Introduction aux tests unitaires

Les tests unitaires sont de simples et très courts programmes permettant de mettre en lumière le fonctionnement d'une partie de l'ensemble d'une application ou d'un composant logiciel.

Parmi tous les axes de qualité, nous pouvons citer :

- La pertinence des tests unitaires.
- La couverture des tests unitaires.
- La documentation des composants.
- L'évolutivité de l'architecture logicielle.
- La centralisation de l'information.

Il est possible d'approcher l'ensemble des réalisations techniques par une approche qualité et des tests de contrôle. Cette approche est nommée « Approche pilotée par les tests ». Son principe repose sur le postulat qu'il est toujours plus simple de maintenir et de valider un composant logiciel ayant une interface ou un contrat d'utilisation simple.

Pour avoir le contrat d'utilisation le plus pratique possible, il suffit d'écrire un court programme utilisant l'interface du composant et validant ses principales fonctionnalités.

Afin de résumer cette approche, commencez toujours par écrire vos tests (qui sont ni plus ni moins des programmes de validation de votre composant) puis écrivez votre composant en lui-même.

Cette démarche va à contre-courant des idées reçues sur la réalisation de code. En effet, l'idée commune nous incite à écrire le code du composant en premier puis les tests unitaires, ce qui pose de nombreux problèmes.

Un composant peut être mal écrit la première fois pour des raisons multiples : inexpérience des développeurs, pression sur les délais de livraison, pression sur le volume des fonctionnalités attendues.

Dans ce cadre, les contrats d'utilisation de vos composants (c'est-à-dire la manière de les utiliser dans un autre programme) sont souvent définis au fil de l'eau.

Un composant logiciel mal désigné est toujours difficile à tester ! Écrire le code d'utilisation de votre

composant va vous permettre de prendre le recul suffisant afin de vous permettre de réfléchir à l'utilisation de votre composant et de définir le contrat d'utilisation optimal dans votre contexte.

Afin d'achever cette introduction, rappelez-vous toujours qu'un composant ayant une interface ou un contrat d'utilisation simple et intuitif est toujours moins coûteux à maintenir. Sachant aujourd'hui que les coûts de maintenance des logiciels sont les premiers postes de dépenses dans le cycle de vie d'un logiciel, il semble important de partir sur l'hypothèse que, plus ces logiciels sont bien conçus, plus la maintenance sera facilitée et rapide.

Dans un processus d'amélioration continue, il est essentiel de tracer toutes les anomalies rencontrées et de les suivre. Pour chacune d'entre elles, il est essentiel d'apporter à la fois une correction, mais également un test unitaire qui permet de la détecter. En rajoutant ce test à la procédure d'intégration continue, il sera alors possible de détecter la réapparition de l'anomalie très rapidement, avant livraison et d'empêcher toute forme de régression.

1. Tests unitaires

Un test unitaire est un programme très court souvent intégré dans un framework ou un module de test. Ce module de test permet de préparer un environnement de test complet et de lancer automatiquement un ensemble de tests cohérents et indépendants.

2. Cas de PHPUnit3

a. Présentation PHPUnit 3

PHPUnit3 est un framework permettant de réaliser des classes de test de vos applications. Ces tests ont pour objectif de permettre de lancer des tests unitaires en masse et aussi de produire des statistiques sur le passage et la non-régression du fonctionnement du code.

b. Installation de PHPUnit 3

```
pear channel-discover pear.phpunit.de
pear channel-discover pear.symfony-project.com
pear install phpunit/PHPUnit
```

c. Exemple de code

Soit une classe Utilisateur ayant un attribut nom.

Code de la classe Utilisateur

```
<?php

class Utilisateur
{
    private $nom;

 public function __contruct($nom) {
    $this->nom=$nom;
}
public function getNom() { return $this->nom;}
}
?>
```

Code de la classe PHPUnit de test

```
<?php

require_once 'PHPUnit/Framework.php';
 require_once 'Utilisateur.class.php';

class UtilisateurTest extends PHPUnit_Framework_TestCase
```

```
{
    public function testCreation()
    {
        $util=new Utilisateur("ENI");
        $this->assertNotNull($util);
        $this->assertType('object', $util) ;
        $this->assertObjectHasAttribute('nom', $util);

        $this->assertEquals("ENI",$util->getNom());
     }
}
?>
```

