@geraldcroes
gerald@php.net
http://croes.org

PLUS QU'UN OBJET, UN OBJET

Ce que j'ai appris de la POO ces 15 dernières années…

Enfin libres…

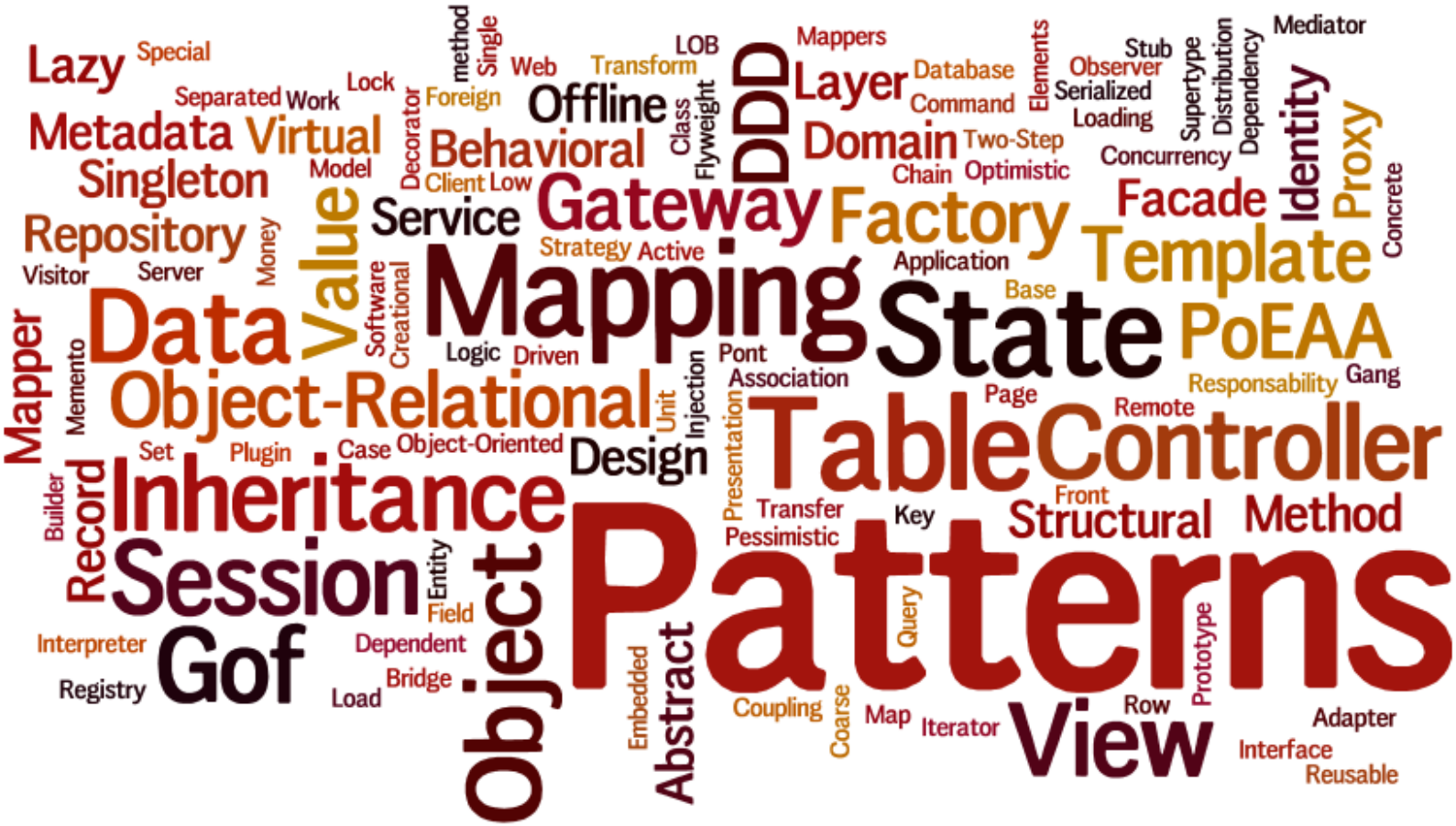
“You Can't Write Perfect Software. Did that hurt? It shouldn't. Accept it as an axiom of life. Embrace it. Celebrate it. Because perfect software doesn't exist. No one in the brief history of computing has ever written a piece of perfect software. It's unlikely that you'll be the first. And unless you accept this as a fact, you'll end up wasting time and energy chasing an impossible dream.”

