Refactoring Legacy Code

By:
Adam Culp
Twitter: @adamculp

https://joind.in/**13560**

Refactoring Legacy Code

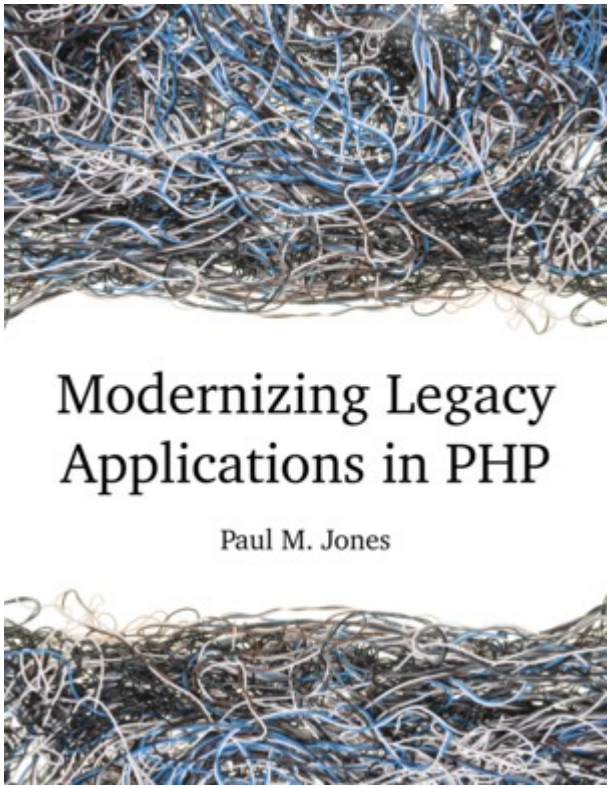
- **About me**
  - PHP 5.3 Certified
  - Consultant at Zend Technologies
  - Zend Certification Advisory Board
  - Organizer SoFloPHP (South Florida)
  - Organized SunshinePHP (Miami)
  - Long distance (ultra) runner
  - Judo Black Belt Instructor

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

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



Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?
  - Is Composer used in your project?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?
  - Is Composer used in your project?
  - Is the project using a framework?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?
  - Is Composer used in your project?
  - Is the project using a framework?
  - Are you unit testing?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?
  - Is Composer used in your project?
  - Is the project using a framework?
  - Are you unit testing?
  - Does your project avoid NIH?

Refactoring Legacy Code

- **What is "Legacy Code"**
  - Is there a coding standard for your project?
  - Is code using OOP?
  - Is Composer used in your project?
  - Is the project using a framework?
  - Are you unit testing?
  - Does your project avoid NIH?

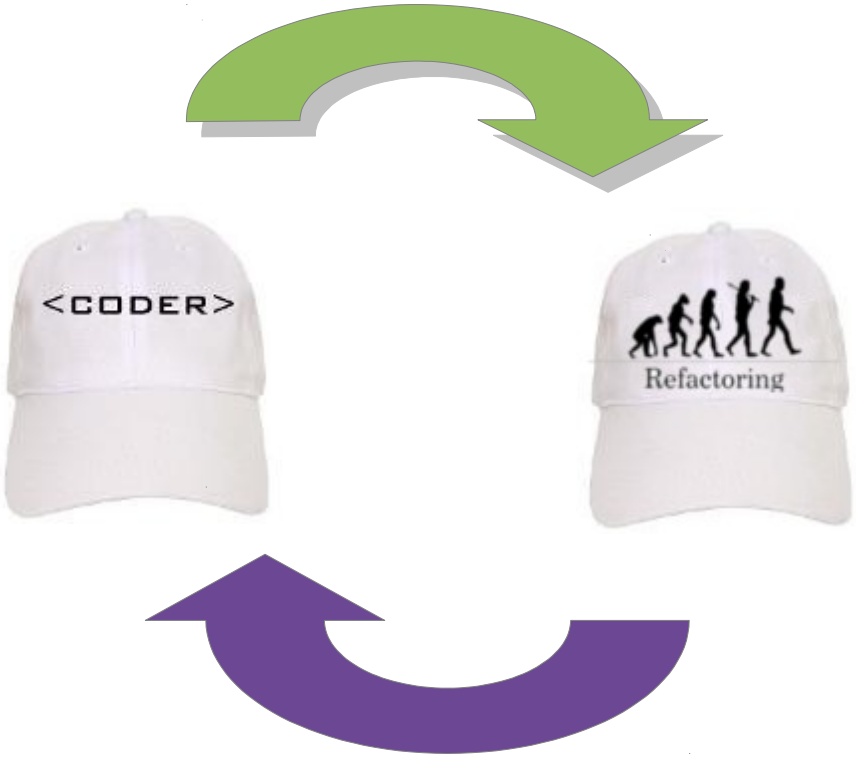
If you can answer "No" to any of these, you may be creating "Legacy Code"!!!

