MCR 2016

JAMES COWIE

Senior Software Engineer at Magento

@jcowie

Object(ivly) thinking

WWW.MAGETITANS.CO.UK

#MageTitansMCR @MageTitans

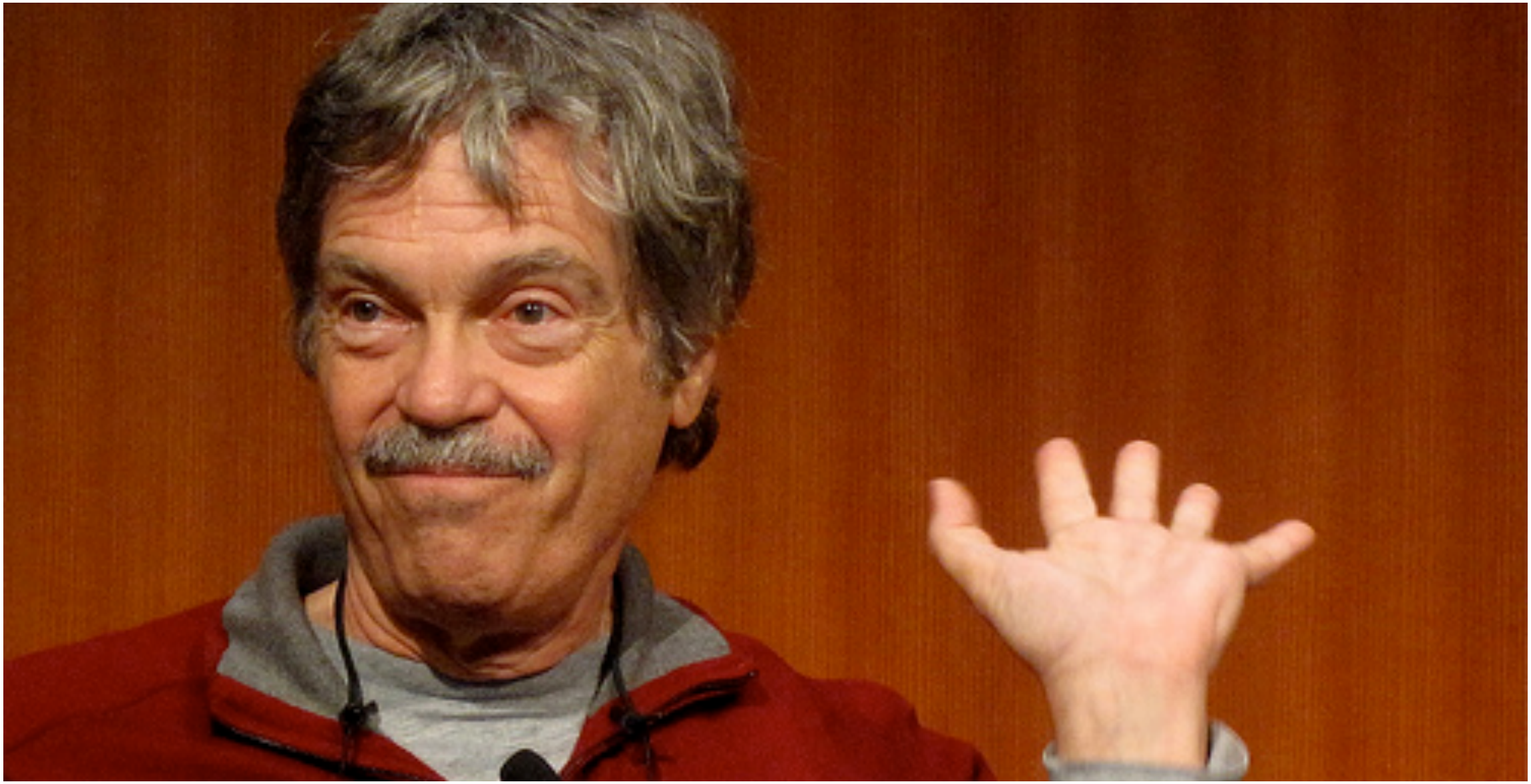
Hello World

@Jcowie
Magento ECG
Mage-Casts
Github: Jamescowie

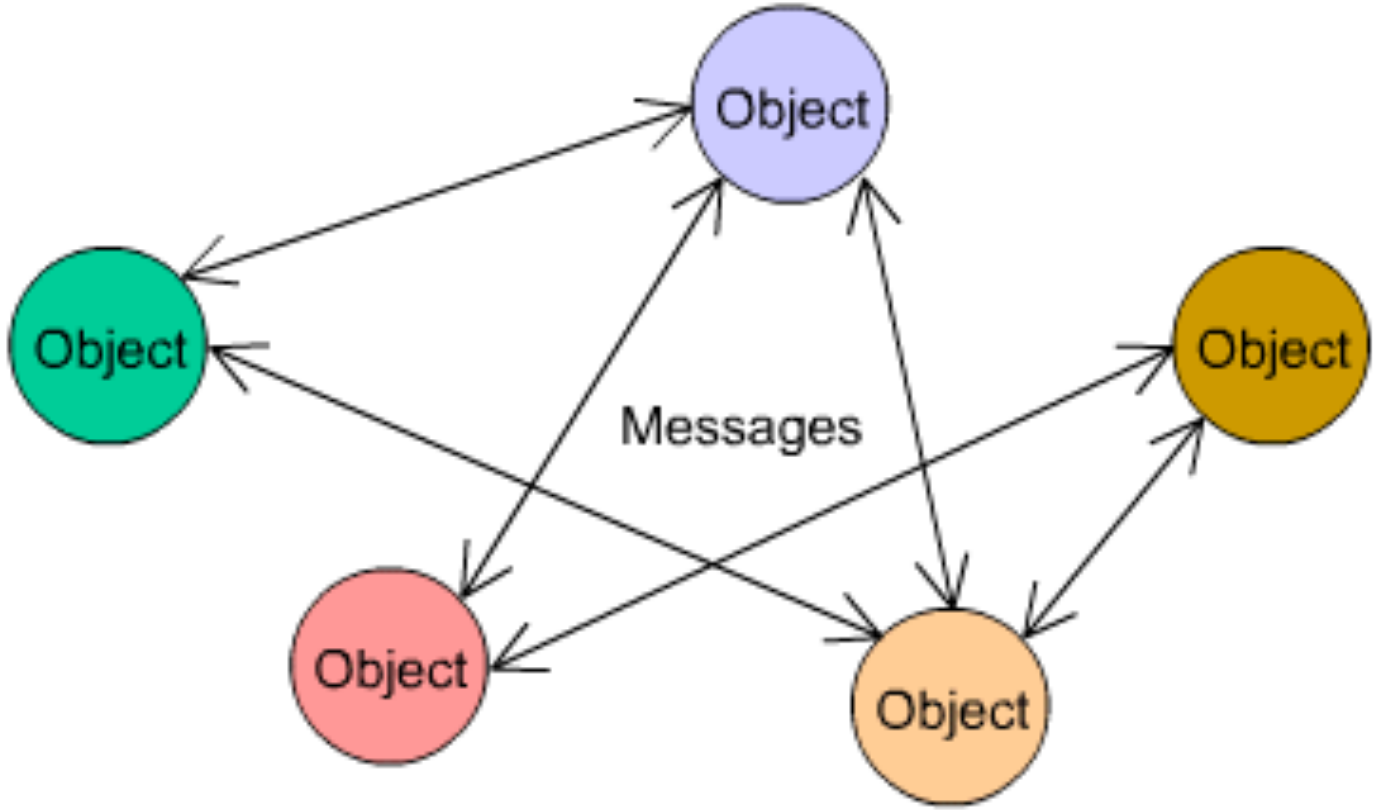
Objectively

Background on OOP
Insight into my thinking
Insight into my workflow
A fun 30 mins

