The Principled Developer

CivicActions

“The” Principle

CivicActions

- **Principle:** A rule or standard, especially of good behavior
- **Software:** (...) and symbolic languages that control the functioning of the hardware (...)
- **Develop:** To aid in the growth of; strengthen.

➔ **To strengthen communications between humans and machines by improving our common language(s)**

- **Are machines the only audience of our communications?**
- **NO!!!:** Our future self, other developers, the ***DOMAIN*** experts, etc

- ➔ **To strengthen communications between humans <-> machines by improving our common language(s)**
- ➔ **Improve:** Clear, Dense/Powerful, Unambiguous, Simple/Accessible

PHP and Drupal

CivicActions

- **Close to the metal:** Primitive data types (string, integer, boolean, etc), operations, If statements
- **Beyond the metal:** Variables, arrays, loops, functions, classes, objects

➔ **Data in Drupal** is represented by **Entities**. A **node** is the **type of entity** used for **content**. **Content** can have different structures. Different **types of nodes** can be created and are known as **content types**. Each **content type** is characterized by which **fields** it possesses

➔ Does the ***code*** match the ***idea***?

```
class Node extends ContentEntityBase implements NodeInterface {
}
```

Software Design Principles

CivicActions

- ➔ **S:** Single responsibility principle
- ➔ **O:** Open/Closed principle
- ➔ **L:** Liskov substitution principle
- ➔ **I:** Interface segregation principle
- ➔ **D:** Dependency Inversion principle

➔ Every subclass/derived class should be substitutable for their base/parent class

```
class Feline {
  public function meows() { return TRUE; }}

class Tiger extends Feline {
  public function meows() {
    return "ROOOOOAAAARRRR!!!"; }}
```

➔ A class should have one and only one reason to change, meaning that a class should have only one job

```
print "<p>Hello World!!!<\p>";
```

➔ A class should have one and only one reason to change, meaning that a class should have only one job

```
$outputter->output(
  $formatter->format("Hello World!!!")
);
```

- ➔ Objects or entities should be open for extension, but closed for modification
- ➔ "Never Hack Core"
  - ◆ hooks, events, plugins, DIC

➔ A client should never be forced to implement an interface that it doesn't use or clients shouldn't be forced to depend on methods they do not use.

Software Design Principles | Interface segregation principle | 2

```
interface CacheInterface {
  public function set($cid, $data);
  public function get($cid);
  public function expire($timestamp);
}
```

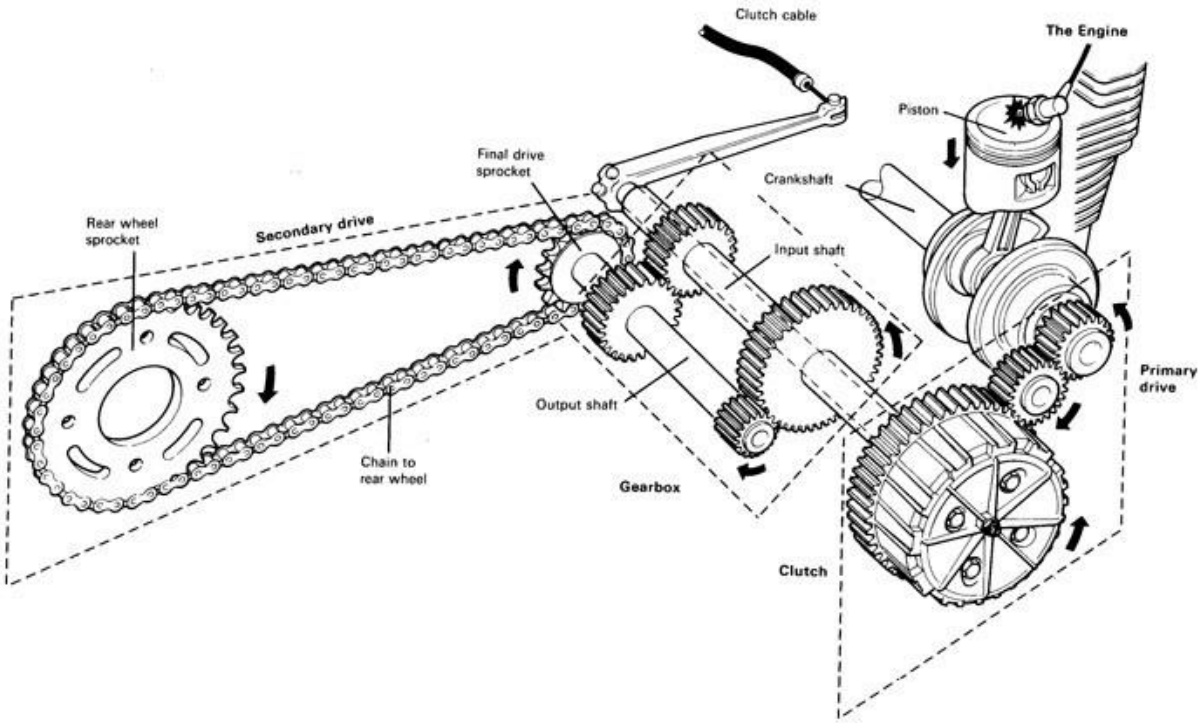
Software Design Principles | Interface segregation principle | 3

```
interface CacheInterface {
  public function set($cid, $data);
  public function get($cid);
}

interface ExpirableCacheInterface extends CacheInterface
{
  public function expire($timestamp);
}
```

➔ Entities must depend on abstractions not on concretions. It states that the high level module must not depend on the low level module, but they should depend on abstractions

Software Design Principles | Dependency inversion principle | 2



CivicActions

➔ Engine -> Clutch

```
class Engine {
  private $clutch;
  public function __construct() {
   $this->clutch = new Clutch();
  }
}
```

➔ Engine -> ClutchInterface <- Clutch

```
class Engine {
  private $clutch;
  public function __construct(ClutchInterface $clutch) {
   $this->clutch = $clutch;
  }
}
```

What about improved communications?

CivicActions

What about improved communications? | Recap

- ➔ Principles are useful
- ➔ "What-if" is the enemy of "what-is"
- ➔ Overengineering?
- ➔ But, isn’t a more principled system a better system?

What about improved communications? | Problem

- ➔ You do not know the correct language around a problem/solution until you do
- ➔ Abstractions inject complexity
- ➔ No abstractions are better than bad abstractions

What about improved communications? | Solution | 1

- ➔ Let the code express the idea
- ➔ Languages should evolve naturally
- ➔ The ***YAGNI*** principle
  - ◆ You ain’t going to need it

What about improved communications? | Solution | 2

➔ ***“Domain/Knowledge Driven Refactoring”***

- Agile, failing fast, lean development

➔ ***“Lots of Refactoring Means Lots of Tests”***

- Lock your intentions

Conclusion

Conclusion

- ➔ Always improve communications by developing a better languages
- ➔ ***SOLID*** is solid but ***YAGNI***
- ➔ Let better languages evolve through ***Domain/Knowledge Driven Refactoring***

Open Discussion

Thank you.

CivicActions