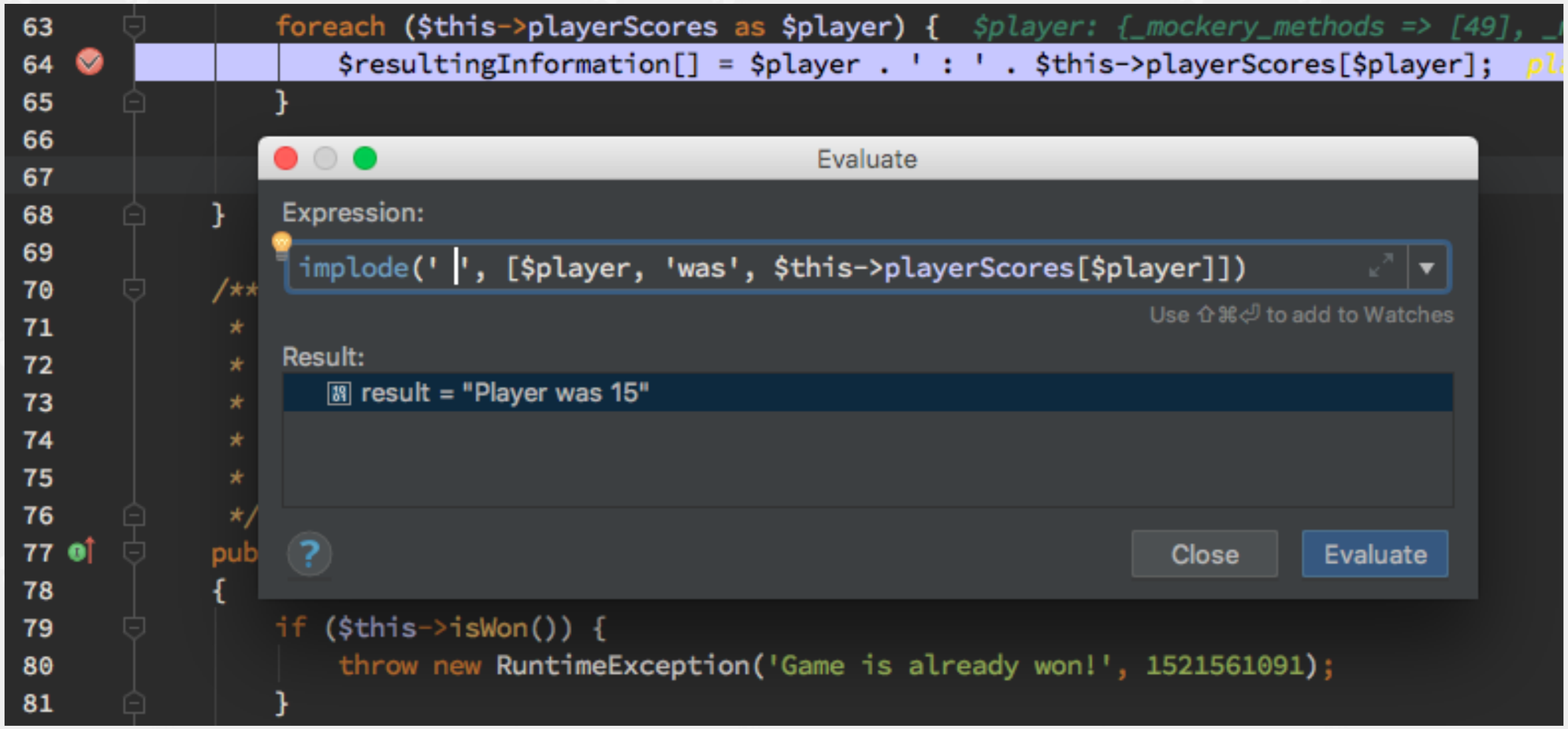
Game.php:64, *ExampleProject|ComplexExamplePackage|Game->getScoreInformation()*
GameTest.php:69, *ExampleProject|ComplexExamplePackage|GameTest->testGetScoreInformation()*
TestCase.php:1071, *ReflectionMethod->invokeArgs()*
TestCase.php:1071, *ExampleProject|ComplexExamplePackage|GameTest->runTest()*
TestCase.php:939, *ExampleProject|ComplexExamplePackage|GameTest->runBare()*
TestResult.php:698, *PHPUnit|Framework|TestResult->run()*
TestCase.php:894, *ExampleProject|ComplexExamplePackage|GameTest->run()*
TestSuite.php:755, *PHPUnit|Framework|DataProviderTestSuite->run()*
TestSuite.php:755, *PHPUnit|Framework|TestSuite->run()*
TestRunner.php:546, *PHPUnit|TextUI|TestRunner->doRun()*
Command.php:195, *PHPUnit|TextUI|Command->run()*
Command.php:148, *PHPUnit|TextUI|Command::main()*
phpunit:53, *{main}()*