Résultat du lancement de phpUnit

```
# phpunit UtilisateurTest
PHPUnit 3.4.4 by Sebastian Bergmann.

F

Time: 0 seconds

There was 0 failureFAILURES!
Tests: 1, Assertions: 4 Failures: 0
```

Voici les deux méthodes pour lancer un test :

- `phpunit MaClasseDeTest`
- `phpunit MaClasseDeTest[MonFichierContenantLaClasse.php]`

d. Ensemble des fonctions de test PHPUnit

Fonctions de test	Fonctions inverses	Description
assertArrayHasKey()	assertNotArrayHasKey()	Test de la présence de la clé dans un tableau PHP.
assertClassHasAttribute()	assertNotClassHasAttribute()	Test de la présence de l'attribut dans une classe PHP.
assertClassHasStaticAttribute()	assertNotClassHasStaticAttribute()	Test de la présence de l'attribut statique dans une classe PHP.
assertContains()	assertNotContains()	Test de la présence de l'élément dans un ensemble PHP.
assertContainsOnly()	assertNotContainsOnly()	Test de la présence de l'élément uniquement dans un ensemble PHP.
assertEqualXMLStructure()	assertNotEqualXMLStructure()	Test d'égalité de chaîne XML.
assertEquals()	assertNotEquals()	Test d'égalité de contenu.
assertFalse()	assertNotFalse()	Test que la valeur passée en argument est égale à faux.
assertFileEquals()	assertNotFileEquals()	Test que le contenu de deux fichiers est équivalent.
assertFileExists()	assertNotFileExists()	Test d'existence de fichier.
assertGreaterThan()	assertNotGreaterThan()	Test de comparaison numérique plus grand que.
assertGreaterThanOrEqual()	assertNotGreaterThanOrEqual()	Test de comparaison numérique plus grand que ou égal.

assertLessThan()	assertNotLessThan()	Test de comparaison numérique plus petit que.
assertLessThanOrEqual()	assertNotLessThanOrEqual()	Test de comparaison numérique plus petit que ou égal.
assertNull()	assertNotNull()	Test de non nullité.
assertObjectHasAttribute()	assertNotObjectHasAttribute()	Test de la présence d'un attribut par un objet PHP.
assertRegExp()	assertNotRegExp()	Test de la validité d'une expression régulière.
assertSame()	assertNotSame()	Test d'égalité en contenu et en type.
assertSelectCount()	assertNotSelectCount()	Test d'égalité de dénombrement d'élément.
assertSelectEquals()	assertNotSelectEquals()	Test d'égalité de contenu d'élément.
assertSelectRegExp()	assertNotSelectRegExp()	Test d'égalité de contenu d'élément sélectionné par expression régulière.
assertStringEndsWith()	assertNotStringEndsWith()	Test d'égalité de fin de chaîne.
assertStringEqualsFile()	assertNotStringEqualsFile()	Test d'équivalence entre un contenu d'une chaîne et le contenu d'un fichier.
assertStringStartsWith()	assertNotStringStartsWith()	Test d'égalité de début de chaîne.
assertTag()	assertNotTag()	Test de présence d'élément XML ou HTML.
assertThat()	assertNotThat()	Test de construction d'expression logique de test.
assertTrue()	assertNotTrue()	Test d'égalité avec la valeur Vrai.

assertType()	assertNotType()	Test d'égalité de type pour une variable.
assertXmlFileEqualsXmlFile()	assertNotXmlFileEqualsXmlFile()	Test d'équivalence entre deux contenus de fichiers XML.
assertXmlStringEqualsXmlFile()	assertNotXmlStringEqualsXmlFile()	Test d'équivalence entre un contenu XML et le contenu de fichier XML.
assertXmlStringEqualsXmlStr()	assertNotXmlStringEqualsXmlStr()	Test d'équivalence entre deux contenus de chaînes XML.

e. Autre élément de la qualité logicielle : Intégration continue

L'intégration continue consiste à mettre en place dès le début du projet une architecture permettant de réaliser un audit constant du code suivant plusieurs critères. Cela permet à tout moment de savoir dans quel état se trouve le projet, mais surtout de se mettre dans une démarche d'amélioration continue et non pas de se poser cette question dans la dernière ligne droite du projet.

Un des logiciels (libre et open source) permettant cela est **SonarQube**. Couplé à **Hudson** ou à **Jenkins**, Il est permet l'intégration continue de code Java, mais s'étend à d'autres langages dont PHP, via une extension dédiée.

