Magento 2: PHP Development Best Practices

Gabriel Somoza

DevelopersParadise
2016 / Opatija / Croatia

Gabriel Somoza

Magento 1 CD+, Magento 2 Developer
Founder @ Strategery
OSS Contributor
PHP Limburg BE

DevelopersParadise
2016 / Opatija / Croatia

Why This Talk?

1. Understand how Magento 2 meets mainstream PHP
2. Get up to speed with the jargon
3. Accelerate towards being productive
4. Choose which topics to strengthen

Before we Start

DevelopersParadise

2016 / Opatija / Croatia

Follow Magento's "Core Coding Standards"

DevelopersParadise
2016 / Opatija / Croatia

PSRs 1 through 4

Code Demarcation Standard

DocBlock Standard

JavaScript Coding Standard

JavaScript DocBlock Standard

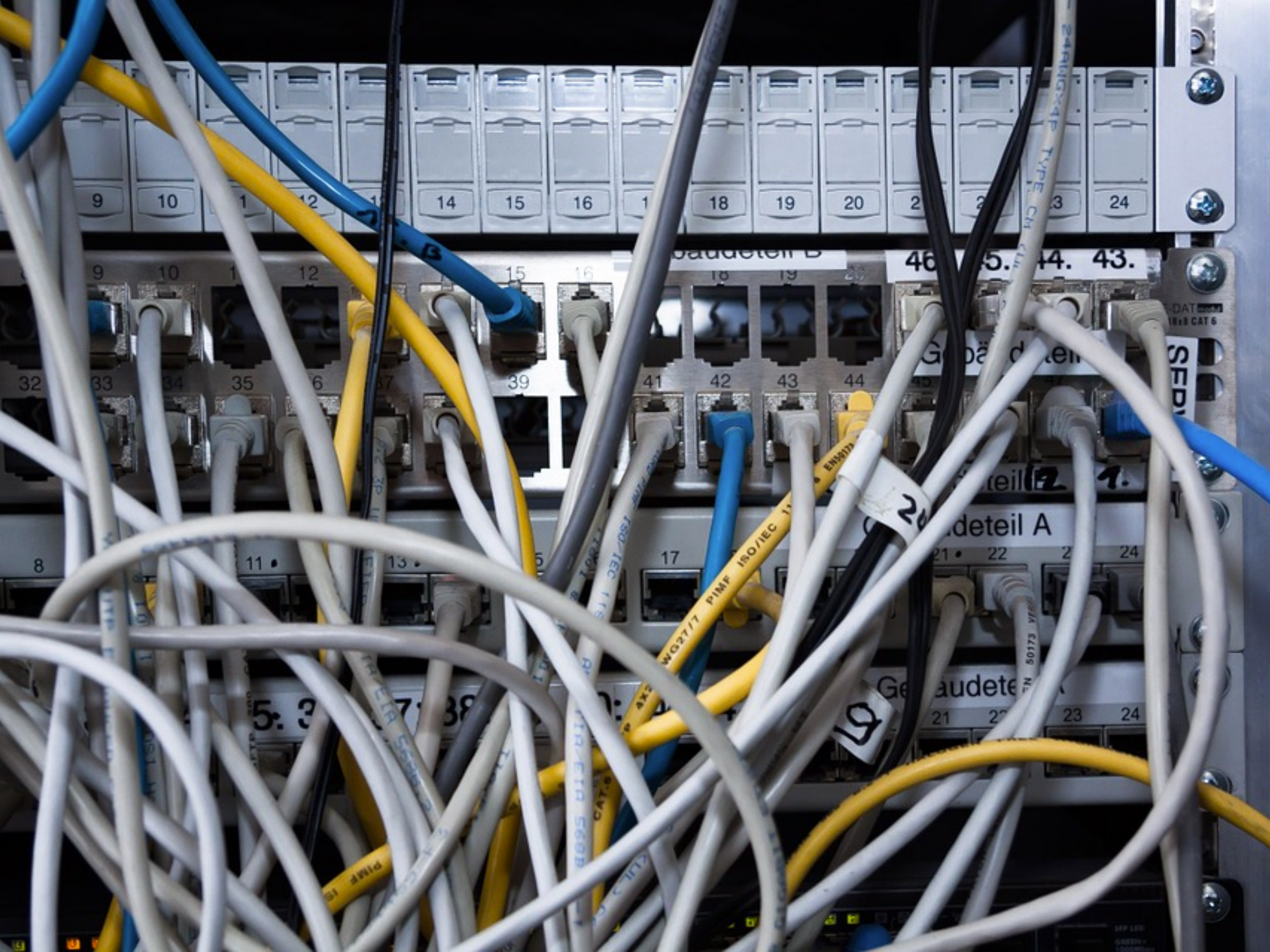
jQuery Widget Standard

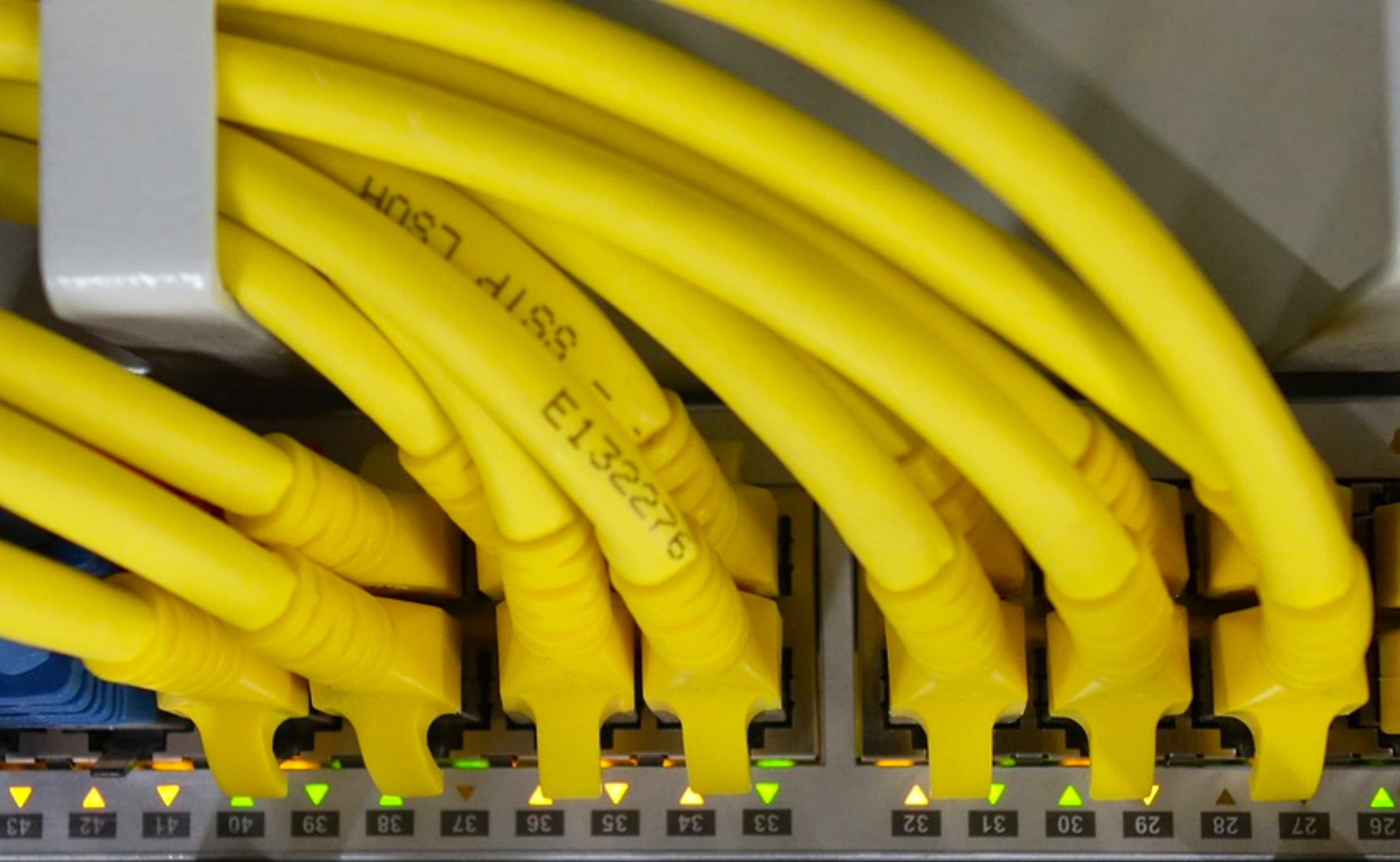
LESS Coding Standard

HTML Style Guide

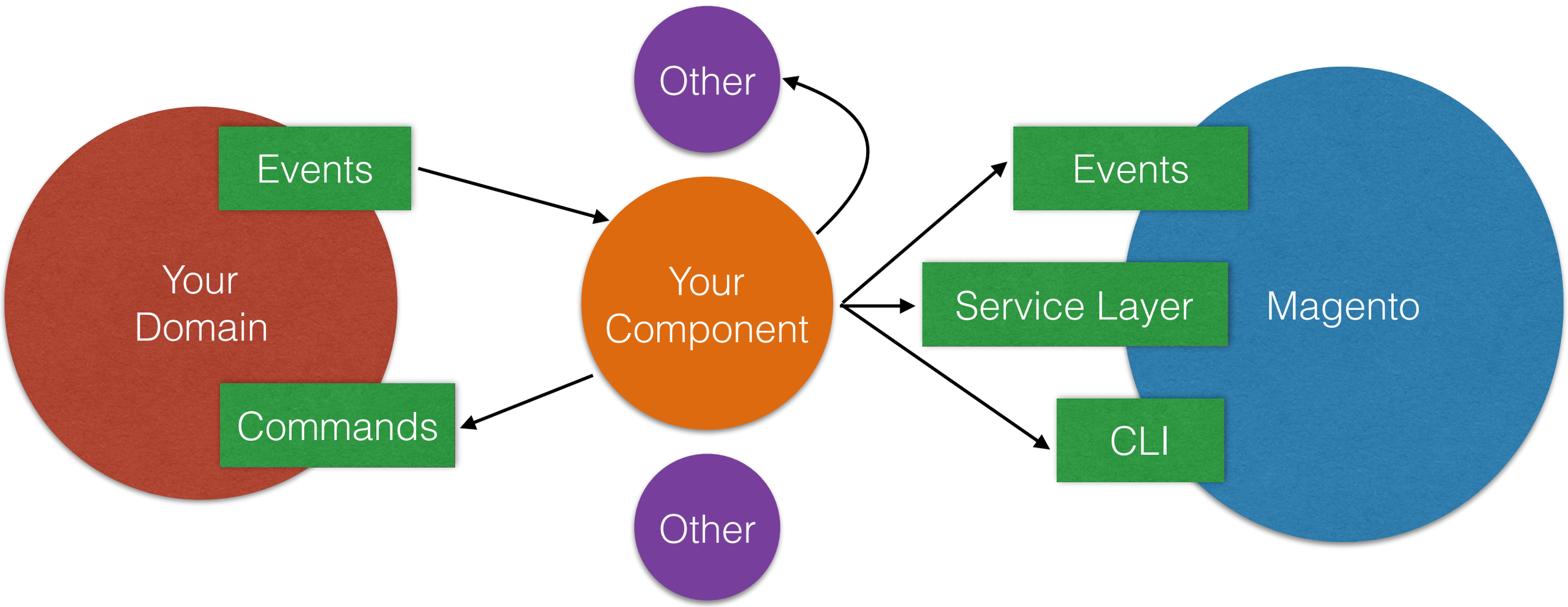
So far, this is a suggestion.

Take it.