Andrew Hunt - *The Pragmatic Programmer: From Journeyman to Master*

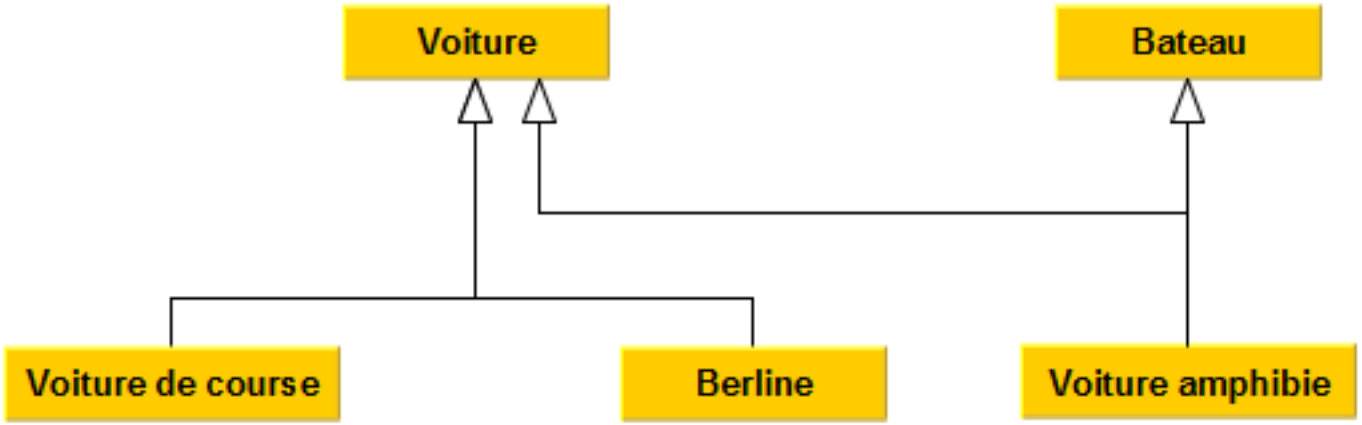
L'objet tel qu'il est présenté

"Object-oriented programming is an approach to designing modular, reusable software systems."

Ce que l'on oublie souvent de nous dire…



A la découverte de l'objet



Object-oriented programming is an approach to designing modular, reusable software systems. Wikipedia

…Before reusable…

"Before software can be reusable it first has to be usable."

-- Ralph Johnson

…Reusable…

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

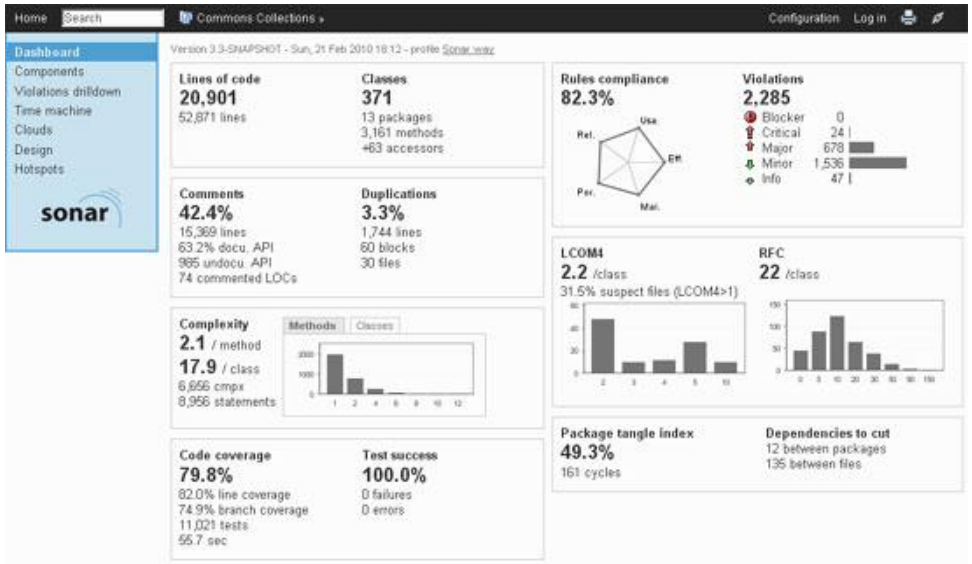
Martin Fowler

DE L'ÉCRITURE DU CODE D'UNE MÉTHODE

Qu'est-ce qu'un code lisible ? Réutilisable ?

Un beau code ?

- Nombre de lignes de codes ?
- lignes par objet ?
- méthodes par objet ?
- Nombre de méthodes ?
- LoC / méthodes ?
- LoC / objet ?
- Commentaires ?
- Couverture de code ?