Object Oriented Programming

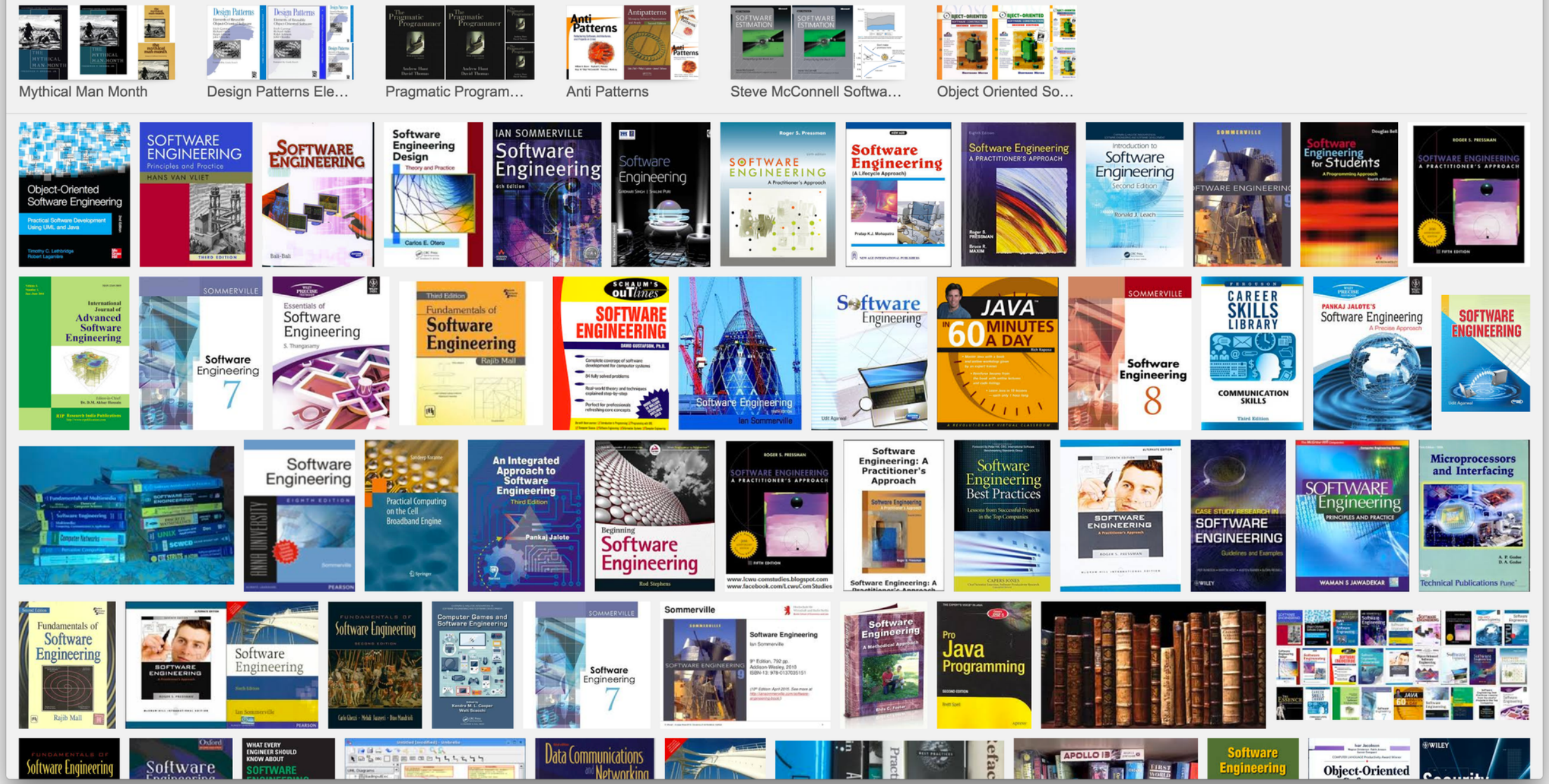


Messaging



Interaction of objects via message passing

Principles of GOOD software


Mythical Man Month
Design Patterns Ele…
Pragmatic Program…
Anti Patterns
Steve McConnell Softwa…
Object Oriented So…
Object-Oriented Software Engineering
Practical Software Development Using UML and Java
SOFTWARE ENGINEERING
Principles and Practice
HANS VAN VLIET
THIRD EDITION
SOFTWARE ENGINEERING
Software Engineering Design
Theory and Practice
Carlos E. Otero
IAN SOMMERVILLE
Software Engineering
6th Edition
Software Engineering
SOFTWARE ENGINEERING
A Practitioner's Approach
Software Engineering
(A Lifecycle Approach)
Software Engineering
A PRACTITIONER'S APPROACH
Introduction to Software Engineering
Second Edition
Ronald J. Leach
Software Engineering for Students
A Programming Approach
SOFTWARE ENGINEERING
A PRACTITIONER'S APPROACH
FIFTH EDITION
International Journal of Advanced Software Engineering
SOMMERVILLE
Software Engineering
7
Essentials of Software Engineering
S. Thangasamy
Third Edition
Fundamentals of Software Engineering
Rajib Mall
SCHAUM'S outlines
SOFTWARE ENGINEERING
Software Engineering
Ian Sommerville
Software Engineering
Udit Agarwal
JAVA
IN 60 MINUTES A DAY
SOMMERVILLE
Software Engineering
8
CAREER SKILLS LIBRARY
COMMUNICATION SKILLS
Third Edition
PANKAJ JALOTE'S
Software Engineering
A Precise Approach
SOFTWARE ENGINEERING
Software Engineering
EIGHTH EDITION
PEARSON
Practical Computing on the Cell Broadband Engine
An Integrated Approach to Software Engineering
Third Edition
Pankaj Jalote
Beginning
Software Engineering
Rod Stephens
SOFTWARE ENGINEERING
A PRACTITIONER'S APPROACH
FIFTH EDITION
www.lcwu-comstudies.blogspot.com
www.facebook.com/LcwuComStudies
Software Engineering: A Practitioner's Approach
Software Engineering
Best Practices
Lessons from Successful Projects in the Top Companies
SOFTWARE ENGINEERING
CASE STUDY RESEARCH IN
SOFTWARE ENGINEERING
Guidelines and Examples
SOFTWARE Engineering
PRINCIPLES AND PRACTICE
WAMAN S JAWADEKAR
Microprocessors and Interfacing
Technical Publications Pune
Fundamentals of Software Engineering
Rajib Mall
SOFTWARE ENGINEERING
Software Engineering
Ian Sommerville
FUNDAMENTALS OF
Software Engineering
SECOND EDITION
Computer Games and Software Engineering
Kendra M. L. Cooper
Walt Scacchi
SOMMERVILLE
Software Engineering
7
Sommerville
Software Engineering
Ian Sommerville
SOFTWARE ENGINEERING
9
Software Engineering
A Methodical Approach
Elvis C. Foster
Pro
Java
Programming
SECOND EDITION
FUNDAMENTALS OF
Software Engineering
Software
Engineering
WHAT EVERY ENGINEER SHOULD KNOW ABOUT
SOFTWARE
Data Communications and Networking
APOLLO 13
Software Engineering
Object-Oriented
WILEY

SOLID

Uncle Bob Martin

Single Responsibility Principle

A class should have one and only one reason to change, meaning that a class should have only one job.

Open Closed

Objects or entities should be open for extension, but closed for modification.

```
/**
 * @api
 */
interface ProductRepositoryInterface
{
.....
}
```

```
<preference
        for="Magento\Catalog\Api\ProductRepositoryInterface"
        type="Magento\Catalog\Model\ProductRepository”
 />
```