Ce logiciel va s'appuyer sur des outils PHP bien connus et utilisables indépendamment qui vont être présentés juste maintenant.

f. Autre élément de la qualité logicielle : Métrique

Le principe de métriques consiste à disposer d'éléments quantitatifs permettant de donner une vision qualitative du code. Une des métriques consiste à déterminer le nombre de modules, de classes, de méthodes et de lignes de code afin de déterminer des ratios qui donneront des indications.

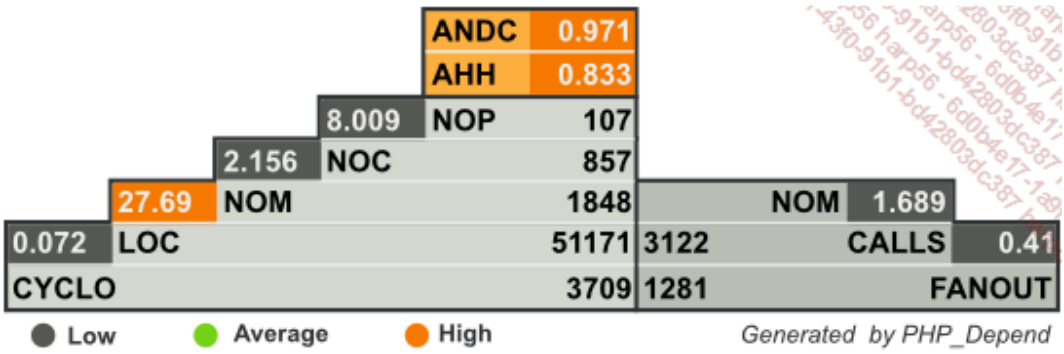
On se base ensuite sur des standards qui veulent, par exemple, qu'une méthode doive avoir un nombre de lignes raisonnable pour être lisible. Trop peu signifie que la fonction n'est peut-être pas utile ou est trop spécialisée. Trop signifie que la méthode doit être découpée de manière à ce que chaque section corresponde à une opération plus élémentaire, de manière à améliorer la lisibilité.

De la même façon, on s'attend à avoir un certain nombre de méthodes par classe et de classes par module. Cependant, sortir des clous ne signifie pas forcément faire une erreur de conception ou mal développer. Il faut donc savoir interpréter ces chiffres.

Une autre métrique importante, faisant référence, est la complexité cyclomatique (CYCLO). Il existe

également la profondeur moyenne de l'arborescence (AHH), le nombre moyen de classes dérivées (ANDC), le nombre d'appels de méthodes (CALLS).

Voici le graphique que l'on peut générer avec PHP Depend :

