Refactoring Legacy Code

By:
Adam Culp
Twitter: @adamculp

https://joind.in/**11658**

Refactoring Legacy Code

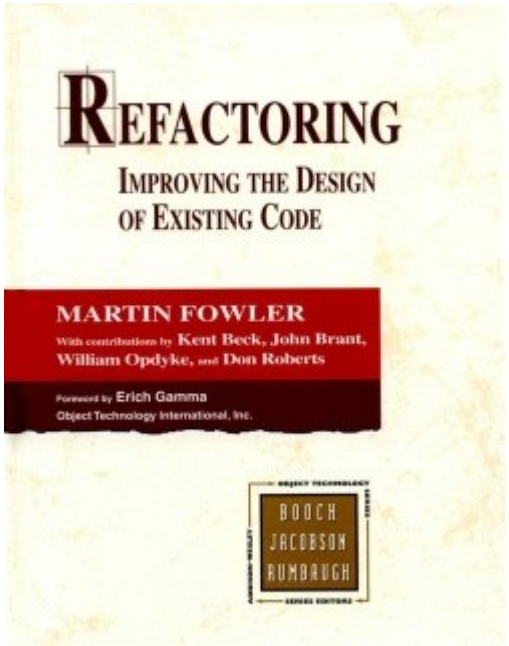
- **About me**
  - PHP 5.3 Certified
  - Consultant at Zend Technologies
  - Zend Certification Advisory Board
  - Organizer SoFloPHP (South Florida)
  - Organized SunshinePHP (Miami)
  - Long distance (ultra) runner
  - Judo Black Belt Instructor

Refactoring Legacy Code

- **Fan of iteration**
  - Pretty much everything requires iteration to do well:
    - Long distance running
    - Judo
    - Development
    - Evading project managers
    - Refactoring!

Refactoring Legacy Code

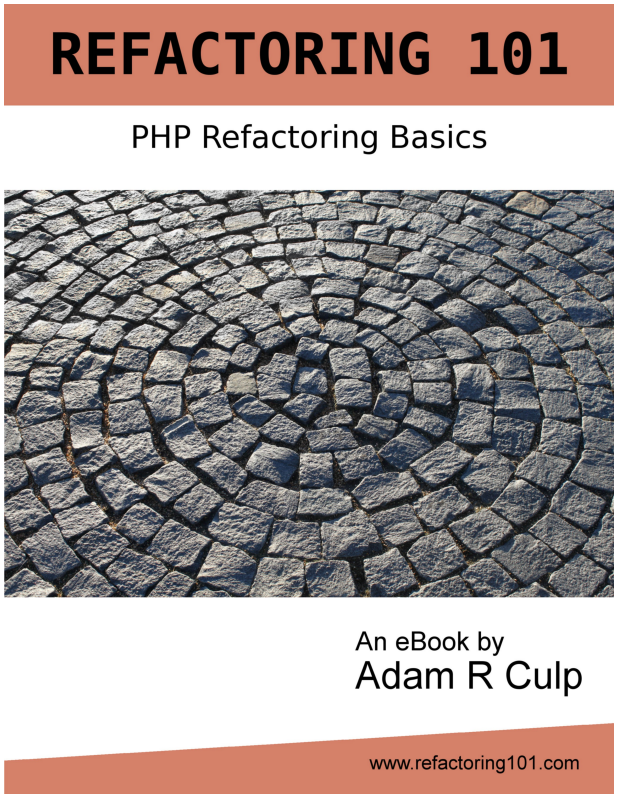
- **Refactoring**
  - "Refactoring; Improving The Design of Existing Code" book, by Martin Fowler.
  - https://github.com/adamculp/refactoring101 - for PHP code samples



