Errors, Exceptions, and Logging

Errors and Exceptions

In PHP, the two different concepts are closely related

Abstraction

There are rules - and they are abstractions

Parisa Tabriz Explains…

“So you want to work in security?”

If I’ve learned anything, I’ve learned that there is no single, standard, or best preparatory path.

Independent of how you acquire it, you’ll benefit from having a strong understanding of applied computer science, or how computers and software work.

“So you want to work in security?”

Much of applied computer science is about solving problems with layers of abstraction, and security is often about finding the flawed assumptions in those abstractions… and then figuring out how to best fix (or exploit) them.

https://www.freecodecamp.org/news/so-you-want-to-work-in-security-bc6c10157d23/

Conway's Law

Mel Conway
@conways_law

Access to tools. Radical simplification. Systems thinking applied to human activity.

Beverly, Massachusetts, USA melconway.com
Joined September 2012

Following

Organizations which design systems…
are constrained to produce designs which are copies of the communication structures of these organizations.

—Melvin Conway (1967)

Single Responsibility Principle — Robert C. Martin

- Every module, class, or function should have responsibility over a single part of the functionality provided by the software, and that responsibility should be entirely encapsulated by the class. All its services should be narrowly aligned with that responsibility.
- Single Responsibility Principle introduced by “Uncle Bob” Martin late 1990s: “each software module should have one and only one reason to change”
- But… we got it wrong.

Single Responsibility Principle

- Uncle Bob Martin, 2014 — *The Single Responsibility Principle is about people.*
- Gather together the things that change for the same reasons. Separate things that change for different reasons.
- A change request from the analytics group should not break (or touch) a feature developed for the marketing group and vice versa.

SRP Expresses Conway's Law

- "The Single Responsibility Principle (SRP) states that each software module should have one and only one reason to change… However it begs the question: *What defines a reason to change?"* — Martin (2014)
- Different reasons to change come from different parts of the organization
- "Single responsibility" means "group's responsibility" rather than "what this piece of code does"

Coupling and Cohesion

- In the same way that parts of your organization work with each other, structure your software with similar coupling and de-coupling
- Think of responsibilities the way your organization thinks of its responsibilities
- If one department's feature does not affect people in the other department, any change in the first department makes should not affect the other department — Single *department's* Responsibility Principle

Responsibility by “Actor”

- Suppose we have a new feature involving signup for a promotion.
- The Single Responsibility Principle tells us to separate this code from, say, the shopping cart. Later change to promotion signup should not affect (or risk breaking) the shopping cart and vice versa.
- But there’s a deeper separation needed… by actor.

Responsibility by Actor

- This (example) feature supports **users** who sign up for the promotion
- This feature also supports **admins** who review reports about the signup's progress
- Suppose we make a change to the signup page layout to fix a problem caused by the latest Chrome or Android release
- Do we want to risk breaking the admin reports? Certainly not!

Different Actors, Same Feature, Different Reasons

- The same feature (promotion signup) contains different reasons to change:
  - **User** signup changes should not affect (or break) reporting
  - **Admin** reporting changes do not happen at the same time, or for the same reasons, as user signup changes
- “Gather together the things that change for the same reasons. Separate the things that change for different reasons.”