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

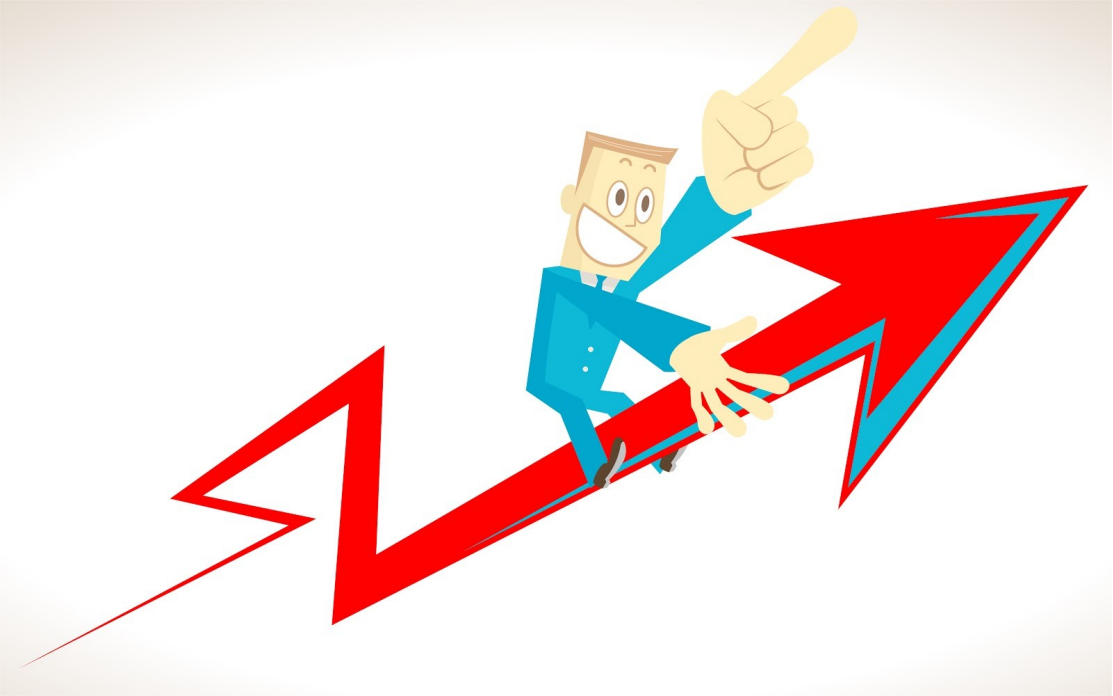
Refactoring Legacy Code

- **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

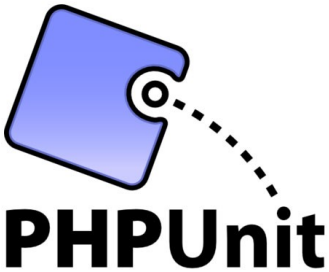
- **Editor/IDE**
  - Makes searching easier
  - Search within project

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
  - Speeds up development



Refactoring Legacy Code

- **Modernizing Steps**
  - Autoloading
  - Consolidate Classes
  - Cleanup Globals
  - Replace “new”
  - Create Tests
  - Extract SQL
  - Extract Logic
  - Replace Remaining “Includes”