SoFloPHP Users Group
WWW.SOFLOPHP.COM

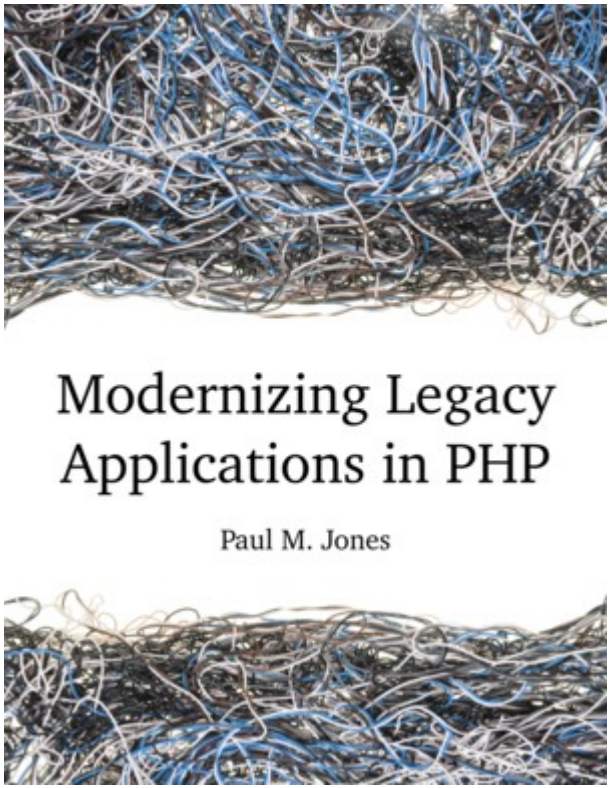
Refactoring Legacy Code

- My book
  - “Refactoring 101” on LeanPub.
  - http://refactoring101.com



Refactoring Legacy Code

- **Modernizing**
  - "Modernizing Legacy Applications in PHP" on LeanPub - by Paul M. Jones
  - http://mlaphp.com

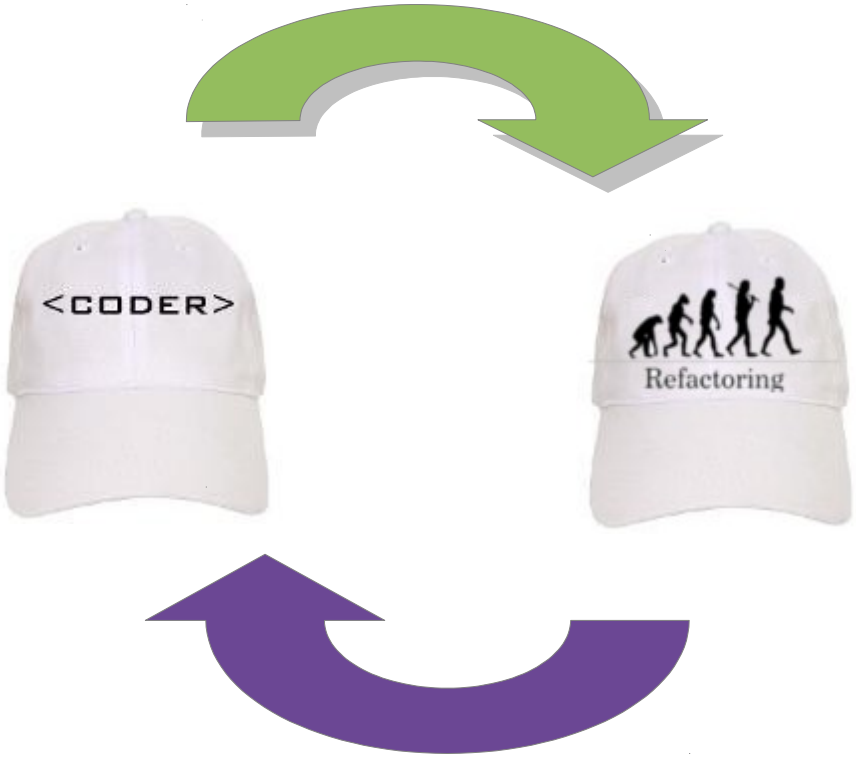


Refactoring Legacy Code

- **What is “refactoring”?**
  - “...process of changing a computer program's source code without modifying its external functional behavior...” *en.wikipedia.org/wiki/Refactoring*
  - No functionality added
  - Code quality

