

XDEBUG

ZooRoyal IT

Sebastian Knott

1/21/2019

BEVOR ES LOS GEHT

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

The screenshot shows a debugger interface with a code editor displaying PHP code. A break point is set at line 143. The call stack shows several frames, and the variables panel shows the state of variables.

```

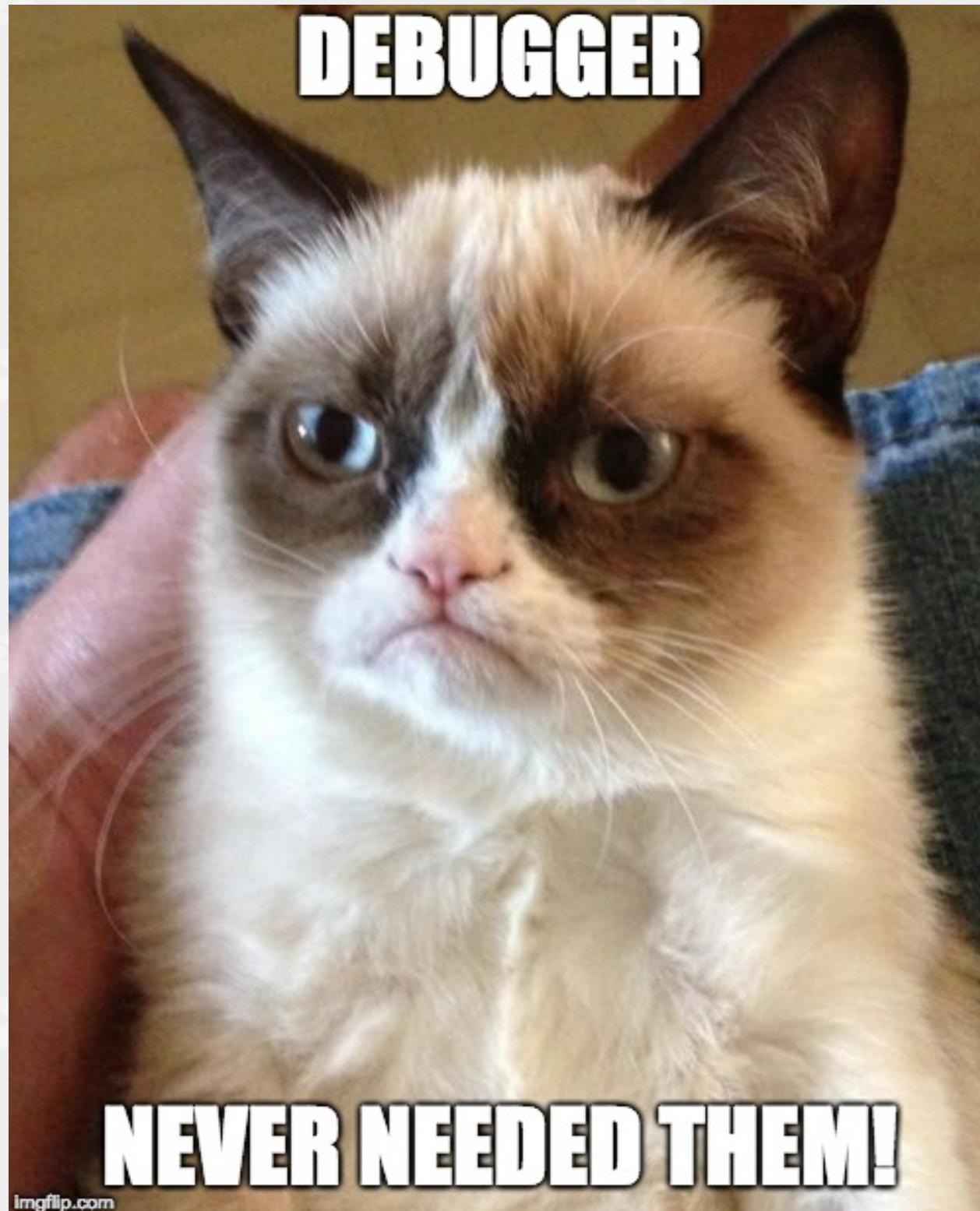
    $newScore = 15;
    break;
    case 15:
        $newScore = 30;
    break;
    case 30:
        $newScore = 40;
        $this->playerReached40[] = $player;
    break;
    case 40:
        $newScore = 40; $newScore = 40
        if (!in_array($player, $this->playerReached40, strict: true) && \count($this->playerReached40) === 1) { $playerReached40[1]
            $this->playerWon = $player; $player = (name > "Player2")? null: null
        }
        if ($this->advancedPlayer === null) {
            $this->advancedPlayer = $player;
        } elseif ($this->advancedPlayer !== $player) {
            $this->advancedPlayer = null;
        } else {
            $this->playerWon = $player;
        }
    break;

    return $newScore;
}

```

- ❖ 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.