Une histoire d'écriture…

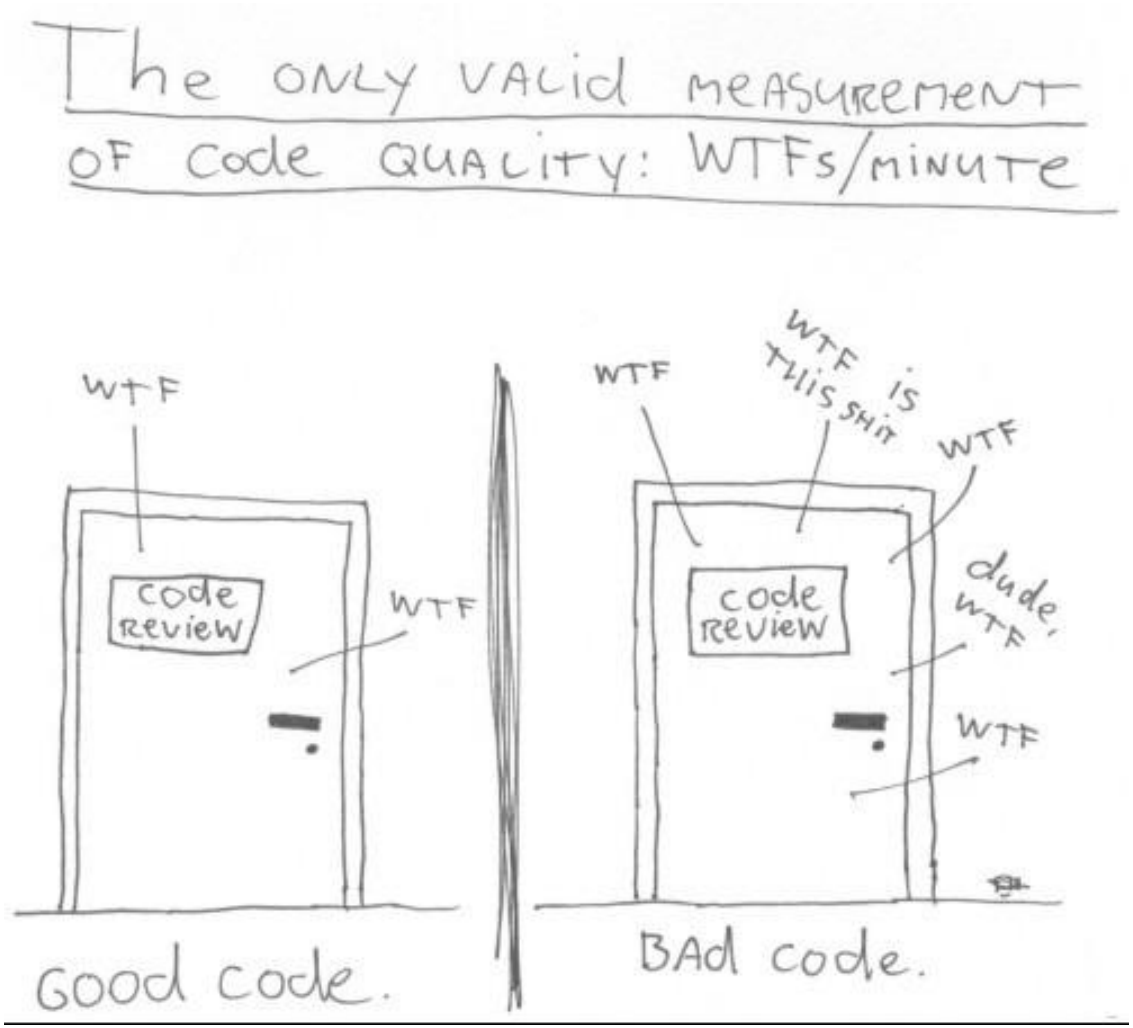


Image prise du livre « Clean Code »

… et de lecture

Pour écrire une ligne, nous en lisons 10

Le choix des mots

- Un nom significatif

```php
class Services
{
    public function process ($a, $b) {
        return $a < $b ? $a : $b;
    }
}
```

Le choix des mots

- Un nom significatif

```php
class Services
{
    public function process ($a, $b) {
        return $a < $b ? $a : $b;
    }
}
```

- … aide à la compréhension

```php
class CompareNombresServices {
    public function getMinimum($nombre1, $nombre2) {
        return $nombre1 < $nombre2 ? $nombre1 : $nombre2;
    }
}
```

Le choix des mots (P.S.)

- Fonctionne aussi avec les boucles for

```
    echo $datas[$i][$j];
```

Le choix des mots (P.S.)

- Fonctionne aussi avec les boucles for

```php
for ($i=0; $i<count($arPersonnes); $i++) {
    for ($j=0; $i<count($arAdresses); $j++) {
        // ...
        // ...
        // 10 lignes dessous
        echo $datas[$i][$j];
    }
}
```

Le choix des mots (P.S.)

- Fonctionne aussi avec les boucles for

```php
for ($i=0; $i<count($arPersonnes); $i++) {
    for ($j=0; $i<count($arAdresses); $j++) {
        // ...
        // ...
        // 10 lignes dessous
        echo $datas[$i][$j];
    }
}
```

Pas de rapprochement mental inutile
« $i ⇔ $numeroPersonne »
« $j ⇔ $numeroAdresse »

```php
for ($numeroPersonne=0; $i<count($arPersonnes); $numeroPersonne++) {
    for ($numeroAdresse=0; $i<count($arAdresses); $numeroAdresse++) {
        // ...
        // ...
        // 10 lignes dessous
        echo $datas[$numeroPersonne][$numeroAdresse];
    }
}
```

Le choix des mots (P.P.S.)