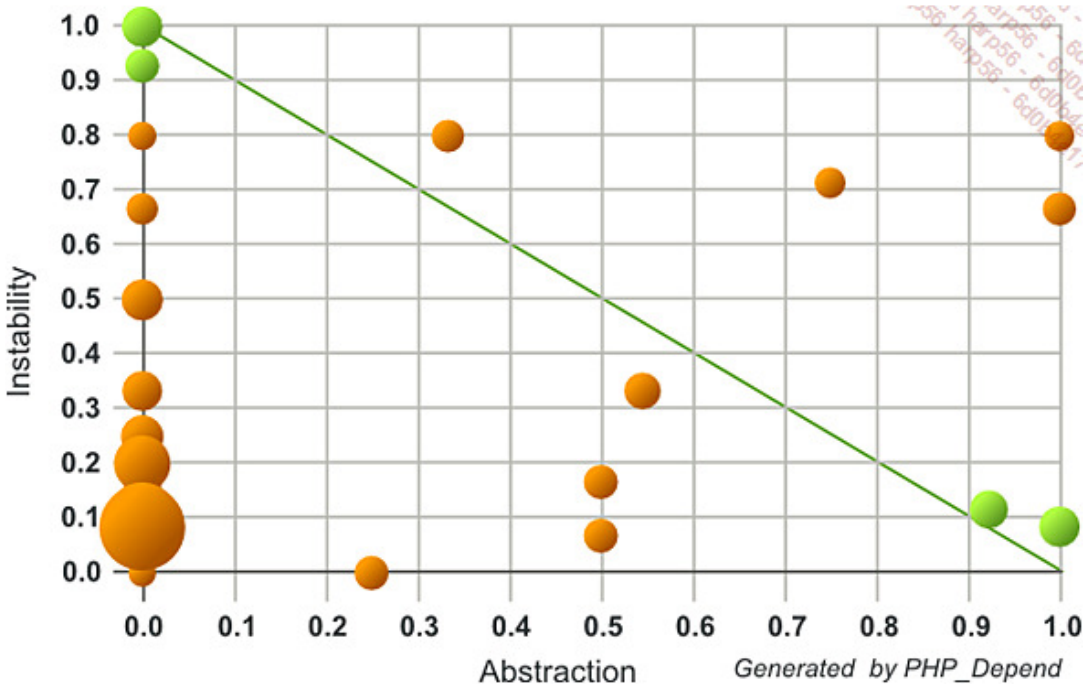


Il existe également l'étude, module par module, du rapport entre complexité et abstraction. Pour bien comprendre ces aspects, il est essentiel d'avoir de bonnes notions de programmation objet, et en particulier de savoir ce qu'est une classe abstraite et en quoi celle-ci est différente d'une interface.

Pour rappel, une interface permet de spécifier la vision que l'on a depuis l'extérieur des objets qui l'implémentent. Il s'agit d'un contrat passé entre les objets qui l'implémentent et ceux qui les utilisent. Une classe abstraite définit un comportement qui peut se spécialiser dans des classes dérivées.

Ces métriques permettent de mettre en exergue des classes abstraites qui auraient trop ou trop peu de dérivées ou qui seraient trop ou trop peu utilisées. Cela peut aider à avoir une meilleure programmation par composants.

Cette métrique peut être illustrée par un autre graphe et l'objectif est que l'ensemble des classes tournent autour de la seconde bissectrice :



La courbe présente le niveau d'abstraction en abscisse et le niveau d'instabilité en ordonnée. Tous deux sont évalués entre 0 et 1.

Les standards de codage définissent un équilibre entre ces deux notions. Ainsi, idéalement, l'ajout du niveau d'abstraction et du niveau d'instabilité doit faire 1. La seconde bissectrice représentée en vert représente donc le but à atteindre.

Tous les points représentés sur le graphique correspondent à des modules (packages) ou des sous-modules (subpackages), tels que définis dans la documentation du code à l'aide de @package et @subpackage. Plus le module est important (en termes de quantité de code) et plus le point qui le représente est gros (l'un des problèmes de ce graphique est qu'un point peut en cacher un autre).

Pour faire simple :

- Plus notre module dépend d'autres modules et plus il est instable.
- Plus d'autres modules dépendent de notre module, plus il est stable.
- Plus le paquet contient de classes abstraites et plus il est abstrait.
- Plus le paquet contient de classes concrètes et moins il est abstrait.

Il faut noter, pour terminer avec cette courte introduction, que les interfaces apportent de l'abstraction. Ainsi, lorsqu'un module ne contient que des classes abstraites, il faut faire en sorte qu'il n'ait pas de dépendances, mais que d'autres modules dépendent de lui. Au contraire, lorsque l'on a un module qui utilise des classes abstraites (typiquement, des classes appartenant à un framework), il faut faire en sorte qu'aucun autre module ne dépende de lui.

Aujourd'hui, les frameworks PHP modernes (comme ZF2) tendent à n'utiliser que peu les classes abstraites pour les remplacer par des classes concrètes avec des hooks. Ils utilisent également beaucoup les interfaces. C'est donc la bonne utilisation de ces interfaces qui va permettre d'orienter correctement les modules au sein du graphique.

Pour avoir une mesure cohérente, il faut bien entendu évaluer son projet web avec ses dépendances (ZF2, Doctrine...) et visualiser le résultat.

Dans la plupart des cas, un projet est constitué de modules sans dépendances entre eux. Chaque module ne dépend que du framework et des différents produits installés dans le répertoire vendor. Ces modules seront donc regroupés en grappe proche de l'origine (instabilité = 0 et abstraction = 0).

Ces outils statistiques sont surtout là pour poser un cadre permettant d'engager des réflexions autour du développement afin de déterminer une architecture valide. De plus, ce qui est très important, au-delà de la métrique est son évolution dans le temps. Ainsi, le fait de constater une dérive doit mener à l'identification de la raison de cette dérive.