Refactoring Legacy Code

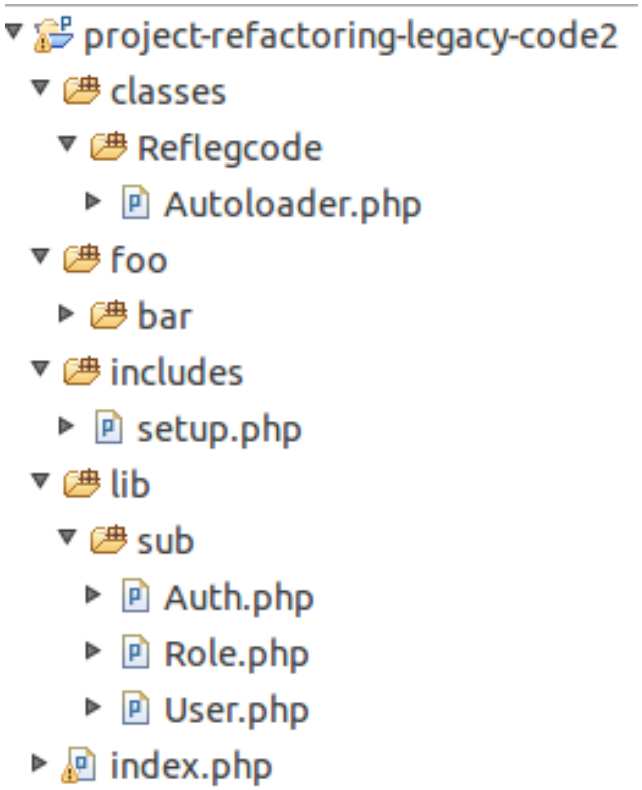
- **Autoloading**
  - Namespaces
  - PSR-0
    - Legacy code typically used long class names
      - Usage = My_Long_Class_Name
      - Translates to "/My/Long/Class/Name.php"
      - Class = My_Long_Class_Name
    - If not, then PSR-4
      - Use My\Great\Namespace\Name
      - Translates to /My/Great/Namespace/Name.php
      - Class = Name

Refactoring Legacy Code

- **Autoloading Approaches**
  - Approaches
    - 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.



Refactoring Legacy Code

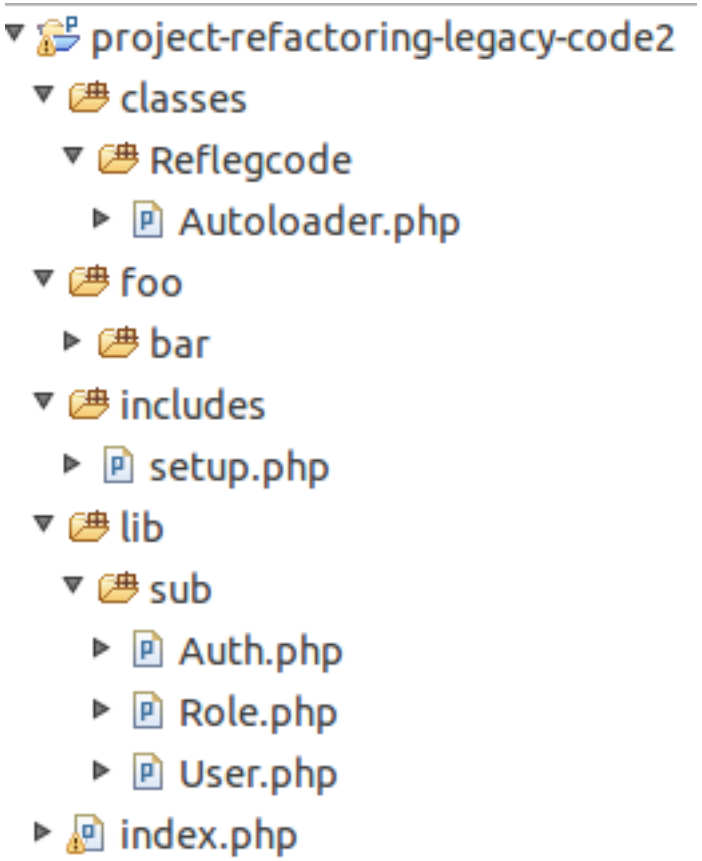
- **Consolidate Classes Step 1**
  - Search for include statements
    - (include, include_once, require, require_once)

