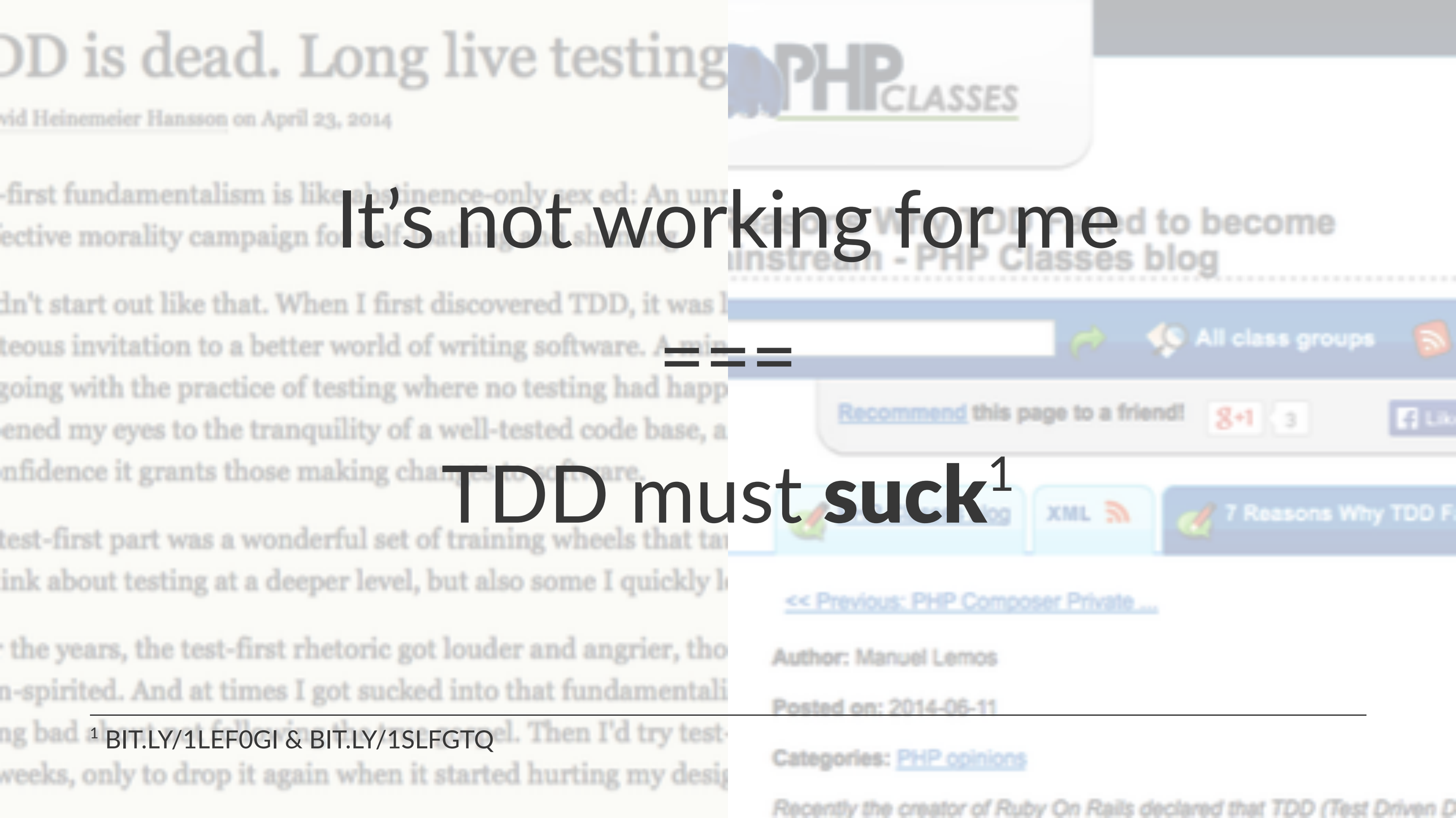
Why Your Test Suite Sucks

(and what you can do about it)

by **Ciaran McNulty** at **PHPUK 2015**



It’s not working for me

===

TDD must **suck**[1]

[1] BIT.LY/1LEF0GI & BIT.LY/1SLFGTQ

The problem is not **TDD**

The problem is **your suite**

Value = Benefits - Costs

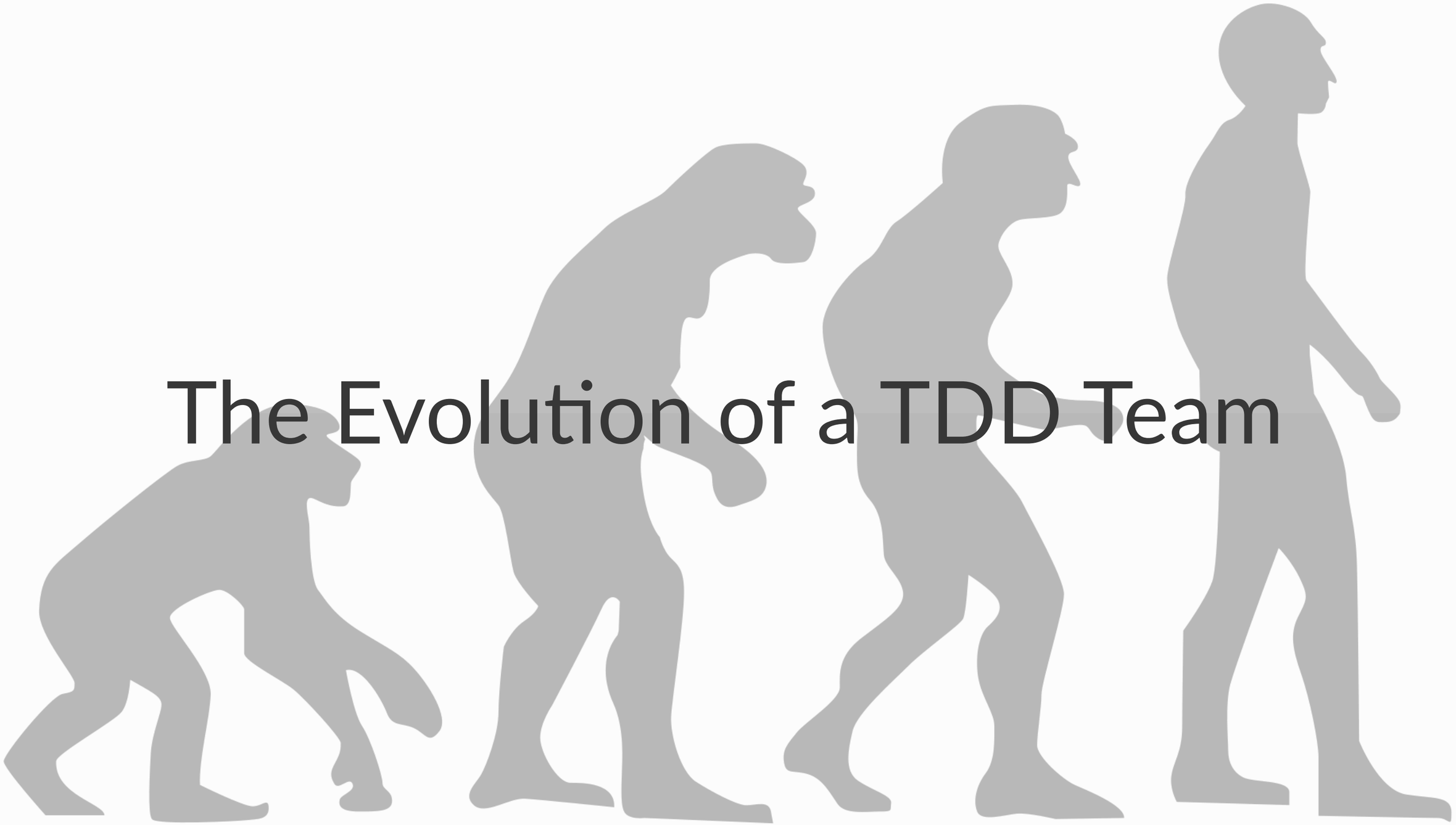
Suites suck when you **aren't getting all the benefits** of TDD