L'outil permettant ces mesures est pdepend et il s'installe ainsi :

```
pear channel-discover pear.pdepend.org

pear install pdepend/PHP_Depend-0.9.14
```

Il s'utilise ainsi (exemple complet) :

```
pdepend --summary-xml=summary.xml --jdepend-chart=chart.svg

--overview-pyramid=pyramid.svg -d memory_limit=512M

/chemin/vers/application
```

g. Autre élément de la qualité logicielle : Détection d'erreurs

Complément à pdepend, phpmd permet de détecter des sections de code trop complexes ou présentant une écriture défaillante (paramètre obligatoire non fourni, etc.).

Voici comment il s'installe :

```
pear channel-discover pear.phpmd.org

pear install --alldeps phpmd/PHP_PMD-0.2.5
```

Voici comment il s'utilise :

```
phpmd /path/to/source text
codesize,unusedcode,naming,/path/to/my/rules.xml
```

Il va présenter une liste de lignes de code potentiellement problématiques avec une explication pour chacune. On peut y trouver de telles instructions :

```
/path/to/file/exemple.php        Avoid excessively long variable names
like $mail_address_exists
/path/to/file/exemple.php        Avoid variables with short names like
$id
```

qui conseillent respectivement :

- Évitez les noms de variables trop longs tels que `$mail_address_exists`.
- Évitez les noms de variables trop courts tels que `$id`.

Le fait d'avoir ces erreurs ne nécessite pas forcément de devoir les corriger. Certaines d'entre elles sont totalement compréhensibles. Par exemple, un attribut id dans une classe est un nom court, mais on peut comprendre qu'il s'agit d'un identifiant relatif à une instance de la classe courante. Le nommer autrement n'apporterait potentiellement rien d'autre qu'une lourdeur. Au contraire, certaines variables sont longues parce qu'elles nécessitent une sémantique précise pour rendre compte de leur contenu et de leur signification.

Au contraire, certaines erreurs telles que « Avoid unused local variables such as » (évitez les variables locales inutilisées telles que) devront impérativement être traitées, parce qu'elles révèlent un problème certain.

Voici d'autres types d'erreurs :

- the method test() has a Cyclomatic Complexity of (la méthode test() a une complexité cyclomatique évaluée à).
- the method test() has an NPath complexity of (la méthode test() a une complexité NPath évaluée à).
- too many fields (trop de champs).
- avoid really long methods (évitez les méthodes trop longues).

- this class has too many methods, consider refactoring it (cette classe a trop de méthodes, envisagez son remaniement).
- the class MyClass has 30 children. Consider to rebalance this class hierarchy (la classe MyClass a plus de 30 dérivés. Envisagez de restructurer sa hiérarchie).
- the method myMethod() contains an exit expression (la méthode myMethod() utilise la fonction exit). À titre d'information, l'utilisation de la fonction est une très mauvaise idée pour beaucoup de raisons, en particulier lorsque l'on utilise un framework.

Les éléments relevés doivent en priorité permettre de se poser des questions sur le code produit et enclencher une réflexion. Ils sont un complément à l'outil précédent permettant non pas d'avoir une vue globale, mais une vue plus fine sur des parties de code.

h. Autre élément de la qualité logicielle : Duplication de code

Lorsque l'on doit réaliser une fonctionnalité qui ressemble, même vaguement, à une autre déjà réalisée, la tentation d'un copier/coller est extrêmement forte. Cependant, il faut à tout prix s'en prémunir et remplacer cela par une réflexion plus générale sur le programme pour voir s'il n'y a pas matière à capitaliser le code.

À terme, cela permet d'avoir un code plus court, d'améliorer la réutilisabilité, la maintenabilité et la qualité globale du code. Autre point important, cela permet d'éviter la propagation des erreurs et donc de diminuer le travail à fournir lors d'une correction.

Voici comment installer le produit phpcpd qui permettra la détection de la duplication de code :

```
pear channel-discover components.ez.no

pear install phpunit/phpcpd-1.3.0
```

Voici comment l'exécuter :

```
phpcpd --min-lines 3 --min-tokens 5 --suffixes php,phtml

/path/to/sources
```

Voici comment se présente la détection d'une duplication de code :

```
path/to/file1.php:551-565
```

```
path/to/file1.php:204-118
```

Ceci signifie donc que le code est exactement similaire et qu'il convient d'engager une action pour éviter cette duplication.

i. Autre élément de la qualité logicielle : Lisibilité

Là, il s'agit d'avoir des informations sur le respect des normes de codage ou « coding style ». Certes, certaines règles sont contraignantes et peuvent paraître obsolètes en particulier ces deux-là :

- Il ne faut pas plus de 85 caractères par lignes.
- Il ne faut pas de tabulation, mais quatre espaces.