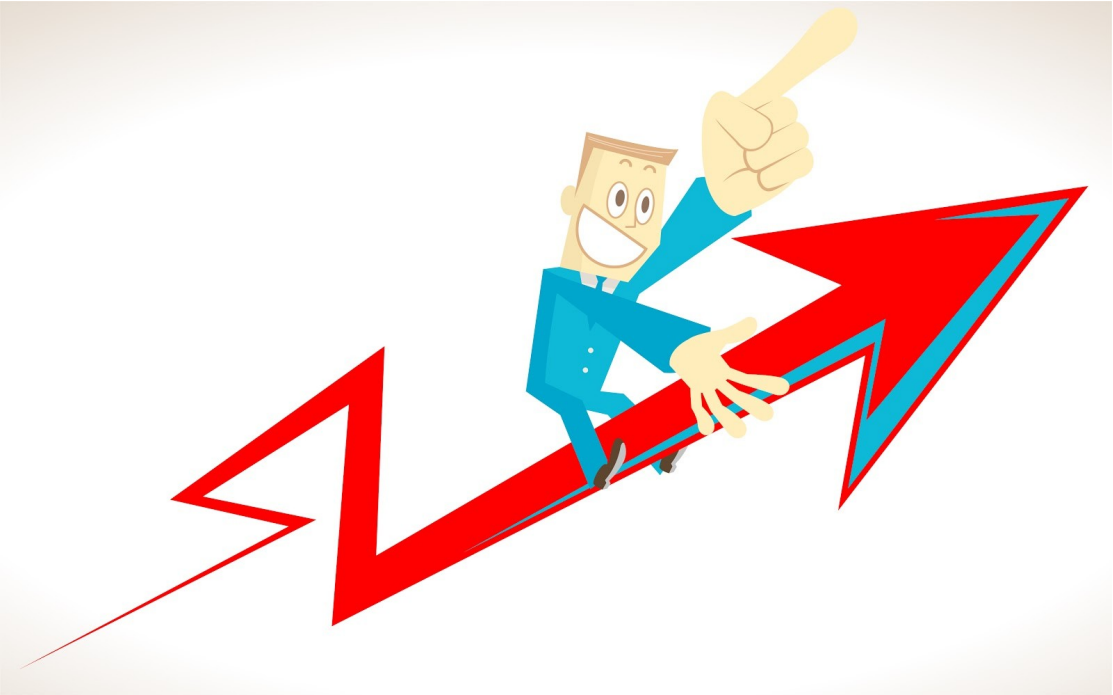
Refactoring Legacy Code

- **Two hats**
  - Adding Functionality Hat
  - Refactoring Hat
  - We add functionality, then refactor, then add more functionality ...



Refactoring Legacy Code

- **Then optimize**
  - Do not optimize while refactoring.
  - Separate step.
  - Refactoring is NOT optimizing.



Refactoring Legacy Code

- **Source Control**
  - Refactor in branch
  - Allows rollback

Refactoring Legacy Code

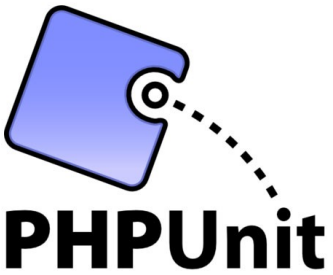
- **Editor/IDE**
  - Files by project
  - Search within project

Refactoring Legacy Code

- **Style Guide**
  - Framework Interop Group
    - http://php-fig.org
    - PSR
  - Faster reading
  - United team

