Make your project SOLID

International PHP Conference 2011

Tobias Schlitt (@tobySen)
Kore Nordmann (@koredn)

October 11, 2011

About us

- Degree in computer sience

About us

- Degree in computer sience
- More than 10 years of professional PHP

About us

- Degree in computer sience
- More than 10 years of professional PHP
- Open source enthusiasts
- Contributing to various FLOSS projects

About us

- Degree in computer sience
- More than 10 years of professional PHP
- Open source enthusiasts
- Contributing to various FLOSS projects

Co-founders of

About us

- Degree in computer sience
- More than 10 years of professional PHP
- Open source enthusiasts
- Contributing to various FLOSS projects

Co-founders of

Helping people to create high quality PHP applications.

About us

- Degree in computer sience
- More than 10 years of professional PHP
- Open source enthusiasts
- Contributing to various FLOSS projects

Co-founders of

Helping people to create high quality PHP applications.

http://qafoo.com

S.O.L.I.D.

S.O.L.I.D.

S.O.L.I.D

- 5 essential principles of object oriented design
- Introduced by Robert C. Martin (Uncle Bob)
  - not the inventor of the principles
- Have proven to lead to better code
- Scientific background (partly)

By Example

A weather loader component

- Fetch weather for a city
- Relevant data:
  - Condition
  - Temperature
  - Wind
- Be service-agnostic
  - Weather service come and go
  - Data licenses may change
- Log service failures
- Make it possible to add service fallbacks later

Single Responsibility Principle

Single Responsibility Principle

Single Responsibility Principle

“There should never be more than one reason for a class to change.”

The wrong way

```php
<?php

namespace qa\Weather\Service;
use qa\Weather\Service, qa\Weather\Structs;

class Google extends Service
{
    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->getFromUrl();
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Structs\Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
    }

    /* ... */
}
```

The wrong way

```php
<?php

namespace qa\Weather\Service;
use qa\Weather\Service, qa\Weather\Structs;

class Google extends Service
{
    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->getFromUrl();
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Structs\Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
    }

    /* ... */
}
```

The wrong way

```php
<?php

namespace qa\Weather\Service;
use qa\Weather\Service, qa\Weather\Structs;

class Google extends Service
{
    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->getFromUrl();
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Structs\Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
    }

    /* ... */
}
```

The wrong way

```php
<?php

namespace qa\Weather\Service;
use qa\Weather\Service, qa\Weather\Structs;

class Google extends Service
{
    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->getFromUrl();
        $weather = $this->extractWeather( $xml );
        return $weather;
    }

    protected function extractWeather( $xml )
    {
        $weather = new Structs\Weather();
        $weather->conditions = $this->parseConditions( $xml );
        // ...
        $weather->windSpeed = $this->convertMilesToKilometer(
            $this->parseWindSpeed( $xml )
        );
    }

    /* ... */
}
```

The better way

```php
<?php

namespace qa\Weather\Service;
use qa\HttpClient,
    qa\Weather\Structs,
    qa\Weather\Service\Google\Parser;

class Google extends Service
{
    public function __construct( HttpClient $client, Parser $parser )
    { /* ... */ }

    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->client->getUrl( $location );
        return $this->parser->parseWeather( $xml );
    }
}
```

The better way

```php
<?php

namespace qa\Weather\Service;
use qa\HttpClient,
    qa\Weather\Structs,
    qa\Weather\Service\Google\Parser;

class Google extends Service
{
    public function __construct( HttpClient $client, Parser $parser )
    { /* ... */ }

    public function getWeatherForLocation( Structs\Location $location )
    {
        $xml = $this->client->getUrl( $location );
        return $this->parser->parseWeather( $xml );
    }
}
```

Single Responsibility Principle

- One responsibility per class
- Responsibilities hard to detect
- Separation of concerns

Liskov Substiturion Principle

Liskov Substiturion Principle

Liskov Substitution Principle

“Functions that use pointers or references to base classes must be able to use objects of derived classes without knowing it.”

