



**DRUPALCON
LONDON**

This Code Stinks!

Presented by Larry Garfield

@Crell

- ❖ Senior Architect, Palantir.net
- ❖ Drupal Association Board of Directors
- ❖ Web Services & Context Initiative Owner
- ❖ Co-Author, *Drupal 7 Module Development*, Packt Publishing
- ❖ Architectural Gadfly
- ❖ Nerf Gunslinger



Code smells

- ❖ *In computer programming, code smell is any symptom in the source code of a program that possibly indicates a deeper problem.*
- ❖ *...code smells are heuristics to indicate when to refactor, and what specific refactoring techniques to use.*
- ❖ *Determining what is and is not a code smell is often a subjective judgment, and will often vary by language, developer and development methodology.*

http://en.wikipedia.org/wiki/Code_smell

"You can smell that there will be a bug
here eventually..."

They're not rules...



They're more like guidelines.

The Seven Stinky Smells...



<http://www.flickr.com/photos/28122162@N04/5979816310/>

#1

And

```
/**
 * Retrieves, populates, and processes a form.
 */
function drupal_form_submit($form_id, &$form_state) {
    // ...
}
```


If I want to retrieve without processing...?

#Fail

SolrPhpClient

```
/**
 * Add an array of Solr Documents to the index all at once
 */
public function addDocuments($documents, ...) {
    // ...
    $rawPost = '<add ... >';
    foreach ($documents as $document) {
        $rawPost .= $this->_documentToXmlFragment($document);
    }
    $rawPost .= '</add>';
    return $this->add($rawPost);
}
```


God Objects

- ❖ Do more than one thing
- ❖ Know too much
- ❖ Remember composition!



#2

Or

This function does X, **or** sometimes Y.

This function returns A, but **sometimes** B.

Good luck figuring out which one...

_registry_check_code()

```
/**
 * Helper to check for a resource in the registry.
 *
 * @param $type
 *     The type of resource we are looking up, or one of the constants
 *     REGISTRY_RESET_LOOKUP_CACHE or REGISTRY_WRITE_LOOKUP_CACHE, which
 *     signal that we should reset or write the cache, respectively.
 * @param $name
 *     The name of the resource, or NULL if either of the REGISTRY_*
constants
 *     is passed in.
 * @return
 *     TRUE if the resource was found, FALSE if not.
 *     NULL if either of the REGISTRY_* constants is passed in as $type.
 */
```

`_registry_check_code()`

What the hell does this function even do???

Or... another function/method

- ✧ Separate function for each task
- ✧ Shared data?
 - ✧ Be classy...

```
class Registry {  
    protected $lookupCache;  
    protected $cacheUpdateNeeded;  
  
    public function lookup($name) { }  
  
    public function clearCache() { }  
  
    public function writeCache() { }  
}
```

```
$r = new Registry();  
spl_autoload_register(array($r, 'lookup'));  
drupal_register_shutdown_function(array($r, 'writeCache'));
```


#3

If

"Overly complex code leads to overly complex bugs."

--My former boss


```
function comment_node_view($node, $view_mode)
```

```

function comment_node_view($node, $view_mode) {
  if (...) {
    if (...) {
      // ...
    }
    elseif (...) {
      if (...) {
        if (...) {
          // ...
          if (...) {
            // ...
          }
        }
      }
      if (...) {
        if (...) {
          // ...
        }
        else {
          // ...
        }
      }
    }
  }
  elseif (...) {
    if (...) {
      // ...
      if (...) {
        // ...
        if (...) {
          // ...
          if (...) {
            // ...
          }
        }
      }
      else {
        // ...
      }
    }
  }
  // ...
  if (...) {
    // ...
  }
}
}

```

Cyclomatic Complexity

"The cyclomatic complexity of a section of source code is the count of the number of linearly independent paths through the source code."

http://en.wikipedia.org/wiki/Cyclomatic_complexity

Unit test that...

```
function comment_node_view($node, $view_mode) {  
  if (...) {  
    if (...) {  
      // ...  
    }  
    elseif (...) {  
      if (...) {  
        if (...) {  
          // ...  
          if (...) {  
            // ...  
          }  
        }  
      }  
      if (...) {  
        if (...) {  
          // ...  
        }  
        else {  
          // ...  
        }  
      }  
    }  
    elseif (...) {  
      if (...) {  
        // ...  
        if (...) {  
          // ...  
          if (...) {  
            // ...  
            if (...) {  
              // ...  
            }  
          }  
        }  
      }  
      else {  
        // ...  
      }  
    }  
  }  
  // ...  
  if (...) {  
    // ...  
  }  
}
```

... I dare you.