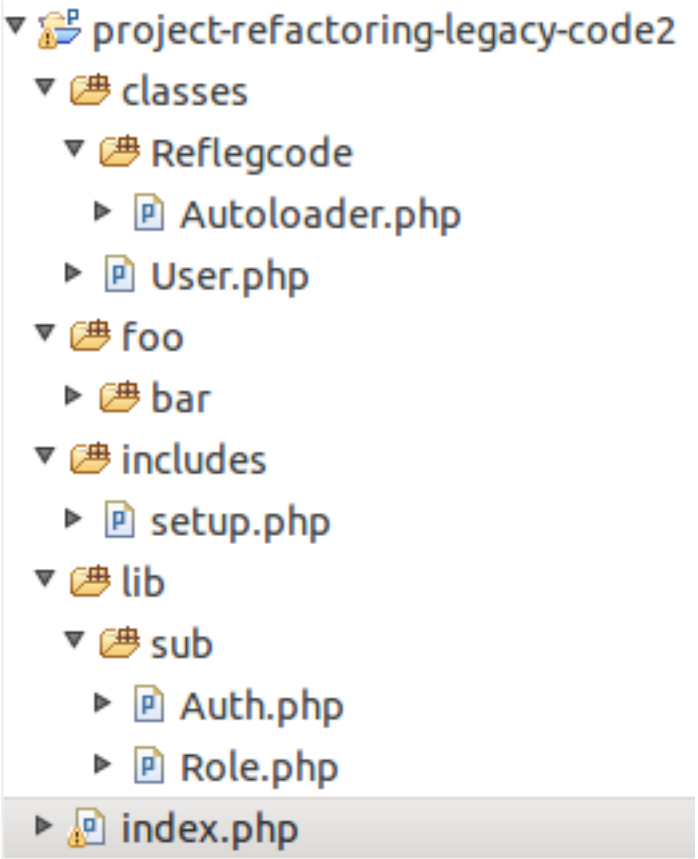
index.php

```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

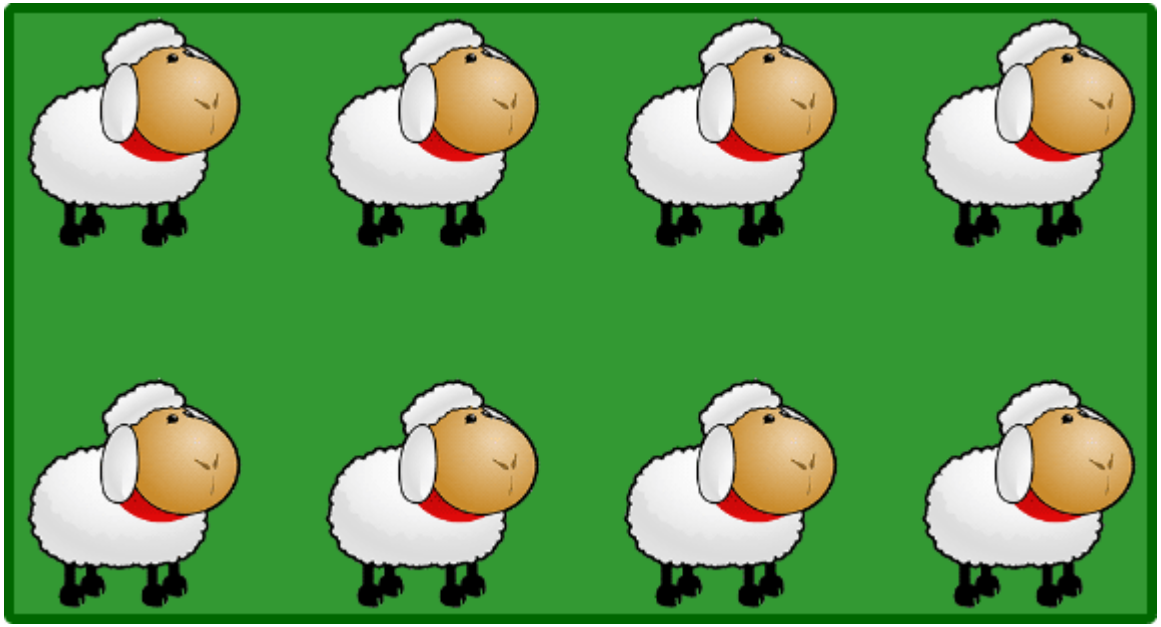
- **Consolidate Classes Step 3**
  - User class is now autoloaded, no more require_once.

index.php

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

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

// ...

?>
```