43
42
41
40
39
38
37
36
35
34
33
32
31
30
29
28
27
26

Write Reusable Components

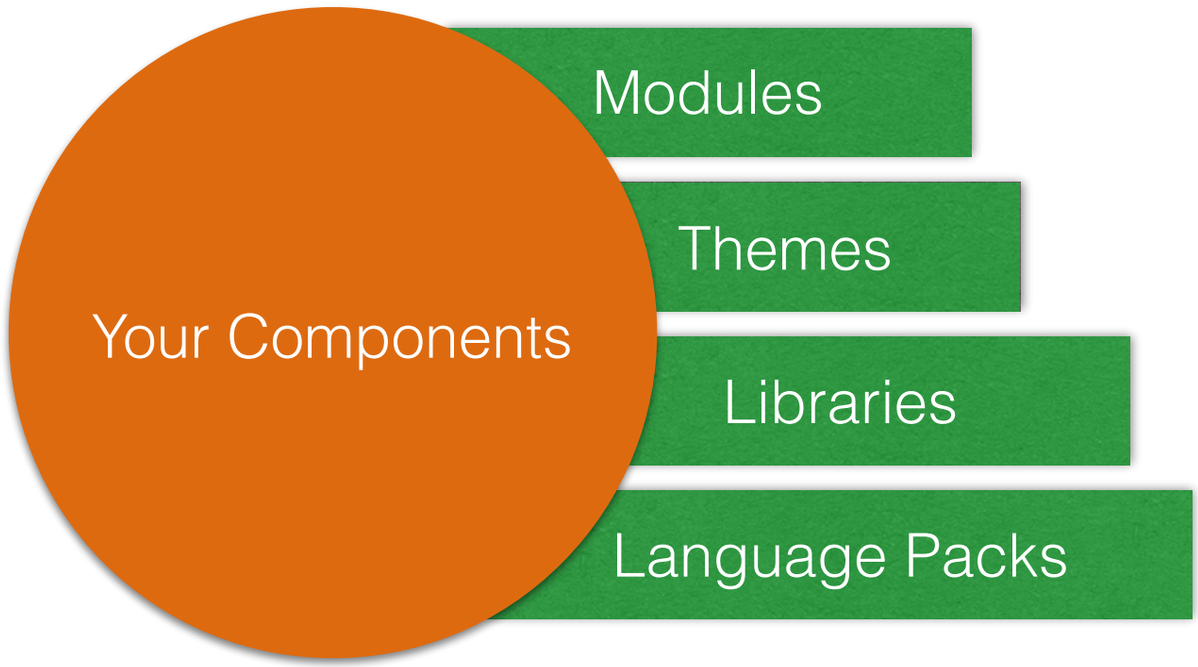

Other
Events
Your Domain
Your Component
Events
Service Layer
Magento
Commands
CLI
Other

For more:

Driving Your Domain by Examples **(13:30)**

Marcos Lopes

Component Types



registration.php

Sample component **composer.json**

```
{
  "name": "acme/magento2-blog",
  "description": "A blog for Magento 2",
  "autoload": {
    "psr-4": {"Acme\\Blog\\":""},
    "files": ["registration.php"]
  },
  "type": "magento2-module",
  "version": "1.0.3",

  // chop, chop..

  "require" : {
    "php": "~7.0",
    "acme/blog-core": "^4.0",
    "acme/magento2-base": "^1.0"
  }
}
```

✓ Package name

✓ Component Type (module, theme, language)