A Normal Class

```php
<?php

namespace qafoo;

class DistanceConverter
{
    const FACTOR = 0.6214;

    public function milesToKilometers( $miles )
    {
        return $miles / self::FACTOR;
    }

    public function kilometersToMiles( $kilometers )
    {
        return $kilometers * self::FACTOR;
    }
}
```

A Normal Class

```php
<?php

namespace qafoo;

class DistanceConverter
{
    const FACTOR = 0.6214;

    public function milesToKilometers( $miles )
    {
        return $miles / self::FACTOR;
    }

    public function kilometersToMiles( $kilometers )
    {
        return $kilometers * self::FACTOR;
    }
}
```

A Normal Class

```php
<?php

namespace qafoo;

class DistanceConverter
{
    const FACTOR = 0.6214;

    public function milesToKilometers( $miles )
    {
        return $miles / self::FACTOR;
    }

    public function kilometersToMiles( $kilometers )
    {
        return $kilometers * self::FACTOR;
    }
}
```

Extending it Incorrectly

```php
<?php

namespace qafoo;

class ValidatingDistanceConverter extends DistanceConverer
{
    const FACTOR = 0.6214;

    public function milesToKilometers( $miles )
    {
        if ( $miles < 0 )
        {
            throw new \InvalidArgumentException( 'Distance negative.' );
        }
        return $miles / self::FACTOR;
    }

    public function kilometersToMiles( $kilometers )
    {
        if ( $kilometers < 0 )
        {
            throw new \InvalidArgumentException( 'Distance negative.' );
        }
        return $kilometers * self::FACTOR;
    }
}
```

Extending it Incorrectly

```php
<?php

namespace qafoo;

class ValidatingDistanceConverter extends DistanceConverer
{
    const FACTOR = 0.6214;

    public function milesToKilometers( $miles )
    {
        if ( $miles < 0 )
        {
            throw new \InvalidArgumentException( 'Distance negative.' );
        }
        return $miles / self::FACTOR;
    }

    public function kilometersToMiles( $kilometers )
    {
        if ( $kilometers < 0 )
        {
            throw new \InvalidArgumentException( 'Distance negative.' );
        }
        return $kilometers * self::FACTOR;
    }
}
```

Liskov Substitution Principle

- Be less strict on input
- Be more strict on output
- Do not change contracts by inheritance
- Methods must work as expected in derived classes
- Users must not distinguish between super- and subclass
- Subtype polymorphism

Dependency Inversion Principle

Dependency Inversion Principle

Dependency inversion principle

“A. High-level modules should not depend on low level modules. Both should depend on abstractions.”

Dependency inversion principle

“A. High-level modules should not depend on low level modules. Both should depend on abstractions.”

“B. Abstractions should not depend upon details. Details should depend upon abstractions.”

Doing it Wrong

```
5   class Loader
6   {
7       /* ... */
8       public function __construct( Service $weatherService, $logFile )
9       {
10          $this->weatherService = $weatherService;
11          $this->log            = fopen( $logFile, 'w' );
12      }
13      public function getWeatherForLocation( Structs\Location $location )
14      {
15          try
16          {
17              $weather = $weatherService->getWeatherForLocation( $location );
18              fwrite( $this->log, "Successfully fetched weather ..." );
19          }
20          catch ( Exceptions\WeatherUnavailableException $e )
21          {
22              fwrite( $this->log, "Failed to fetch ..." );
23              throw $e;
24          }
25          return new Structs\LocatedWeather( /* ... */ );
26      }
27  }
```

Doing it Wrong

```
5   class Loader
6   {
7       /* ... */
8       public function __construct( Service $weatherService, $logFile )
9       {
10          $this->weatherService = $weatherService;
11          $this->log            = fopen( $logFile, 'w' );
12      }
13      public function getWeatherForLocation( Structs\Location $location )
14      {
15          try
16          {
17              $weather = $weatherService->getWeatherForLocation( $location );
18              fwrite( $this->log, "Successfully fetched weather ..." );
19          }
20          catch ( Exceptions\WeatherUnavailableException $e )
21          {
22              fwrite( $this->log, "Failed to fetch ..." );
23              throw $e;
24          }
25          return new Structs\LocatedWeather( /* ... */ );
26      }
27  }
```

