[***Techcrunch kill off unnecessary code reviews***](http://techcrunch.com/2015/10/16/codacy-raises-raises-1-1m-to-kill-off-unnecessary-code-reviews/)

[***Codacy Website***](https://www.codacy.com/)

*“ Developers spend a lot of time reviewing code (between 20%-50%), even today, when code reviewing can in fact be automated much of the time. Code reviews are described as “technical debt” and some say this ‘debt’ amounts to a trillion dollars globally.*

*Co-founder Jaime Jorge told me: “We believe the right way to introduce a technology like static analysis, that has been present for decades, is to tightly couple it with the workflow that evolved drastically in the last couple of years. Building code analysis in-house gives us an advantage as we can provide more meaningful results and react faster to the specific needs of programming communities.” (Techcruch article) “*





*Error Prone 85%*













***What about the controllers ?***

1. Is the code readable as written?  Does it require additional comments, better naming, or general refactoring to be easily understandable?
2. Does each function contain a (very) brief comment describing functionality, inputs, and outputs?

***PHP Standard Recommendation 1 (PSR 1)***

* Files MUST use only <?php and <?= tags.
* Files MUST use only UTF-8 without BOM for PHP code.
* Files SHOULD *either* declare symbols (classes, functions, constants, etc.) *or* cause side-effects (e.g. generate output, change .ini settings, etc.) but SHOULD NOT do both.
* Class names MUST be declared in StudlyCaps.
* Class constants MUST be declared in all uppercase with underscore separators.
* Method names MUST be declared in camelCase.

***PHP Standard Recommendation 2 (PSR 2)***

* Code MUST follow a “coding style guide” PSR [[PSR-1](https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-1-basic-coding-standard.md)].
* Code MUST use 4 spaces for indentation, not tabs.
* There MUST NOT be a hard limit on line length; the soft limit MUST be 120 characters; lines SHOULD be 80 characters or less.
* There MUST be one blank line after the namespace declaration, and there MUST be one blank line after the block of use declarations.
* Opening braces for classes MUST go on the next line, and closing braces MUST go on the next line after the body.
* Opening braces for methods MUST go on the next line, and closing braces MUST go on the next line after the body.
* Visibility MUST be declared on all properties and methods; abstract and final MUST be declared before the visibility; static MUST be declared after the visibility.
* Control structure keywords MUST have one space after them; method and function calls MUST NOT.
* Opening braces for control structures MUST go on the same line, and closing braces MUST go on the next line after the body.
* Opening parentheses for control structures MUST NOT have a space after them, and closing parentheses for control structures MUST NOT have a space before.

**DocBlock Levels:** DocBlocks should be placed at specific levels in the code, for example:

* **File Level DocBlock -**  Identifies the purpose of the file, ownership, version and contingencies
* **Class Level DocBlock -**  Identifies the purpose of the class, ownership and version
* **Function/Method Level DocBlock -**  Identifies the purpose of the function/method as well as ownership/version.  Also includes definitions of parameters and should always include code samples

<?php

/\*\*

\* Gig.php controller for Gigs at SCC

\* @package ITC 260 Gig Central CodeIgnitor

\* @subpackage Gig Controller

\* @author Patricia Barker <patriciabethbarker@gmail.com>

\* @version 2.2 2015/06/12

\* @link http://www.tcbcommercialproperties.com/sandbox/codeignitor/

\* @license http://www.apache.org/licenses/LICENSE-2.0

\* @see Gig\_model.php

\* @see view/gigs/index.php

\* @see view/gigs/view.php

\* @see view/gigs/add.php

\* @todo none

\*/

/\*\*

\* Gigs\_form controller

\*

\*

\* @see Gig\_model.php

\* @todo none

\*/

**class Gig extends CI\_Controller**

**{**//begin controller

  /\*\*

 \* Loads default data into object

 \* @param none

 \* @return void

 \* @todo none

 \*/

**public function \_\_construct()**

**{**//begin constructor

**parent::\_\_construct();**

**$this->load->model('gig\_model');**

**$this->config->set\_item('banner', 'Global News Banner');**

**}#end constructor**

**public function index()**

**{**//begin function index

**$data['gigs'] = $this->gig\_model->get\_gigs();**

**$data['title']= 'Gigs';**

**$this->load->view('gigs/index', $data);**

**}**#end function index

**public function view($slug = NULL)**

**{**//begin function index

**$data['gig'] = $this->gig\_model->get\_gigs($slug);**

**if (empty($data['gig']))**

**{**

**show\_404();**

**}**

**$data['title']= 'Gig';**

**$this->load->view('gigs/view', $data);**

**}**#end function index

**public function add()**

**{**

**$this->load->helper('form');**

**$this->load->library('form\_validation');**

**$data['title'] = 'Add a new gig';**

**$this->form\_validation->set\_rules('Name', 'Company Name', 'required');**

**$this->form\_validation->set\_rules('Email', 'Email', 'required');**

**if ($this->form\_validation->run() === FALSE)**

       {//create form to add gigs

        **$this->load->view('gigs/add', $data);**

**}**

**else**

**{**//this processes

**$data['gigs'] = $this->gig\_model->get\_gigs();**

**$data['title']= 'Gigs';**

**$this->gig\_model->add\_gig();**

**$this->load->view('gigs/success');**

**//            $this->load->view('gigs/index', $data);**

**}**

**}**#end function addForm**()**

**public function sendnewsletter()**

**{**

 // XXX TODO FIXME add authentication to this so not just anybody can spam everybody with newsletters

    // Ensure that the request is a POST. POST signifies an intentional action, as opposed to passively requesting information.

**if( TRUE || $this->input->method(/\*capitalizeReturnValue=\*/TRUE) === "POST" )**

**{**

**//** XXX TODO check for a "secret" post value to ensure that a GigCentral process authorized this. Random POST requests shouldn't trigger this

**$this->load->library('email');**

**$this->config->load('email');**

**$this->load->helper('url');**

 // XXX add config option for this

**$email = $this->config->item('autoemail\_from\_address'); $name = $this->config->item('autoemail\_from\_name');**

**$subject = $this->config->item('autoemail\_from\_name');**

**$data['gigs'] = $this->gig\_model->get\_gigs();**

**ob\_start();** // Begin capture of view output for email

**$this->load->view('gigs/gignewsletter-email',$data);**

**$message = ob\_get\_contents();**

**ob\_end\_clean();**

// discard the output buffer. it's contents are saved into $message, and we don't want to print it directly

**$this->load->model('profile\_model');**

**foreach( $this->profile\_model->get\_profiles(/\*slug=\*/FALSE,** /\*subscribedOnly=\*/**TRUE) as $subscribedUser)**

**{**

**$this->email->from($email, $name);**

**$this->email->to(  $subscribedUser['email'] );**

**$this->email->subject($subject);**

**$this->email->set\_mailtype("html");**

**$this->email->message($message);**

**if(! $this->email->send())**

**{**

// XXX TODO: Log failure-to-sends; too many failures in a row may mean the email is no longer valid

**}**

**echo "Sending mail to " . $subscribedUser['email'] . " : <br />" . $message . "<hr />\n";**

**}**

**}**

**else**

**{**

**$this->output->set\_status\_header('405');**

**$this->output->set\_header("Allow: POST");**

**echo "Sorry, we don't allow random GET requests to kick off subscription emails; perhaps try a POST?";**

**}**

**}**

**}**#end Gigs class/controller