Interface Segregation Principle

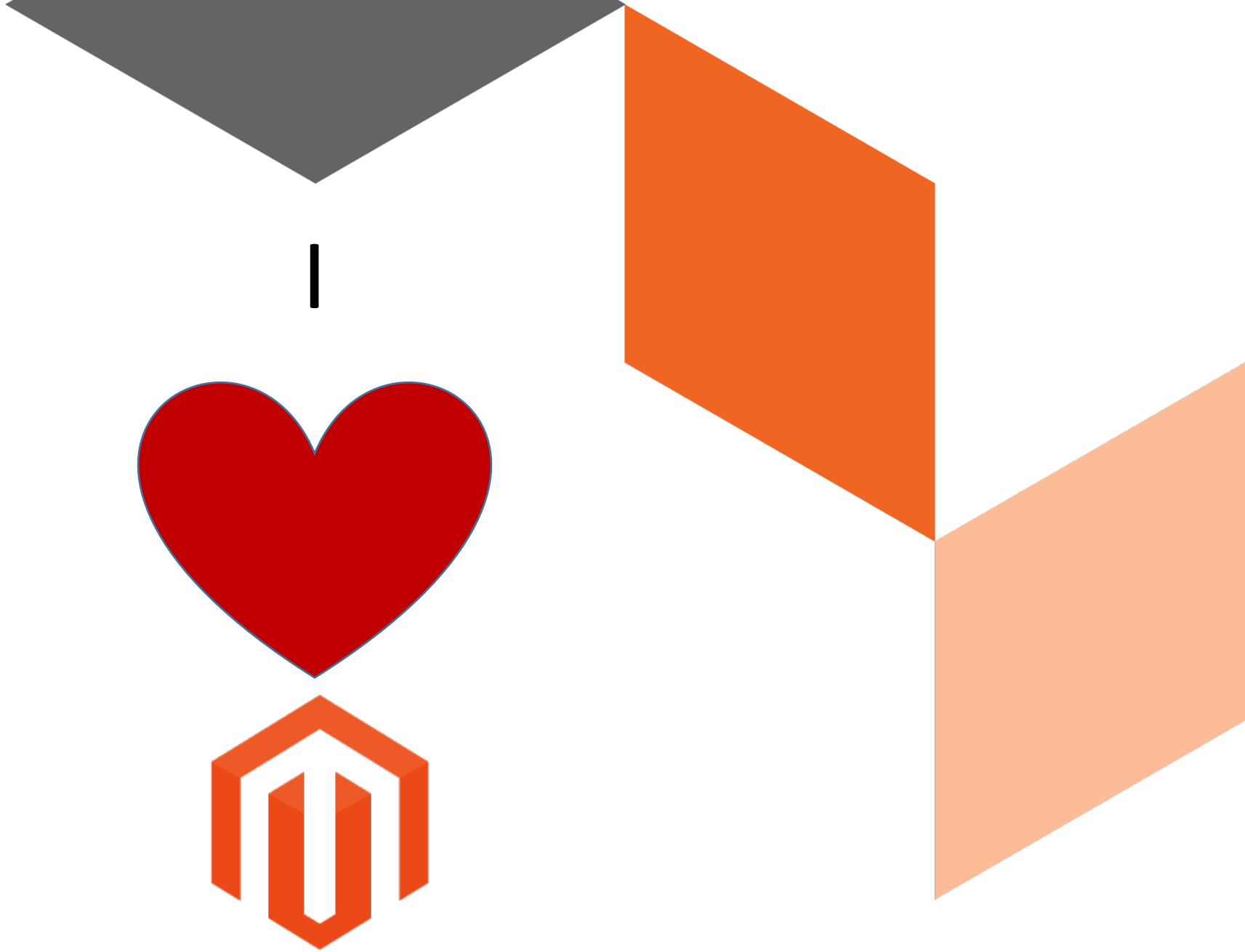
A client should never be forced to implement an interface that it doesn’t use or clients shouldn’t be forced to depend on methods they do not use.

Dependency Inversion Principle

Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions.

```
/**
 * @api
 */
interface ProductRepositoryInterface
{
.....
}
```

```
<preference
        for="Magento\Catalog\Api\ProductRepositoryInterface"
        type="Magento\Catalog\Model\ProductRepository”
 />
```



A framework is just one of the tools to help you developer “Better” and “faster”