- Fonctionne aussi avec les boucles foreach

```
    $liste->add($key, $value);
```

Le choix des mots (P.P.S.)

- Fonctionne aussi avec les boucles foreach

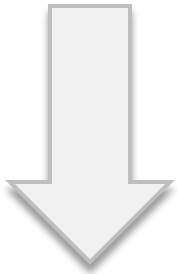
```
    $liste->add($key, $value);
```

```
    $produitsAFacturer->add($nomProduit, $prix);
```

Le choix des mots (P.P.S.)

- Fonctionne aussi avec les boucles foreach

```php
foreach ($arDatas as $key=>$value) {
    $liste->add($key, $value);
}
```



```php
foreach ($arCommandeProduit as $nomProduit=>$prix) {
    $produitsAFacturer->add($nomProduit, $prix);
}
```

Les programmes se lisent par fragments, dans le désordre

Soyez spécifiques…

- Des noms précis, sans ambiguïtés

```
class PdfGenerator
{
    public function getPdf ($string, $name) {
```

Soyez spécifiques…

- Des noms précis, sans ambiguïtés

```php
class PdfGenerator
{
    public function getPdf ($string, $name) {
```

Quel format de chaîne ? Html ? Rtf ?
Texte plein ? Markup ? Lequel ?

En sortie, le contenu pdf ? Le succès ? Un objet ?
Un chemin de fichier ?

Le nom du fichier ? du modèle ? De la police par défaut ? de l'application ? De celui qui demande le document ?

Soyez spécifiques…

- Des noms précis, sans ambiguïtés

```
class PdfGenerator
{
    public function getPdf ($string, $name) {
```

Quel format de chaîne ? Html ? RTF ?
Texte plein ? Markup ? Lequel ?

En sortie, le contenu pdf ? Le succès ? Un objet ?
Un chemin de fichier ?

Le nom du fichier ? du modèle ? De la police par défaut ? de l'application ? De celui qui demande le document ?

```
class PdfGenerator
{
    public function generateFile ($htmlString, $resultingPdfFilePath)
    {
        // ...
```

On ne plaisante pas avec le code

```php
class Martine {
    private static $theRing = false;

    public static function bigbang () {
        if (self::$theRing === false) {
            self::$theRing = new Martine();
        }
        return self::$theRing;
    }

    public function laPetiteEnveloppe (PersonnelCollection $personnelCollection) {
        //...
    }
    //...
}
```

Le code est un jeu sérieux

```php
class ServiceComptable {
    private static $instance = false;

    public static function create () {
        if (self::$instance === false) {
            self::$instance = new ServiceComptable();
        }
        return self::$instance;
    }

    public function calculDesPayes (PersonnelCollection $personnelCollection) {
        // ...
    }
    // ...
}
```

Autres conseils

- Evitez les recoupements de rôles
  - DossierStatusInspector / DossierStatusManager
- Evitez la désinformation
- Prononçables, Faciles à retenir,
  - genDythxhw / genDithxs

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

Les commentaires…

"Good code is its own best documentation. As you're about to add a comment, ask yourself, 'How can I improve the code so that this comment isn't needed?'"

-- Steve McConnell

Les commentaires comme aveu de faiblesse.

```php
class MesServices {

    // ...

    public function getPrix (Produit $produit) {
        if ($this->getDate() >= $produit->getDate()
            && $this->getDate() <= $this->dateServices->getIncreasedBy($produit->getDate(), '6 month')) {
            return $produit->getPrixUnitaire() * 0.90;
        }
    }
```

Les commentaires comme aveu de faiblesse.

```php
class ServicesDeFacturation {

    // ...

    public function getPrix (Produit $produit) {
        if ($this->enPeriodeDeLancement($produit)) {
            return $produit->getPrixUnitaire() * 0.90;
        }
    }

    private function enPeriodeDeLancement (Produit $produit) {
        return $this->getDate() >= $produit->getDate()
               && $this->getDate() <= $this->dateServices->getIncreasedBy($produit->getDate(), '6 month');
    }
```

Le cas PHPDoc

```php
/**
 * Génération de PDF
 */
class PdfGenerator
{
    /**
     * Récupère le PDF
     *
     * @param $string
     * @param $name
     */
    public function getPdf ($string, $name) {

    }
}
```

Pallier au manque du langage

- @return
- @throws
- Typage des propriétés
- Choix d'architecture, de réalisation
- Limitations connues

Derniers points

- Prenez des conventions
- Respectez vos conventions
- Popularisez les conventions dans toute l'équipe