Suites suck when the tests **are hard to maintain**

Reason 1:

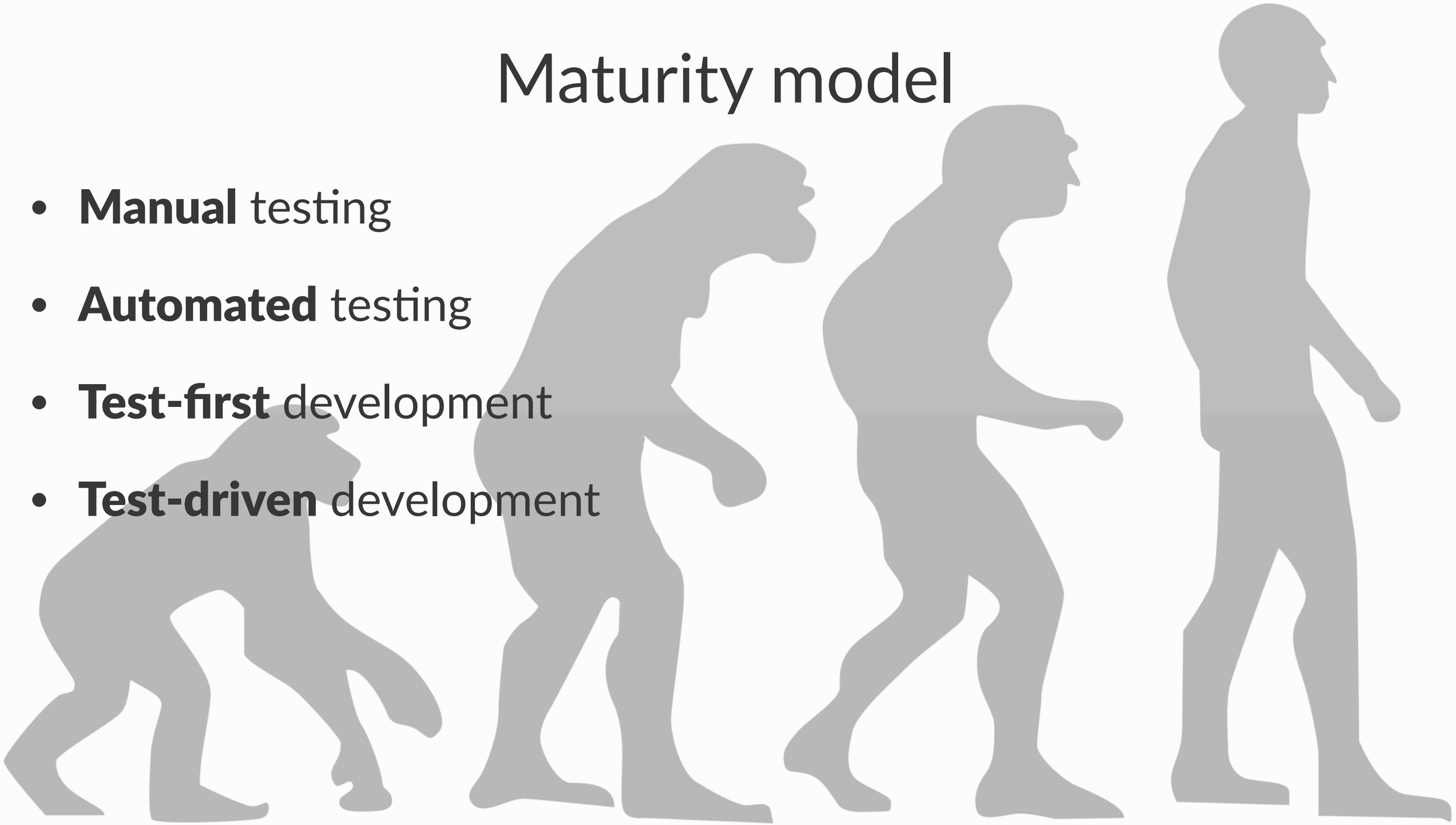
You don't have a test suite

The Evolution of a TDD Team



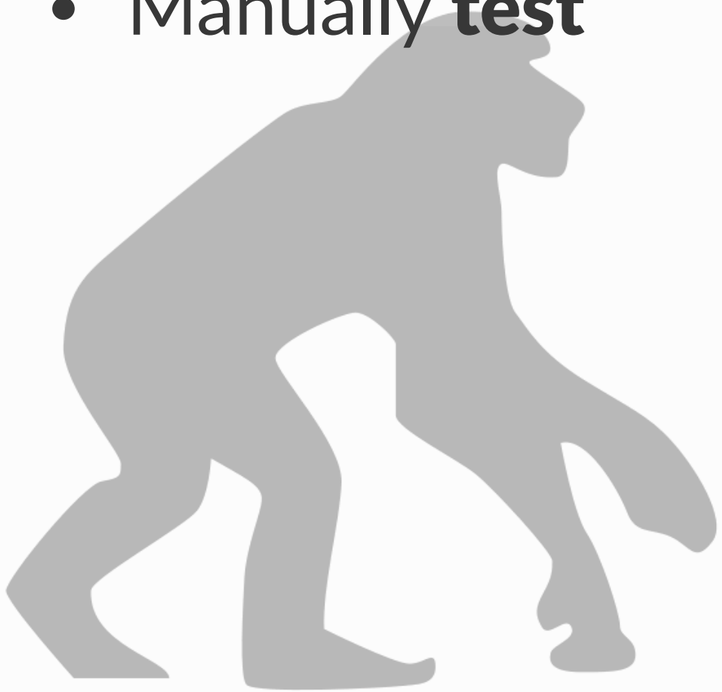
Maturity model

- **Manual** testing
- **Automated** testing
- **Test-first** development
- **Test-driven** development