- Symfony documentation

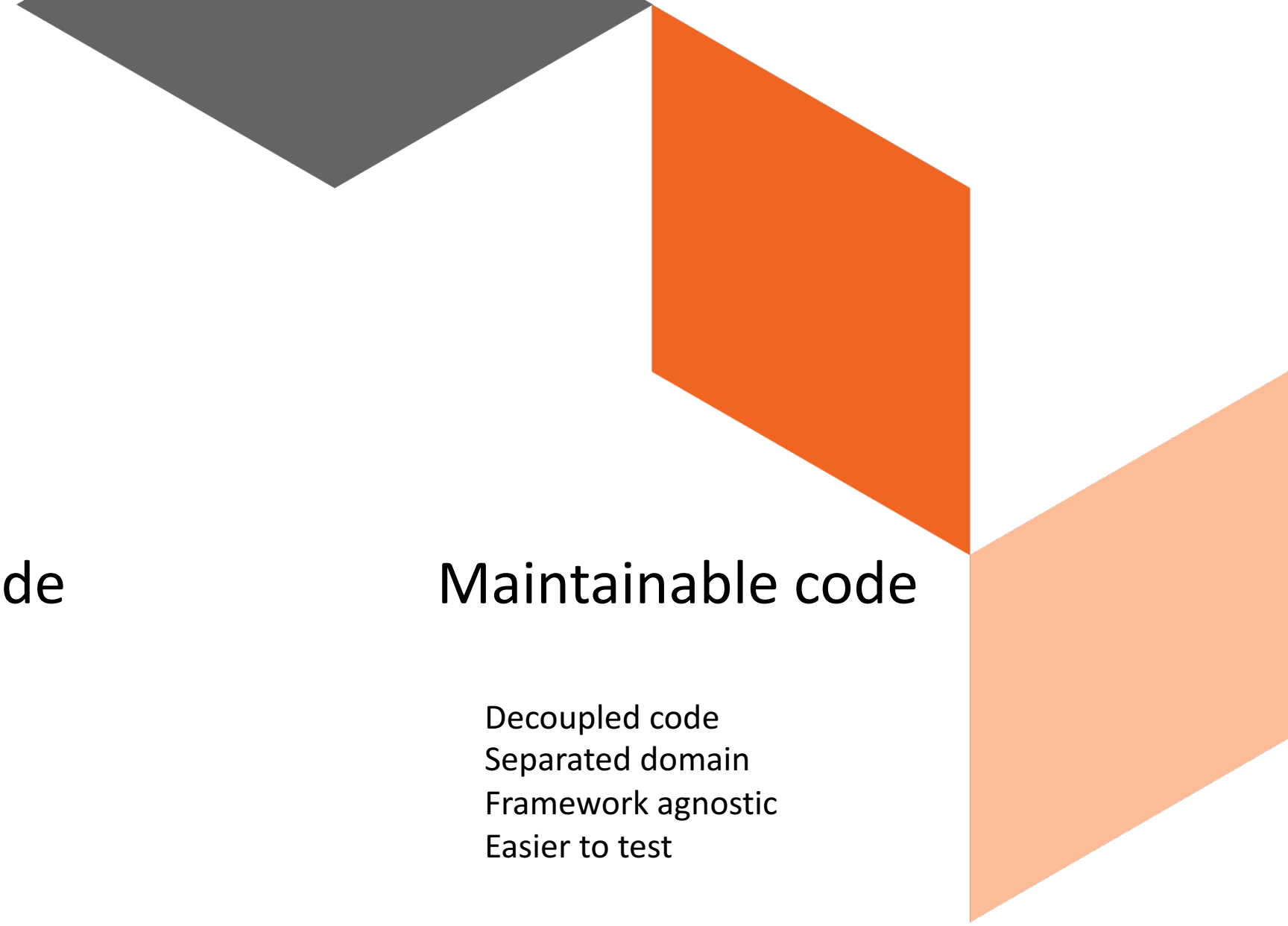
Convenient code

Maintainable code

Convenient code

Coupled code
Mixed responsibility
Bound to framework
Harder to test

Maintainable code



Convenient code

Coupled code
Mixed responsibility
Bound to framework
Harder to test

Maintainable code

Decoupled code
Separated domain
Framework agnostic
Easier to test

Magento can be convenient

```
public function execute()
{
    if ($this->_request->getParam(\Magento\Framework\App\ActionInterface::PARAM_NAME_URL_ENCODED)) {
        return $this->resultRedirectFactory->create()->setUrl($this->_redirect->getRedirectUrl());
    }
    $category = $this->_initCategory();
    if ($category) {
        $this->layerResolver->create(Resolver::CATALOG_LAYER_CATEGORY);