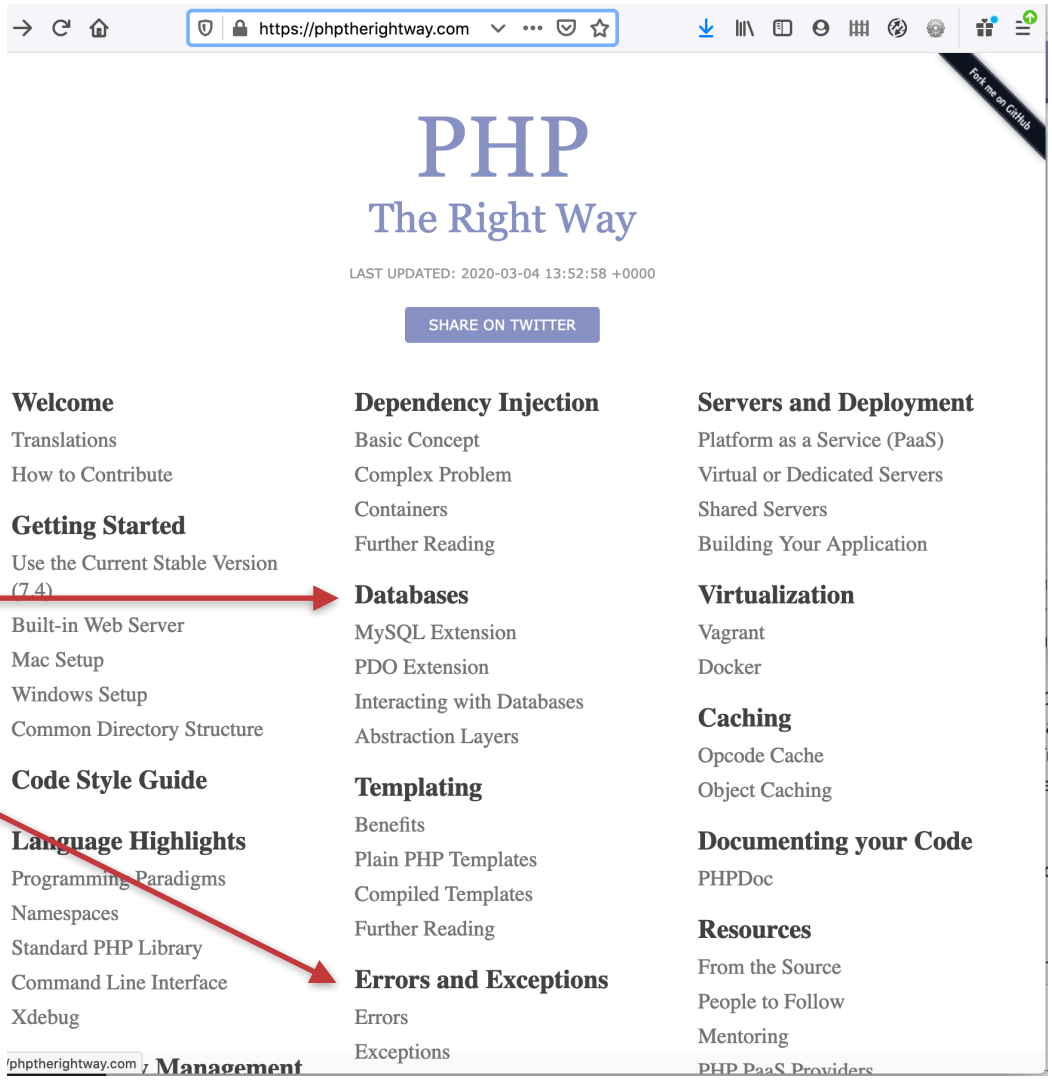
Another Important Separation

- We separate business logic from infrastructure
- But the infrastructure itself has layers and separations
  - The Model-View-Controller (MVC) pattern is a system of separations
  - Errors, Exceptions, and Logging have layers of abstraction on top of the native PHP functionality