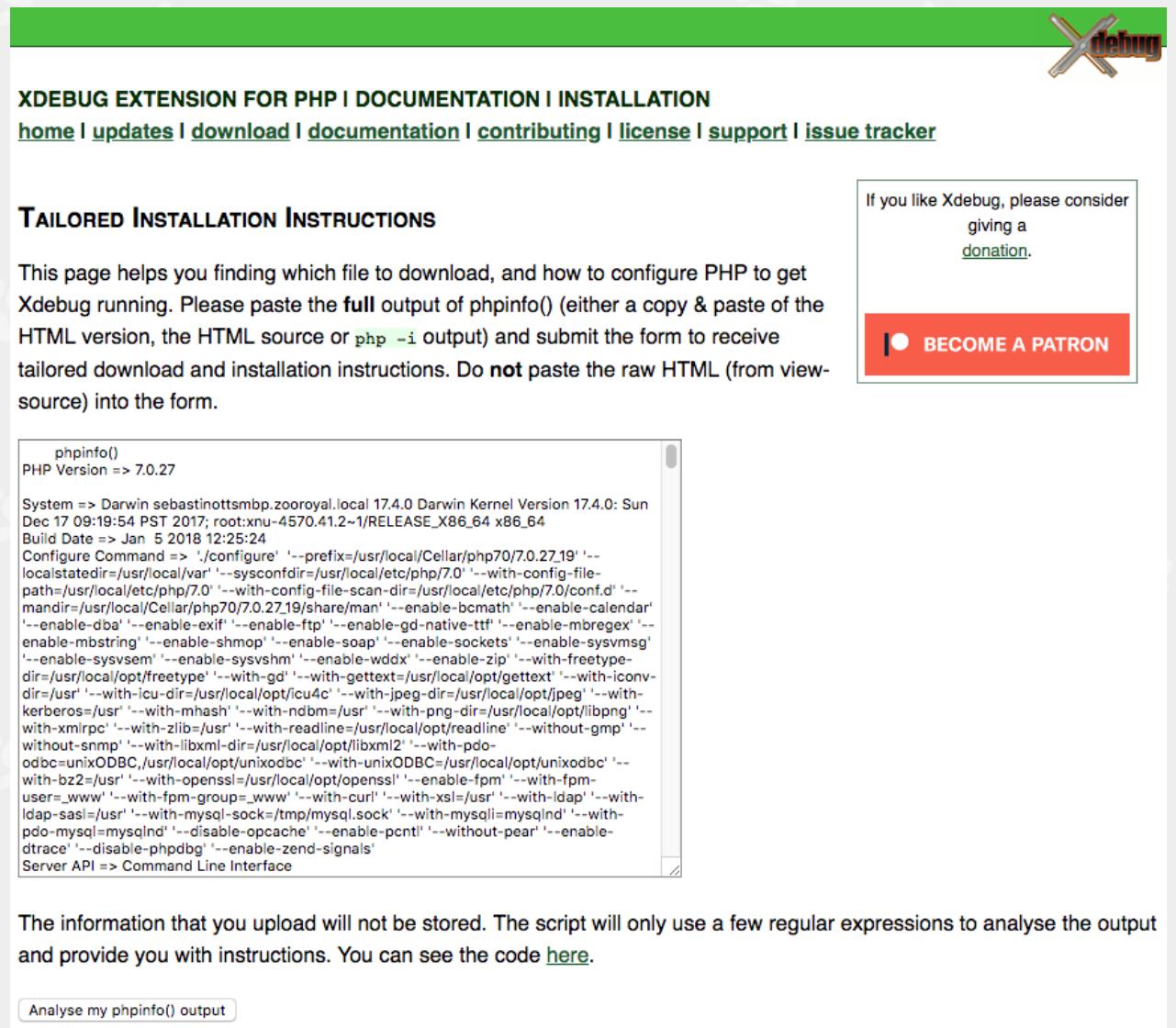
BINARIES INSTALLIEREN MIT WIZZARD

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



The screenshot shows the Xdebug Installation Wizard interface. At the top right is the Xdebug logo. Below it is a navigation bar with links: home | updates | download | documentation | contributing | license | support | issue tracker. To the right of the navigation is a green box containing text: "If you like Xdebug, please consider giving a donation." Below this is a red button labeled "BECOME A PATRON". The main content area has a green header "XDEBUG EXTENSION FOR PHP | DOCUMENTATION | INSTALLATION". Underneath is a section titled "TAILORED INSTALLATION INSTRUCTIONS" with the following text: "This page helps you finding which file to download, and how to configure PHP to get Xdebug running. Please paste the full output of `phpinfo()` (either a copy & paste of the HTML version, the HTML source or `php -i` output) and submit the form to receive tailored download and installation instructions. Do **not** paste the raw HTML (from view-source) into the form." Below this is a large text box containing the `phpinfo()` output. At the bottom of the page is a note: "The information that you upload will not be stored. The script will only use a few regular expressions to analyse the output and provide you with instructions. You can see the code [here](#)". At the very bottom is a button labeled "Analyse my `phpinfo()` output".

```

phpinfo()
PHP Version => 7.0.27

System => Darwin sebastianottsmbp.zooroyal.local 17.4.0 Darwin Kernel Version 17.4.0: Sun Dec 17 09:19:54 PST 2017; root:xnu-4570.41.2~1/RELEASE_X86_64 x86_64
Build Date => Jan 5 2018 12:25:24
Configure Command => './configure' '--prefix=/usr/local/Cellar/php70/7.0.27.19' '--localstatedir=/usr/local/var' '--sysconfdir=/usr/local/etc/php/7.0' '--with-config-file-path=/usr/local/etc/php/7.0' '--with-config-file-scan-dir=/usr/local/etc/php/7.0/conf.d' '--mandir=/usr/local/Cellar/php70/7.0.27.19/share/man' '--enable-bcmath' '--enable-calendar' '--enable-dba' '--enable-exif' '--enable-ftp' '--enable-gd-native-ttf' '--enable-mbregex' '--enable-mbstring' '--enable-shmop' '--enable-soap' '--enable-sockets' '--enable-sysvmsg' '--enable-sysvsem' '--enable-sysvshm' '--enable-wddx' '--enable-zip' '--with-freetype-dir=/usr/local/opt/freetype' '--with-gd' '--with-gettext=/usr/local/opt/gettext' '--with-iconv-dir=/usr' '--with-icu-dir=/usr/local/opt/icu4c' '--with-jpeg-dir=/usr/local/opt/jpeg' '--with-kerberos=/usr' '--with-mhash' '--with-ndbm=/usr' '--with-png-dir=/usr/local/opt/png' '--with-xmlrpc' '--with-zlib=/usr' '--with-readline=/usr/local/opt/readline' '--without-gmp' '--without-snmp' '--with-libxml-dir=/usr/local/opt/libxml2' '--with-pdo-odbc=unixODBC,/usr/local/opt/unixodbc' '--with-unixODBC=/usr/local/opt/unixodbc' '--with-bz2=/usr' '--with-openssl=/usr/local/opt/openssl' '--enable-fpm' '--with-fpm-user=_www' '--with-fpm-group=_www' '--with-curl' '--with-xsl=/usr' '--with-ldap' '--with-ldap-sasl=/usr' '--with-mysql-sock=/tmp/mysql.sock' '--with-mysqli=mysqlnd' '--with-pdo-mysql=mysqlnd' '--disable-opcache' '--enable-pcntl' '--without-pear' '--enable-dtrace' '--disable-phpdbg' '--enable-zend-signals'
Server API => Command Line Interface
  