Variables

$player = {Mocke
$resultingInforma
$this = {ExampleF
$_SERVER = {arra
$GLOBALS = {arr
Constants

ZooRoyal

CODE EXECUTION



Ausführen von PHP-Code, wenn die Applikation angehalten wurde.

- Zugriff auf Objekte im Scope
- Einfluss auf Programmablauf
- Direkte Ausgabe

```
60
61      $resultingInformation = [];  $resultingInformation: [0]
62
63      foreach ($this->playerScores as $player) {  $player: {_mockery_methods => [49], _mockery_expectatior
64          $resultingInformation[] = $player . ' : ' . $this->playerScores[$player];  playerScores: SplObje
65      }
66
67      return implode( glue: ', ', $resultingInformation);
68  }
69
```

\ExampleProject\ComplexExamplePackage › Game › getScoreInformation()

Debug: GameTest

Debugger | Console

Frames

Game.php:64, *ExampleProject|ComplexExamplePackage|Game->getScoreInformation()*
GameTest.php:69, *ExampleProject|ComplexExamplePackage|GameTest->testGetScoreInformation()*
TestCase.php:1071, *ReflectionMethod->invokeArgs()*
TestCase.php:1071, *ExampleProject|ComplexExamplePackage|GameTest->runTest()*
TestCase.php:939, *ExampleProject|ComplexExamplePackage|GameTest->runBare()*
TestResult.php:698, *PHPUnit|Framework|TestResult->run()*
TestCase.php:894, *ExampleProject|ComplexExamplePackage|GameTest->run()*
TestSuite.php:755, *PHPUnit|Framework|DataProviderTestSuite->run()*
TestSuite.php:755, *PHPUnit|Framework|TestSuite->run()*
TestRunner.php:546, *PHPUnit|TextUI|TestRunner->doRun()*
Command.php:195, *PHPUnit|TextUI|Command->run()*
Command.php:148, *PHPUnit|TextUI|Command::main()*
phpunit:53, *{main}()*

Variables

$player = {Mocke
$resultingInforma
$this = {ExampleF
$_SERVER = {arra
$GLOBALS = {arr
Constants