Refactoring Legacy Code

- **Testing**
  - Consistent results
  - Prevents breaks



Behat

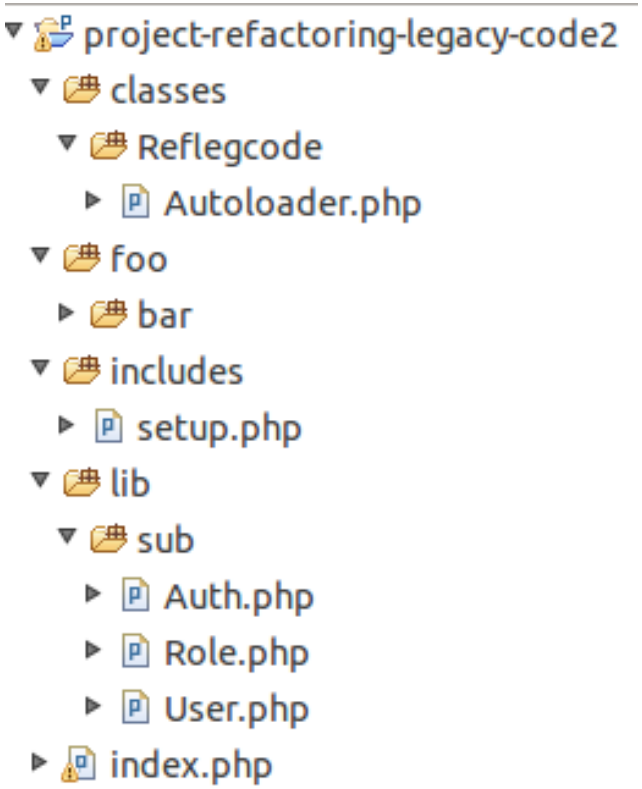
SoFloPHP Users Group
WWW.SOFLOPHP.COM

Refactoring Legacy Code

- **Autoloading**
  - Namespaces
  - PSR-0
    - Because legacy code typically used long class names rather than namespace separators.
  - Methods
    - Global function
    - Closure
    - Static or Instance Method *(preferred, if possible)*
    - __autoload() - PHP v 5.0
  - Need a central place for classes

Refactoring Legacy Code

- **Consolidate Classes**
  - Move to one location
    - Could be named “includes”, “classes”, “src”, “lib”, etc.
  - Search for include statements (include, include_once, require, require_once)



Refactoring Legacy Code

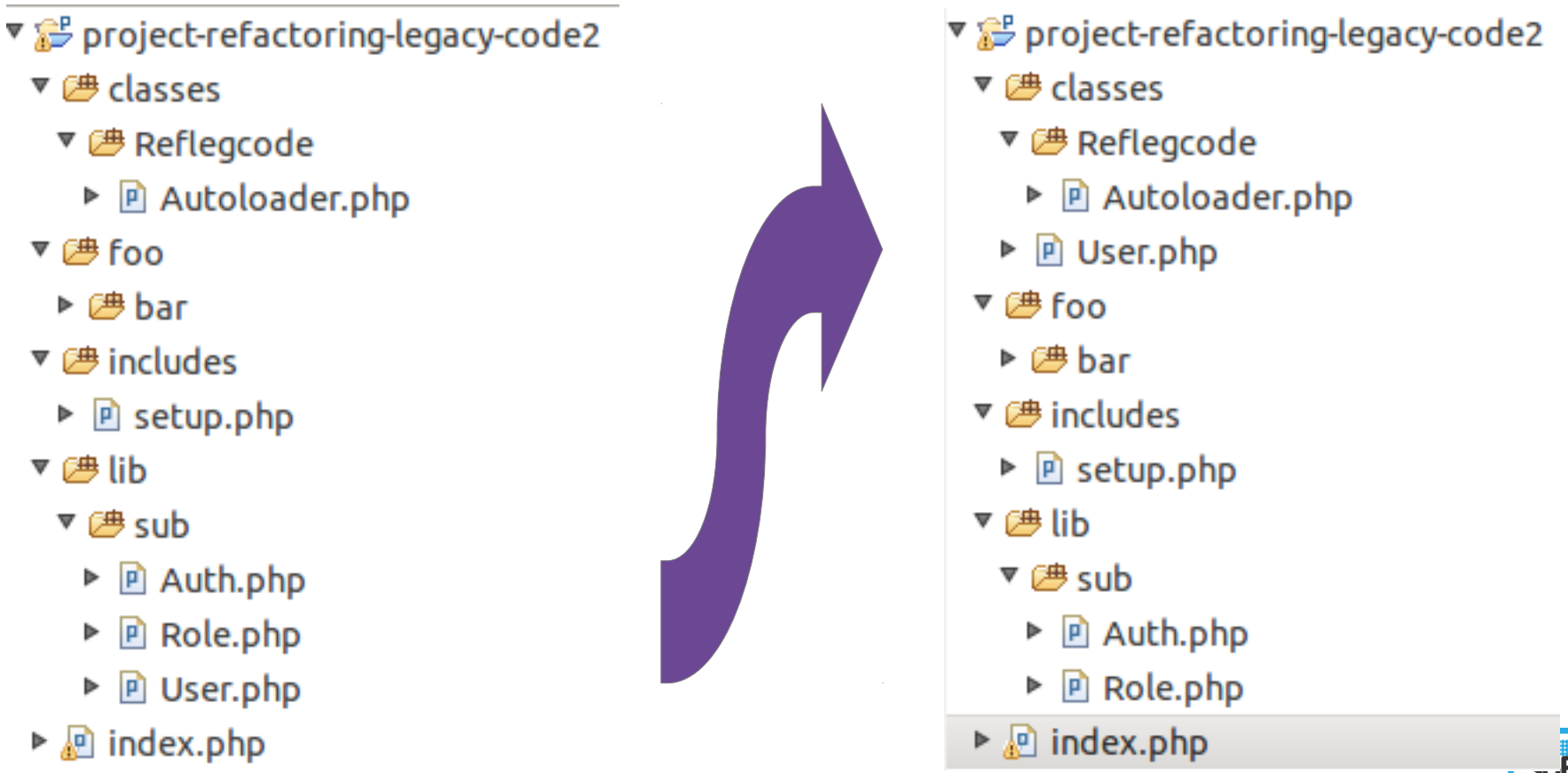
- **Consolidate Classes Step 1**
  - Search for include statements (include, include_once, require, require_once)