```

KONFIGURATION



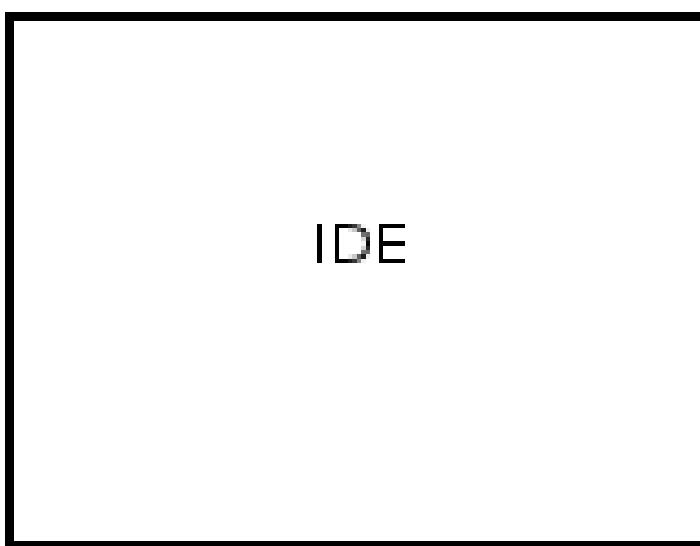
≡

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

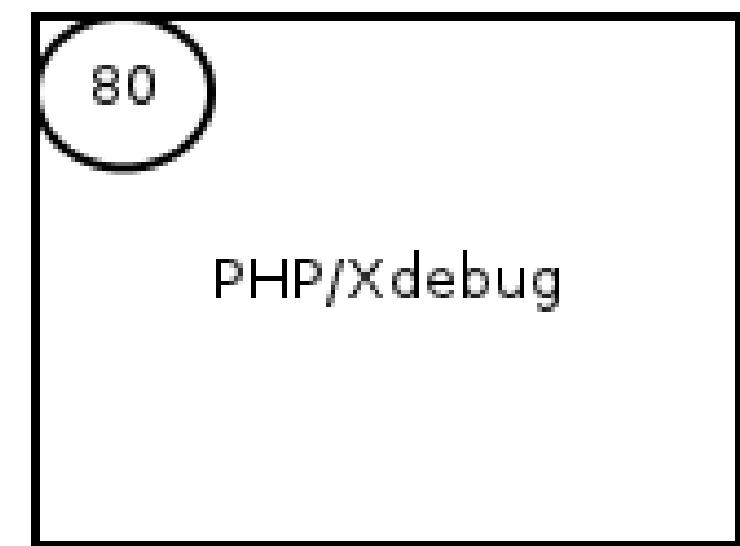
REMOTE DEBUGGING

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

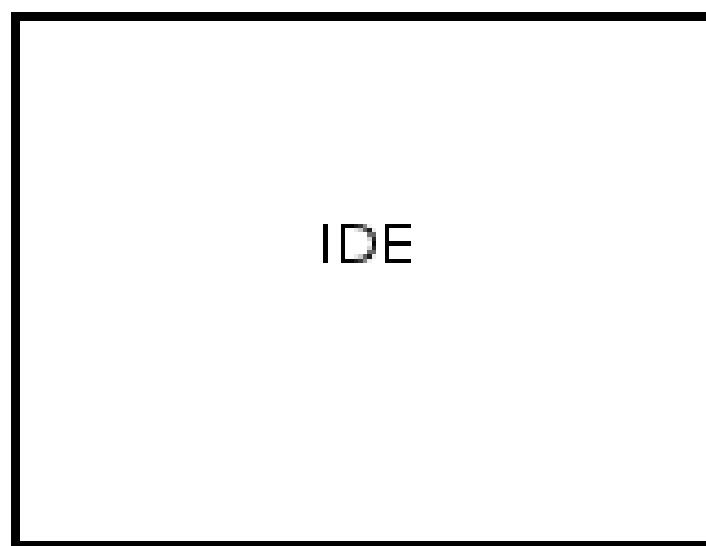


IDE

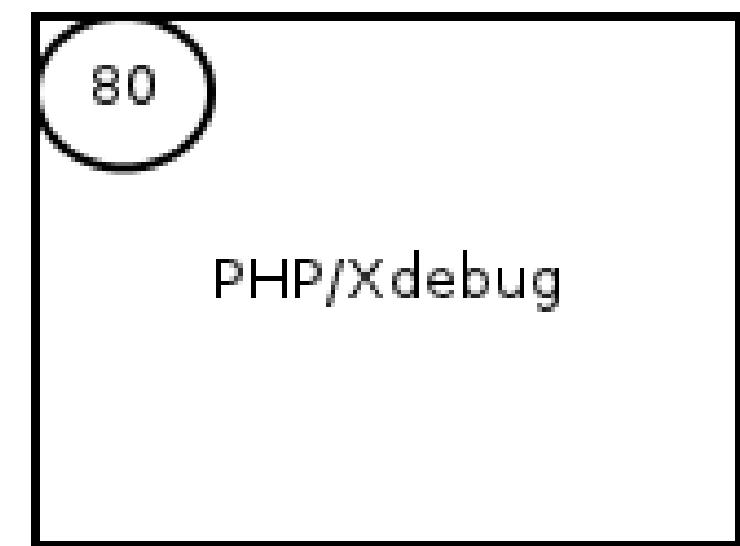


PHP/Xdebug

IP=10.0.1.2



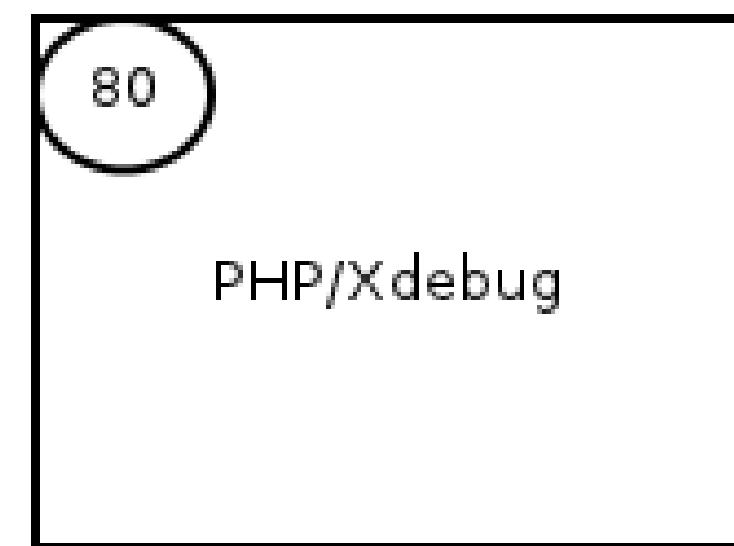
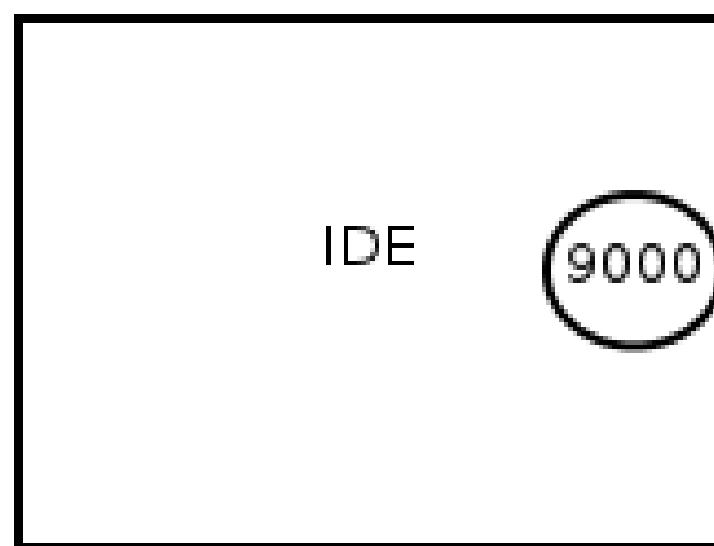
IDE