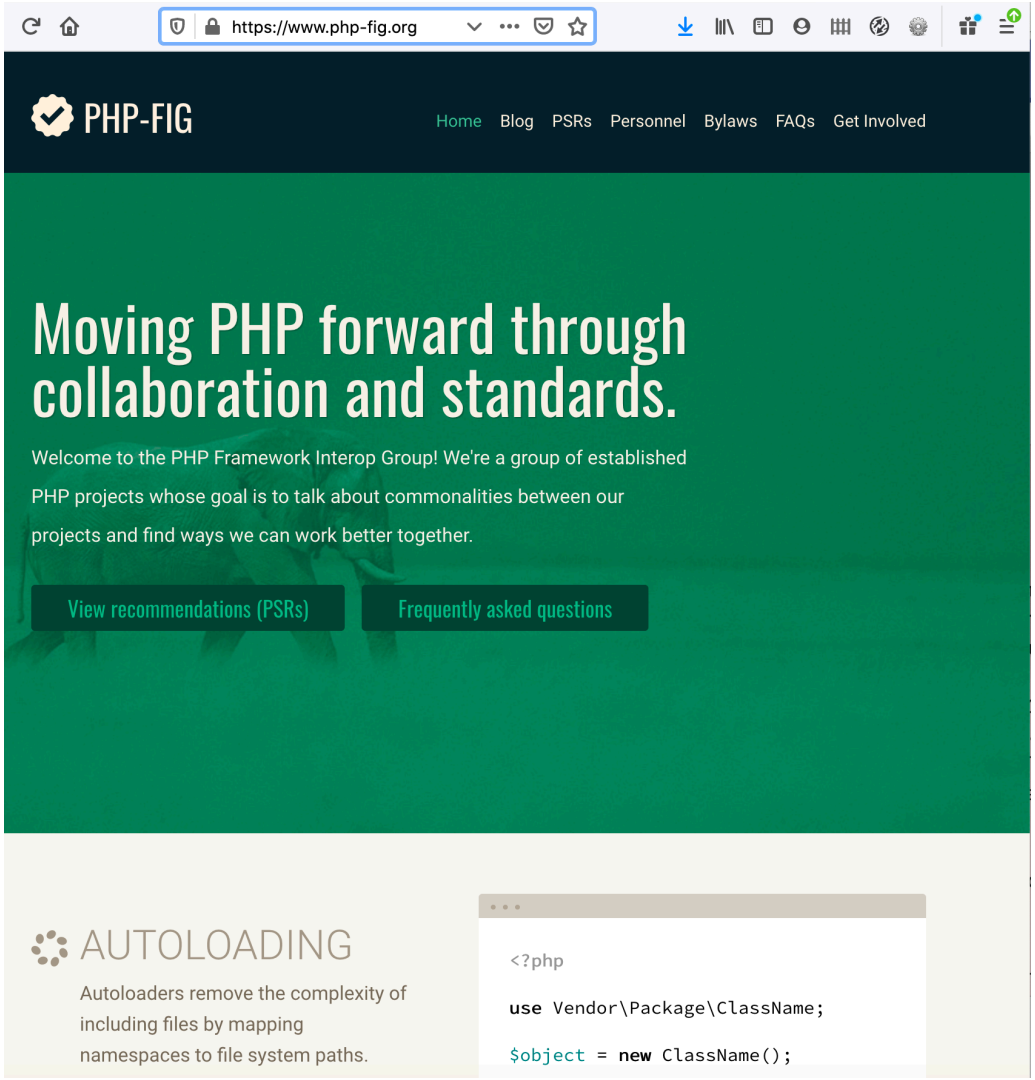
Errors, Exceptions, and Logging

- Different frameworks, with different philosophies or emphasis, provide abstractions that “wrap” the native PHP functionality
- Learn the native PHP functionality so you can understand each abstraction’s intent
- The PHP community recognizes these problems:
  - PHP The Right Way https://phptherightway.com/
  - PHP Framework Interoperability Group https://www.php-fig.org/

PHP The Right Way



PHP Framework Interoperability Group (FIG)



PSR-12 Style: Ease Sharing Code Between Authors

PSR-12: Extended Coding Style

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

Overview

This specification extends, expands and replaces PSR-2, the coding style guide and requires adherence to PSR-1, the basic coding standard.

Like PSR-2, the intent of this specification is to reduce cognitive friction when scanning code from different authors. It does so by enumerating a shared set of rules and expectations about how to format PHP code. This PSR seeks to provide a set way that

PSR-3: Logger Interface

PSR-3: Logger Interface

This document describes a common interface for logging libraries.

The main goal is to allow libraries to receive a `Psr\Log\LoggerInterface` object and write logs to it in a simple and universal way. Frameworks and CMSs that have custom needs MAY extend the interface for their own purpose, but SHOULD remain compatible with this document. This ensures that the third-party libraries an application uses can write to the centralized application logs.

The key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” in this document are to be interpreted as described in RFC 2119.

The word `implementor` in this document is to be interpreted as someone implementing the `LoggerInterface` in a log-related library or framework. Users of loggers are referred to as `user`.

Errors and Exceptions

Inspired by “PHP The Right Way”
https://phptherightway.com/#errors_and_exceptions

Errors and Exceptions

- In many exception-based languages, when something goes wrong an exception is thrown
- That’s not true for PHP
- We added exceptions to PHP with PHP 5 (if I recall aright)

Errors and Exceptions

- Many of the PHP built-in libraries were written prior to PHP 5
  - These libraries generally return FALSE for failure or error
  - Based on how C libraries worked
- Error/failure handling is gloriously inconsistent throughout PHP

How Might We Explore Errors and Exceptions?