"Tabs are 8 characters, and thus indentations are also 8 characters.

Now, some people will claim that having 8-character indentations makes the code move too far to the right, and makes it hard to read on a 80-character terminal screen. **The answer to that is that if you need more than 3 levels of indentation, you're screwed anyway, and should fix your program."**

--<http://www.kernel.org/doc/Documentation/CodingStyle>

Run-Time Type Identification

```
function entity_label($entity_type, $entity) {  
    switch ($entity_type) {  
        case 'node':  
            return $entity->title;  
        case 'user':  
            return $entity->name;  
        case 'comment':  
            return $entity->subject;  
    }  
}
```


Polymorphism (procedural)

```
function entity_label($entity_type, $entity) {  
    $label = FALSE;  
    $info = entity_get_info($entity_type);  
    if (isset($info['label callback']) &&  
function_exists($info['label callback'])) {  
        $label = $info['label callback']($entity, $entity_type);  
    }  
    elseif (!empty($info['entity keys']['label']) &&  
isset($entity->{$info['entity keys']['label']})) {  
        $label = $entity->{$info['entity keys']['label']};  
    }  
  
    return $label;  
}
```

Polymorphism (OOP)

```
function get_label(Entity $entity) {  
    $entity->label();  
}
```

// or just

```
$entity->label();
```

#4

DrupalWebTestCase

Unit testing

"[U]nit testing is a method by which individual units of source code are tested to determine if they are fit for use."

"A unit is the smallest testable part of an application."

DrupalWebTestCase

```
class DrupalWebTestCase {  
    protected function setUp() {  
        // Generate complete fake database install.  
        // Generate complete language environment.  
        // Screw around with shutdown functions.  
  
        // Create a files directory(!)  
        // Change PHP environment.  
        // Delete a bunch of globals(!)  
        // Set a bunch of other globals(!!!)  
  
        // Run a complete install of Drupal.  
        // Populate the registry.  
        // Install various modules.  
        // Reset/rebuild all data structures after enabling the modules.  
  
        // Run cron(?)  
        // Simulate a login.  
        // Muck about with variable_set(), which is global.  
    }  
}
```


DrupalWebTestCase

Unit = 1 Drupal install

Fail...

System testing

- ❖ Conducted on a complete, integrated system
- ❖ "Testing the whole system"
- ❖ Yep, that's `DrupalWebTestCase`

DrupalUnitTestCase

- ❖ No fresh install
- ❖ Empty database connection
- ❖ Empty directory
- ❖ ... 1000x times faster

If you can't unit test it, your code is wrong.

Being more testable

- ❖ Avoid globals
- ❖ Avoid statics
- ❖ Dependency injection, dependency injection, dependency injection
- ❖ Minimize singletons (e.g., function calls)

#5

Documentation

You can't teach what you don't know.

You don't know what you can't teach.

You don't understand what you can't document.

I can't understand what you don't document.

```
abstract class FileTransferFTP extends FileTransfer {
    /**
     * Return an object which can implement the FTP protocol.
     *
     * @param string $jail
     * @param array $settings
     * @return FileTransferFTP
     *     The appropriate FileTransferFTP subclass based on available
     *     options. If the FTP PHP extension is available, use it.
     */
    static function factory($jail, $settings) { }
}
```


Date.module

```
/**
 * Getter callback to return date values as timestamp in
 * UTC from the field.
 */
function date_entity_metadata_field_getter($object, array
$options, $name, $obj_type, &$amp;context) { }
```

Lack of comments

- ❖ Laziness
- ❖ Lack of comprehension
- ❖ Indifference
- ❖ Embarrassment

What to document

- ❖ *Every* function
- ❖ *Every* method
- ❖ *Every* class
- ❖ *Every* object property
- ❖ *Every* constant
- ❖ *Every* parameter

- ❖ *No* exceptions

#6

Inappropriate intimacy



<http://www.flickr.com/photos/usachicago/4483603789>

Inappropriate intimacy

Inappropriate Intimacy is a Code Smell that describes a method that has too much intimate knowledge of another class or method's inner workings, inner data, etc.