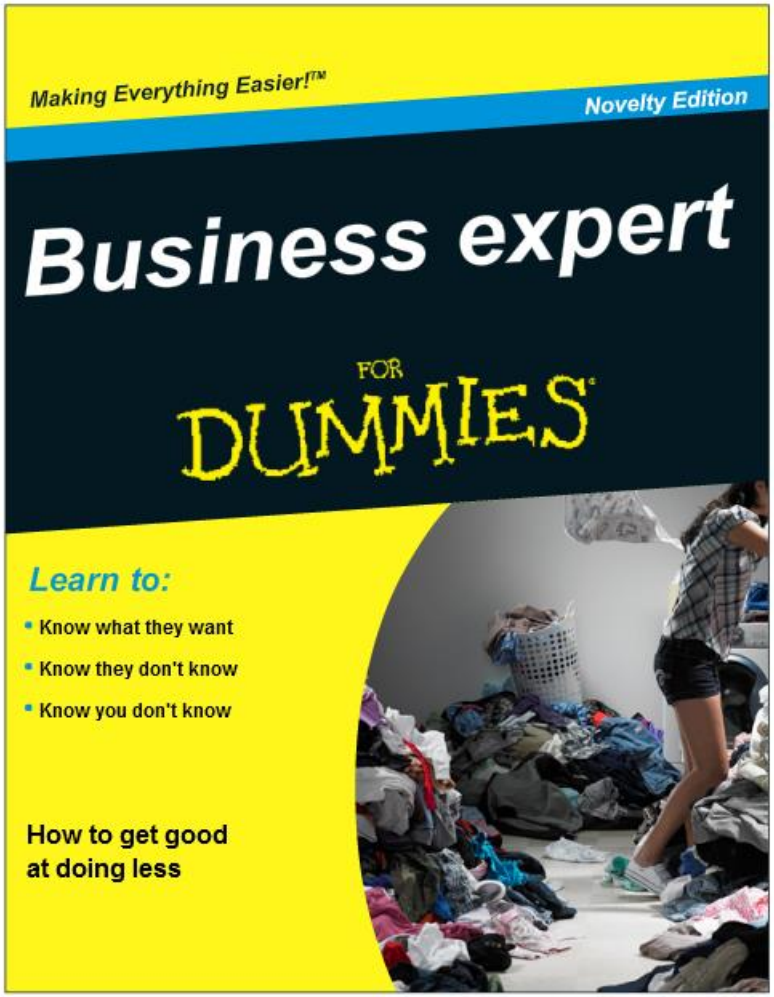
Un état d'esprit

DE L'ÉCRITURE **DU CODE** D'UNE MÉTHODE

Quels principes d'écriture ? Quels outils ? Quelles techniques ?

Le paradoxe de la programmation

- Eternel néophyte
- Un futur incertain
- Un présent cahotique



YAGNI – You ain't gonna need it

"Build what you need as you need it, **aggressively refactoring** as you go along; don't spend a lot of time planning for grandiose, unknown future scenarios. **Good software can evolve into what it will ultimately become.**"

-- Jeff Atwood

YAGNI

"If debugging is the process of removing software bugs, then programming must be the process of putting them in."

-- Edsger Dijkstra

KISS - Keep It Simple, Stupid

"Do the simplest thing that could possibly work"

-- Pratique XP

KISS

"Debugging is twice as hard as writing the code in the first place. Therefore, if you write the code as cleverly as possible, you are, by definition, not smart enough to debug it."

-- Brian W. Kernighan

KISS & YAGNI en pratique

```php
interface CataloguesProduits {
    public function getListProduits();
}

class CatalogueProduitSocieteMartin implements CataloguesProduits {
    public function getListProduits () {
        return array (
            new Produit('id', 'nom', 3.55)
        );
    }
}
```

Ce qui répond simplement au besoin d'aujourd'hui pourra être facilement modifié demain

Personne n’a besoin d’objets

« Vous avez besoin d’une interface simple et efficace »

Design Pattern – Rappels

- Abstract Factory, **Factory**, Builder, Factory Method, Prototype , ***Singleton***

- **Adapter**, Bridge, Composite, **Decorator**, Façade, Flyweight, **Proxy**

- **Chain of responsibility**, Command, Interpreter, Iterator, Mediator, Memento, **Observer**, State, **Strategy**, Template method, Visitor

Design Pattern – Quelques références

- *GOF - Design Patterns: Elements of Reusable Object-Oriented Software*
- Martin Fowler – PoEAA - *Patterns of Enterprise Application Architecture*
- *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.*
- Eric Evans – DDD – Domain Driven Design

Design Pattern

- The Good
  - Catalogue d'architecture
  - Des problèmes et des solutions connus
  - Fait travailler la conception

Design Pattern

- The Good
  - Catalogue d'architecture
  - Des problèmes et des solutions connus
  - Fait travailler la conception
- The Bad
  - Des solutions inadaptées

Design Pattern

- The Good
  - Catalogue d'architecture
  - Des problèmes et des solutions connus
  - Fait travailler la conception
- The Bad
  - Des solutions inadaptées
- The Ugly
  - Générateur d'incompréhension

Design Pattern – The Ugly

```php
class PriceFactory
{
    public function getPrixCommande(Commande $commande) {
        $somme = 0;
        foreach ($commande->getProduits() as $qte=>$produit) {
            $somme += $qte * $produit->getPrixUnitaire();
        }
        return $somme;
    }

    public function getPrixCommercial (Commande $commande) {
        return $this->getPrixCommande($commande) * 0.9;
    }
}
```

N'utilisez, nommez les modèles de conception dans votre code à la seule condition que vous les maîtrisez parfaitement

A propos du découplage

"In the one and only true way. The object-oriented version of 'Spaghetti code' is, of course, 'Lasagna code'. (Too many layers).“

-- Roberto Waltman.