- With unit tests!
- We can even keep the explorations as part of the code base if it makes sense to do so — they document behavior

First Step - Always

- Write a test that fails
- Ensure that it DOES fail
- This proves to yourself that you have your test harness set up correctly

ExploreErrorTest.php | SingletonTest.php | SingletonPro

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\ErrorException;

use PHPUnit\Framework\TestCase;

class ExploreErrorTest extends TestCase
{
    public function testFail(): void
    {
        static::fail(__FUNCTION__);
    }
}
```

Tests failed: 1, passed: 19 of 20 tests – 141 ms

```
Testing started at 6:49 PM ...
/usr/local/opt/php@7.3/bin/php /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/php
PHPUnit 8.5.3 by Sebastian Bergmann and contributors.


testFail
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php:12



Time: 217 ms, Memory: 14.00 MB

FAILURES!
Tests: 20, Assertions: 36, Failures: 1.
Process finished with exit code 1
```

Notice Error

```
10    public function testUndefined(): void
11    {
12        $a = $foo;
13        self::assertTrue( condition: true,  message: 'Dummy assertion');
14    }
```

Notice Errors Mostly Ignored

- PHP will generally ignore a notice error and continue on
- Depends on server configuration
- PHP processing continues even though we now have corrupt or uninitialized data
- Watch for notice errors in your dev environment and FIX THEM

Error Severities

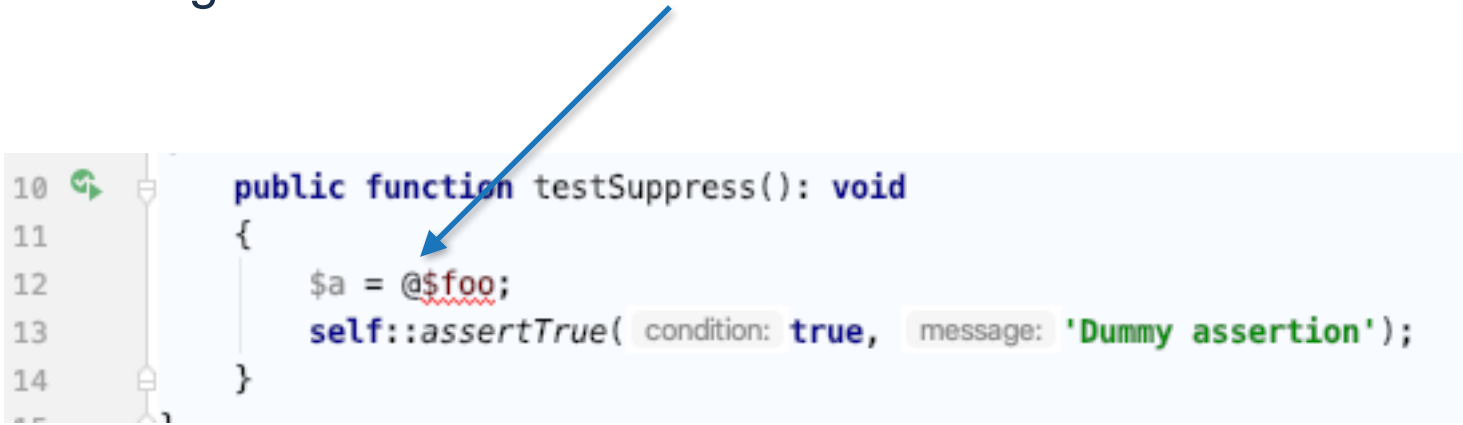
- E_NOTICE, E_WARNING, E_ERROR
- Other errors exist such as “this feature is deprecated”
- Turn on E_STRICT to find places code should be improved
- Use error_reporting() to change error levels being reported

