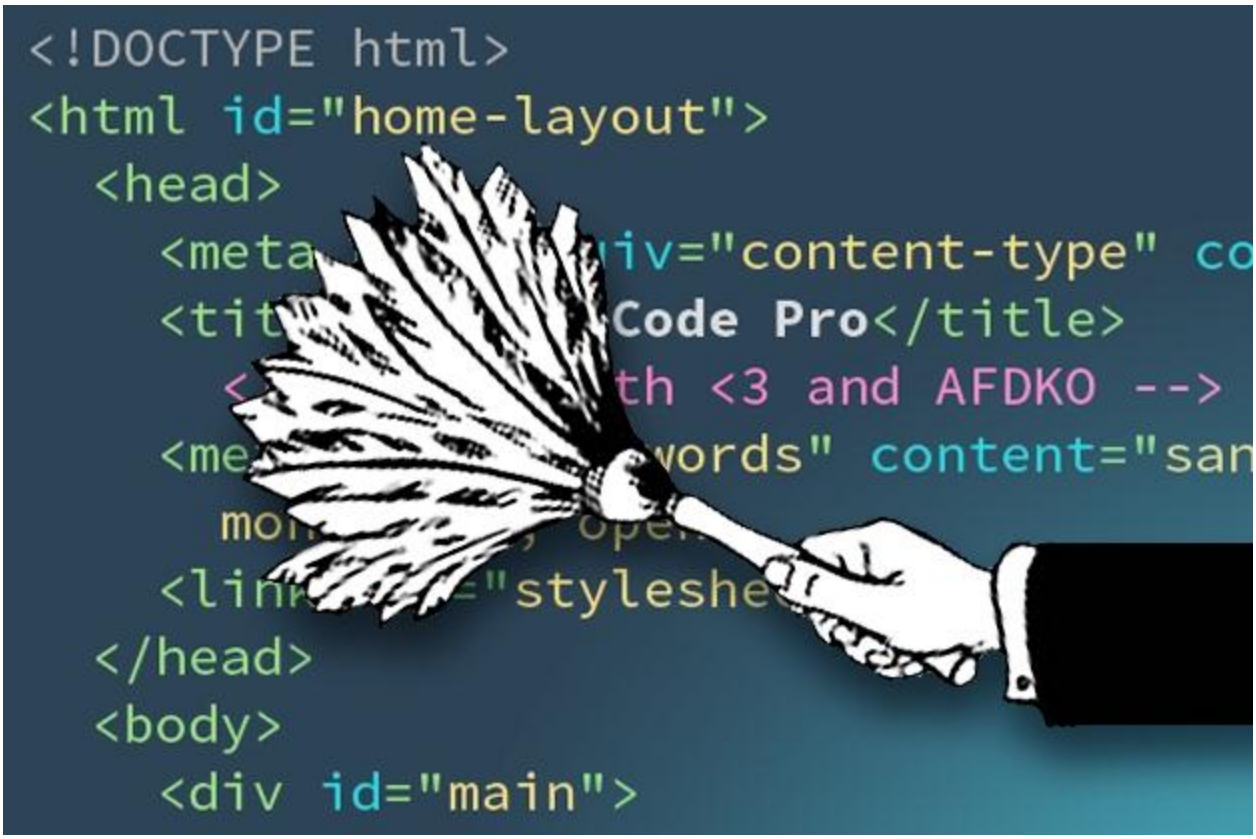


Clean, high quality code: a guide on how to become a better programmer

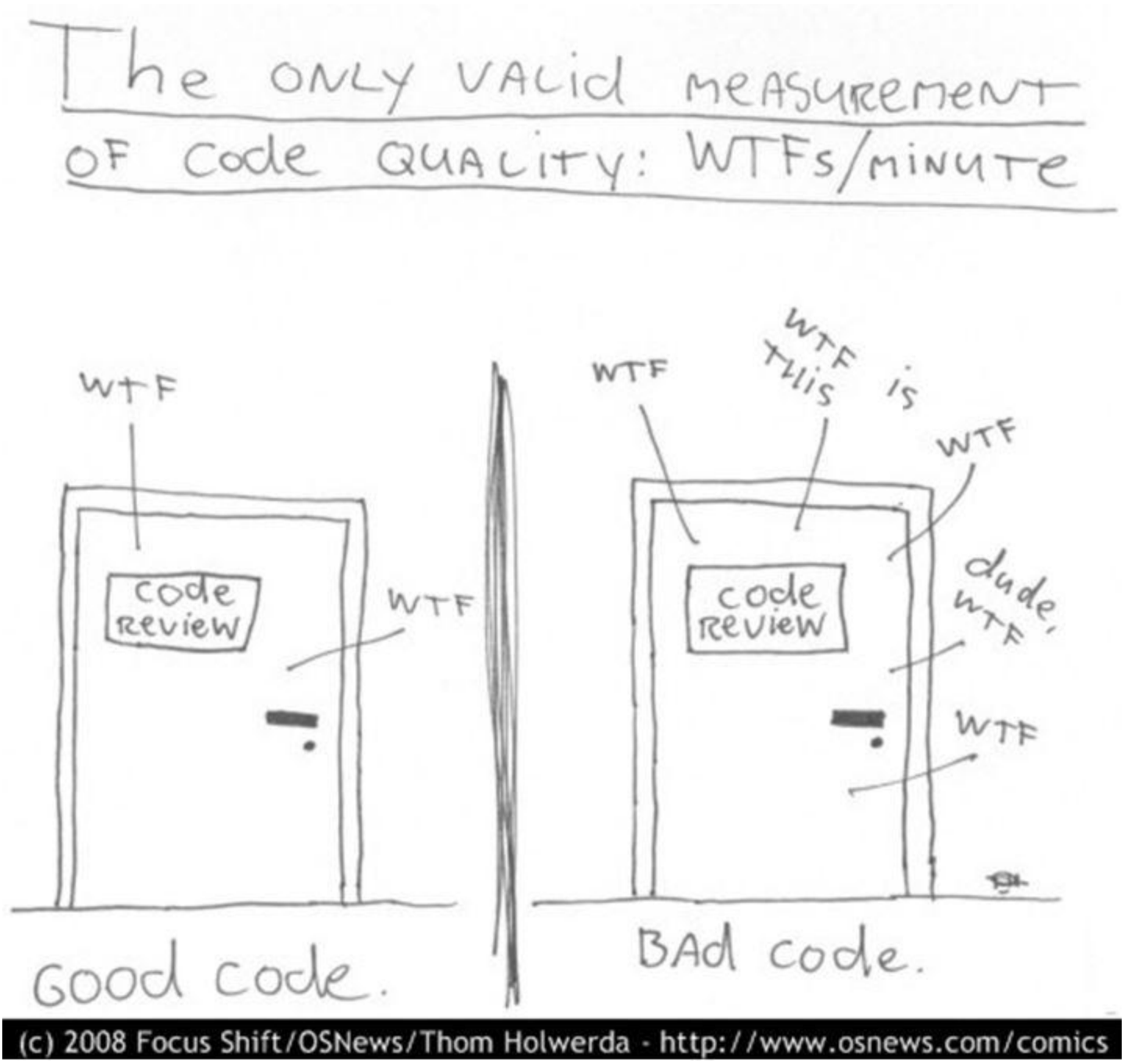
Category:
Web Development

05.05.2014

"Any fool can write code that a computer can understand. Good programmers write code that humans can understand." - Martin Fowler

What is clean code and why do we need it?

