Clean code

Goals

- Reach team consensus about clean code
- New code should be “clean”
- Old code should get “cleaner”

Pillars

- Readability
- Maintainability
- Testability

Readability - basics

- Short classes (max ~1k lines)
- Short methods (max ~60 lines; one page)
- Sensible variable and method names
- No duplication
- PSR-2, Airbnb eslint
- “Express intent”, “obviously correct”...

Readability – classes

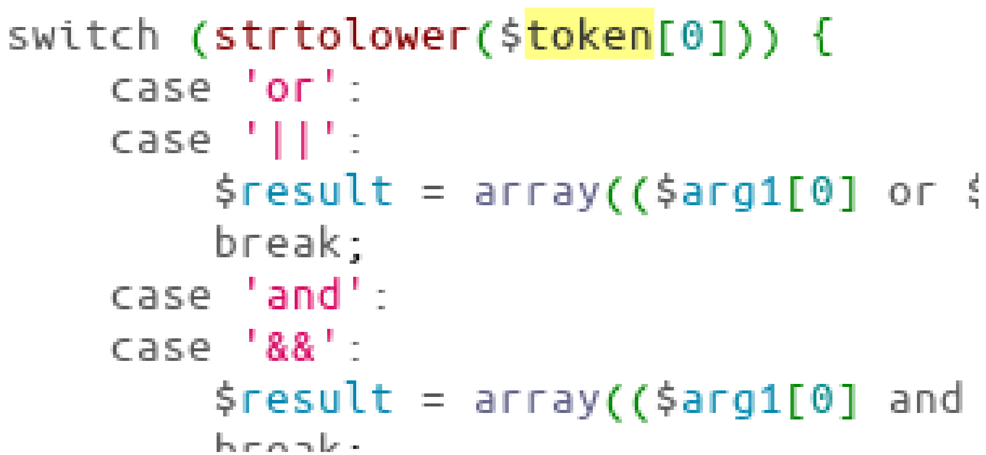
- One responsibility
- ActiveRecord – only responsible for database access
- Controllers – only gluing things together
- Views – only template code

Readability – methods

- No more than 5 arguments
  (If more – factor out new class)
- Clear relationship between input and output
  (Should be described in one sentence)
- No strange side-effects
  Example: Echo only happens at top-level
  Example 2: Pass object instead of object id

Express intent – arrays

- Using array as tuple:
- Better: OOP
- $token->getValue();
- Tip: Only use arrays as arrays (not hash tables, not tuples)
- No performance difference



```
switch (strtolower($token[0])) {
    case 'or':
    case '||':
        $result = array(($arg1[0] or $
        break;
    case 'and':
    case '&&':
        $result = array(($arg1[0] and
```

Scrutinizer

- “F” means: hard to read
- “A” means: *probably* easy to read
- New code rated “F”? More classes!

Testability

- Integration tests take time to run
- Unit tests necessary for quick feedback
- Smaller classes and methods are easier to test
- Dependencies must be explicit to be mockable
- “new” is a dependency? Replace with factories

Business logic

- “Business logic should be in M in MVC”
- But: M is a layer
- Database access is one part of that layer
  (In Yii, ActiveRecord)
- Solution: Service classes (instead of helper functions)

Services 1

- Suggestion: Put business logic in service classes
- Services are part of model layer
- Other parts are: AR, CFormModel, Data Value Objects

Services 2

- Put in application/models/services/<domain>
- Reasonably framework agnostic
- Dependency injection (__constructor, later DI container)
- Factories instead of “new”
- Highly testable

Services 3

- Long helper function → service class
- Long static method → service class
- More glue code (but can be automated later)
- Testable code is more abstract than imperative (spaghetti vs ravioli; layer of indirection)

Services 4

- $survey->activate()
  activated = 1, save
- Service: SurveyActivator, 1k LoC?
  application/models/services/survey/SurveyActivator.php
- *Object reification*
  ”Object for a concept”, or “Object for a problem”
  Further: “Patterns for expressing design intent in code”

Services 5

- Example in branch bug/15747-refactor-theme-converter-to-service-class

Maintainability

- “Possible to add new features without touching old code”
- Inheritance, events, reflection
  (new $class, $class → $method)
- Hard!
- Further reading: The expression problem

Docs and specs

- Would top-down design increase code quality?
- “Think before you do”
- UML?
- Use-case, scenario?
- Manual

Stress

- High pressure → ugly code?
- When and why are we stressed?
- When and why do we compromise on code quality?

End

- “Perfect is the enemy of good”
- “Good-enoughness”
- “Number of wtf” - subjective?
- Messy code that works > clean code that doesn’t work?