Ceci dit, contrairement à ce que l'on peut lire parfois, elles ne sont pas le fruit de l'application bête et méchante d'une doctrine, mais sont les conséquences de potentiels problèmes techniques. Par exemple, l'accès à un environnement de production peut n'être possible qu'à partir de terminaux dont la largeur ne dépassera pas 85 caractères, d'où la limitation de la longueur des lignes afin de préserver la lisibilité. De la même manière, le caractère espace est compressible alors que la tabulation ne l'est pas, ce qui a un impact lors du parsing (analyse et extraction des informations) du fichier PHP.

Généralement, les règles sont orientées pour faciliter la lecture du code et de toutes les manières, si une règle ne plaît pas, il suffit de la retirer puisque l'on peut choisir avec précision les règles que l'on décide de paramétrer.

Voici comment installer le produit PHP CodeSniffer :

```
pear install PHP_CodeSniffer-1.3.0RC1
```

Voici comment le lancer :

```
phpcs . --extensions=php
```

Voici un exemple en autorisant les tabulations (en les remplaçant par quatre caractères lors de l'évaluation du code) :

```
phpcs . --extensions=php --tab-width=4
```

Il est possible d'avoir un résumé des statistiques en capitalisant le nombre d'avertissements et d'erreurs par fichier :

```
phpcs . --extensions=php --report=summary
```

Les erreurs détectées par cet outil doivent impérativement toutes être corrigées, à partir du moment où ce dernier est bien paramétré pour correspondre à ses attentes.

Introduction aux tests fonctionnels

Parmi tous les axes de qualité, nous pouvons citer :

- La pertinence des tests fonctionnels.
- La couverture des tests fonctionnels.
- Les temps de réponse.
- La stabilité en fonctionnement.
- La robustesse de l'application PHP.
- La pertinence des réponses.
- La robustesse du code face aux erreurs.
- La pertinence des messages d'erreur.

Le test fonctionnel est à l'application ce que le test unitaire est au code. Le rôle d'un test fonctionnel est de mettre en évidence le fonctionnement correct d'une fonctionnalité de l'application indépendamment du fonctionnement du code et de la manière dont l'application est réalisée techniquement.

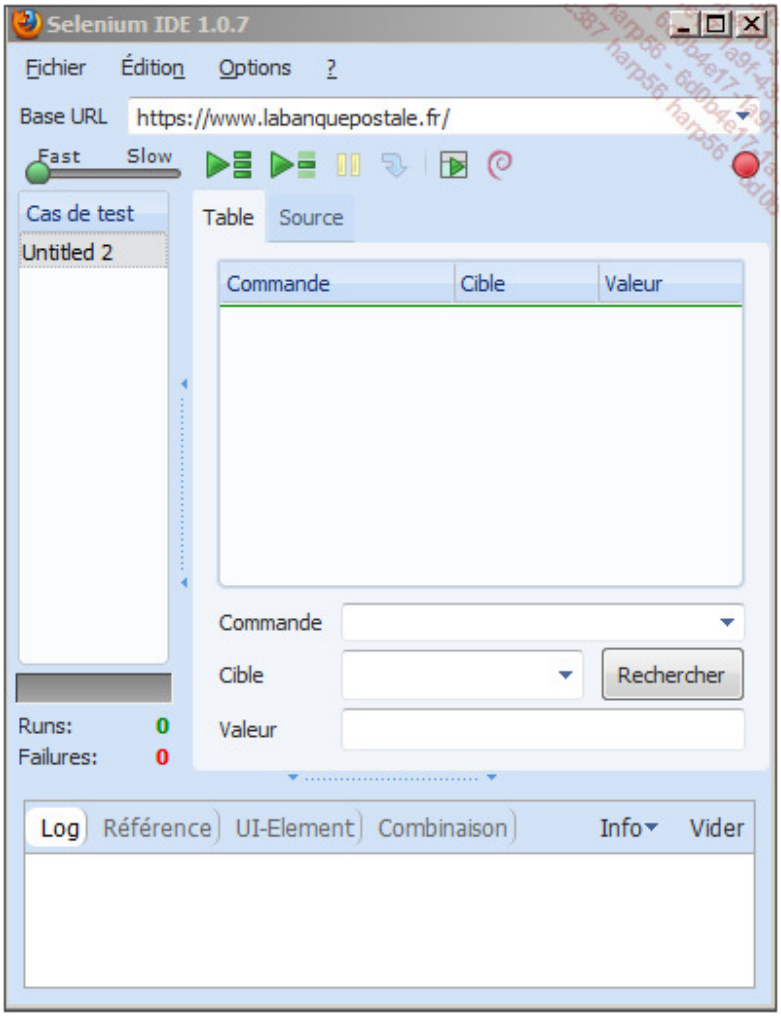
Il est donc important de trouver un bon outil permettant de réaliser l'ensemble des étapes pour valider une fonctionnalité. Simple ou complexe, la fonctionnalité est avant tout un processus permettant de valider la réception d'une information attestant le bon fonctionnement de l'application.