PHP/Xdebug

IP=10.0.1.2

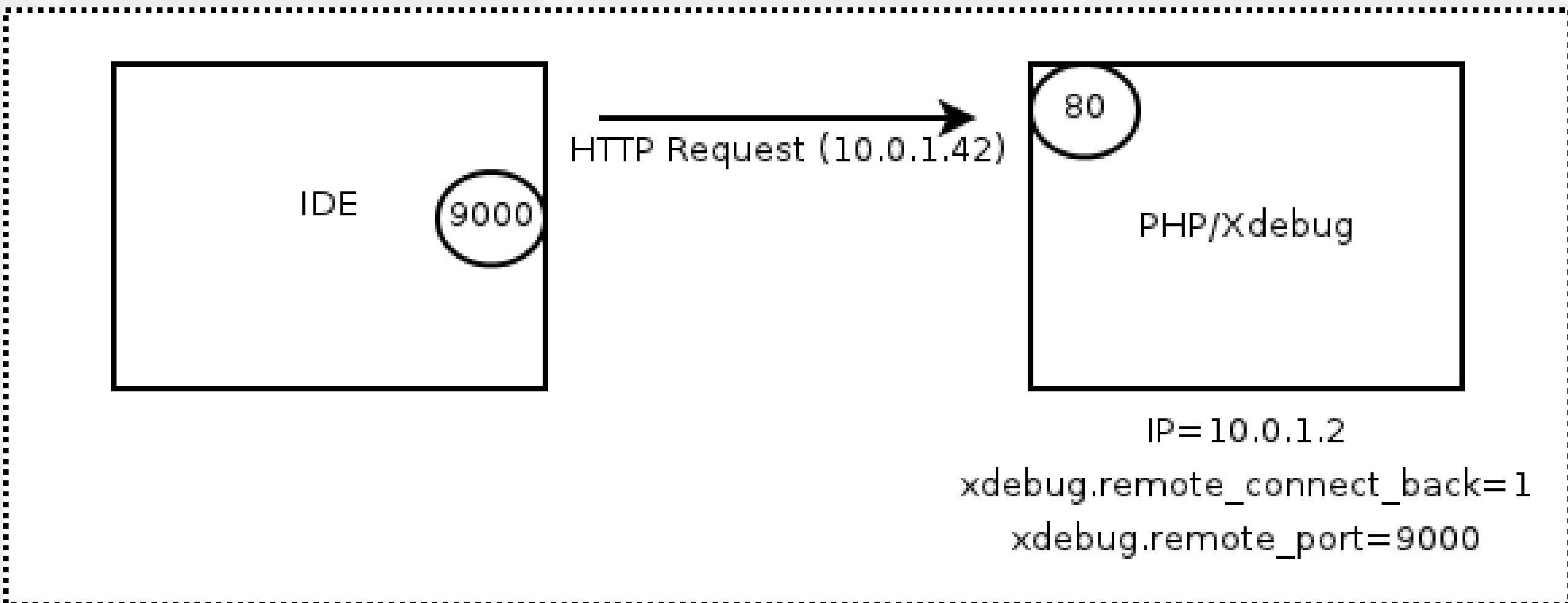
xdebug.remote_connect_back=1

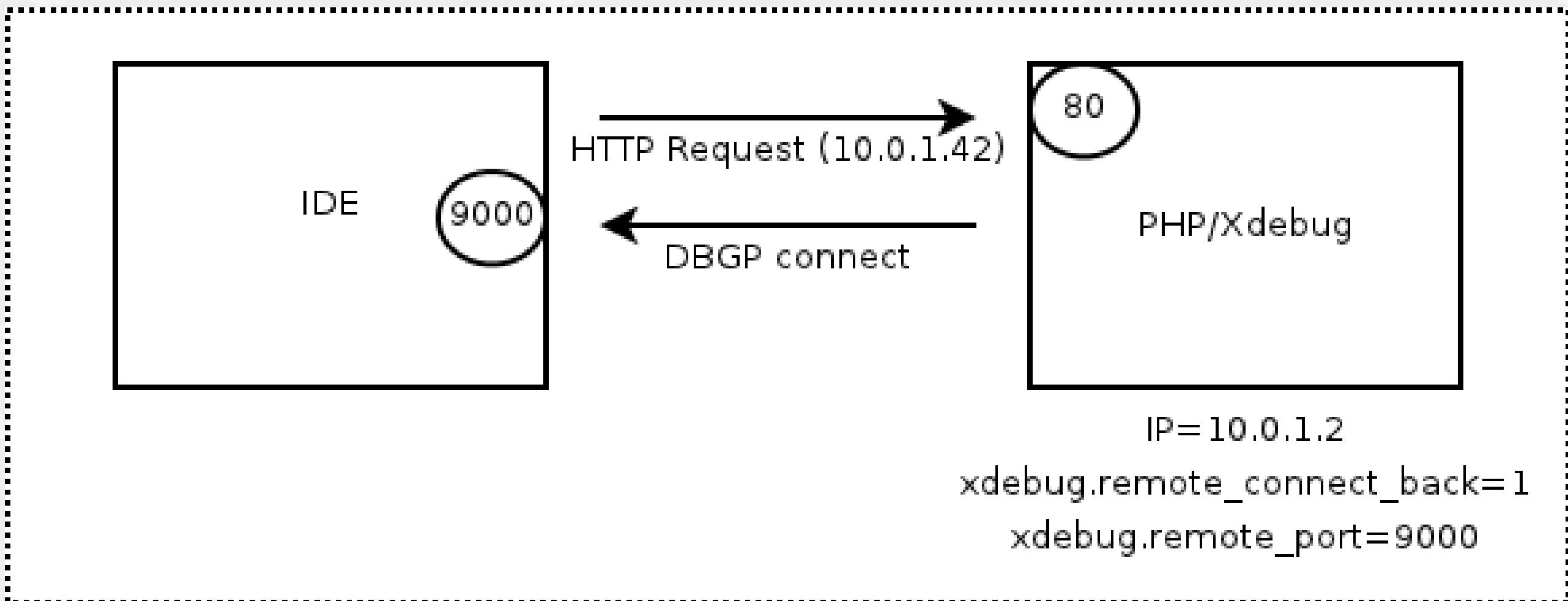


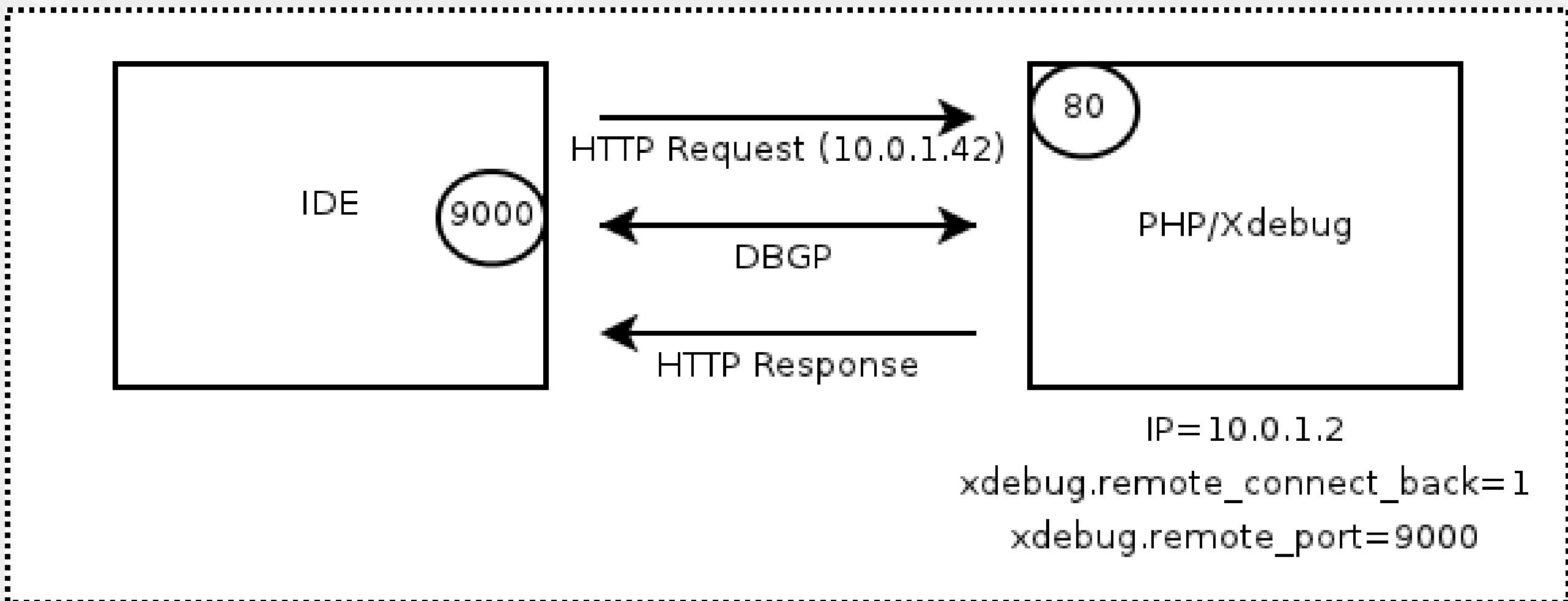
IP=10.0.1.2

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.

INTERACTIVE DEBUGGING

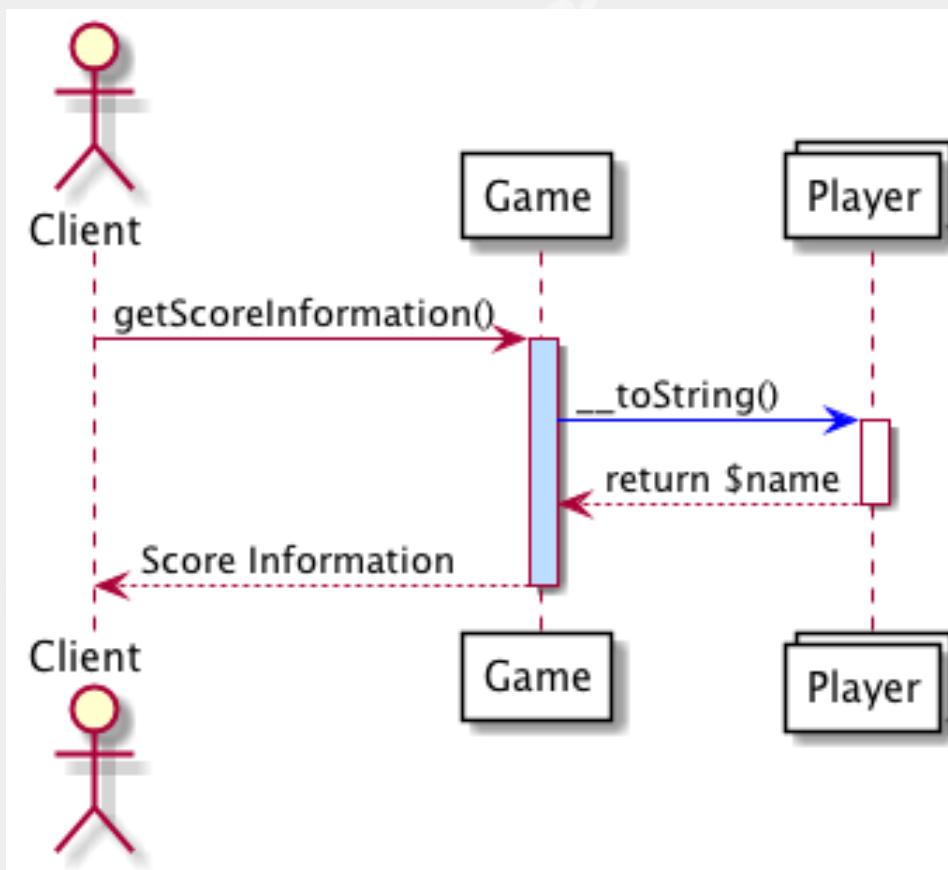
BREAKPOINTS

Breakpoint halten die Application an einer beliebigen Operation an

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

STEP INTO

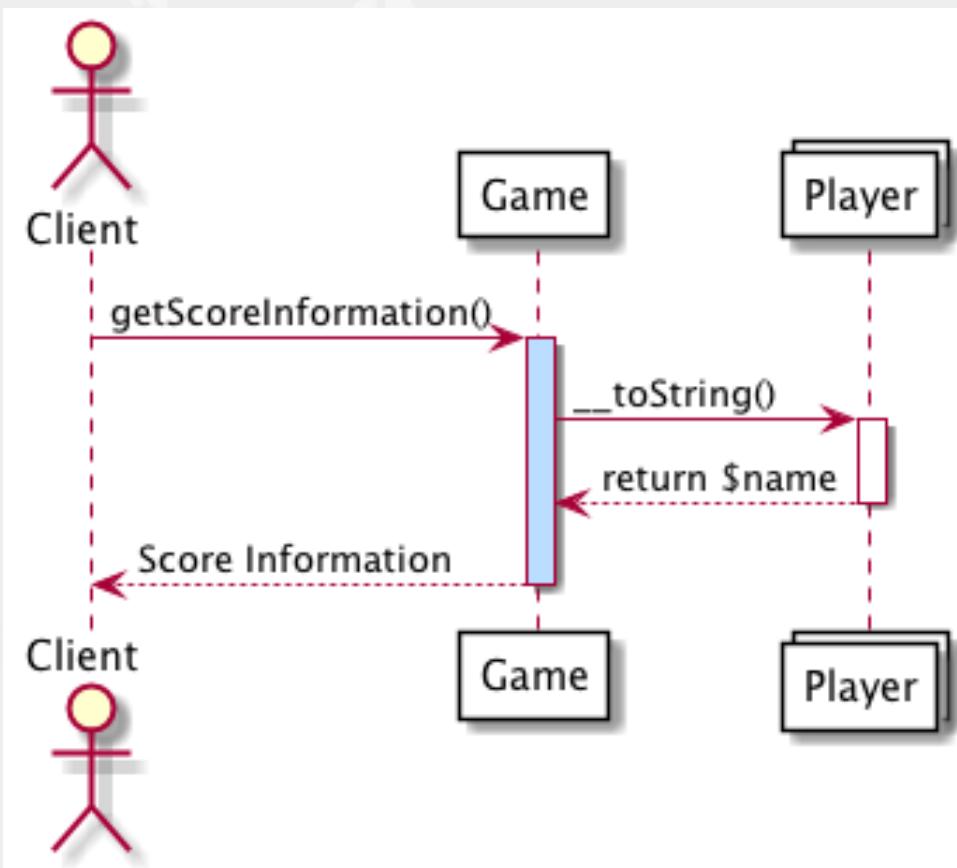
```
$resultingInformation[] = $player . ' : ' . $this->playerScores[$player];
```