Doing it Wrong

```
5   class Loader
6   {
7       /* ... */
8       public function __construct( Service $weatherService, $logFile )
9       {
10          $this->weatherService = $weatherService;
11          $this->log            = fopen( $logFile, 'w' );
12      }
13      public function getWeatherForLocation( Structs\Location $location )
14      {
15          try
16          {
17              $weather = $weatherService->getWeatherForLocation( $location );
18              fwrite( $this->log, "Successfully fetched weather ..." );
19          }
20          catch ( Exceptions\WeatherUnavailableException $e )
21          {
22              fwrite( $this->log, "Failed to fetch ..." );
23              throw $e;
24          }
25          return new Structs\LocatedWeather( /* ... */ );
26      }
27  }
```

Doing it Right

```
4   use qa\Logger; // Abstract base class for Logger\File, Logger\Db, ...
5
6   class Loader
7   {
8       /* ... */
9       public function __construct( Service $weatherService, Logger $logger = null )
10      {
11          $this->weatherService = $weatherService;
12          $this->logger         = $logger;
13      }
14      public function getWeatherForLocation( Structs\Location $location )
15      {
16          try
17          {
18              $weather = $weatherService->getWeatherForLocation( $location );
19              $this->logger->log( "Successfully fetched weather ..." );
20          }
21          catch ( Exceptions\WeatherUnavailableException $e )
22          {
23              $this->logger->log( "Failed to fetch ..." );
24              throw $e;
25          }
26          return new Structs\LocatedWeather( /* ... */ );
27      }
28  }
```

Doing it Right

```
4   use qa\Logger; // Abstract base class for Logger\File, Logger\Db, ...
5
6   class Loader
7   {
8       /* ... */
9       public function __construct( Service $weatherService, Logger $logger = null )
10      {
11          $this->weatherService = $weatherService;
12          $this->logger         = $logger;
13      }
14      public function getWeatherForLocation( Structs\Location $location )
15      {
16          try
17          {
18              $weather = $weatherService->getWeatherForLocation( $location );
19              $this->logger->log( "Successfully fetched weather ..." );
20          }
21          catch ( Exceptions\WeatherUnavailableException $e )
22          {
23              $this->logger->log( "Failed to fetch ..." );
24              throw $e;
25          }
26          return new Structs\LocatedWeather( /* ... */ );
27      }
28  }
```

Doing it Right

```
4   use qa\Logger; // Abstract base class for Logger\File, Logger\Db, ...
5
6   class Loader
7   {
8       /* ... */
9       public function __construct( Service $weatherService, Logger $logger = null )
10      {
11          $this->weatherService = $weatherService;
12          $this->logger         = $logger;
13      }
14      public function getWeatherForLocation( Structs\Location $location )
15      {
16          try
17          {
18              $weather = $weatherService->getWeatherForLocation( $location );
19              $this->logger->log( "Successfully fetched weather ..." );
20          }
21          catch ( Exceptions\WeatherUnavailableException $e )
22          {
23              $this->logger->log( "Failed to fetch ..." );
24              throw $e;
25          }
26          return new Structs\LocatedWeather( /* ... */ );
27      }
28  }
```

Dependency inversion principle

- Use abstraction to encapsulate low level modules
- Depend on interfaces, not realizations
- Only required facilities belong into abstractions
- Finding abstractions is not easy

Interface Segregation Principle

Interface Segregation Principle

Interface Segregation Principle

“Clients should not be forced to depend upon interfaces that they do not use.”