index.php

```php
<?php
require 'includes/setup.php';
require_once 'lib/sub/User.php';

// ...

$user = new User();

// ...

?>
```

Refactoring Legacy Code

- Consolidate Classes Step 2



Refactoring Legacy Code

- **Consolidate Classes Step 3**
  - User class is now autoloaded, no more require_once.

index.php

```php
<?php
require 'includes/setup.php';

// ...
// User class is now autoloaded
$user = new User();

// ...

?>
```

Refactoring Legacy Code

- **Global Dependencies**
  1. Search for global reference
  2. Move global calls to constructor
  3. Convert call to a constructor parameter
  4. Update call to class to pass parameter (DI)
  5. Repeat

Refactoring Legacy Code

- Global Use Example

Example.php | setup.php

```php
<?php
// some setup code, then:
$db = new Db('hostname', 'username', 'password');
?>
```

Example.php | setup.php

```php
<?php
class Example
{
    public function fetch()
    {
        global $db;
        return $db->query(...);
    }
}
?>
```

Refactoring Legacy Code

- **Global Cleanup Step 1**
  - Move global call to constructor

Example.php

```
1  <?php
2  class Example
3  {
4      protected $db;
5
6      public function __construct()
7      {
8          global $db;
9          $this->db = $db;
10     }
11
12     public function fetch()
13     {
14         return $this->db->query(...);
15     }
16 }
17 ?>
```

Refactoring Legacy Code

- **Global Cleanup Step 2**
  - Convert call to a constructor parameter

Example.php

```
1  <?php
2  class Example
3  {
4      protected $db;
5
6      public function __construct(Db $db)
7      {
8          $this->db = $db;
9      }
10
11     public function fetch()
12     {
13         return $this->db->query(...);
14     }
15 }
16 ?>
```

SoFloPHP Users Group
WWW.SOFLOPHP.COM

Refactoring Legacy Code

- **Global Cleanup Step 3**
  - Update call to class to pass parameter (DI)

page_script.php

```php
<?php
// a setup file that creates a $db variable
require 'includes/setup.php';

// ...

$example = new Example($db);
?>
```

Refactoring Legacy Code

- **Global Cleanup Repeat**
  - Look for more instances to clean up

Refactoring Legacy Code

- **Replacing “new”**
  1. Extract instantiation to constructor parameter. (one time)
  2. Extract block of creation code to new Factory class. (repeated)
  3. Update instantiation calls
  4. Repeat

Refactoring Legacy Code

- Replacing “new” Step 1 *(Single)*

ItemsGateway.php

```php
<?php
class ItemsGateway
{
    protected $db_host;
    protected $db_user;
    protected $db_pass;
    protected $db;

    public function __construct($db_host, $db_user, $db_pass)
    {
        $this->db_host = $db_host;
        $this->db_user = $db_user;
        $this->db_pass = $db_pass;

        $this->db = new Db($this->db_host, $this->db_user, $this->db_pass);
    }
    // ...
}
?>
```

Refactoring Legacy Code

- **Replacing "new" Step 2 *(Single)***
  - Pass Db object into class constructor. **(DI)**

ItemsGateway.php

```php
<?php
class ItemsGateway
{
    protected $db;

    public function __construct(Db $db)
    {
        $this->db = $db;
    }
    // ...
}
?>
```