- 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

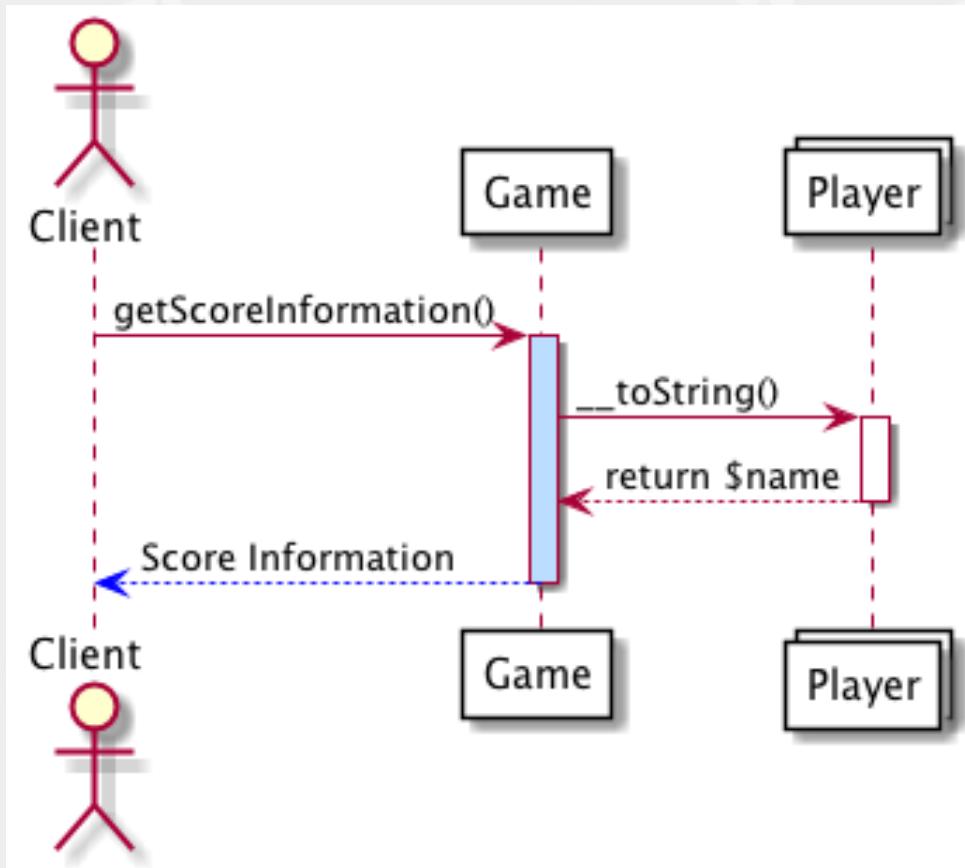
```
player->__toString
```

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

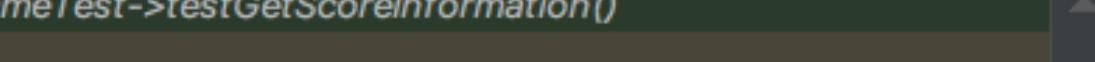
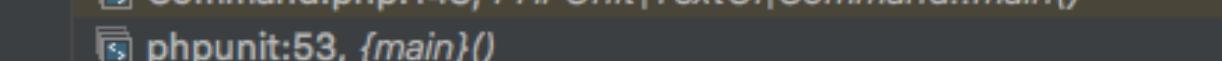
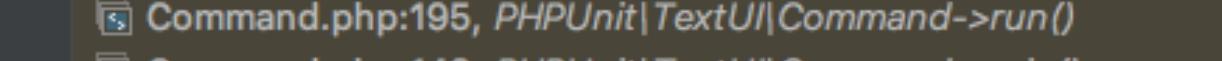
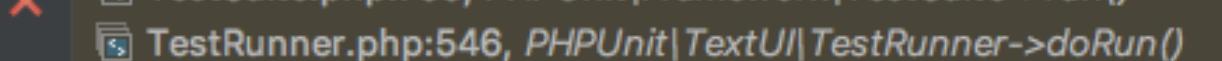
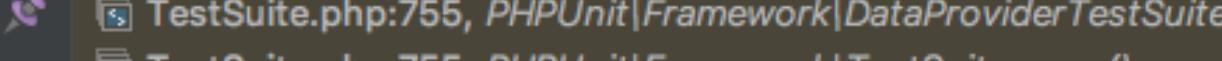
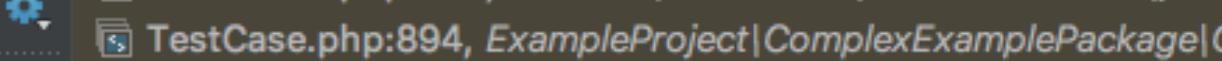
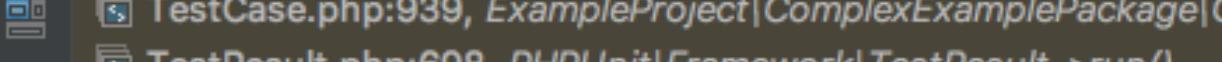
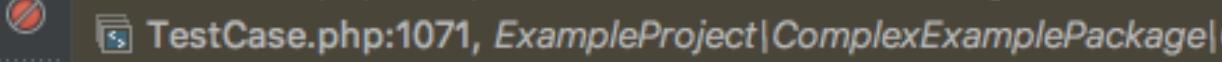
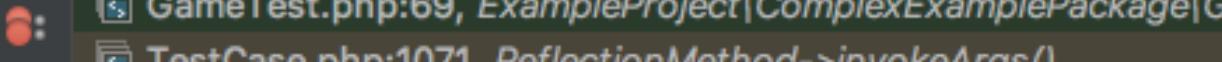
STEP OUT