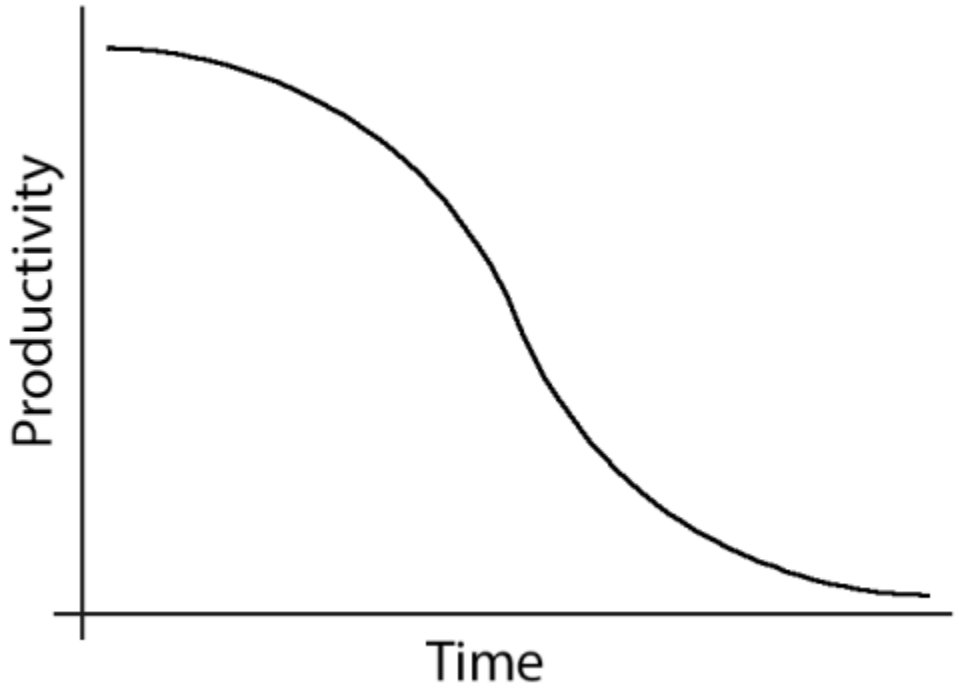
As eloquently noted by Robert Martin in his book “Clean Code,” the only valid measurement of code quality is the number of WTFs per minute as represented in the below diagram:



“Are we debugging in a panic, poring over code that we thought worked? Are customers leaving in droves and managers breathing down our necks? How can we make sure we wind up behind the right door when the going gets tough? The answer is: craftsmanship.” - Robert C. Martin

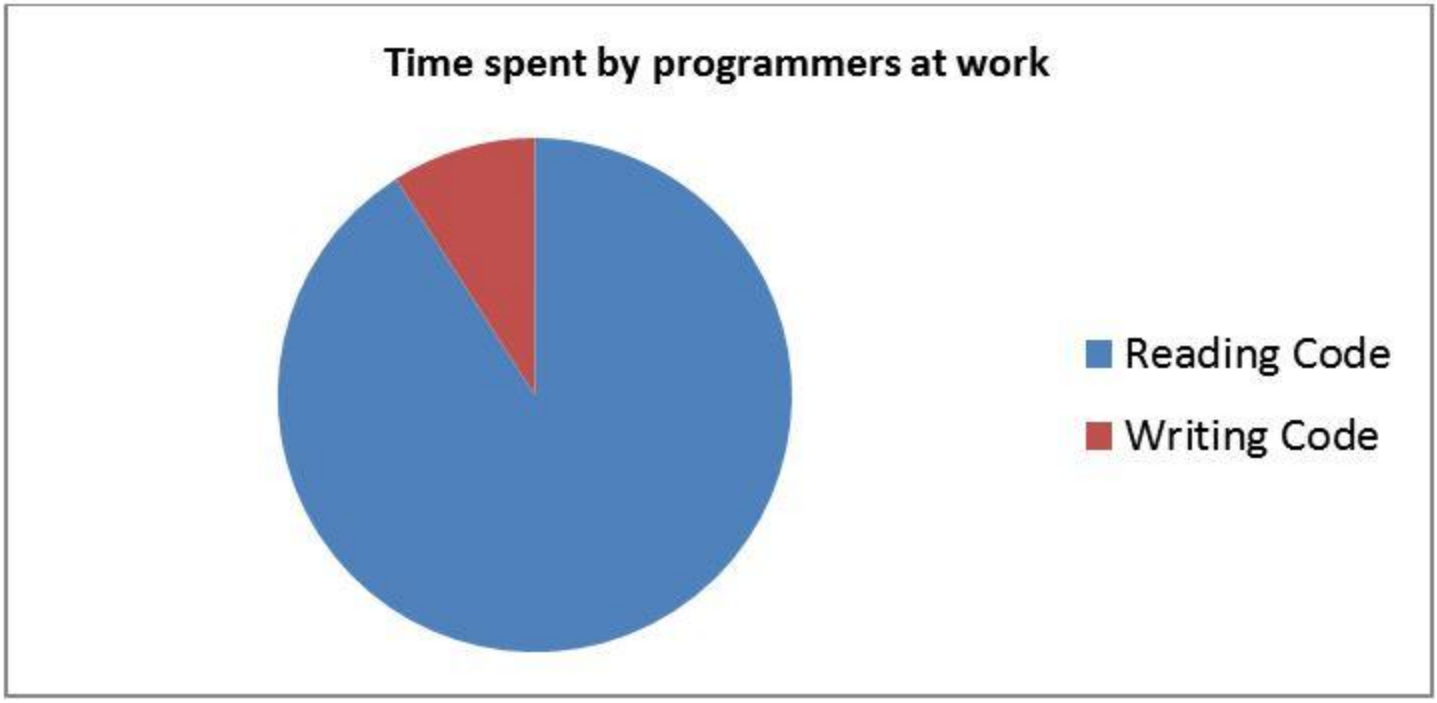
There are countless examples of bad code bringing companies down or making a disaster of an otherwise good product. Some famous and (fairly serious!) examples include space rockets being mislaunched due to a badly transcribed formula (a single line was missing that meant the rocket had to be terminated 293 seconds after launch ultimately costing around 20 million dollars), or a race condition bug in a medical machine that caused the death of three people, when they were exposed to lethal radiation doses – and you think**you've** had a bad day!