Manual testing

- **Design** an implementation
- **Implement** that design
- Manually **test**



Growing from Manual to Automated

- “When we **make changes**, it breaks things!”
- “We spend **a lot of time** on manual testing”
- “**Bugs** keep making it into production”

Enabled by:
- Tutorials, training
- Team policies

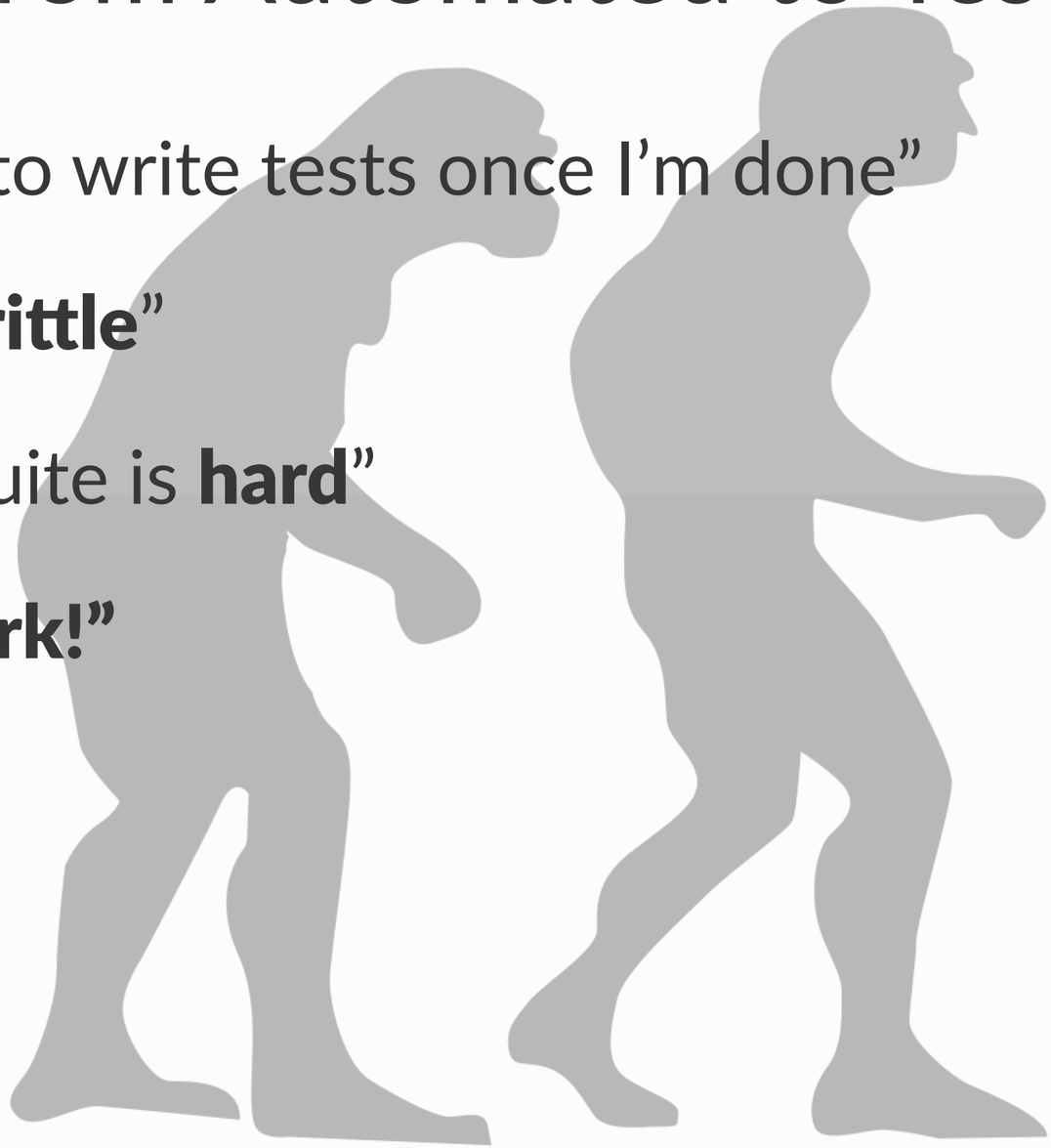
Automated testing

- **Design** an implementation
- **Implement** that design
- **Write tests** to verify it

Growing from Automated to Test-first

- “It’s a **pain** having to write tests once I’m done”
- “My test suite is **brittle**”
- “Maintaining the suite is **hard**”
- **”TDD Doesn’t Work!”**