STACK TRACE

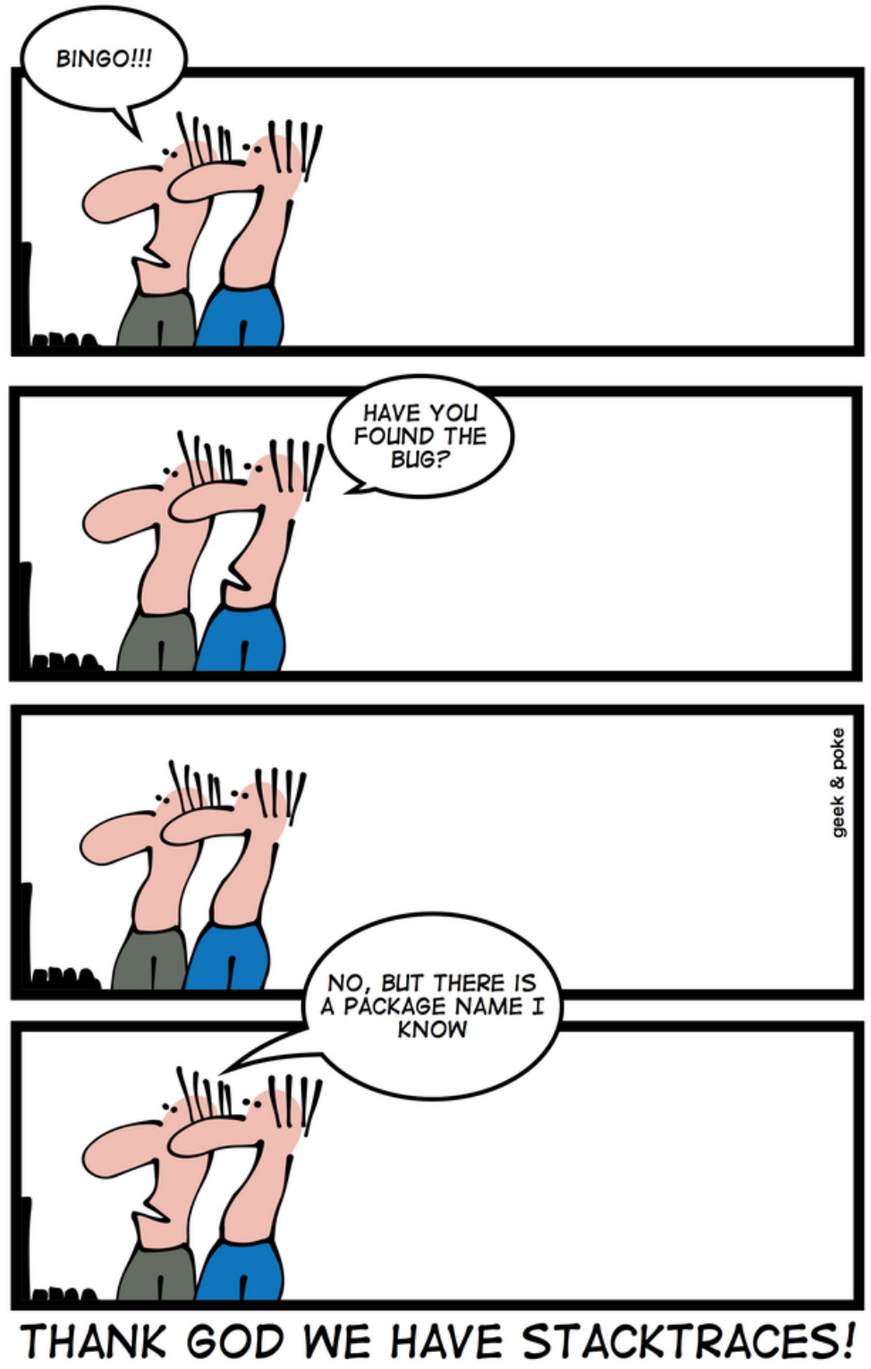
XDEBUG UND STACK TRACES

Xdebug ergänzt die Ausgabe von Stack Traces um …

- Methodenparameter
- Lokale Variablen
- Superglobals

```php
<?php
ini_set('xdebug.collect_vars', 'on');
ini_set('xdebug.collect_params', '4');
ini_set('xdebug.dump_globals', 'on');
ini_set('xdebug.dump.SERVER', 'PWD');
ini_set('xdebug.show_local_vars', 'on');

function foo($array)
{
    $word = 'a word';
    throw new RuntimeException($word);
}

$class = new stdClass;
$class->bar = 100;

$array = [
    42 => false,
    'foo' => 912124,
    $class,
    fopen('/etc/passwd', 'r')
];

foo($array);
```

```
Stack trace:
#0 latex/xDebug/Quellen/stack-trace.php(24): foo(Array)
#1 {main}
  thrown in latex/xDebug/Quellen/stack-trace.php on line 11

Fatal error: Uncaught RuntimeException: a word in latex/xDebug/Quellen/stack-trace.php on line
11

RuntimeException: a word in latex/xDebug/Quellen/stack-trace.php on line 11

Call Stack:
    0.0008     358432   1. {main}() latex/xDebug/Quellen/stack-trace.php:0
    0.0020     360232   2. foo($array = array (42 => FALSE, 'foo' => 912124, 43 => class
stdClass { public $bar = 100 }, 44 => resource(5) of type (stream)))
latex/xDebug/Quellen/stack-trace.php:24

Dump $_SERVER
   $_SERVER['PWD'] = 'latex/xDebug/Quellen'

Variables in local scope (#2):
  $array = array (42 => FALSE, 'foo' => 912124, 43 => class stdClass { public $bar = 100 }, 44
=> resource(5) of type (stream))
  $word = 'a word'
```


BINGO!!!
HAVE YOU FOUND THE BUG?
NO, BUT THERE IS A PACKAGE NAME I KNOW
geek & poke
THANK GOD WE HAVE STACKTRACES!

ZooRoyal

CODE PROFILING

Xdebug kann für Aufrufe von PHP-Applikationen Ablaufprofile erzeugen.

- Call Tree
- Memory Allocation
- Time Tracking

KONFIGURATION

Der Profiler muss mittels php.ini konfiguriert werden

```
xdebug.profiler_enable=1
```

aktiviert den Profiler

```
xdebug.profiler_enable_trigger=1
```

aktiviert den Profiler, wenn die richtig POST-, GET-, oder COOKIE-Variable gesetzt sind

```
xdebug.profiler_output_dir=/path/asd
```

gibt den Speicherort für Profildateien an

PROFILDATEIEN

- Dateiname standardmäßig `cachegrind.out.<id >`
- Muss mit einem Viewer geöffnet werden
  - PHPStorm
  - qcachegrind

BEISPIEL PHP-APPLIKATION

```php
<?php

firstfunction();

function firstFunction()
{
    $data = ['a', 'bb', 'ccc'];
    secondFunction($data);
    wasteTime();
}

function secondFunction($data)
{
    for ($i = 0 ; $i < 10000; $i++) {
        $items = implode(',', $data);
        echo "items $items \n" ;
    }
}

function wasteTime()
{
    echo 'Wasting Time' . "\n";
    sleep(1);
}
```

CALL TREE

Mittels eines Call Trees kann der Entwickler nachvollziehen, welche Methoden in welcher Reihenfolge abgearbeitet worden sind