The quality of the code is directly correlated to the maintainability of the product. Programming productivity has an arccotangent relationship with time.



As features are added and changes are made, time passes and the original developers move on or forget some of the project details, or if the quality of the code is not good, changes become increasingly risky and more complex.

Programmers are really authors, and your target audience is not the computer it is other programmers (and yourself). The ratio of time spent by a programmer reading code to writing code is normally 10 to one. You are constantly reading old code in order to write new code.



Writing clean code makes the code easier to understand going forward and is essential for creating a successful maintainable product.

“Always code as if the guy who ends up maintaining your code will be a violent psychopath who knows where you live.”- Martin Golding

It is hard to write clean code

Remember the second law of thermodynamics? It implies that disorder in a system will always increase unless you spend energy and work to keep it from increasing.

Similarly, it takes a hell of a lot of more effort to write clean code. Writing clean code is hard work. It needs a lot of practice and focus during execution.

To be able to write clean code you should train your mind over a period of time.The hardest part is simply making a start, but as time goes by and your skillset improves, it becomes easier. Writing clean code is all about readability, so even the smallest things like changing you habits for naming variables make the biggest difference.

We should be consciously trying to improve code quality and decrease code entropy. The boy scouts of America have a simple rule:

“Leave the campground cleaner than you found it.”

The same rule applies to programmers. Always leave the checked in code cleaner than the code that is checked out.

Examples of what you can improve

1. Variable and method name

1.1. Use intention-revealing name

This is bad:

```
protected $d; // elapsed time in days
```

This is good:

```
protected $elapsedTimeInDays;
protected $daysSinceCreation;
protected $daysSinceModification;
protected $fileAgeInDays;
```

1.2. Use pronounceable name

This is bad:

```
public $genymdhms;
public $modymdhms;
```

This is good:

```
public $generationTimestamp;
public $modificationTimestamp;
```

1.3. Use namespaces instead of prefixing names

Most modern programming languages (including PHP and Python) support namespaces.

This is bad:

```
class Part {

   private $m_dsc;

}
```

This is good:

```
class Part {

   private $description;

}
```

1.4. Don't be cute

This is bad:

```
$program->whack();
```

This is good:

```
$program->kill();
```

1.5. Use one word per concept

Be consistent. For example, don’t use **get** and **fetch** to do the same thing in different classes

1.6. Use solution domain names

People reading your code will be other programmers so they understand solution domain terminology, so make the most of it. - For example, jobQueue is better than jobs

1.7. Use verbs for function names and nouns for classes and attributes

```
class Product {

   private $price;

   public function increasePrice($dollarsToAddToPrice)

   {

      $this->price += $dollarsToAddToPrice;

   }

}
```

2. Better Functions

2.1. The smaller the better

2.2. A function should only do one thing

2.3. No nested control structure

2.4. Less arguments are better

More than three arguments are evil. For example:

```
Circle makeCircle(Point center, double radius);
```

Is better than

```
Circle makeCircle(double x, double y, double radius);
```

2.5. No side effects

Functions must only do what the name suggests and nothing else.