Supported by:
- Coaching, pairing
- Leadership



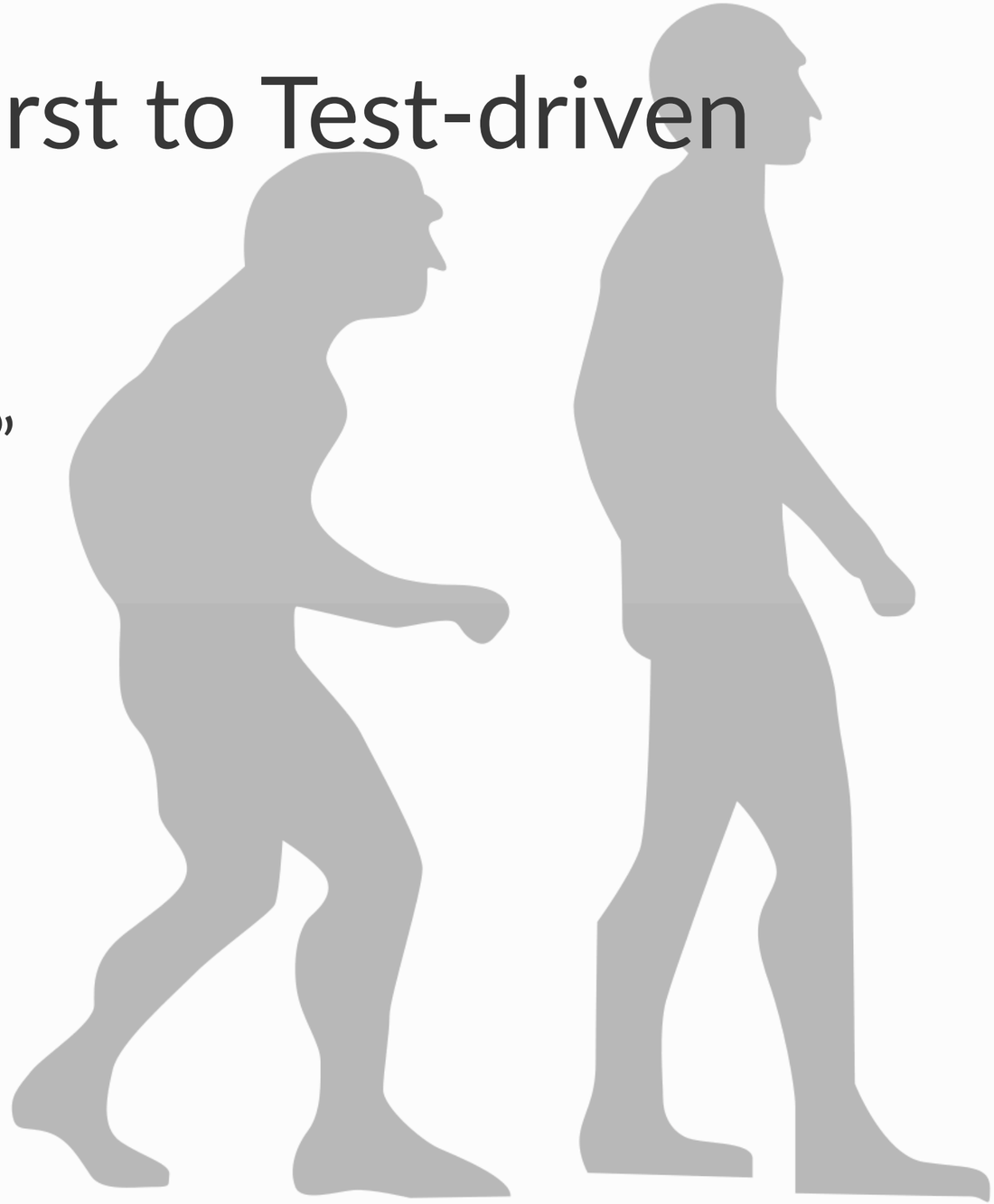
Test-first development

- **Design** an implementation
- **Write tests** to verify that implementation
- Implement that design

Growing from Test-first to Test-driven

- “Test suite is **still** brittle”
- “Maintaining the suite is **still** hard”
- **”TDD Doesn’t Work!”**

Supported by:
- Practice



Test-driven development

- **Write tests** that describe behaviour
- **Implement** a system that has that behaviour

Problems:
- “I don’t know what to do with all this **spare time**!”
- “Where should we keep this **extra money**?”

Reason 2:

You are testing implementation

Problem: Mocking Queries

- A query is a method that **returns a value without side effects**
- We **shouldn’t care** how many times (if any) a query is called
- Specifying that detail **makes change harder**

```
function testItGreetsPeopleByName()
{
    $user = $this->getMock('User');

    $user->expects($this->once())
         ->method('getName')
         ->willReturn('Ciaran');

    $greeting = $this->greeter->greet($user);

    $this->assertEquals('Hello, Ciaran!', $greeting);
}
```

```
function testItGreetsPeopleByName()
{
    $user = $this->getMock('User');

    $user->method('getName')
         ->willReturn('Ciaran');

    $greeting = $this->greeter->greet($user);

    $this->assertEquals('Hello, Ciaran!', $greeting);
}
```

Don’t mock queries, just use stubs

Do you really care how many times it's called?

Problem: Testing the sequence of calls

- Most of the time we don’t care **what order calls are made in**
- Testing the sequence makes it **hard to change**

```php
public function testBasketTotals()
{
    $priceList = $this->getMock('PriceList');

    $priceList->expect($this->at(0))
              ->method('getPrices')
              ->willReturn(120);

    $priceList->expect($this->at(1))
              ->method('getPrices')
              ->willReturn(200);

    $this->basket->add('Milk', 'Bread');
    $total = $this->basket->calculateTotal();

    $this->assertEqual(320, $total);
}
```

```php
public function testBasketTotals()
{
    $priceList = $this->getMock('PriceList');

    $priceList->method('getPrices')
              ->will($this->returnValueMap([
                 ['Milk', 120],
                 ['Bread', 200]
              ]));

    $this->basket->add('Milk', 'Bread');
    $total = $this->basket->calculateTotal();

    $this->assertEqual(320, $total);
}
```

Test behaviour you care about, don't test implementation

PROTIP: The best way to do this is to not have an implementation yet

Reason 3:

Because your design sucks

Problem: Too many Doubles

- It is **painful** to have to **double lots of objects** in your test
- It is a signal your **object has too many dependencies**

```php
public function setUp()
{
    $chargerules = $this->getMock('ChargeRules');
    $chargetypes = $this->getMock('ChargeTypes');
    $notifier = $this->getMockBuilder('Notifier')
                     ->disableOriginalConstructor()->getMock();

    $this->processor = new InvoiceProcessor(
         $chargerules, $chargetypes, $notifier
    );
}
```

```php
class InvoiceProcessor
{
    public function __construct(
        ChargeRules $chargeRules,
        ChargeTypes $chargeTypes,
        Notifier $notifier
    )
    {
        $this->chargeRules = $chargeRules;
        $this->chargeTypes = $chargeTypes;
        $this->notifier = $notifier;
    }

    // …
```

Too many doubles in your test mean your class has **too many dependencies**

Problem: Stubs returning stubs

- A related painful issue is **stubs returning a stubs**
- It is a signal our object **does not have the right dependencies**

```
public function it_notifies_the_user(
    User $user,
    Contact $contact,
    Email $email,
    Emailer $emailer
)
{
    $this->beConstructedWith($emailer);

    $user->getContact()->willReturn($contact);
    $contact->getEmail()->willReturn($email);

    $this->notify($user);

    $emailer->sendTo($email)->shouldHaveBeenCalled();
}
```

```
class Notifier
{
    public function __construct(Emailer $emailer)
    {
        // …
    }

    public function notify(User $user)
    {
        $email = $user->getContact()->getEmail()

        $this->emailer->sendTo($email);
    }
}
```

```
class Notifier
{
    public function __construct(Emailer $emailer)
    {
        // …
    }

    public function notify(User $user)
    {
        $this->emailer->sendTo($user);
    }
}
```

```
class Notifier
{
    public function __construct(Emailer $emailer)
    {
        // …
    }

    public function notify(Email $email)
    {
        $this->emailer->sendTo($email);
    }
}
```

```
public function it_notifies_the_user(
    Email $email,
    Emailer $emailer
)
{
    $this->beConstructedWith($emailer);

    $this->notify($email);

    $emailer->sendTo($email)->shouldHaveBeenCalled();
}
```

Stubs returning stubs indicate **dependencies are not correctly defined**

Problem: Partially doubling the object under test

- A **common question** is “how can I mock methods on the SUT?”
- Easy to do in some tools, **impossible in others**
- It is a signal **our object has too many responsibilities**

```php
class Form
{
    public function handle($data)
    {
        if ($this->validate($data)) {
            return 'Valid';
        }

        return 'Invalid';
    }

    protected function validate($data)
    {
        // do something complex
    }
}
```

```
function testItOutputsMessageOnValidData()
{
    $form = $this->getMock('Form', ['validate']);
    $form->method('validate')->willReturn(true);

    $result = $form->handle([]);

    $this->assertSame('Valid', $result);
}
```

```php
class Form
{
    protected function __construct(Validator $validator)
    {
        // …
    }

    public function handle($data)
    {
        if ($this->validator->validate($data)) {
            return 'Valid';
        }

        return 'Invalid';
    }
}
```

```
function testItOutputsMessageOnValidData()
{
    $validator = $this->getMock(‘Validator’);
    $validator->method('validate')->willReturn(true);

    $form = new Form($validator);

    $result = $form->handle([]);

    $this->assertSame('Valid', $result);
}
```

If you want to mock part of the SUT it means **you should really have more than one object**

How to be a better software designer:

- **Stage 1**: Think about your code before you write it
- **Stage 2**: Write down those thoughts
- **Stage 3**: Write down those thoughts as code

How long will it take without TDD?