```
10    public function testUndefined(): void
11    {
12        error_reporting( level: 0);
13        $a = $foo;
14        self::assertTrue( condition: true,  message: 'Dummy assertion');
15    }
```

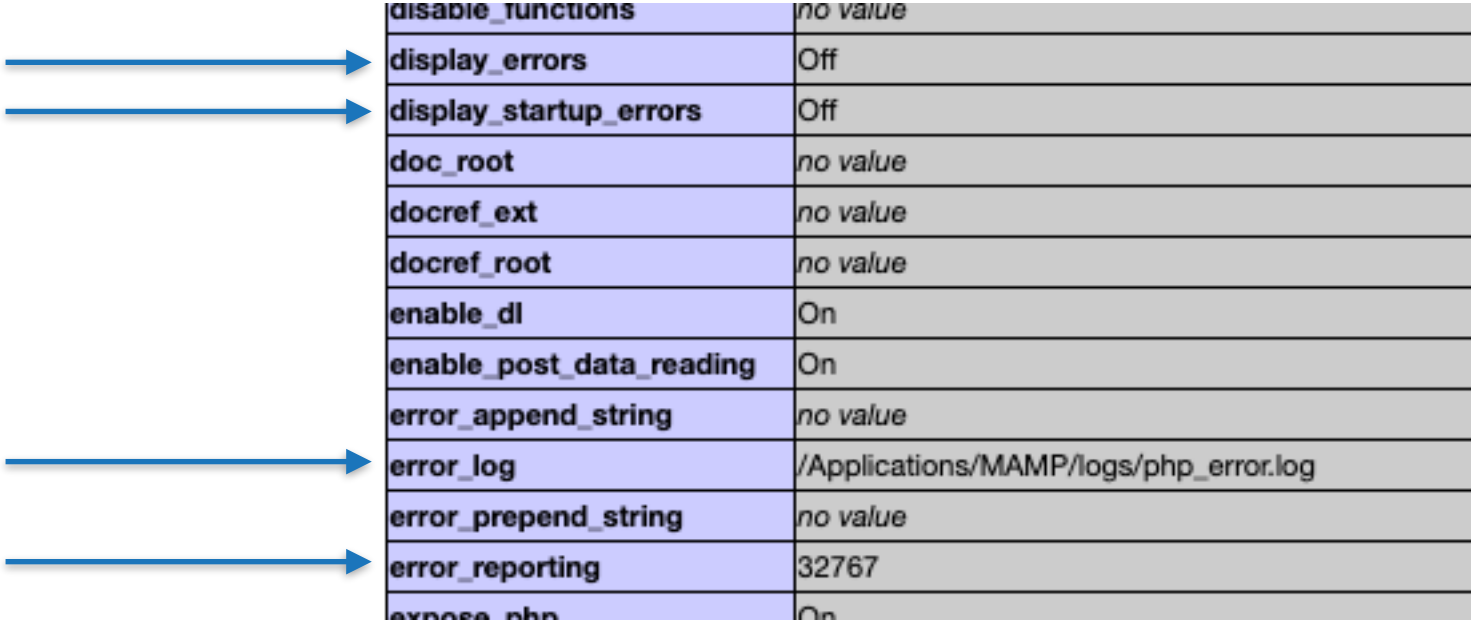
```
Tests passed: 20 of 20 tests – 192 ms
Testing started at 7:07 PM ...
/usr/local/opt/php@7.3/bin/php /Us
PHPUnit 8.5.3 by Sebastian Bergman

Time: 284 ms, Memory: 14.00 MB

OK (20 tests, 36 assertions)
Process finished with exit code 0
```

Inline Error Suppression - Don’t Do This!

- Place an @ character in front of the expression
- This suppresses any error from being emitted by that expression — it won’t be logged anywhere
- Remarkably bad idea to cover up errors — but you may see it in existing PHP code



Error Log in ini file, shown by phpinfo()



disable_functions	*no value*
display_errors	Off
display_startup_errors	Off
doc_root	*no value*
docref_ext	*no value*
docref_root	*no value*
enable_dl	On
enable_post_data_reading	On
error_append_string	*no value*
error_log	/Applications/MAMP/logs/php_error.log
error_prepend_string	*no value*
error_reporting	32767
expose_php	On

Error Logs

```
Edwards-MacBook-Pro:design-pattern edwardbarnard$ pushd /Applications/MAMP/logs/
/Applications/MAMP/logs ~/PhpstormProjects/DesignPattern/design-pattern
Edwards-MacBook-Pro:logs edwardbarnard$ ls -al
total 32
drwxrwxrwx   6 edwardbarnard  admin   192 Apr  1 13:53 .
drwxrwxr-x@ 20 edwardbarnard  admin   640 Feb 17 08:30 ..
-rw-r--r--   1 edwardbarnard  admin   998 Apr  1 16:18 apache_error.log
-rw-r--r--   1 edwardbarnard  admin     0 Apr  1 13:17 cloud_mamp.log
-rw-r-----   1 edwardbarnard  admin  6887 Apr  3 08:31 mysql_error_log.err
-rw-r--r--   1 edwardbarnard  admin   122 Apr  1 13:53 php_error.log
Edwards-MacBook-Pro:logs edwardbarnard$
```

```php
<?php
declare(strict_types=1);

namespace App\Test\TestCase\ErrorException;

use InvalidArgumentException;
use PHPUnit\Framework\TestCase;

class ExploreErrorTest extends TestCase
{
    public function testException(): void
    {
        try {
            $this->attitude();
        } catch (InvalidArgumentException $e) {
            static::fail('Exception message: ' . $e->getMessage());
        }
    }

    private function attitude(): void
    {
        throw new InvalidArgumentException( message: 'Wrong again');
    }
}
```

```php
try {
    $this->attitude();
} catch (InvalidArgumentException $e) {
    static::fail('Exception message: ' . $e->getMessage());
}
```

```
Testing started at 7:28 PM ...
/usr/local/opt/php@7.3/bin/php /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/ph
PHPUnit 8.5.3 by Sebastian Bergmann and contributors.