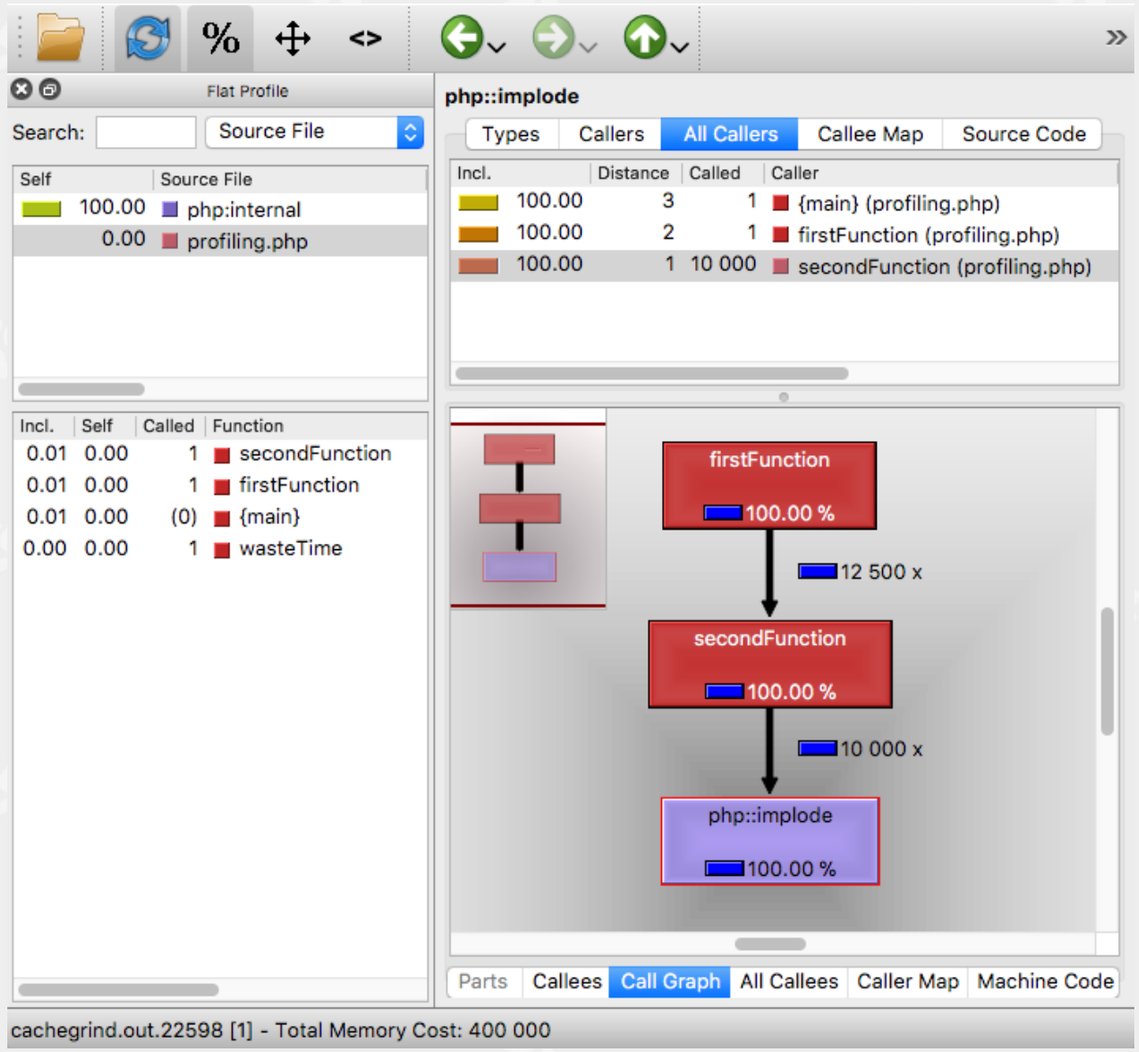
```php
firstfunction();

function firstFunction()
{
    $data = ['a', 'bb', 'ccc'];
    secondFunction($data);
    wasteTime();
}

function secondFunction($data)
{
    for ($i = 0 ; $i < 10000; $i++) {
        $items = implode(',', $data);
        echo "items $items \n" ;
    }
}

function wasteTime()
{
    echo 'Wasting Time' . "\n";
    sleep(1);
}
```

Callable	Time	Calls
<All scripts>	1.274 100,0%	10.005 100,0%
/Users/sknott/Documen	1.274 100,0%	1 0,0%
firstFunction	1.274 100,0%	1 0,0%
wasteTime	1.000 78,5%	1 0,0%
php::sleep	1.000 78,5%	1 0,0%
secondFunction	256 20,1%	1 0,0%
php::implode	0 0,1%	10.000 100,0%