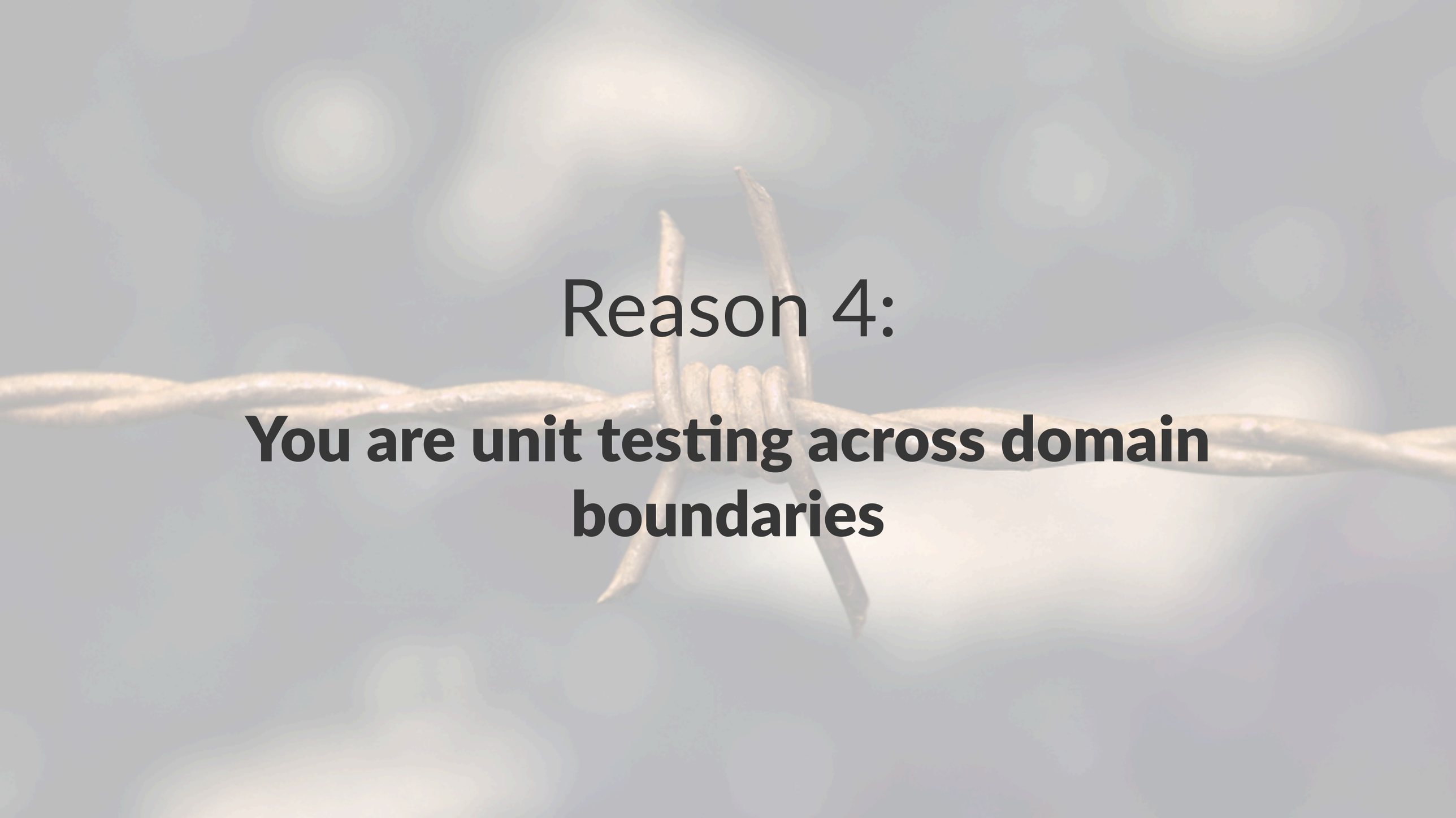
- **1 hour** thinking about how it should work
- **30 mins** Implementing it
- **30 mins** Checking it works

How long will it take with TDD?

- **1 hour** Thinking in code about how it should work
- **30 mins** Implementing it
- **1 min** Checking it works

Use your test suite as a way to
reason about the code

Customers should not dictate your **design process**



Reason 4:

You are unit testing across domain boundaries

Problem: Testing 3rd party code

- When we extend a library or framework we **use other people's code**
- To test our code we end up **testing their code** too
- This time is **wasted** – we should trust their code works

```php
class UserRepository extends \Framework\Repository
{
    public function findByEmail($email)
    {
        $this->find(['email' => $email]);
    }
}
```

```
public function it_finds_the_user_by_email(
    \Framework\DatabaseConnection $db,
    \Framework\SchemaCreator $schemaCreator,
    \Framework\Schema $schema,
    User $user
)
{
    $this->beConstructedWith($db, $schemaCreator);

    $schemaCreator->getSchema()->willReturn($schema);
    $schema->getMappings()->willReturn('email'=>'email');

    $user = $this->findByEmail('bob@example.com');

    $user->shouldHaveType('User');
}
```

```php
class UserRepository
{
    private $repository;

    public function __construct(\Framework\Repository $repository)
    {
        $this->repository = $repository;
    }

    public function findByEmail($email)
    {
        return $this->repository->find(['email' => $email]);
    }
}
```

```
public function it_finds_the_user_by_email(
    Repository $repository,
    User $user
)
{
    $this->beConstructedWith($repository);

    $repository->find(['email' => 'bob@example.com'])
               ->willReturn($user);

    $this->findByEmail('bob@example.com');
         ->shouldEqual($user);
}
```

When you extend third party code, you **take responsibility** for its design decisions