        $settings = $this->_catalogDesign->getDesignSettings($category);

        // apply custom design
        if ($settings->getCustomDesign()) {
            $this->_catalogDesign->applyCustomDesign($settings->getCustomDesign());
        }

        $this->_catalogSession->setLastViewedCategoryId($category->getId());
        .....
}
```

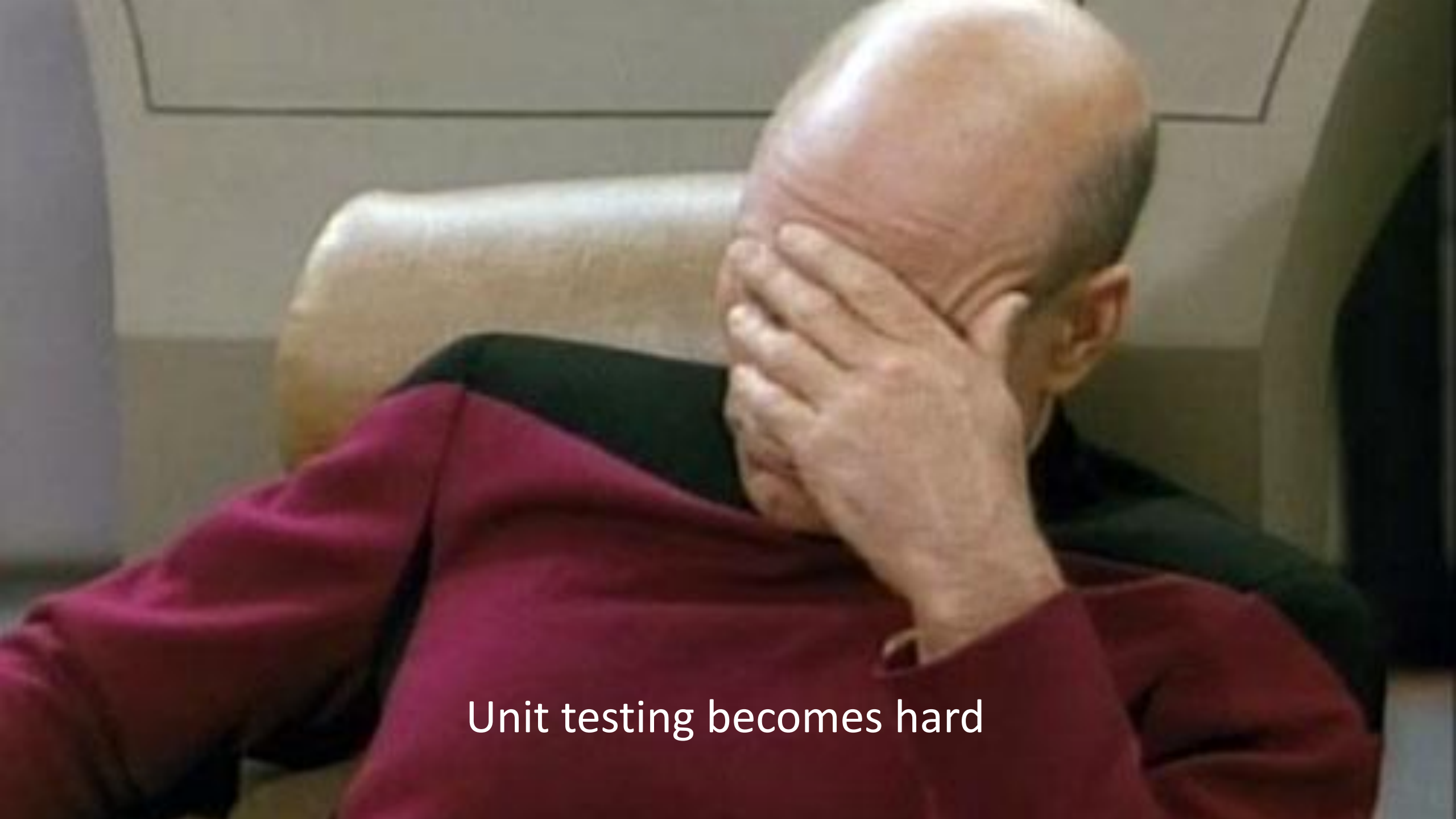
```
class Index extends \Magento\Framework\App\Action\Action
```

```php
/** @var \Magento\Framework\View\Result\PageFactory  */
protected $resultPageFactory;

public function __construct(
    \Magento\Framework\App\Action\Context $context,
    \Magento\Framework\View\Result\PageFactory $resultPageFactory
) {
    $this->resultPageFactory = $resultPageFactory;
    parent::__construct($context);
}
```