- 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_expectations => [50]}, $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player> }
64     ✓
65 }
66
67 return implode( glue: ' ', $resultingInformation);
68
69 }
```

Debug: GameTest



STEP BY STEP DEBUGGING



MONKEYUSER.COM

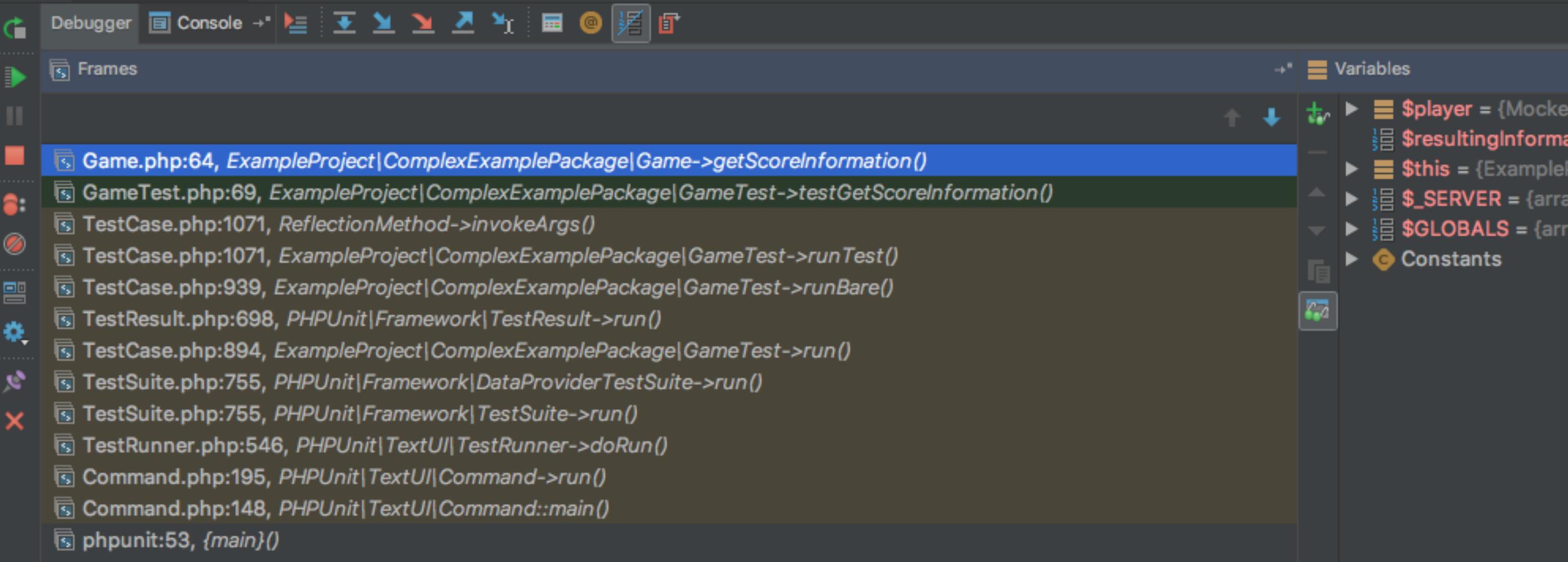
CONDITIONS

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

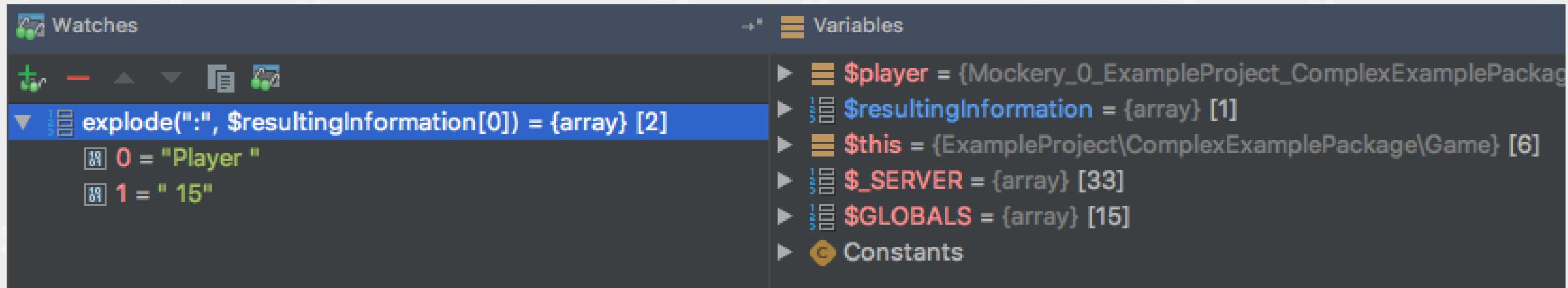
- ❖ Rekursionen
- ❖ Schleifen
- ❖ Exceptions
- ❖ In vielen Kontexten genutzter Code

```
60
61     $resultingInformation = []; $resultingInformation: [0]
62
63     foreach ($this->playerScores as $player) { $player: {_mockery_methods => [49], _mockery_expectation => [50], _mockery_isMockObject => true}, $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player> }
64     $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player>
65 }
66
67 return implode( glue: ' ', $resultingInformation);
68 }
69 }
```

Debug: GameTest



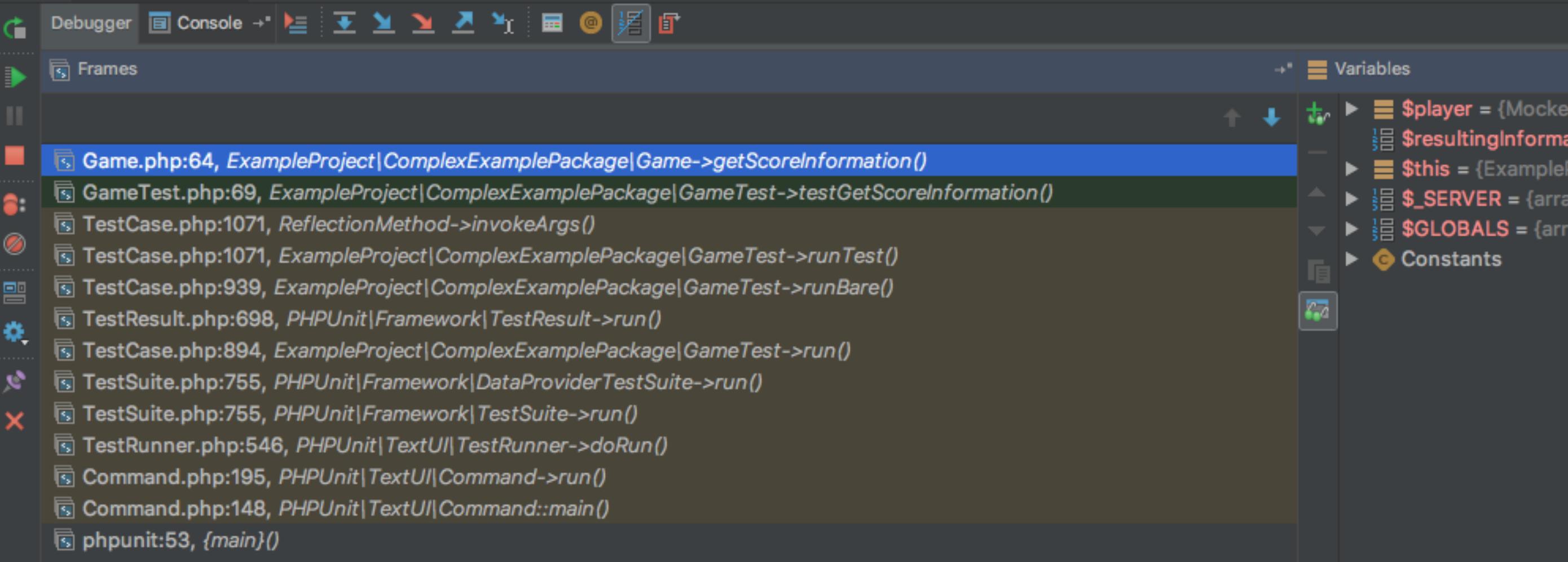
WATCHES



- 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_expectation => [50], _mockery_isMockObject => true}, $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player> }
64     $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player>
65 }
66
67 return implode( glue: ' ', $resultingInformation);
68 }
69 }
```

Debug: GameTest



CODE EXECUTION

The screenshot shows a PHP code editor with a debugger evaluation dialog open. The code being evaluated is:

```
foreach ($this->playerScores as $player) { $player: {_mockery_methods => [49], _} $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; } }
```

The expression being evaluated is:

```
implode(' | ', [$player, 'was', $this->playerScores[$player]])
```

The result of the evaluation is:

```
result = "Player was 15"
```

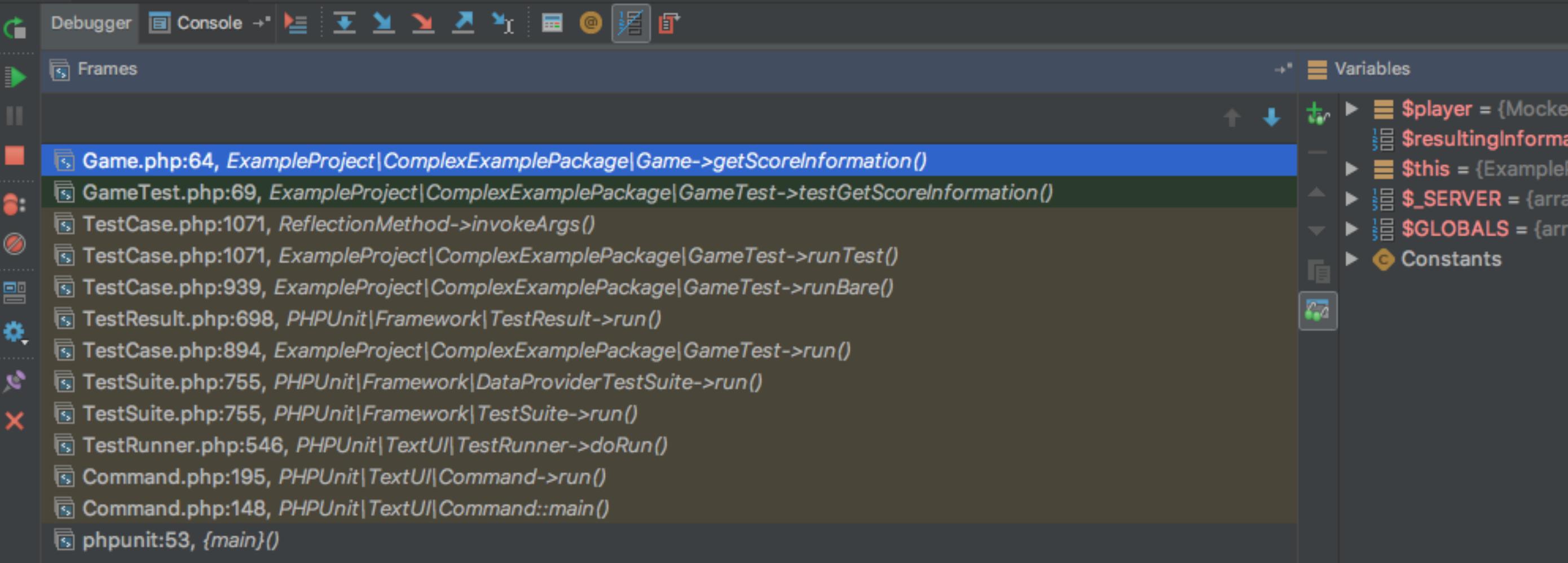
The code editor background shows lines 63 through 81 of the PHP file.