MEMORY ALLOCATION

Aus Profildateien kann auch eine Statistik zum Speicherbedarf erzeugt werden

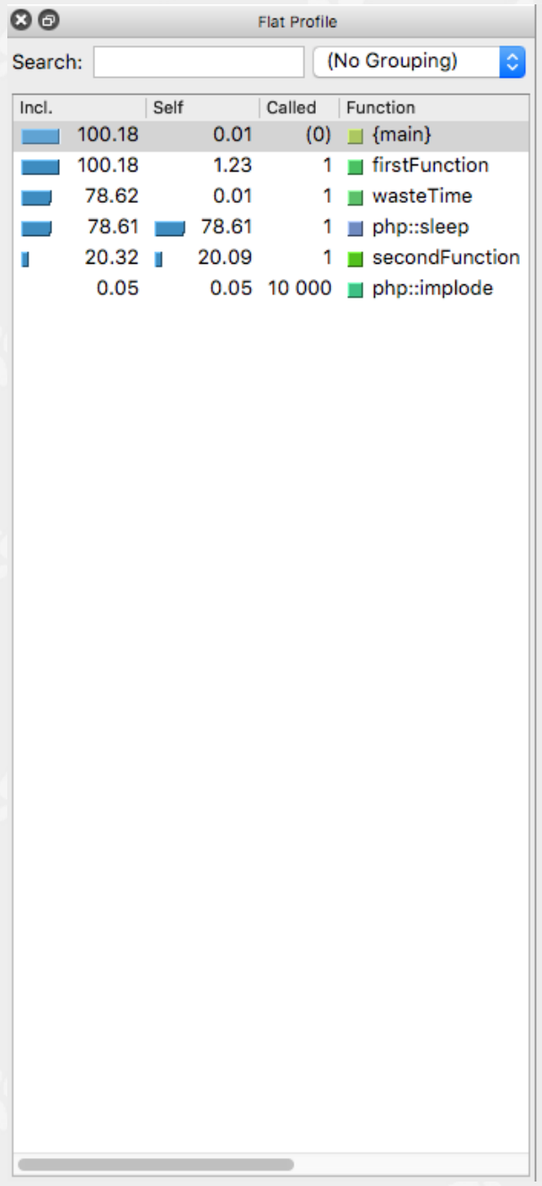
```php
firstfunction();

function firstFunction()
{
    $data = ['a', 'bb', 'ccc'];
    secondFunction($data);
    wasteTime();
}

function secondFunction($data)
{
    for ($i = 0 ; $i < 10000; $i++) {
        $items = implode(',', $data);
        echo "items $items \n" ;
    }
}

function wasteTime()
{
    echo 'Wasting Time' . "\n";
    sleep(1);
}
```