Large number of dependencies

\Magento\Framework\App\Action\Context $context,
\Magento\Framework\View\Result\PageFactory $resultPageFactory
\Magento\Framework\Registry $coreRegistry,
\Magento\Store\Model\StoreManagerInterface $storeManager,
CategoryRepositoryInterface $categoryRepository


Unit testing becomes hard

The “convenient” controller requires more Integration Tests

Potential code smell

Lets see and Example

```
protected function configure()
{
    $this->setName('generate:showcaseproducefeed');
    $this->setDescription('Generate a produce feed of showcase products in json');
    parent::configure();
}
```

```php
protected function execute(InputInterface $input, OutputInterface $output)
{
    $product = $this->products->get('24-MB01');
    $writer = $this->filesystem->getDirectoryWrite('var');
    $file = $writer->openFile('showcase.json', 'w');

    try {
        $file->lock();
        try {
            $file->write(json_encode(['product_name'  => $product->getName(),
                                      'product_price' => $product->getPrice()
            ]));
        } finally {
            $file->unlock();
        }
    } finally {
        $file->close();
    }

    $output->writeln("Feed Generated");
}
```

Can we add XML feed generation

```
protected function execute(InputInterface $input, OutputInterface $output)
{
    $product = $this->products->get('24-MB01');
    $writer = $this->filesystem->getDirectoryWrite('var');
    $file = $writer->openFile('showcase.json', 'w');
    $xmlFile = $writer->openFile('showcase.xml', 'w');
    try {
        $file->lock();
        try {
            $file->write(json_encode(['product_name'  => $product->getName(),
                                      'product_price' => $product->getPrice()
            ]));

            $showcaseXML = new \SimpleXMLElement("<showcase></showcase>");
            $showcaseXML->addAttribute('showcase-products', 'today');
            $showcaseAttributes = $showcaseXML->addChild($product->getName());
            $showcaseAttributes->addAttribute('price', $product->getPrice());

            $xmlFile->write($showcaseXML->asXML());

            $file->unlock();
        }...
    $output->writeln("Feed Generated");
}
```

Our class had reason to change

How can we test this ?

```
public function __construct(
    \Magento\Catalog\Api\ProductRepositoryInterface $productRepository,
    \Magento\Framework\App\State $state,
    \Magento\Framework\Filesystem $filesystem
) {
    $state->setAreaCode('frontend');
    $this->products = $productRepository;
    $this->filesystem = $filesystem;
    parent::__construct();
}
```

```
$showcaseXML = new \SimpleXMLElement("<showcase></showcase>");
```

Scenario: Products are presented in JSON file
  Given The generate command exists
  When I run the generate command
  Then I should see a JSON file created

```
use Behat\Behat\Context\Context;
use Symfony\Component\Process\Process;
use \Symfony\Component\Filesystem\Filesystem;
```

```php
class FeatureContext implements Context
{
    private $output;
    private $filesystem;
    public function __construct()
    {
        $this->filesystem = new Filesystem();
    }
}
```

```
/**
 * @Given The generate command exists
 */
public function theGenerateCommandExists()
{
    return true;
}
```

```php
/**
 * @When I run the generate command
 */
public function iRunTheGenerateCommand()
{
    $process = new Process("php " . getcwd()
            . "/../../../generate:showcaseproducefeed");
    $process->run();
    $this->output = $process->getOutput();
}
```

```
/**
 * @Then I should see a JSON file created
 */
public function iShouldSeeAJsonFileCreated()
{
    expect(file_exists(__dir__ .
'/../../../../../../var/showcase.json'))->toBe(true);
}
```

Tests Pass

```
~/P/P/P/M/a/a/c/T/Showcase ❯❯❯ b features/showcase.feature:9
Feature: As a customer
  I need to be able to consume a list of showcase products
  So that I can keep up to date with top deals

  Scenario: Products are presented in JSON file # features/showcase.feature:9
    Given The generate command exists           # FeatureContext::theGenerateCommandExists()
    When I run the generate command             # FeatureContext::iRunTheGenerateCommand()
    Then I should see a JSON file created       # FeatureContext::iShouldSeeAJsonFileCreated()

1 scenario (1 passed)
3 steps (3 passed)
0m2.02s (52.34Mb)
~/P/P/P/M/a/a/c/T/Showcase ❯❯❯
```

Scenario: Products are presented in XML file

Given The generate command exists

When I run the generate command

Then I should see a XML file created

Tests Fail

```
Scenario: Products are presented in XML file # features/showcase.feature:14
  Given The generate command exists           # FeatureContext::theGenerateCommandExists()
  When I run the generate command             # FeatureContext::iRunTheGenerateCommand()
  Then I should see a XML file created
```

We could add this into the command buuuut.