SOLID

- Single Responsability Principle
- Open / Close Principle
- Liskov Substitution Principle
- Interface Segregation Principle
- Dependency Inversion Principle

SOLID - Single Responsability Principle

- Une et une seule responsabilité
- Une seule raison de changer

"The SRP is one of the simplest of the principle, and one of the hardest to get right"

-- Robert C. Martin

SOLID - Single Responsability Principle

- Un vélo qui sait gonfler ses pneus
- Un article qui envois un email
- ActiveRecord
- Un bus qui sait trouver son chemin
- Une chemise qui sait se nettoyer toute seule

« Décomposez les responsabilités de votre application »

S**O**LID - Open Close Principle

« Une nouvelle fonctionnalité, une nouvelle classe »

Du ‘chemin le plus rapide en temps’ vers le ‘chemin le moins long en km’.

SOLID - Liskov Substitution Principle

« Les classes filles doivent respecter le contrat des classes mères »

```php
class Message
{
    public function send ($messageTexte) {
        // ...
    }
}

class MessageMail extends Message
{
    public function prepare ($eMailDestinataire) {
        // ...
    }

    public function send ($messageTexte) {
        // ... nécessite une préparation préalable
    }
}

class MessageEcran extends Message
{
    public function send ($messageTexte, $couleur) {
        // ...
    }
}
```

FAUX : Change le contrat d'utilisation (workflow)

FAUX : Change le contrat d'utilisation (signature)

SOLID - Interface Segregation

Dépendez du strict nécessaire

Taille et Diplomes ne servent à rien dans le calcul des impôts

```php
interface Personne
{
    public function getNom();
    public function getPrenom();

    public function getDateDeNaissance();
    public function getTaille();
    public function getDiplomes();

    public function getEnfants();
    public function getConjoint();

    public function getRevenus();
}

class HabitantFrance implements Personne {
    // ...
}

interface Imposition {
    public function getAvisImposition (Personne $personne);
}

class TresorPublic implements Imposition {
    // ...
}
```

SOLID - Interface Segregation

```php
interface Imposition {
    public function getAvisImposition (DeclarationDeRevenus $declarationDeRevenus);
}

interface DeclarationDeRevenus {
    public function getNom();
    public function getPrenom();
    public function getNombreEnfants();
    public function hasConjoint();
    public function GetRevenusAnnuel();
}

class DeclarationDeRevenusAdaptatorPersonne implements DeclarationDeRevenus {
    public function __construct (Personne $personne) {
        // ...
    }

    public function getNombreEnfants() {
        //logique de conversion des données
    }
}
```

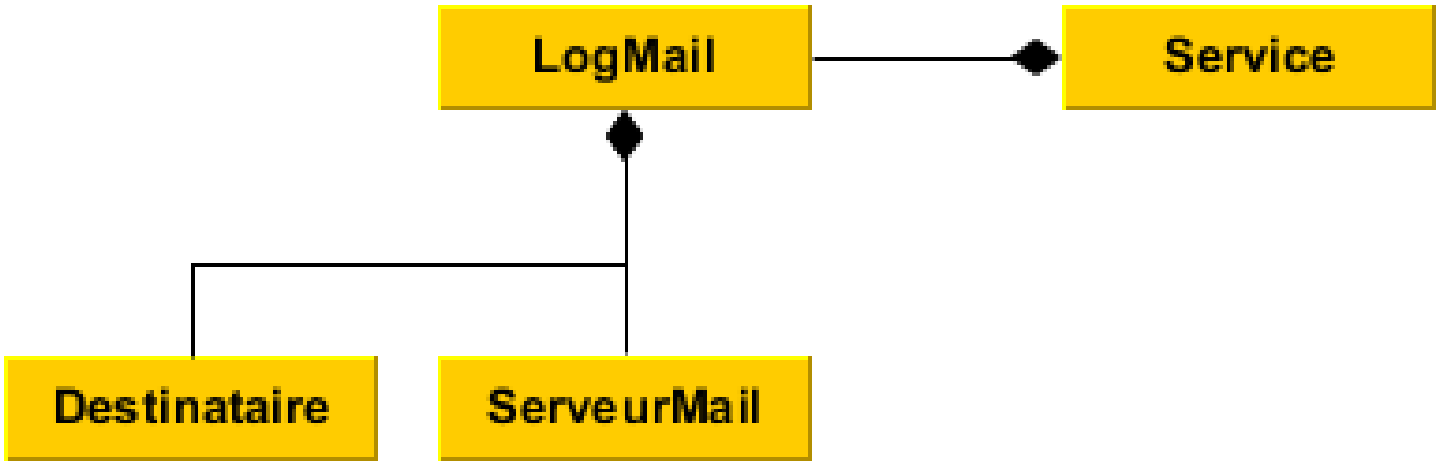
SOLID - Dependency Inversion Principle

- High level modules should not depend upon low-level modules. Both should depend upon abstractions.
- Abstractions should never depend upon details. Details should depend upon abstractions.