✓ External libraries

✓ Magento 2 Dependencies

✓ Its OK to have dozens of Composer packages

✓ Use Toran Proxy (commercial) or Satis (free, OSS) for private packages

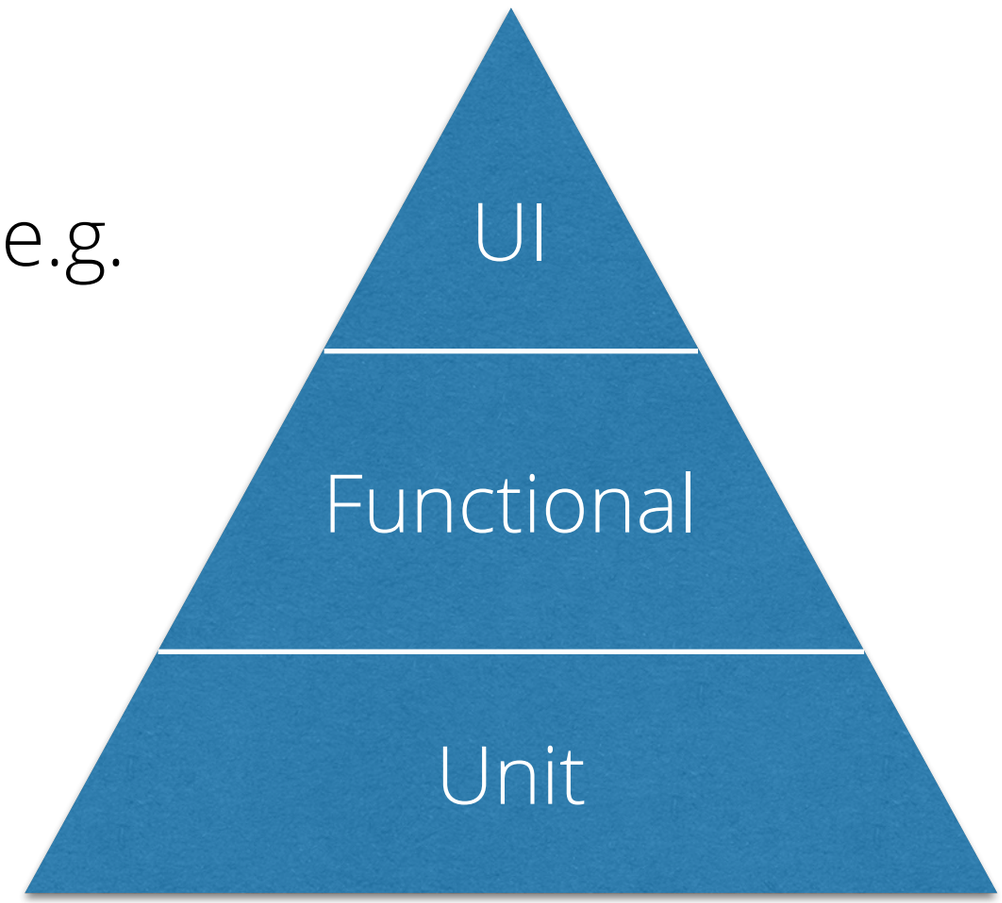
For more:

Magento 2 + Composer **(12:30)**

Peter Jaap

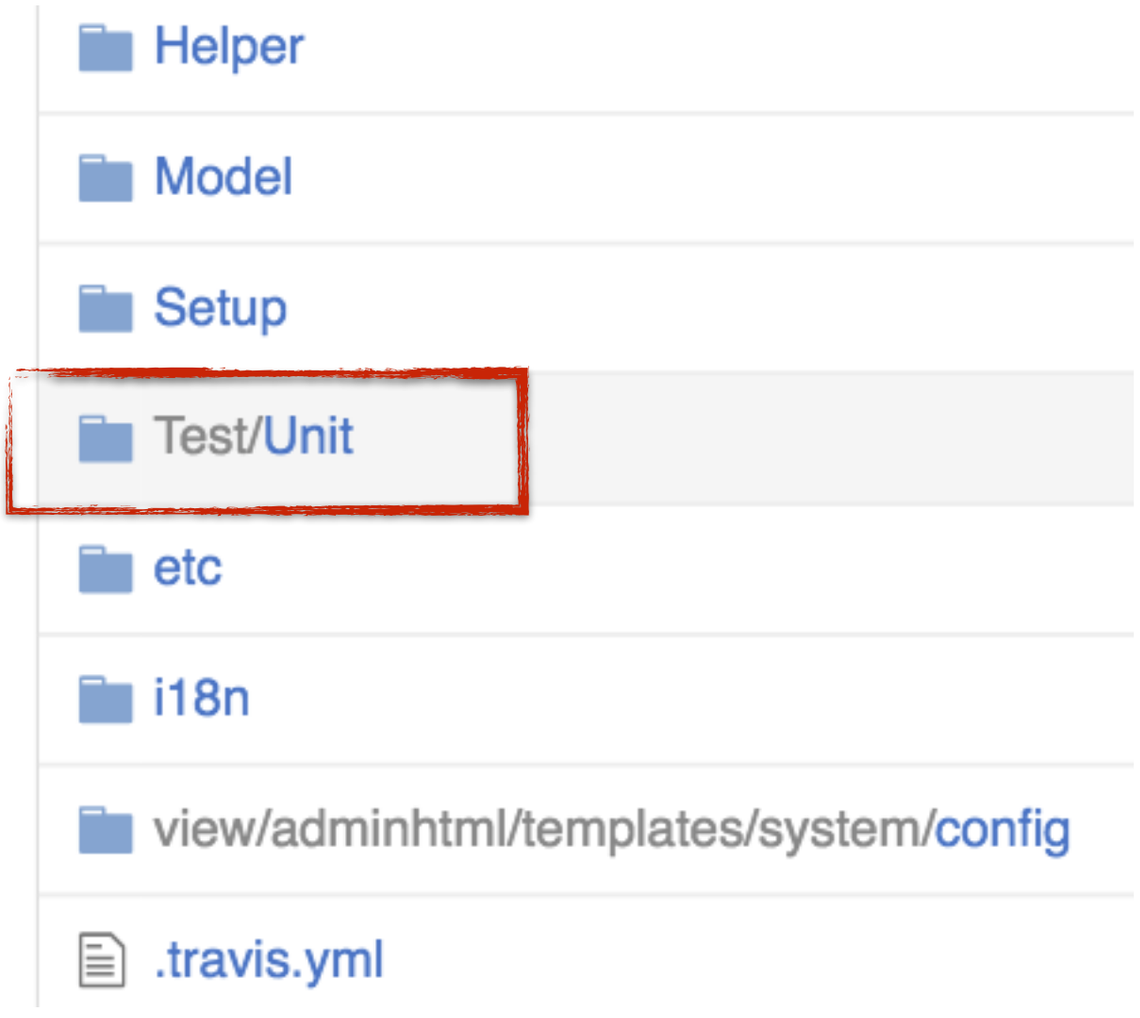
Write Automated Tests

1. Unit test: **generously**
2. Integration test: **interactions** (e.g. with 3rd party systems)
3. Functional/UI test: for **acceptance**
4. Follow core standards



Mike Cohn

Distribute unit tests with your component



✓ Better than just: build passing

http://shields.io/

✓ Documentation!

Understand the Object Manager (OM)

- Dependency Injection
- Lifecycle Management
- Compilation
- Scopes
- Special Patterns
- Translations

Don’t use it Directly

```php
class MyTask
{
    public function doSomething()
    {
        $dependency = $this->getObjectManager()
            ->get('My\Dependency');

        return $dependency->doSomethingElse();
    }
}
```

- Testing = nightmare
- Coupling to the API
- There’s a better way

Automatic Constructor Injection!

```php
// class \Acme\Mailer\Controller\Api
public function __construct(
    \Acme\Mailer\Helper\Data $data
) {
```


Type Definitions

etc/di.xml

```
<type name="Magento\Authorization\Model\ResourceModel\Rules">
    <arguments>
        <argument name="rootResource" xsi:type="object">
            Magento\Framework\Acl\RootResource\Proxy
        </argument>
        <argument name="aclCache" xsi:type="object">
            Magento\Framework\Acl\Cache\Proxy
        </argument>
    </arguments>
</type>
```

- ✓ Declare types
- ✓ Define arguments
- ✓ Replace arguments (caution!)

Be aware of what you think
when you see the word

"TYPE"

```
<virtualType
    name="Magento\Framework\Message\Session\Storage"
    type="Magento\Framework\Session\Storage">
    <arguments>
        <argument name="namespace" xsi:type="string">
            message
        </argument>
    </arguments>
</virtualType>
<type name="Magento\Framework\Message\Session">
    <arguments>
        <argument name="storage" xsi:type="object">
            Magento\Framework\Message\Session\Storage
        </argument>
    </arguments>
</type>
```

etc/di.xml

Preferences

```
<preference
 for="Psr\Log\LoggerInterface"
 type="Magento\Framework\Logger\Monolog" />
```

etc/di.xml

✓ Use interfaces (./Api/)

✓ Contextual replacements (caution!)

Proxy Objects

```xml
<type name="Magento\Authorization\Model\ResourceModel\Rules">
    <arguments>
        <argument name="rootResource" xsi:type="object">
            Magento\Framework\Acl\RootResource\Proxy
        </argument>
        <argument name="aclCache" xsi:type="object">
            Magento\Framework\Acl\Cache\Proxy
        </argument>
    </arguments>
</type>
```

etc/di.xml

- ✓ Delay initialization of “expensive” objects that:
  - Have to be instantiated (e.g. for typing)
  - But might not YET need to be initialized
- ✓ Caching

Plugins (Interception)

etc/di.xml

```
<type name="Class\To\Intercept">
    <plugin name="my_plugin" type="My\Plugin\Class"/>
</type>
```

```
// \Magento\SampleInterception\Plugin\PluginAfter
public function afterBaseMethodUppercase(
    $subject,
    $interceptedOutput
) {
    return '(after) ' . $interceptedOutput . '(/after)';
}
```


Add before / after / around behaviour.