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_expectation => [50]} }
64     $resultingInformation[] = $player . ' : ' . $this->playerScores[$player]; playerScores: SplObjectStorage<Player>
65 }
66
67 return implode( glue: ' ', $resultingInformation);
68
69 }
```

Debug: GameTest



STACK TRACE

XDEBUG UND STACK TRACES



Xdebug ergänzt die Ausgabe von Stack Traces um ...

- ❖ Methodenparameter
- ❖ Lokale Variablen
- ❖ Superglobals

```
<?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:

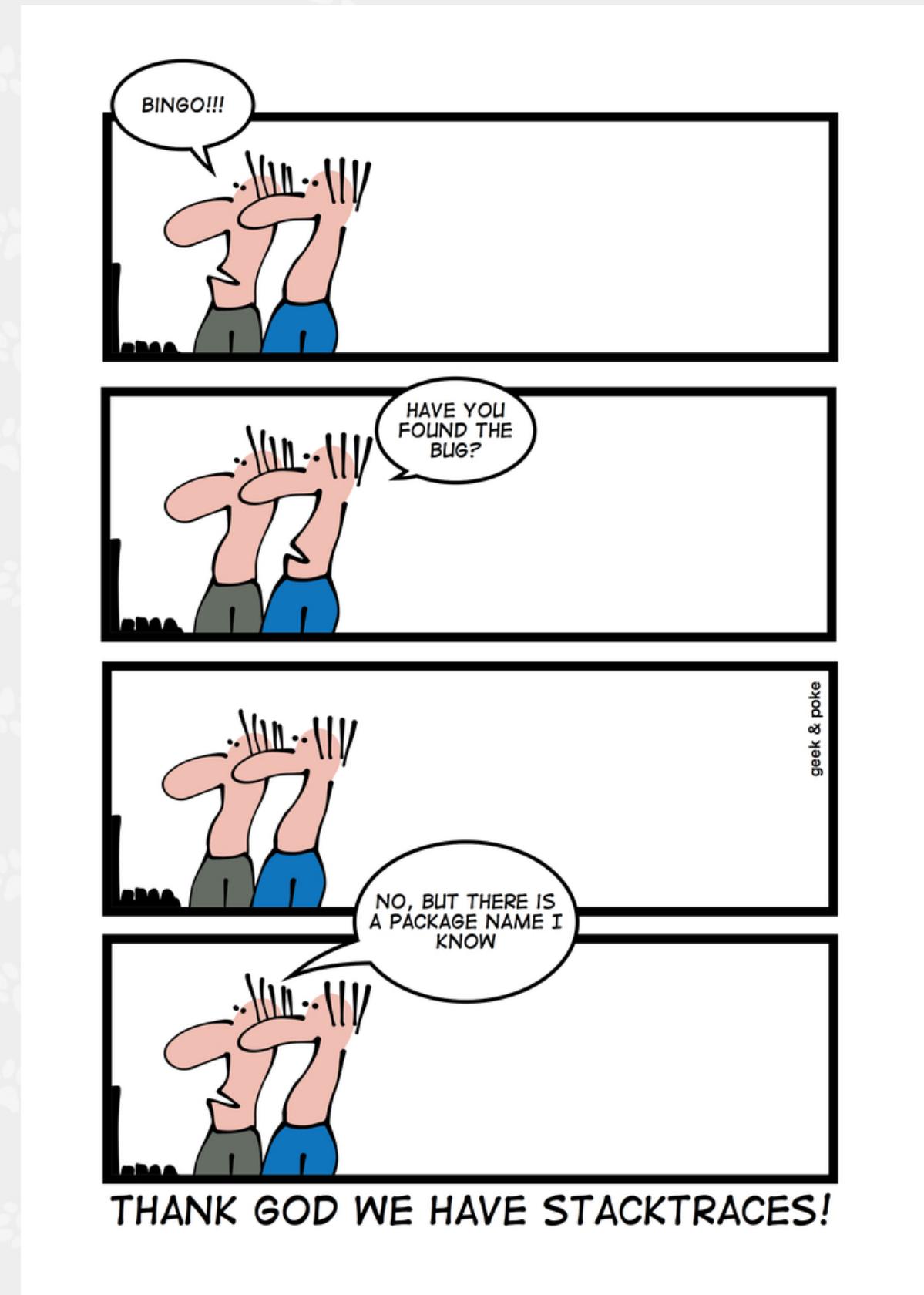
```
0.0008      358432  1. {main}() latex/xDebug/Quellen/stack-trace.php:0
0.0020      360232  2. foo($array = array (42 => FALSE, 'foo' => 912124, 43 => class
stdClass { public $bar = 100 }, 44 => resource(5) of type (stream)))
latex/xDebug/Quellen/stack-trace.php:24
```

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'
```



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

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

```
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%
▼ fx firstFunction	1.274 100,0%	1 0,0%
▼ fx wasteTime	1.000 78,5%	1 0,0%
fx php::sleep	1.000 78,5%	1 0,0%
▼ fx secondFunction	256 20,1%	1 0,0%
fx php::implode	0 0,1%	10.000 100,0%