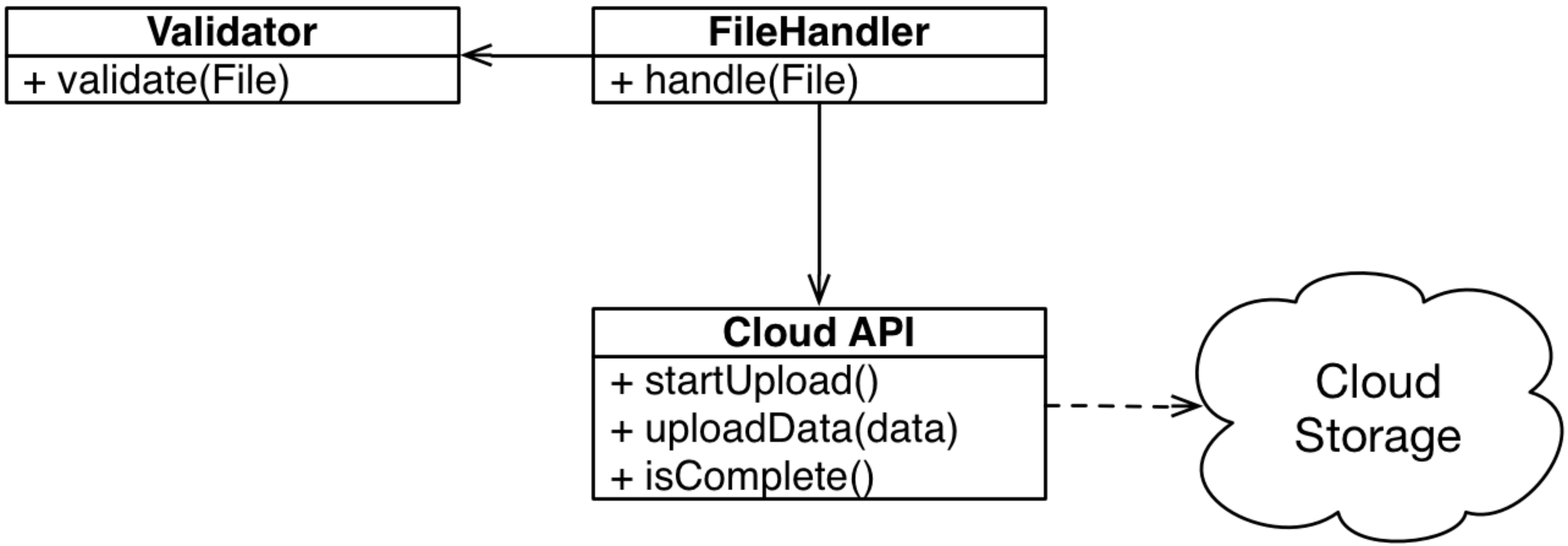
Favour **composition** over **inheritance**

Problem: Doubling 3rd party code

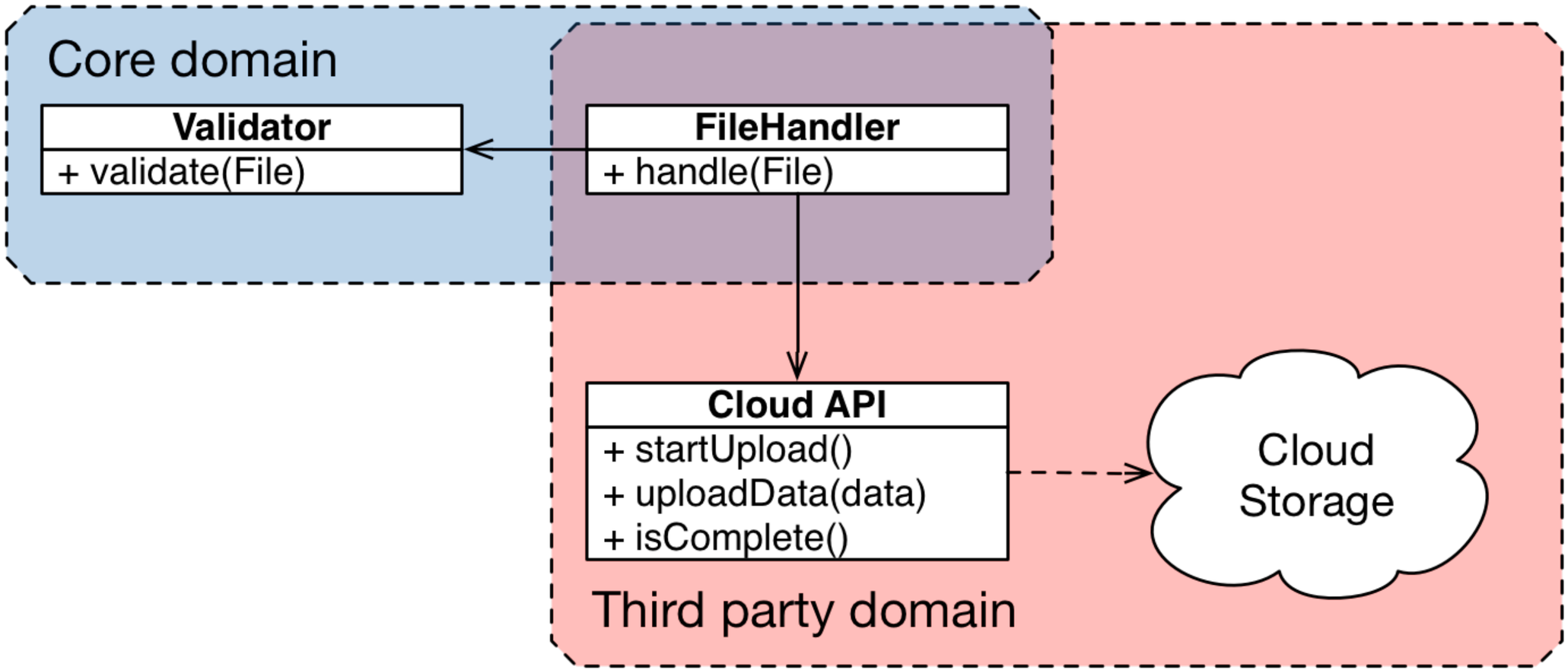
- Test Doubles (Mocks, Stubs, Fakes) are used to test **communication** between objects
- Testing against a Double of a 3rd party object **does not give us confidence** that we have described the communication correctly

```php
class FileHandlerSpec extends ObjectBehaviour
{
    public function it_uploads_data_to_the_cloud_when_valid(
        CloudApi $client, FileValidator $validator, File $file
    )
    {
        $this->beConstructedWith($client, $validator);

        $validator->validate($file)->willReturn(true);

        $client->startUpload()->shouldBeCalled();
        $client->uploadData(Argument::any())->shouldBeCalled();
        $client->uploadSuccessful()->willReturn(true);

        $this->process($file)->shouldReturn(true);
    }
}
```