Refactoring Legacy Code

- **Consolidate Classes Repeat**
  - Search for more instances



Refactoring Legacy Code

- **Cleanup "Global" Dependencies Steps**
  1. Search for global reference
  2. Move global calls to constructor
  3. Convert global call to a constructor parameter
  4. Update global call to a class
  5. Instantiate new class and pass as parameter (DI)
  6. Repeat

Refactoring Legacy Code

- **Global Cleanup Step 1**
  - Search for global reference

Example.php | setup.php

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

Example.php | setup.php

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

Refactoring Legacy Code

- **Global Cleanup Step 2 & 3**
  - Move global call to constructor
  - Pass values as properties

Example.php

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

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

    public function fetch()
    {
        return $this->db->query(...);
    }
}
?>
```

Refactoring Legacy Code

- **Global Cleanup Step 4**
  - Convert call to a constructor parameter

Example.php

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

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

    public function fetch()
    {
        return $this->db->query(...);
    }
}
?>
```

Refactoring Legacy Code

- **Global Cleanup Step 5**
  - Instantiate new class
  - Inject as 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

- **Steps to Replacing "new"**
  1. Search for "new"
  2. Extract instantiation to constructor parameter. (if one time)
     - Or extract block of creation code to new Factory class. (if repeated)
  3. Update instantiation calls
  4. Repeat

Refactoring Legacy Code

- **Replacing "new" Step 1 *(Single)***
  - Before

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)***
  - Inject Db object into class constructor. **(DI)**
  - No longer instantiating within class

ItemsGateway.php

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

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

Refactoring Legacy Code

- Replacing "new" *(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 3 *(Multiple)***
  - Create factory
  - Extract "new" call to new factory

ItemFactory.php

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

Refactoring Legacy Code

- **Replacing "new" Step 4 *(Multiple)***
  - Update instantiation calls

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

- **Replacing "new" Step 4 *(Multiple)***
  - Call to factory

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;
    }
}
?>
```

Refactoring Legacy Code

- Replacing “new” Repeat

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 relevant 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

- **Additional Possibilities**
  - Can now implement framework
  - Leverage services
  - Leverage events
  - Use Composer

Refactoring Legacy Code

- **Why refactor**
  - Less bugs
  - Faster development
  - More stable
  - Easier onboarding
  - Save $$$

Refactoring Legacy Code

- **When/How to refactor**
  - Ask boss for time
  - “Leave it cleaner than you found it”
  - Do it on your own, it makes YOUR life easier

Refactoring Legacy Code

- **Convince the boss**
  - Three kinds of managers:
    - Tech Savvy
    - Quality Centric
    - Time Driven

Refactoring Legacy Code

- **Tech Savvy boss**
  - Lower cyclomatic complexity
  - SOLID
  - Separation of concerns
  - MVC/Framework
  - Less bugs
  - Easier onboarding

Refactoring Legacy Code

- **Quality Centric boss**
  - Code quality
  - Tests
  - Happier customers
  - Less bugs

Refactoring Legacy Code

- **Time driven boss**
  - Faster feature delivery
  - Higher productivity - less time reading
  - Faster onboarding
  - Less testing time
  - Less debugging

Refactoring Legacy Code

- **Concluding Thoughts**
  - Do not refactor a broken application
  - 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/13560

Adam Culp

http://www.geekyboy.com

http://RunGeekRadio.com

Twitter @adamculp

Questions?