Doing it Wrong

```
7   class Google extends Service
8   {
9       // ...
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12
13      public function getForecastForLocation( Structs\Location $location, $weekDay )
14      { /* ... */
15  }
```

```
5   class Loader
6   {
7       public function __construct( Service $weatherService, \qa\Logger $logger )
8       { /* ... */ }
9
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12  }
```

Doing it Wrong

```
7   class Google extends Service
8   {
9       // ...
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12
13      public function getForecastForLocation( Structs\Location $location, $weekDay )
14      { /* ... */
15  }


5   class Loader
6   {
7       public function __construct( Service $weatherService, \qa\Logger $logger )
8       { /* ... */ }
9
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12  }
```

Doing it Wrong

```
7   class Google extends Service
8   {
9       // ...
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12
13      public function getForecastForLocation( Structs\Location $location, $weekDay )
14      { /* ... */
15  }
```

```
5   class Loader
6   {
7       public function __construct( Service $weatherService, \qa\Logger $logger )
8       { /* ... */ }
9
10      public function getWeatherForLocation( Structs\Location $location )
11      { /* ... */ }
12  }
```

Doing it Right

```php
class Google extends Service implements LocationWeatherProvider
{
    // ...

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }

    public function getForecastForLocation( Structs\Location $location, $weekDay )
    { /* ... */ }
}
```

```php
interface LocationWeatherProvider
{
    function getWeatherForLocation( Structs\Location $location );
}
```

```php
class Loader
{
    public function __construct( LocationWeatherProvider $provider, \qa\Logger $logger )
    { /* ... */ }

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }
}
```

Doing it Right

```php
class Google extends Service implements LocationWeatherProvider
{
    // ...

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }

    public function getForecastForLocation( Structs\Location $location, $weekDay )
    { /* ... */ }
}
```

```php
interface LocationWeatherProvider
{
    function getWeatherForLocation( Structs\Location $location );
}
```

```php
class Loader
{
    public function __construct( LocationWeatherProvider $provider, \qa\Logger $logger )
    { /* ... */ }

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }
}
```

Doing it Right

```php
class Google extends Service implements LocationWeatherProvider
{
    // ...

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }

    public function getForecastForLocation( Structs\Location $location, $weekDay )
    { /* ... */ }
}
```

```php
interface LocationWeatherProvider
{
    function getWeatherForLocation( Structs\Location $location );
}
```

```php
class Loader
{
    public function __construct( LocationWeatherProvider $provider, \qa\Logger $logger )
    { /* ... */ }

    public function getWeatherForLocation( Structs\Location $location )
    { /* ... */ }
}
```

Interface Segregation Principle

- Avoid not needed dependencies
- Design interfaces from a usage point of view
- Do not let unnecessary functionality float in

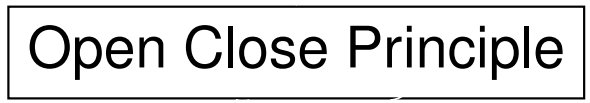
Open/Close Principle

Open/Close Principle

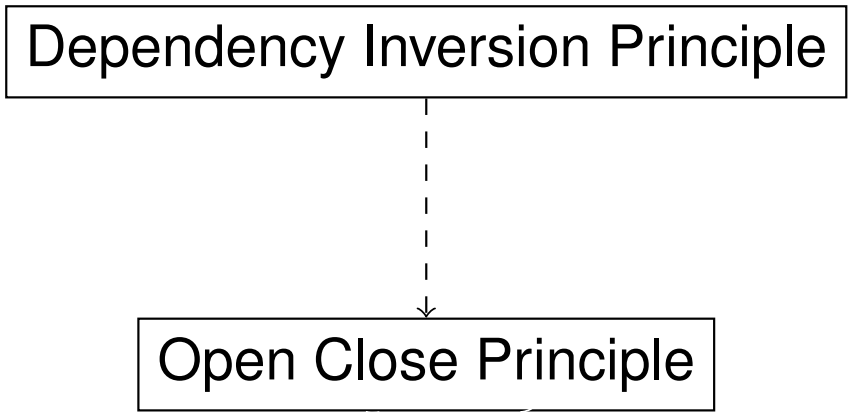
Open/Close Principle

*“Software entities (classes, modules, functions, etc.)
should be open for extension, but closed for modification.”*