Refactoring Legacy Code

- Replacing “new” Step 3 *(Multiple)*

ItemsGateway.php

```php
<?php
class ItemsGateway
{
    protected $db;

    public function __construct(Db $db)
    {
        $this->db = $db;
    }

    public function selectAll()
    {
        $rows = $this->db->query("SELECT * FROM items ORDER BY id");
        $item_collection = array();
        foreach ($rows as $row) {
            $item_collection[] = new Item($row);
        }
        return $item_collection;
    }
}
?>
```

Refactoring Legacy Code

- **Replacing "new" Step 4 *(Multiple)***
  - Create factory

ItemFactory.php

```php
<?php
class ItemFactory
{
    public function newInstance(array $item_data)
    {
        return new Item($item_data);
    }
}
?>
```

Refactoring Legacy Code

- Replacing “new” Step 5 *(Multiple)*

ItemsGateway.php

```php
<?php
class ItemsGateway
{
    protected $db;
    protected $item_factory;

    public function __construct(Db $db, ItemFactory $item_factory)
    {
        $this->db = $db;
        $this->item_factory = $item_factory;
    }

    public function selectAll()
    {
        $rows = $this->db->query("SELECT * FROM items ORDER BY id");
        $item_collection = array();
        foreach ($rows as $row) {
            $item_collection[] = $this->item_factory->newInstance($row);
        }
        return $item_collection;
    }
}
?>
```

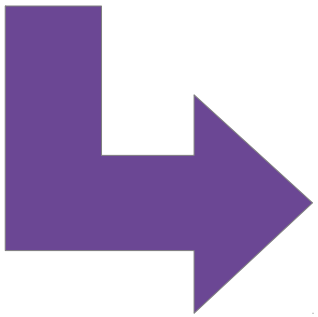
SoFloPHP Users Group
WWW.SOFLOPHP.COM

Refactoring Legacy Code

- Replacing “new” Step 6 *(Multiple)*

page_script.php

```php
<?php
// a setup file that creates a $db variable
require 'includes/setup.php';

// ...

// create a gateway
$items_gateway = new ItemsGateway($db_host, $db_user, $db_pass);

?>
```



page_script.php

```php
<?php
// a setup file that creates a $db variable
require 'includes/setup.php';

// create a gateway with its dependencies
$db = new Db($db_host, $db_user, $db_pass);
$item_factory = new ItemFactory;
$items_gateway = new ItemsGateway($db, $item_factory);
?>
```

Refactoring Legacy Code

- **Write Tests**
  - Code is fairly clean
  - Write tests for entire application
  - If not testable, refactor
    - Extract method
    - Replace temp with query
    - Etc.

Refactoring Legacy Code

- **Extract SQL**
  1. Search for SQL
  2. Move statement and relevent logic to Gateway class
  3. Create test for new class
  4. Alter code to use new method
  5. Repeat

Refactoring Legacy Code

- **Extract Logic**
  1. Search for uses of Gateway class outside of Transaction classes
  2. Extract logic to Transaction classes
  3. Test
  4. Write new tests where needed
  5. Repeat

Refactoring Legacy Code

- **Replace “includes”**
  - Search for left over includes
  - If in current class
    1. Copy contents into file directly
    2. Refactor for: no globals, no 'new', DI, return instead of output, no includes
  - More often
    1. Copy contents of include as-is to new class method
    2. Replace with in-line instantiation
    3. Search for other uses of same, and update them as well
    4. Delete original include file, regression test
  - Test, create new tests if needed
  - Repeat

Refactoring Legacy Code

- **Framework**
  - Code is able to be upgraded to framework
  - Create models
  - Create factories
  - Create modules
  - Move models & factories
  - Create/Update tests
  - Create controllers and views
  - Add service
  - Use events
  - Use 3rd party (vendor) libraries

Refactoring Legacy Code

- **Conclusion**
  - Do not refactor a broken application
  - **Always** have tests in place prior to refactor
    - Unit tests or
    - Functional tests or
    - Manual tests
  - Do things in small steps
  - Love iteration!

- **Thank you!**
  - Please rate at: https://joind.in/11658

Adam Culp

http://www.geekyboy.com

Twitter @adamculp

Questions?