--<http://c2.com/cgi/wiki?InappropriateIntimacy>

Tight coupling

- ❖ Content coupling (implementation details)
- ❖ Common coupling (shared globals)
- ❖ External coupling (common exchange format)
- ❖ Control coupling (one controls another)
- ❖ Data-structured coupling (excessive data)
- ❖ Data coupling (parameters only)
- ❖ Message coupling (intermediary for data)

High

Low

What is the knock-on effect when
implementation details change?

How badly does this optimization break the API?

Any Drupal examples...?

- ✧ Form API
- ✧ Render API / `hook_page_alter()`
- ✧ Field /Language API
- ✧ Node API (`hook_node_load()`)

... Crap

```

$fields = array('n.nid', 'n.title', 'u.name');
$tables = array(
    'n' => array(
        'type' => NULL,
        'table' => 'node',
        'alias' => 'n',
        'condition' => array(),
        'arguments' => NULL,
        'all fields' => FALSE,
    ),
    'u' => array(
        'type' => 'INNER JOIN',
        'table' => 'user',
        'alias' => 'u',
        'condition' => 'u.uid = n.nid',
        'arguments' => array(),
        'all_fields' => FALSE,
    ),
);
$where = array(
    array(
        'field' => 'u.status',
        'value' => 1,
        'operator' => '=',
    ),
    array(
        'field' => 'n.created',
        'value' => REQUEST_TIME - 3600,
        'operator' => '>',
    ),
);
$order_by = array(
    'n.title' => 'ASC',
);

db_select($tables, $fields, $where, NULL, $order_by, array(), NULL, array(0, 5));

```

```
$select = db_select('node', 'n');  
$select->join('user', 'u', 'u.uid = n.uid');  
$select  
->fields('n', array('nid', 'title'))  
->fields('u', array('name'))  
->condition('u.status', 1)  
->condition('n.created', REQUEST_TIME - 3600, '>')  
->orderBy('n.title', $direction)  
->execute();
```


Solution

- ✧ Interfaces
- ✧ Documented interfaces
- ✧ Well-documented interfaces

#7

Impurity



<http://www.flickr.com/photos/fransdewit/5661512661/>

Pure function

1. Give the same obvious input, get the same output
2. No (observable) side effects or I/O

Signs of Impurity

- ✧ Side effects
- ✧ Globals
- ✧ Cannot be repeated

Unless side effects are the goal...

Keep those self-contained.

```
function drupal_theme_initialize() {
    global $theme, $user, $theme_key;

    // If $theme is already set, assume the others are set, too.
    if (isset($theme)) {
        return;
    }

    // ...

    $custom_theme = menu_get_custom_theme();
    $theme = !empty($custom_theme) ? $custom_theme : $theme;

    // Store the identifier for retrieving theme settings with.
    $theme_key = $theme;

    // ...

    // Themes can have alter functions, so reset the cache.
    drupal_static_reset('drupal_alter');

    drupal_add_js($setting, 'setting');
}
```

```
class Theme {  
    protected $themeKey;  
    public function __construct($user) { }  
  
    public function theme($hook, $vars = array()) {  
        $this->themeKey;  
    }  
  
    public function getJs() {  
        return $this->JsInfo;  
    }  
}
```

```
function theme($hook, $vars = array()) {  
    static $theme;  
  
    if (empty($theme)) {  
        $theme = new Theme($user);  
        drupal_add_js($theme->getJs());  
    }  
  
    return $theme->theme($hook, $vars);  
}
```


Can't I do anything right?

Good smells

- ❖ Single-purpose
- ❖ Self-contained
- ❖ Predictable
- ❖ Repeatable
- ❖ Unit testable
- ❖ Documented



See also...

<http://chicago2011.drupal.org/sessions/aphorisms-api-design>



See also...

- ❖ <http://www.codinghorror.com/blog/2006/05/code-smells.html>
- ❖ <http://wiki.java.net/bin/view/People/SmellsToRefactorings>
- ❖ <http://www.joelonsoftware.com/articles/Wrong.html>
- ❖ <http://TheDailyWTF.com/>

What did you think?

Locate this session on the
DrupalCon London website:

<http://london2011.drupal.org/conference/schedule>

Click the “Take the survey” link

THANK YOU!