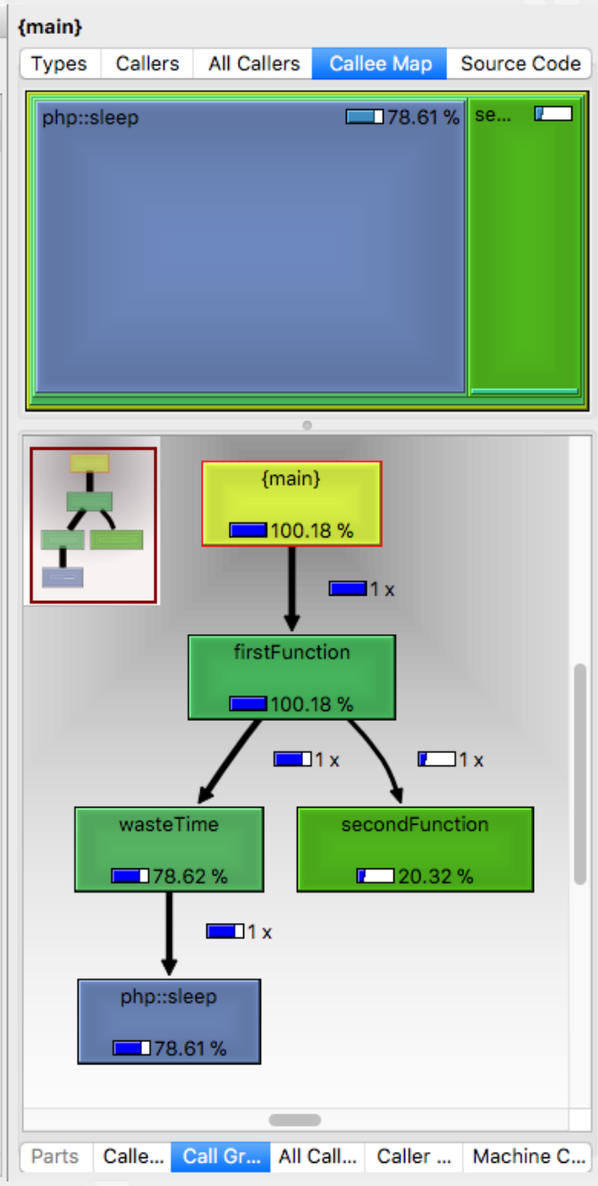


TIME TRACKING

Xdebug zeichnet in Profildateien Laufzeiten von Methoden und Funktoinen auf

```php
firstfunction();

function firstFunction()
{
    $data = ['a', 'bb', 'ccc'];
    secondFunction($data);
    wasteTime();
}

function secondFunction($data)
{
    for ($i = 0 ; $i < 10000; $i++) {
        $items = implode(',', $data);
        echo "items $items \n" ;
    }
}

function wasteTime()
{
    echo 'Wasting Time' . "\n";
    sleep(1);
}
```





ZooRoyal

CODE COVERAGE ANALYSIS

FEATURES

- Zeichnet bei einem Aufruf durchlaufene Zeilen auf
- Exportiert Daten in verschiedener Form
- Integriert von PHP CodeCoverage

PHPUNIT

KONFIGURATION

- CodeCoverage in phpunit.xml konfiguriert