Exception message: Wrong again
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php:16



Time: 287 ms, Memory: 14.00 MB

FAILURES!
Tests: 20, Assertions: 36, Failures: 1.
Process finished with exit code 1
```

```php
try {
    $this->attitude();
} catch (InvalidArgumentException $e) {
    static::fail('Exception message: ' . $e->getTraceAsString());
}
```

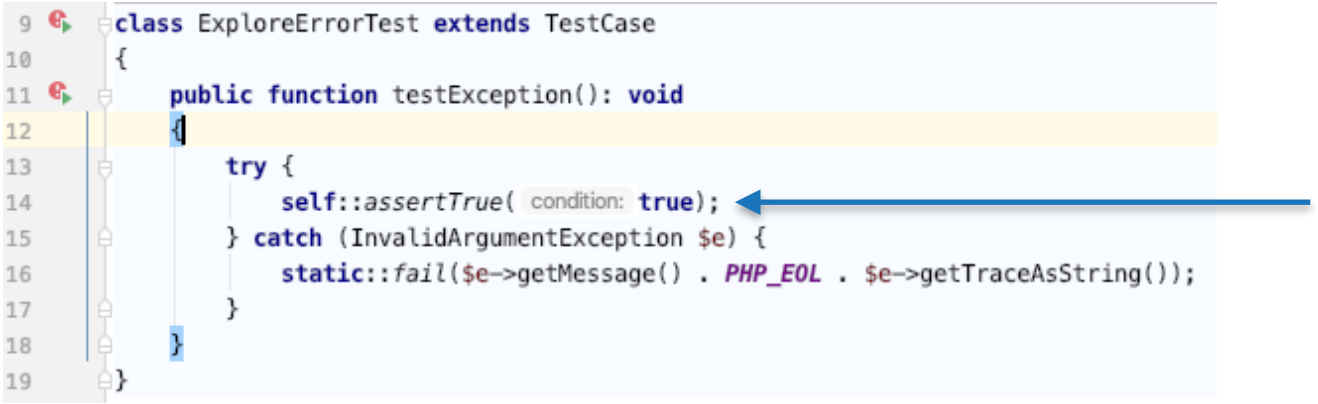
```
Testing started at 7:32 PM ...
/usr/local/opt/php@7.3/bin/php /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/phpunit --configuration /Use
PHPUnit 8.5.3 by Sebastian Bergmann and contributors.