Parmi les bons outils permettant la validation et le test, **SeleniumHq** est un système de test des applications Web permettant d'automatiser les opérations de navigation au travers de Firefox. Essentiellement basée sur une puissante extension de Firefox, elle permet d'enregistrer des scénarios et de les rejouer à volonté.

Dès que votre jeu de tests est réalisé, il vous permet de valider les versions suivantes, de produire des plans de supervision réguliers et de réaliser un plan de tests réaliste ainsi qu'un plan de non-régression.

SeleniumHq est disponible à l'adresse suivante : http://seleniumhq.org/projects/ide/

L'essentiel de Selenium est une extension Firefox. Cependant, il possède l'ensemble des fonctionnalités pour le contrôler à distance et permettre une intégration totale au sein des environnements de tests automatisés et des environnements de développement.



L'enregistrement commence dès le clic sur l'icône d'enregistrement à droite.

Translation diagramme de classe UML vers PHP objet et SQL

Il existe de nombreuses limitations dans les langages techniques ne permettant pas de supporter l'intégralité des concepts objet offerts par le langage UML et son méta-modèle.

Ainsi il est difficile de traduire des concepts d'interface UML en langage SQL alors que cela paraît d'une simplicité et d'une cohérence évidente en langage PHP orienté objet. En effet, une interface UML trouvera sa traduction naturelle en interface PHP car le concept objet d'interface est le même entre UML et le PHP.

Un autre problème de taille est la traduction des types UML en PHP !

Le langage PHP étant un langage d'inférence, c'est-à-dire un langage dont le type de chaque variable est défini au moment de l'affectation de valeur, le passage du type dans la traduction en PHP ne peut être que l'objet d'un compromis.

Une des solutions consiste à poser systématiquement un commentaire indiquant précisément le type de donnée attendue dans chaque attribut de la classe.

L'ensemble du processus de traduction présent dans ce chapitre permet de traduire des diagrammes UML directement en code PHP.

Dans la première partie de l'exemple, la traduction de diagramme UML en langage PHP est abordée, ainsi que la génération de classes de test pour PHPUnit et la mise en place des bases de documentation pour phpDocumentor afin de faciliter la génération de la documentation du projet.

De nombreux outils existent pour la traduction de modèle. Cependant, les traducteurs fournis par les ateliers de modélisation UML en source libre tels que **StarUML** ou **argoUML** ne produisent qu'un code ne permettant pas une traduction fine des modèles produits pour vos projets.

L'expérience d'une traduction manuelle de votre conception, au moins une fois dans votre carrière de développeur PHP en environnement de conception UML orienté objet, vous permettra de mieux comprendre ce que des changements dans une classe introduisent comme changement dans les modèles UML et inversement.

Il est impératif de bien maîtriser ce processus et donc de comprendre comment une classe UML peut se traduire en PHP.

Le code proposé comme traduction n'est pas une vérité absolue mais plutôt une solution reflétant au mieux le diagramme UML.

La traduction SQL peut être automatique si on utilise une solution de type ORM comme Propel, Doctrine et Doctrine2 offrant notamment la configuration par annotation d'entité. Les annotations des classes PHP est le format le plus abordable, son utilisation est intuitive et les configurations sont souvent plus rapides qu'avec les autres formats (YAML, le XML).

1. Traduction d'une classe vide en PHP

Prenons une classe simple : Utilisateur. Voici donc sa représentation UML :

Un générateur UML avec un profil très simple pourra générer le code suivant :

```
<?php
class Utilisateur {


}
?>
```

Un générateur UML avec un profil moins trivial pourra générer plusieurs codes tels que :

- Le code source pré-documenté pour phpDocumentor.
- La classe de test sur la base de PHPUnit.
- Le code SQL de création de table.
- Le code basique d'une classe permettant d'accéder aux objets via la base de données.