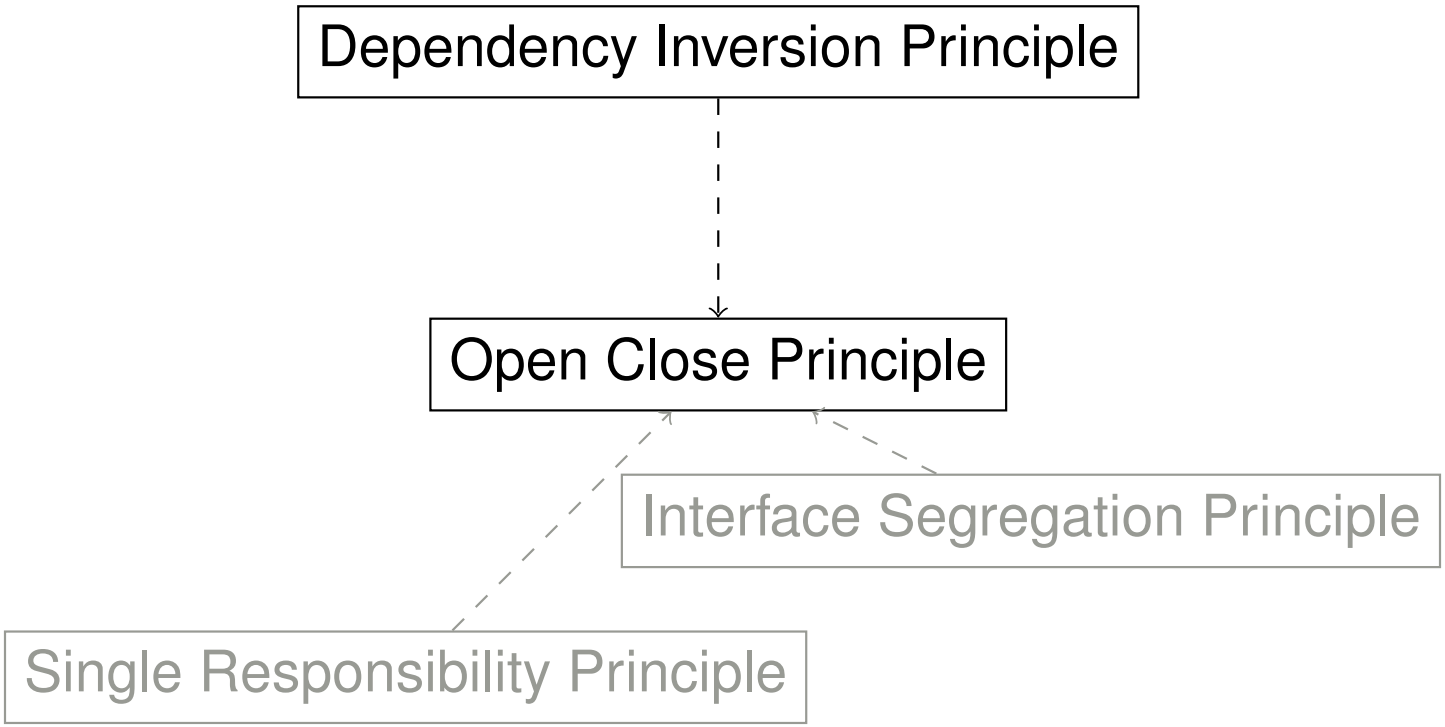
Open/Close Principle



Open/Close Principle



Open/Close Principle



Open/Close Principle

- Changes introduce errors
  - Especially cascading changes
- Write once, change never!
- Extend software by new code
  - New interface implementations
  - Inheritance
  - Aggregation

Conclusion

S.O.L.I.D.

Conclusion

You want S.O.L.I.D.

Thanks for listening

Questions? Comments? Critics? Ideas?

Thanks for listening

Please rate this talk at
`http://joind.in/3889`

(Slides will be linked there)

Thanks for listening

Please rate this talk at
`http://joind.in/3889`

Stay in touch

- Tobias Schlitt
- toby@qafoo.com
- @tobySen / @qafoo

- Kore Nordmann
- kore@qafoo.com
- @koredn / @qafoo

Rent a PHP quality expert:
`http://qafoo.com`

Credits

Thanks for motivational posters ”SOLID in pictures” to

http://lostechies.com/derickbailey/2009/02/11/solid-development-principles-in-motivational-pictures/