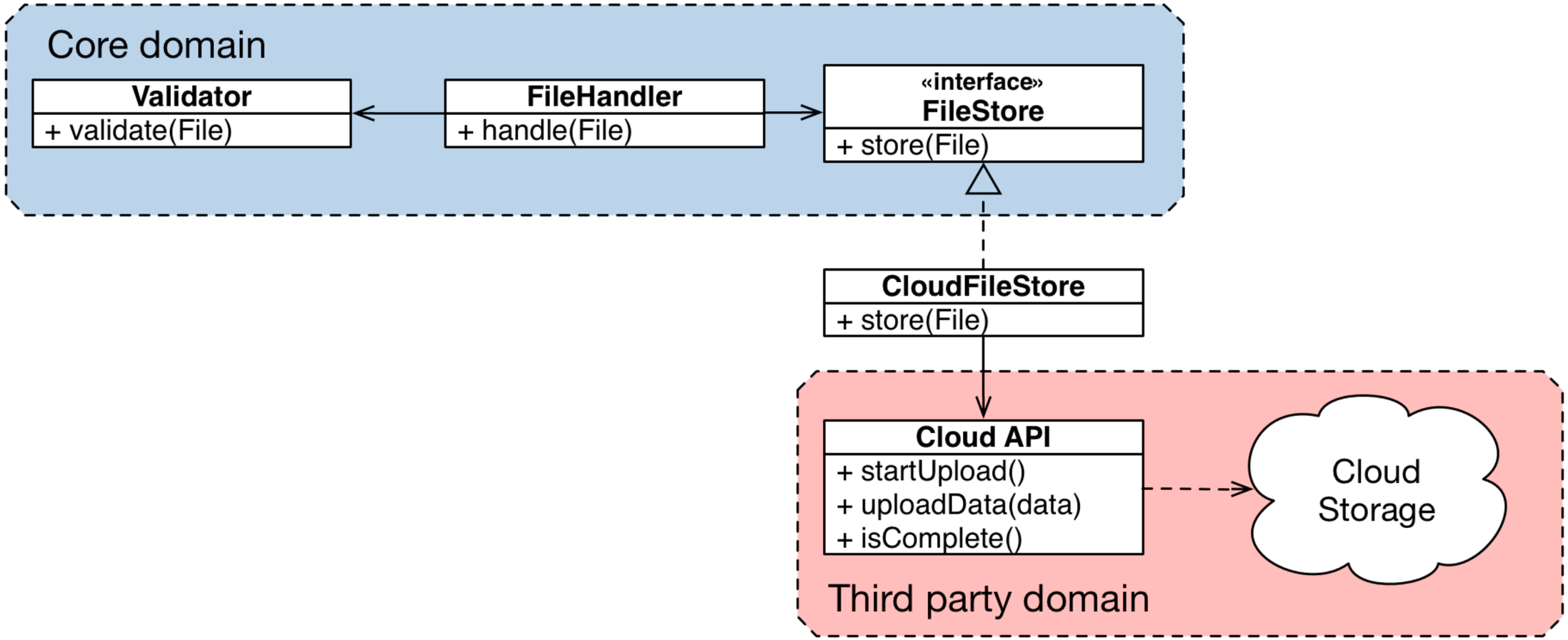
Coupled architecture



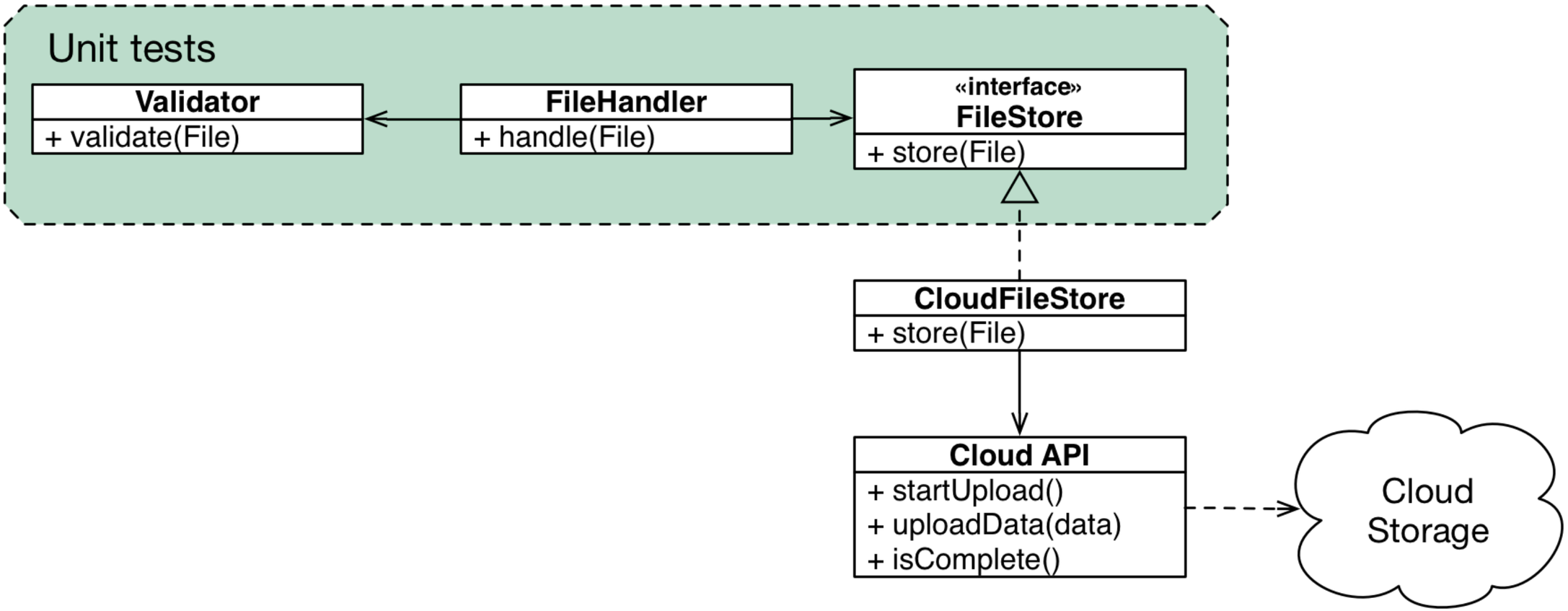
Coupled architecture



Layered architecture

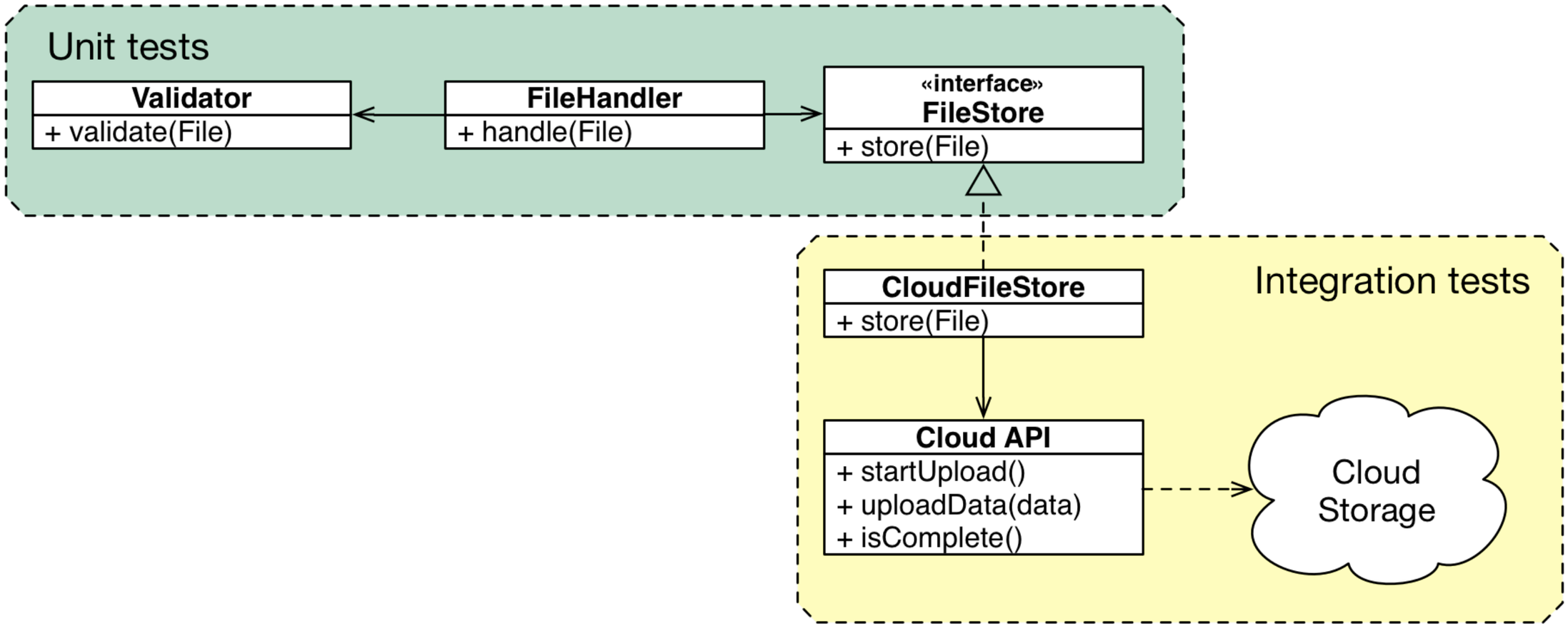


Testing layered architecture



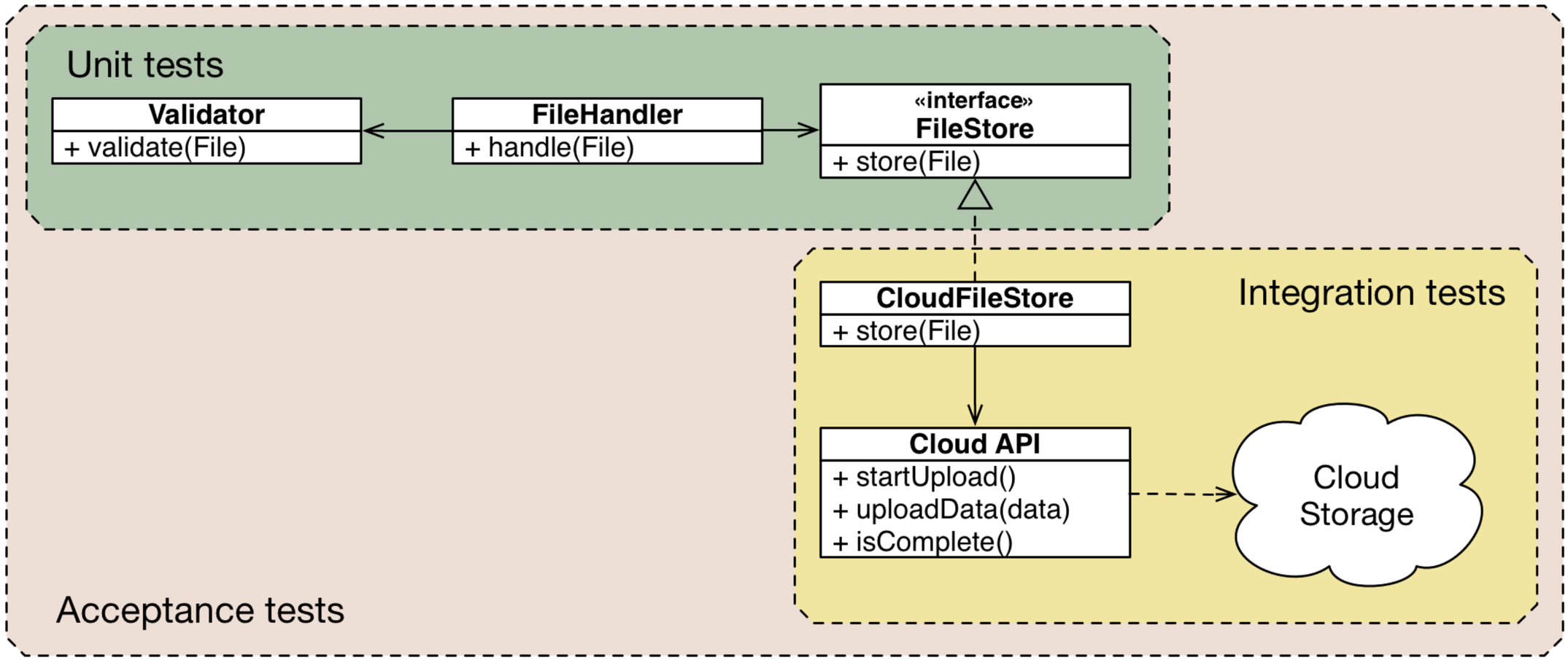
```
class FileHandlerSpec extends ObjectBehaviour
{
    public function it_uploads_data_to_the_cloud_when_valid(
        FileStore $filestore, FileValidator $validator, File $file
    )
    {
        $this->beConstructedWith($filestore, $validator);
        $validator->validate($file)->willReturn(true);

        $this->process($file);

        $filestore->store($file)->shouldHaveBeenCalled();
    }
}
```

Testing layered architecture



```php
class CloudFilestoreTest extends PHPUnit_Framework_TestCase
{
    function testItStoresFiles()
    {
        $testCredentials = …
        $file = new File(…);

        $apiClient = new CloudApi($testCredentials);
        $filestore = new CloudFileStore($apiClient);

        $filestore->store($file);

        $this->assertTrue($apiClient->fileExists(…));
    }
}
```

Testing layered architecture



Don’t test other people’s code

You don't know how it's supposed to **behave**

Don’t test how you talk to other people’s code

You don't know how it's supposed to **respond**

Putting it all together

- Use TDD to **drive development**
- Use TDD to **replace existing practices**, not as a new extra task

Putting it all together

Don’t ignore tests that suck, improve them:

- Is the test too **tied to implementation**?
- Does the **design of the code** need to improve?
- Are you **testing across domain boundaries**?

Image credits

- **Human Evolution** by Tkgd2007
  http://bit.ly/1DdLvy1 (CC BY-SA 3.0)
- **Oops** by Steve and Sara Emry
  https://flic.kr/p/9Z1EEd (CC BY-NC-SA 2.0)
- **Blueprints for Willborough Home** by Brian Tobin
  https://flic.kr/p/7MP9RU (CC BY-NC-ND 2.0)
- **Border** by Giulia van Pelt
  https://flic.kr/p/9YT1c5 (CC BY-NC-ND 2.0)

Thank you!

https://joind.in/13393
https://github.com/ciaranmcnulty
http://www.slideshare.net/CiaranMcNulty
@ciaranmcnulty