✓ Alternative to before / after events
✓ Around is powerful for decorating or conditional logic (e.g. authorization)
✓ Manipulate return values (use with caution!)

```php
// \Magento\SampleInterception\Plugin\PluginAfter
public function afterBaseMethodUppercase(
    $subject,
    $interceptedOutput
) {
    return '(after) ' . $interceptedOutput . '(/after)';
}
```

```php
// another interceptor plugin
public function afterBaseMethodUppercase(
    $subject,
    $result
) {
    if (strlen($result) < 10) {
        // do something important
    }
}
```


Better use case:

```
// \Magento\Customer\Model\Cart\ConfigPlugin
public function afterGetConfig(
    \Magento\Checkout\Block\Cart\Sidebar $subject,
    array $result
) {
    return array_merge_recursive(
        $result,
        $this->configProvider->getConfig()
    );
}
```

✓ Aligned to the purpose of the intercepted function

Good applications:

- ✓ Cleaner alternative to creating/requesting “before” and “after” events
- ✓ Part of a coherent chain of incremental atomic transformations designed towards a clear goal

“Less good” applications (*):

- As a replacement for existing events
- Arbitrary transformations to data

* subject to the type of component you’re building

"VIRTUAL TYPES"

A virtual (no real class!), configurable sub-type of another TYPE

✓ Fine-grained control over DI configuration

✓ Can help avoid repetitive argument replacement

- Confusing at first
- Weigh their value vs. simplicity
- You will hardly need them

For more: What is the difference between type and virtualType

Find the right balance

For more:

Porting a complex extension to Magento 2 **(10:00)**

Fabian Schmengler

Leverage Events

etc/events.xml

```
<event name="some_magento_event">
    <observer name="my_observer"
        instance="My\Observer\Class" />
</event>
```

- ✓ Similar to Magento 1
- ✓ Generally unobtrusive
- ✓ Nicely decoupled code
- ✓ Use them whenever possible

Observers In Magento 2 - Cool Ryan

Write Defensive Components

Fail Fast(er)

composer require beberlei/assert

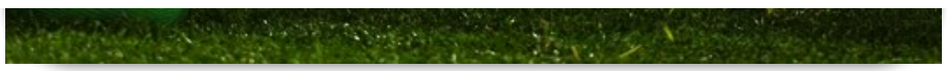
```php
/**
 * Sends a test email
 *
 * @throws \Assert\InvalidArgumentException
 * @throws \Assert\LazyAssertionException
 */
public function execute()
{
    $emailAddress = $this->getRequest()->getParam('email');
    \Assert\that($emailAddress)->email();
    Assertion::email($emailAddress);
    // => \Assert\InvalidArgumentException

    // useful for checking array item types:
    $orders = $this->getOrders(); // => array
    \Assert\lazy()
        ->that($orders, 'not empty')->notEmpty()
        ->that($orders, 'types')
            ->all()->isInstanceOf(OrderInterface::class)
        ->verifyNow();
    // => \Assert\LazyAssertionException
}
```


...and don't forget to CATCH exceptions

```php
/**
 * Sends a test email
 *
 * @throws \Assert\InvalidArgumentException
 * @throws \Assert\LazyAssertionException
 */
public function execute()
{
    try {
        $emailAddress = $this->getEmailParam();
        $orders = $this->getOrders();
        $this->mailSender->send($emailAddress, $orders);

    } catch (InvalidArgumentException $e) {
        $this->messageManager->addError($e->getMessage());
    }
}
```



Default to Strict Encapsulation

```php
interface AbandonedEmailSenderInterface
{
    /**
     * Gathers data and sends email
     * @return void
     * @throws InsufficientDataException
     */
    public function sendForCart(
        CartInterface $cart
    );

    /** Returns the email transport */
    public function getEmailTransport();

    /** Sets the email transport **/
    public function setEmailTransport(
        EmailTransportInterface $emailSender
    );
}
```

WHY!?