Open Closed Principle

```php
<?php

namespace Titans\Showcase\Api;

interface BuilderInterface
{
    public function build($data);
}
```

Create a scenario for extraction

Scenario: Class file does return json
When I call the JSONBuilder class
Then I should be returned a json string

```
public function __construct()
{
    $this->builder =
        new Titans\Showcase\Commands\Builders\JsonBuilder();
}
```

```php
/**
 * @When I call the JSONBuilder class
 */
public function iCallTheJsonbuilderClass()
{
    $this->output = $this->builder->build(['name' => 'james']);
}

/**
 * @Then I should be returned a json string
 */
public function iShouldBeReturnedAJsonString()
{
    expect($this->output)->toBe(json_encode(['name' => 'james']));
}
```

```php
<?php

namespace Titans\Showcase\Commands\Builders;

class JsonBuilder implements
\Titans\Showcase\Api\BuilderInterface
{
    public function build($data)
    {
        return json_encode($data);
    }
}
```

```php
<?php

namespace Titans\Showcase\Commands\Builders;

class JsonBuilder implements \Titans\Showcase\Api\BuilderInterface
{
    public function build($data)
    {
        return json_encode($data);
    }
}
```

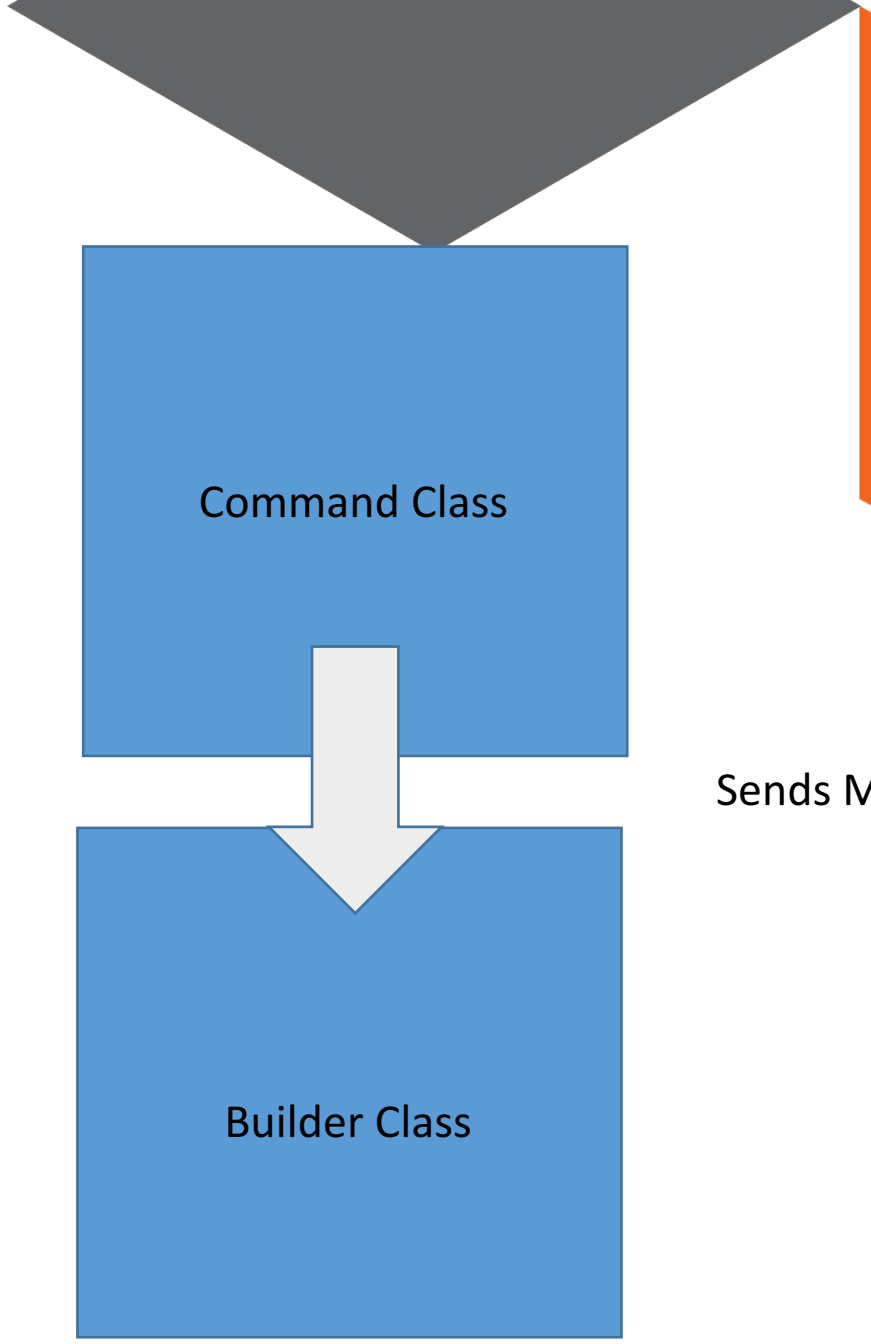
```php
<?php

namespace Titans\Showcase\Commands\Builders;

class XmlBuilder implements \Titans\Showcase\Api\BuilderInterface
{
    public function build($data)
    {
        return ....
    }
}
```