  - Whitelist definiert beobachtete Dateien
  - Log definiert Art der Ausgabe

PHPUNIT IM TERMINAL AUFRUFEN

```
vendor/bin/phpunit --config=phpunit.xml
```

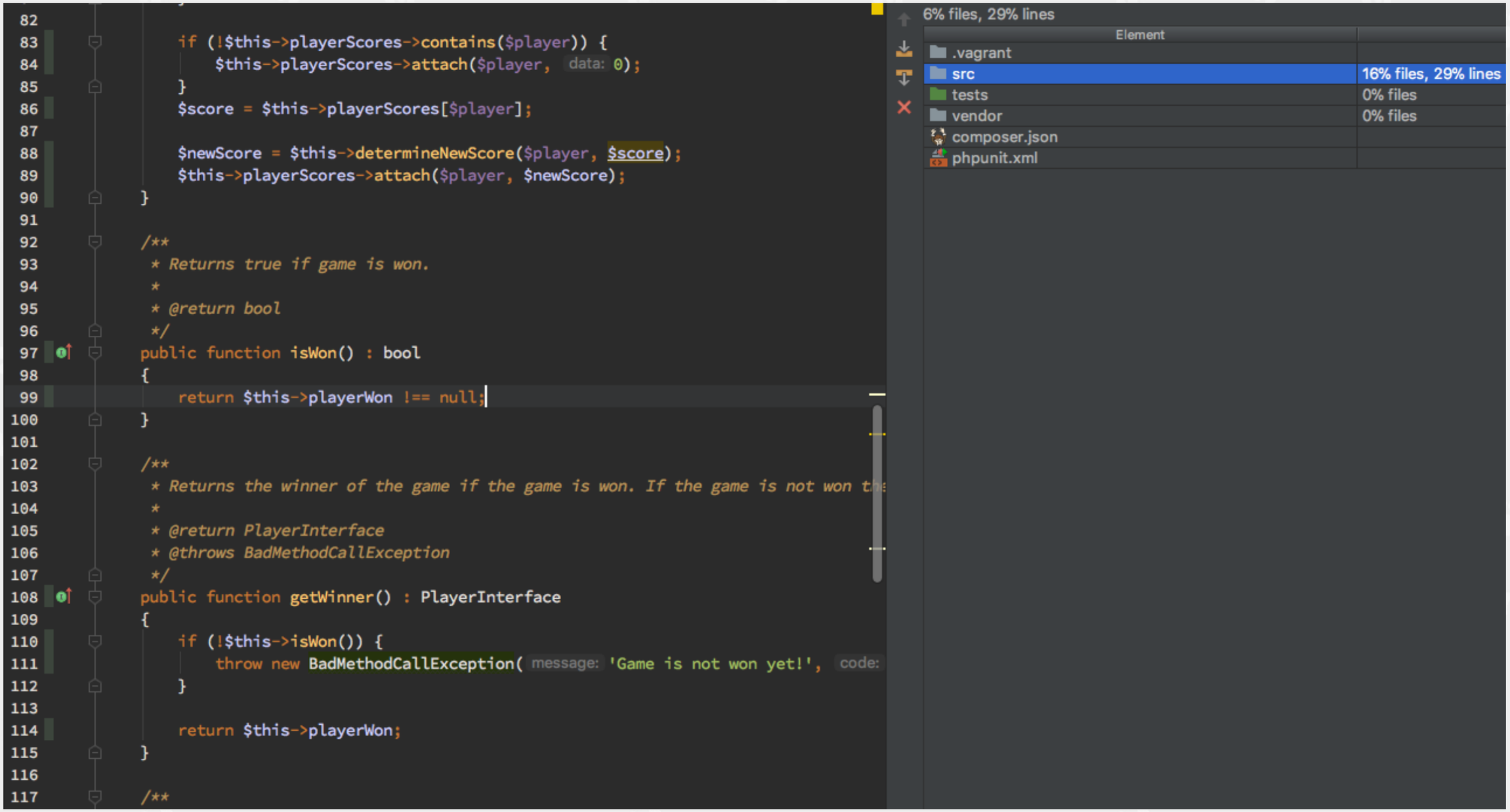
PHPUNIT COVERAGE TERMINAL

```
└─ php vendor/bin/phpunit --config=phpunit.xml
PHPUnit 6.5.8 by Sebastian Bergmann and contributors.

..............                                            14 / 14 (100%)

Time: 252 ms, Memory: 6.00MB

OK (14 tests, 8 assertions)


Code Coverage Report:
  2018-04-27 13:33:24

 Summary:
  Classes: 100.00% (1/1)
  Methods: 100.00% (6/6)
  Lines:   100.00% (46/46)

\ExampleProject\ComplexExamplePackage::ExampleProject\ComplexExamplePackage\Game
  Methods: 100.00% ( 6/ 6)   Lines: 100.00% ( 46/ 46)
```