“Getter and setter methods are evil

Allen Hollub, 2003

“Setters don’t mean anything, anyway

Marco Pivetta (@ocramius)

Getter & Setter Swarm

```
setStatusHistories([statusHistories : array|\Magento\Sales\Api\Da
setState(state : string):$this
setStatus(status : string):$this
setCouponCode(code : string):$this
setProtectCode(code : string):$this
setShippingDescription(description : string):$this
setIsVirtual(isVirtual : int):$this
setStoreId(id : int):$this
setCustomerId(id : int):$this
setBaseDiscountAmount(amount : float):$this
setBaseDiscountCanceled(baseDiscountCanceled : float):$this
setBaseDiscountInvoiced(baseDiscountInvoiced : float):$this
setBaseDiscountRefunded(baseDiscountRefunded : float):$this
setBaseGrandTotal(amount : float):$this
setBaseShippingAmount(amount : float):$this
setBaseShippingCanceled(baseShippingCanceled : float):$this
setBaseShippingInvoiced(baseShippingInvoiced : float):$this
setBaseShippingRefunded(baseShippingRefunded : float):$this
setBaseShippingTaxAmount(amount : float):$this
```

You can avoid it!

```php
interface AbandonedEmailSenderInterface
{
    /**
     * Gathers data and sends email
     * @return void
     * @throws InsufficientDataException
     */
    public function sendForCart(
        CartInterface $cart
    );
}
```


✓ Dependencies are injected

✓ Accessed directly or by private/ protected methods

```php
final class LoggingAbandonedEmailSender
    implements Api\AbandonedEmailSenderInterface
{
    private $logger;
    private $transport;

    public function __construct(
        TransportInterface $transport,
        LoggerInterface $logger
    ) {
        $this->transport = $transport;
        $this->logger = $logger;
    }

    /**
     * Gathers data and sends email
     * @return void
     * @throws InsufficientDataException
     */
    public function sendForCart(
        CartInterface $cart
    ) {
        // etc
    }
}
```


```php
final class LoggingAbandonedEmailSender
    implements Api\AbandonedEmailSenderInterface
{
    private $logger;
    private $transport;

    private function foo() {}
}
```

✓ Its easy to expose private API

◉ Its hard to do the opposite (BC breaks)

Some benefits:

- ✓ Prevent unnecessary BC breaks
- ✓ Encourage more expressive APIs
- ✓ Encourage better abstractions
- ✓ Simplify test-cases

IMPORTANT

- Avoid **final** for classes used in the OM's Proxy pattern

More on Defensive Programming

Extremely Defensive PHP - Marko Pivetta

Defensive Programming - Wikipedia

Write lots of good tests

7

Don’t Forget to Document!

At least with a README.md

README.md

View all of README.md

empty

README.md

magento2-extension-example

Magentostudy_News
This is a very basic custom module developed by considering all coding standards and structure of latest version of Magento 2. By referring this module magento developers can learn easily how to create a custom module in Magento 2
This module have following features:

- Table and data creation schema file
- News Creation/Edition/Deletion from Admin panel
- Admin Grid listing all News
- Image upload with each news item from admin panel
- List all news on frontend with pagination
- News access link added in footer
- Send news to all customers in email using Cron

1. Create a "code" folder inside your_magento_webroot/app folder.
2. Put this Module inside "code" folder. app/code/Magentostudy/News
3. execute command "php magento setup:upgrade"

What to include in README.md?

- ✓ Build Status
- ✓ Description
- ✓ Installation
- ✓ Features / Configuration
- ✓ LICENSE
- ✓ Support & Author
- ✓ Contributing Guide

Leverage Generic Composer Packages

DevelopersParadise

2016 / Opatija / Croatia

beberlei/assert

monolog/monolog

mockery/mockery

league/factory-muffin

league/tactician

prooph/event-sourcing

Stay Up to Date

devdocs.magento.com

phptherightway.com

magentotherightway.com

mostly Magento 1 ATM

alanstorm.com

For security best practices:

Leveraging Magento 2 to combat the OWASP Top Ten

(tomorrow 13:00)

Tales Seeparsan

Questions?

DevelopersParadise

2016 / Opatija / Croatia

Thank You!

DevelopersParadise

2016 / Opatija / Croatia