« Dépendez des concepts et non des implémentations »

« Utilisez des prises et des connecteurs, ne soudez pas les câbles »

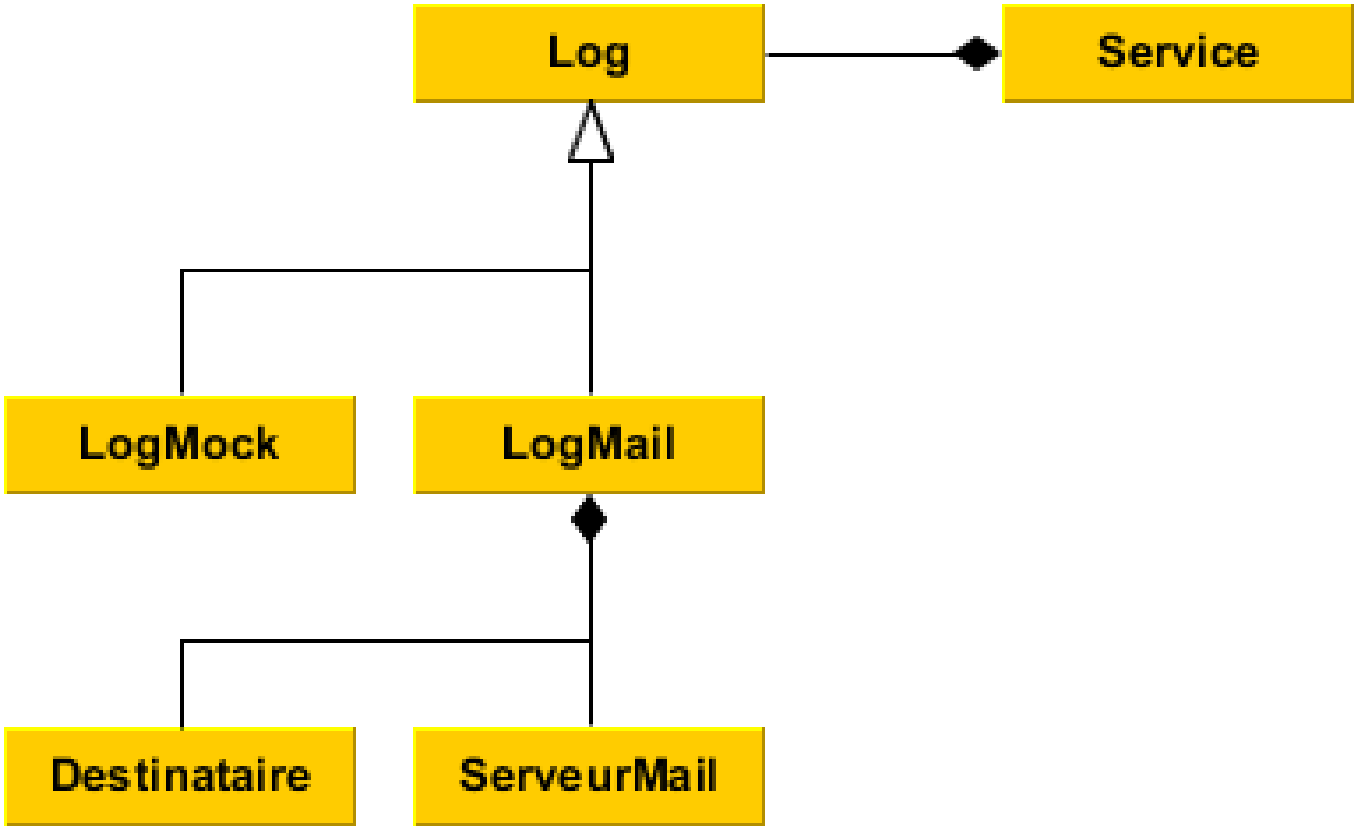
SOLID - Dependency Inversion Principle



Pas de log sans serveur de mail

Pas de test sans serveur de mail

SOLID - Dependency Inversion Principle



Injection de dépendance

- Une solution de découplage
- Une implémentation du « DIP »

```php
class ArticleMySqlRepository implements ArticleRepository {
    /**
     * @var PDO
     */
    private $pdo;

    public function __construct () {
        $this->pdo = new Pdo('dsn');
    }

    // ...
}
```