Exception message: #0 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php(14): App\
#1 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(1415): App\Test\TestCase\Erro
#2 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(1035): PHPUnit\Framework\Test
#3 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestResult.php(691): PHPUnit\Framework\Tes
#4 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(763): PHPUnit\Framework\TestR
#5 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestSuite.php(597): PHPUnit\Framework\Test
#6 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestSuite.php(597): PHPUnit\Framework\Test
#7 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/TestRunner.php(621): PHPUnit\Framework\TestSu
#8 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/Command.php(200): PHPUnit\TextUI\TestRunner->
#9 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/Command.php(159): PHPUnit\TextUI\Command->run
#10 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/phpunit(61): PHPUnit\TextUI\Command::main()
#11 {main}
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php:16
```

The Traceback Trick

- When I am trying to trace through PHP code, sometimes I can't tell what chain events got to that code
- I can throw an exception and examine the backtrace
- Like this…

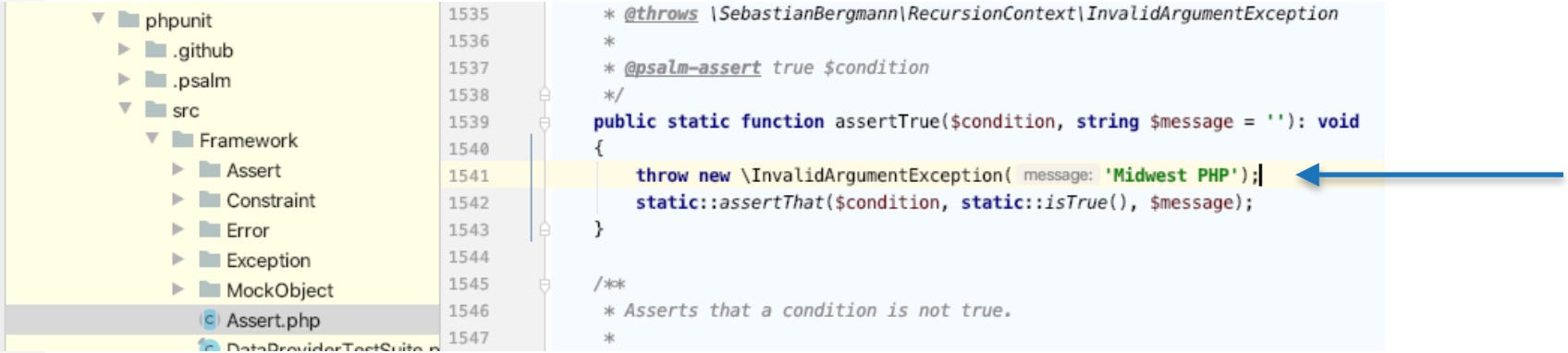
Place the mysterious call inside try / catch

- We want to trace assertTrue()



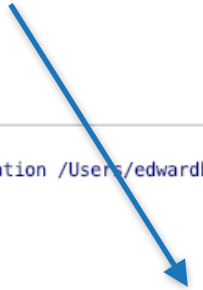
Throw exception inside code you need to trace

- Here is assertTrue(), line 1539 in one of the PHPUnit files



Run the Code

```
11    public function testException(): void
12    {
13        try {
14            self::assertTrue( condition: true);
15        } catch (InvalidArgumentException $e) {
16            static::fail($e->getMessage() . PHP_EOL . $e->getTraceAsString());
17        }
18    }
```



```
Testing started at 7:44 PM ...
/usr/local/opt/php@7.3/bin/php /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/phpunit --configuration /Users/edwardbarnard/Phpsto
PHPUnit 8.5.3 by Sebastian Bergmann and contributors.


Midwest PHP
#0 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php(14): PHPUnit\Framework\Assert::assertTrue(true)
#1 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(1415): App\Test\TestCase\ErrorException\ExploreError
#2 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(1035): PHPUnit\Framework\TestCase->runTest()
#3 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestResult.php(691): PHPUnit\Framework\TestCase->runBare()
#4 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestCase.php(763): PHPUnit\Framework\TestResult->run(Object(App\T
#5 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestSuite.php(597): PHPUnit\Framework\TestCase->run(Object(PHPUni
#6 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/Framework/TestSuite.php(597): PHPUnit\Framework\TestSuite->run(Object(PHPUn
#7 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/TestRunner.php(621): PHPUnit\Framework\TestSuite->run(Object(PHPUnit
#8 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/Command.php(200): PHPUnit\TextUI\TestRunner->doRun(Object(PHPUnit\Fr
#9 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/src/TextUI/Command.php(159): PHPUnit\TextUI\Command->run(Array, true)
#10 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/vendor/phpunit/phpunit/phpunit(61): PHPUnit\TextUI\Command::main()
#11 {main}
 /Users/edwardbarnard/PhpstormProjects/DesignPattern/design-pattern/tests/TestCase/ErrorException/ExploreErrorTest.php:16
```

Logging

This will be quick

So Many Options!

- Why are we logging?
- Who consumes the logs?
- Performance considerations
- Sanitized/sensitive data

“Proper” Logging

- Looks complicated
- Unnecessary (in my view) learning curve
- Let your framework simplify
- Entire companies are built on log file analysis

Logging

Symfony comes with a minimalist PSR-3 logger: `Logger`. In conformance with the twelve-factor app methodology, it sends messages starting from the `WARNING` level to stderr.

Summary

- **Errors and Exceptions** — PHP is gloriously inconsistent
- **Single Responsibility Principle** — Gather together the things that change at the same time for the same reason, requested by the same person, and separate the things that change at different times for different reasons and support different persons (actors)
- **Conway's Law** — Organizations which design systems… are constrained to produce designs which are copies of the communication structures of those organizations
- **Cole's Law** — Thinly sliced cabbage