MEMORY ALLOCATION

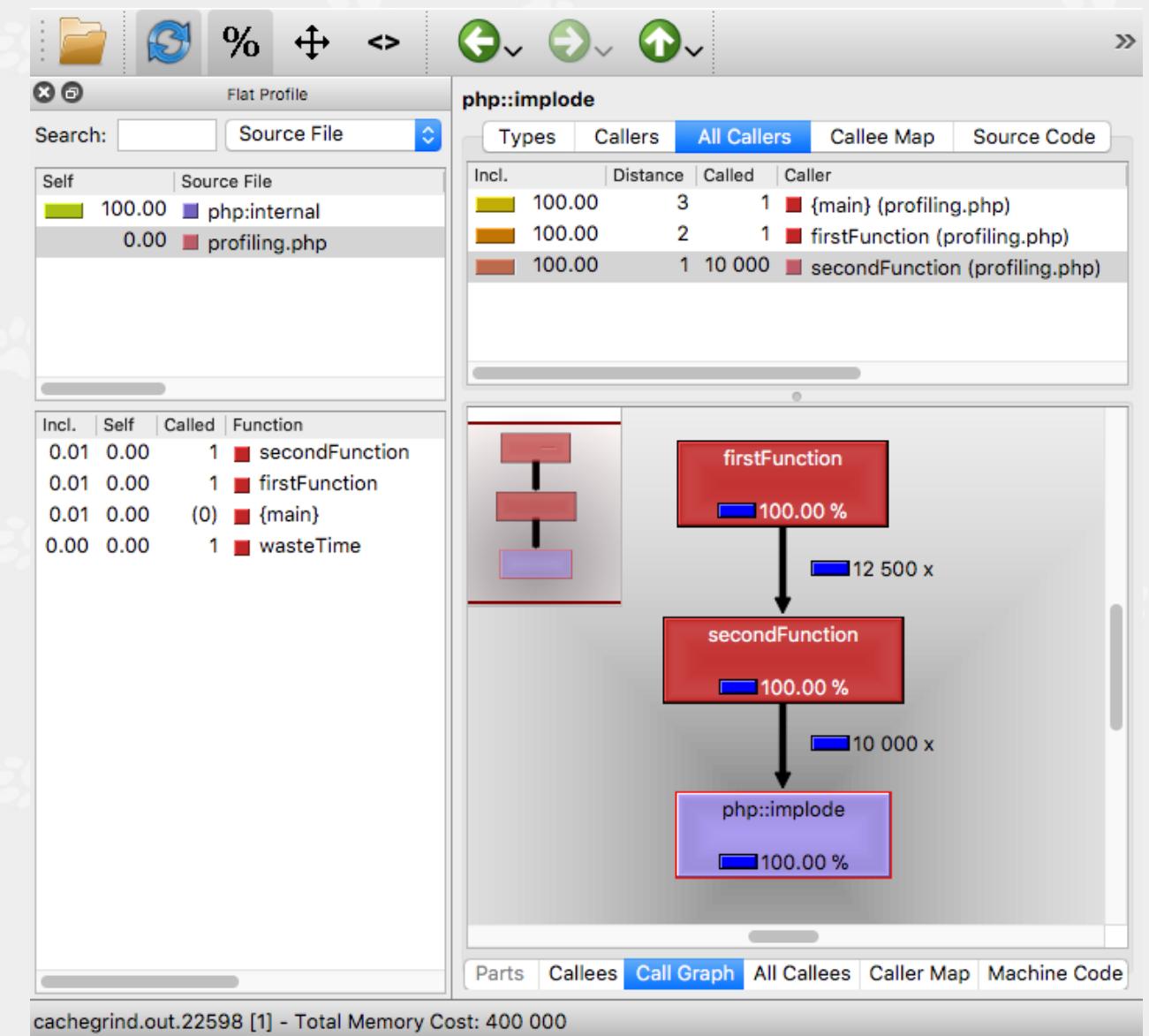
Aus Profildateien kann auch eine Statistik zum Speicherbedarf erzeugt werden

```
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

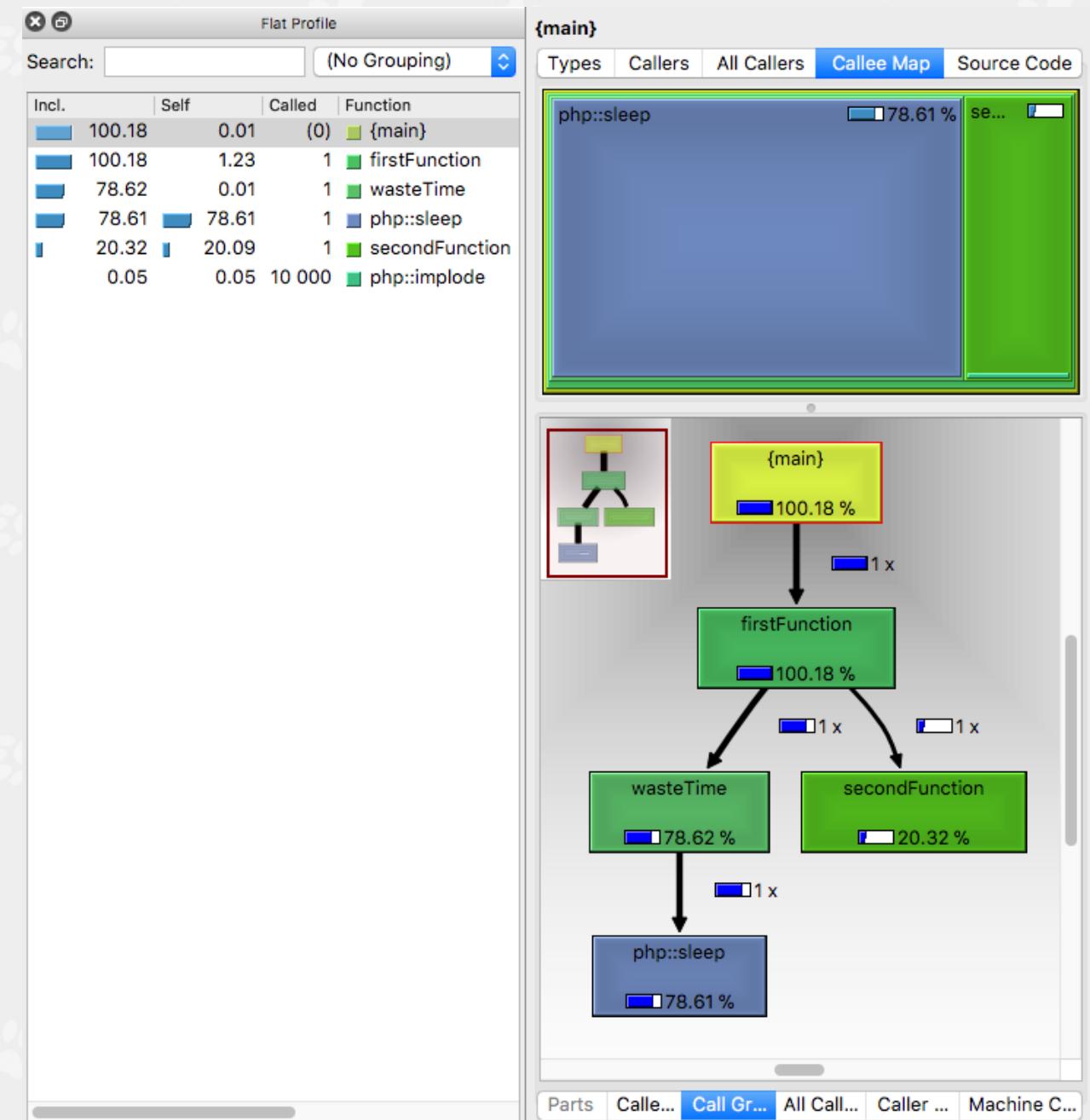
Xdebug zeichnet in Profildateien Laufzeiten von Methoden und Funktoinen auf

```
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);
}
```



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.
.....  
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)
```

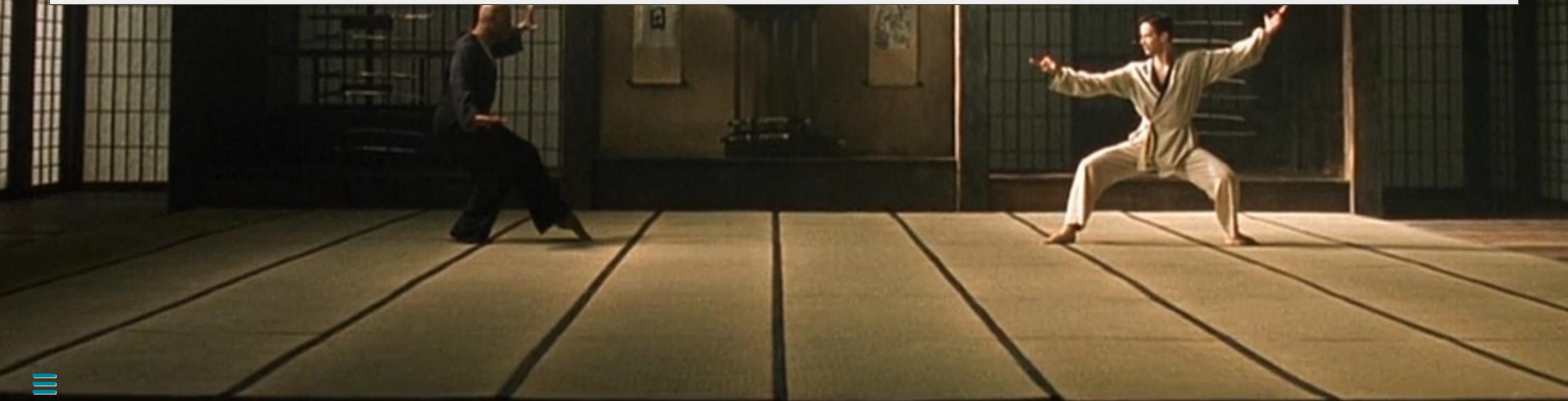
PHPUNIT COVERAGE IN PHPSTORM

TestCoverage kann auch direkt in PHPStorm angezeigt werden.

The screenshot shows a code editor in PHPStorm displaying a file with PHP code. On the left, the code is shown with line numbers from 82 to 117. Annotations on the left margin indicate coverage status: green for covered lines, yellow for partially covered lines, and orange for uncovered lines. A vertical bar on the right indicates the current line of execution. To the right of the editor is a Coverage Analysis tool window. The title bar of the window says "6% files, 29% lines". It contains a tree view of project files:

Element	Percentage
.vagrant	0% files
src	16% files, 29% lines
tests	0% files
vendor	0% files
composer.json	
phpunit.xml	

CODE DOJO



DAS DOJO

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

```
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;  
}
```



Lieben
Danke!