Injection de dépendance

- Par construction
- Via « setters »

```php
class ArticleMySqlRepository implements ArticleRepository {
    /**
     * @var ConnecteurBdd
     */
    private $connecteurBdd;

    public function __construct (ConnecteurBdd $connecteurBdd) {
        $this->connecteurBdd = $connecteurBdd;
    }

    // ...
}
```

Injection de dépendance

- Frameworks à la rescousse

```
sso_client:
    class: ProjectName\Sso\SsoClientRecetteProxy

jeton_applicatif_factory:
    class: ProjectName\Sso\JetonApplicatifFactory
    arguments: [60, @mongo_client, %db_name%]

jeton_applicatif_repository:
    class: ProjectName\Sso\Repositories\JetonApplicatifMongoRepository
    arguments: [@mongo_client, %db_name%]

authenticate_service:
  class: ProjectName\Sso\Services\AuthenticateServices
  arguments: [@utilisateur_repository, @jeton_applicatif_repository]
```

UYB – Use Your Brain !

- Des principes, pas des règles absolues !

TDD

"I don't care if it works on your machine! We are not shipping your machine!“

-- Vidiu Platon.

DE L'ÉCRITURE DU CODE **D'UNE MÉTHODE**

Une façon de concevoir les applications.

DDD / BBBB *

Big Boring Blue Book

(it's a joke, please don't hit me)

DDD / BBBB


push state change with
DOMAIN EVENTS
access with
REPOSITORIES
access with
SERVICES
express model with
express model with
ENTITIES
encapsulate with
act as root of
express model with
encapsulate with
AGGREGATES
express model with
MODEL-DRIVEN DESIGN
VALUE OBJECTS
encapsulate with
model gives structure to
isolate domain with
encapsulate with
define model within
LAYERED ARCHITECTURE
encapsulate with
FACTORIES
UBIQUITOUS LANGUAGE
cultivate rich model with
keep model unified by
CONTINUOUS INTEGRATION
SHARED KERNEL
names enter
BOUNDED CONTEXT
CORE DOMAIN
overlap allied contexts through
CUSTOMER /SUPPLIER
relate allied contexts as
work in autonomous, clean
assess/overview relationships with
CONTEXT MAP
minimize translation
CONFORMIST
avoid overinvesting in
GENERIC SUBDOMAINS
segregate the conceptual messes
support multiple clients through
free teams to go
translate and insulate unilaterally with
OPEN HOST SERVICE
formalize as
PUBLISHED LANGUAGE
BIG BALL OF MUD
ANTI-CORRUPTION LAYER
SEPARATE WAYS

Le métier, le métier, et le métier

"Perhaps the greatest strength of an object-oriented approach to development is that it offers a mechanism that captures a model of the real world.“

Booch, Grady (1986).

Software Engineering with Ada

Le langage commun

- Les experts métiers ont un langage
- Le programme doit reposer sur ce langage

Les objets valeurs

- Caractérisés par leurs propriétés
- Un ensemble d'objets caractérisés par leurs propriétés ou leurs capacités de traitement. Ces objets sont immutables.

Value object

Les entités

- Les éléments dont l'identité compte plus que ses propriétés.
- Les objets du modèle dont le cycle de vie est important.

Les agrégats

- Il est parfois difficile de garantir l'intégrité du modèle lorsqu'il contient des associations complexes.
- Assemblez les objets complexes autours d'une entité racine, et autorisez le modèle à l'utiliser que cette dernière.

Les services

- Lorsqu'un processus n'appartient à aucune entité, aucun objet valeur, ajoutez ce dernier dans le modèle grâce à une interface dédiée.

Les fabriques

- Contient la logique et responsabilité de création des objets valeurs, entités et agrégats.

Les entrepôts

- Les entrepôts convertissent les données en objets métiers, et inversement.
- Ils utilisent les fabriques pour construire les objets.

Le secret de la programmation objet se situe à mi-chemin entre l'art de l'écriture du code, le choix des mots, le choix des acteurs, la capacité à dresser une scène cohérente et compréhensible…

Et à mi-chemin entre l'usage de techniques et principes technologiques pour mener à bien cette vision, pour lui permettre de s'exprimer dans le temps.

CONCLUSION

“There are two ways of constructing a software design.
One way is to make it so simple that there are obviously no deficiencies.
And the other way is to make it so complicated that there are no obvious deficiencies.”
-- Charles Antony Richard Hoare

"Programming today is a race between software engineers striving to build bigger and better idiot-proof programs, and the universe trying to build bigger and better idiots. So far, the universe is winning."
- Rick Cook

"Perfection is achieved, not when there is nothing more to add, but when there is nothing left to take away."
- Antoine de Saint-Exupéry

QUESTIONS ?

Sources

- Clean-Code
- Refactoring Patterns
- Gof
- PoEAA
- Codinghorror
- Refactoring Patterns
- https://www.youtube.com/watch?v=B93QezwTQpI