2.6. Avoid output arguments

If returning something is not enough then your function is probably doing more than one thing.

For example

```
email.addSignature();
```

Is better than

```
addSignature(email);
```

2.7. Error Handling is one thing

Throwing exceptions is better than returning different codes dependent on errors.

Asking for forgiveness is easier than requesting permission. Use try/catch instead of conditions if possible

2.8. Don’t repeat yourself

Functions must be atomic

3. Comments

3.1. Don’t comment bad code, rewrite it

3.2. If code is readable you don’t need comments

This is bad:

```
// Check to see if the employee is eligible for full
benefits

if ($employee->flags && self::HOURLY_FLAG && $employee->age
> 65)
```

This is good:

```
if ($employee->isEligibleForFullBenefits())
```

3.3. Explain your intention in comments

For example:

```
// if we sort the array here the logic becomes simpler in
calculatePayment() method
```

3.4. Warn of consequences in comments

For example:

```
// this script will take a very long time to run
```

3.5. Emphasis important points in comments

For example:

```
// the trim function is very important, in most cases the
username has a trailing space
```

3.6. Always have your PHPDoc comments

Most IDEs do this automatically, just select the shortcut.

Having doc comments are especially important in PHP because methods don't have argument and return types. Having doc comments lets us specify argument and return types for functions.

If your function name is clear, you don't need a description of what the method is doing in the doc comment.

For example:

```
/**
* Return a customer based on id
* @param int $id
* @return Customer
*/
Public function getCustomerById($id)
```

3.7. Any comments other than the above should be avoided

3.8. Noise comments are bad

For example:

```
/** The day of the month. */
private $dayOfMonth;
```

3.9. Never leave code commented

Perhaps this is the most important point in this whole article. With modern source control software such as Git you can always investigate and revert back to historical code. It is ok to comment code when debugging or programming, but you should never ever check in commented code.

4. Other code smells and heuristics

There are a lot more that you can do to identify and avoid bad code. Below is a list of some code smells and anti-patterns to avoid. Refer to the sources of this blog for more information.

- Dead code
- Speculative Generality - no need “what if?”
- Large classes
- God object
- Multiple languages in one file
- Framework core modifications
- Overuse of static (especially when coding in Joomla!)
- Magic numbers - replace with const or var
- Long if conditions - replace with function
- Call super’s overwritten methods
- Circular dependency
- Circular references
- Sequential coupling

- Hard-coding
- Too much inheritance - composition is better than inheritance

Conclusion

As I mentioned before it takes a lot of practice and effort to write clean code. However it is definitely worth the extra effort. Writing clean code is integral to the success of the project.

Adopting coding standards such as (PSR-2) will make it a lot easier to write clean code. If you are using PhpStorm (IntelliJ) you can set your coding style to PSR-2 and the IDE's internal code inspector will warn you when your code is not good.

Also adopting Code Sniffer will help a lot. You can install Joomla's CodeSniffer and only commit when it is green.

Perfect code is not always feasible but It is important to try and write code as cleanly as possible. Programming is an art. Let other programmers enjoy your code.

To finish, I would like to share a telling quote from Robert Martin:
"[Most managers] may defend the schedule and requirements with passion; but that's their job. It's your job to defend the code with equal passion"

Sources

"Clean Code: A Handbook of Agile Software Craftsmanship" by Robert C. Martin

https://github.com/php-fig/fig-standards/blob/master/accepted/PSR-2-coding-style-guide.md

http://docs.joomla.org/Joomla_CodeSniffer

http://www.javacodegeeks.com/2013/02/my-favourite-programming-quotes.html