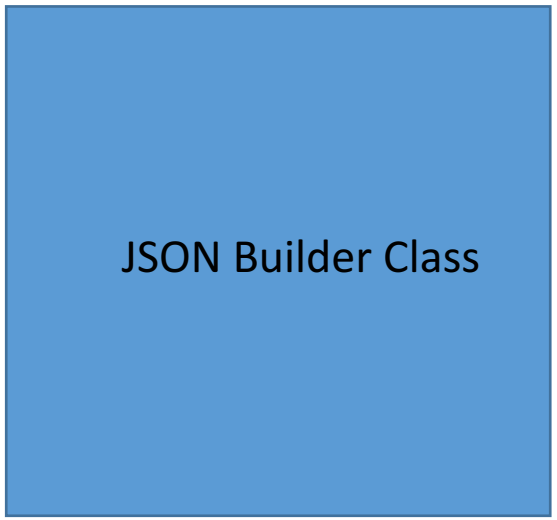
```php
public function __construct(
    \Titans\Showcase\Commands\Builders\JsonBuilder $jsonBuilder
) {
    $this->jsonBuilder = $jsonBuilder;
    parent::__construct();
}
```

```
protected function execute(InputInterface $input, OutputInterface
$output)
{
    $product = $this->products->get('24-MB01');
    $this->writer->write(
            'showcase.json',
            $this->jsonBuilder->build(
                ['product_name' => $product->getName()]
            )
    );
}
```



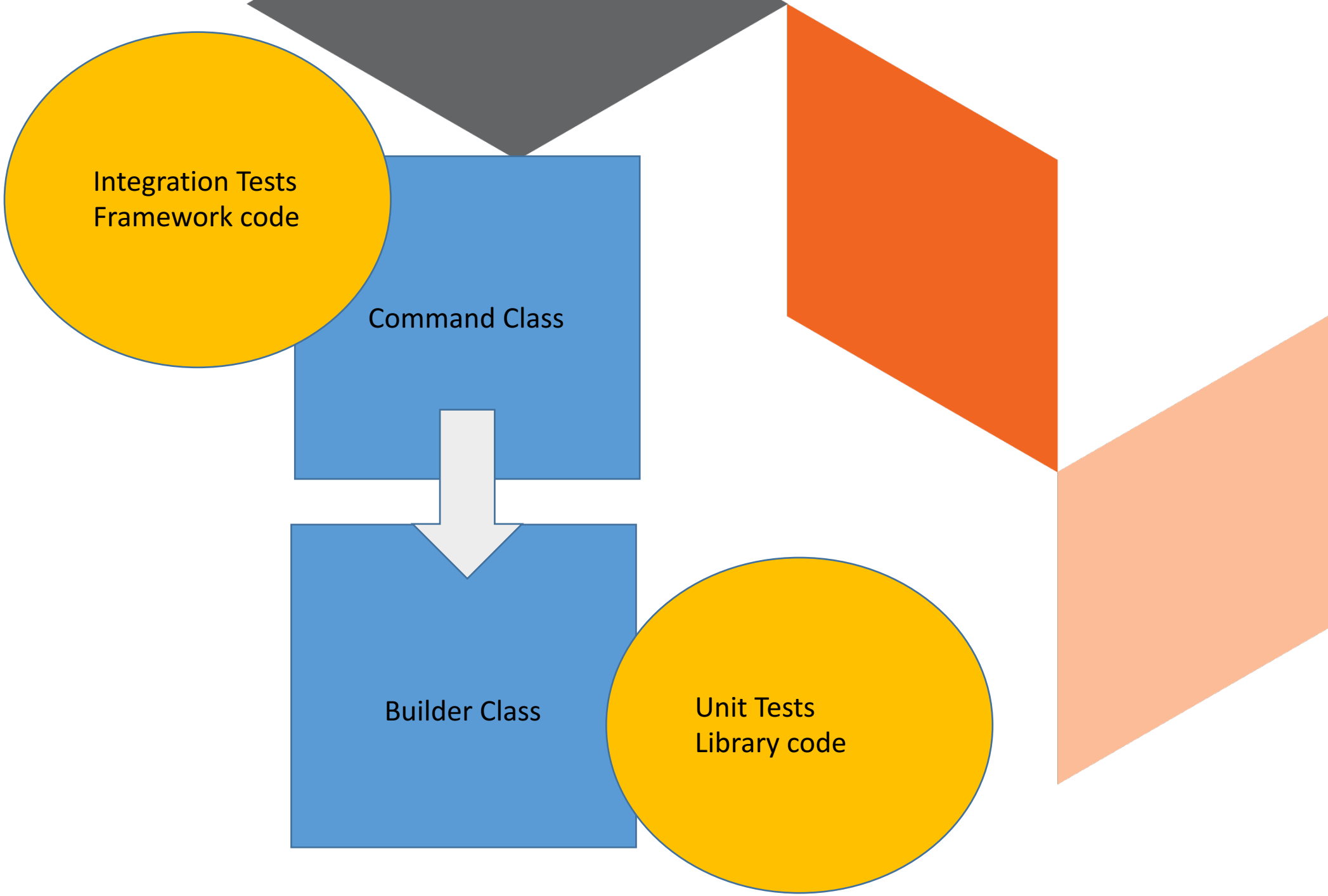
Sends Message

Responsibility for handling the command



Responsibility Building JSON based on a message

Value Object.


Integration Tests
Framework code
Command Class
Builder Class
Unit Tests
Library code

What are the benefits ?

We now pass messages between objects
We are testing to some level using Behat
We have clear Objects
We use Value Objects