ZooRoyal

PHPUNIT COVERAGE IN PHPSTORM

TestCoverage kann auch direkt in PHPStorm angezeigt werden.



CODE DOJO

DAS DOJO

- Kleine, unabhängige, fokussierte, in sich geschlossene Übung
- Übt die Ausführung und Herangehensweise
- Bietet Raum für gemeinsames Lernen
- Lösung der Aufgabe erklärtes Nicht-Ziel

DER FOKUS

- Pair Programming + TDD = TDD-Game
- Keine PHPMD Regel darf gebrochen werden
- Absichtsvolles Testen
- Einsatz des Debuggers

DAS KATA

HANGMAN

- Der Spieler muss ein geheimes Wort erraten
- Der spieler kann 10 Mal eine Aktion ausführen
  - Einen Buchstaben erraten
  - Das Wort erraten
- Die geratenen Buchstaben werden an der richtigen Position im Wort angezeigt
- Nicht erratene Buchstaben werden durch _ markiert

ZooRoyal

```php
interface hangman(){
    /** Erhält zu Spielbeginn das zu ratende Wort */
    public function __construct(string $secret);

    /**
    * Lässt den Spieler einen Buchstaben raten.
    * @return bool true wenn der Buchstabe im Wort ist sonst false
    */
    public function guessLetter(string $letter):bool;

    /** Zeigt die erratenen Buchstaben im Wort */
    public function getState():string;

    /** Gibt verbleibende Versuche zurück */
    public function getTriesLeft():int;

    /** Lässt den Spieler das Wort raten */
    public function solve(string guess):bool;

    /** Gibt $secret zurück wenn der Spieler verloren hat */
    public function getSecret():string;
}
```