Voici ce qu'il pourra générer comme code PHP et comme classe de test.

a. Traduction en classe PHP avec documentation

Ici, le processus de traduction est clair et simple. Une classe UML équivaut à une classe PHP.

```
<?php
/**
* Classe Utilisateur
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
```

```
* @version 0.1
* @package base
*/
class Utilisateur {


}
?>
```

b. Traduction en classe PHP de test PHPUnit

La classe de test est plus rudimentaire et peut être générée afin de permettre de faciliter la réalisation du code de tests unitaires basiques.

Ici, les tests sont très simples :

- Création d'objet.
- Validation du type de l'objet.
- Validation de la non-nullité de l'objet.

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'src/base/Utilisateur.class.php';


/**
* Classe TestUtilisateur
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
```

```
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class TestUtilisateur extends PHPUnit_Framework_TestSuite
{
/**
* @var objet Utilisateur
* @access private
**/
private $util1;

/**
* initialise l'objet Utilisateur
* @access protected
**/
protected function setUp()
    {
        $this->util1=new Utilisateur();
    }

/**
* libère l'objet Utilisateur
* @access protected
**/
protected function tearDown()
    {
```

```
        $this->util1 = NULL;
    }


/**
* teste la bonne création de l'utilisateur
* @access public
**/
public function testCreationUtilisateur()
    {

    $this->assertNotNull( $this->util1);
    }

/**
* teste le type de l'utilisateur
* @access public
**/
public function testObjetUtilisateur()
    {

    $this->assertType('object', $this->util1);
    }
}

?>
```

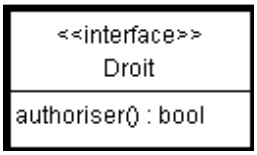
c. Traduction en code SQL

Même si la classe UML Utilisateur ne possède aucun attribut, lors de la génération de la table UTILISATEUR qui sera l'équivalent au sens SQL de cette classe, il est impératif d'ajouter un identifiant unique ID_UTILISATEUR qui permettra de jouer le rôle d'identifiant d'objet dans notre modèle.

Ainsi, chaque objet de type classe Utilisateur aura un équivalent au sens strict du terme en base de données.

```
CREATE TABLE 'base'.'UTILISATEUR' (
'ID_UTILISATEUR' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

2. Traduction d'une interface



Voici ce qui pourra être généré comme code PHP et comme classe de test.

a. Traduction en classe PHP avec documentation

Depuis la version 5 de PHP, un véritable langage objet est apparu, introduisant une nouvelle syntaxe pour déclarer des interfaces. Voici donc une traduction plus naturelle de l'interface en UML.

```
<?php
/**
* Interface Droit
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
```

```
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
interface Droit {
    public function authoriser() ;
}
?>
```

Une seconde traduction peut avoir lieu, il s'agit de traduire l'interface en classe abstraite.

```
<?php
/**
* classe abstraite pour Droit
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
abstract class Droit {
    public abstract function authoriser() ;
}
?>
```

b. Traduction en classe PHP de test PHPUnit

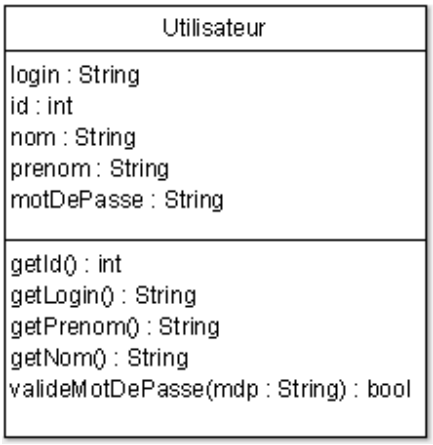
Le générateur ne pourra pas générer automatiquement de code de test de bonne qualité pour une interface.

En effet, une interface n'a de sens que lorsqu'une classe concrète réalise ou implémente les méthodes qu'expose l'interface.

c. Traduction en code SQL

Une interface pose un problème fondamental de traduction. En effet, l'interface ne porte pas d'information en soi et n'existe qu'au travers de classes de réalisation (souvent des classes concrètes) traduites elles-mêmes en SQL par des tables comportant déjà des colonnes d'information relatives à chaque attribut de la classe.

3. Traduction d'une classe et de ses propriétés



a. Traduction en classe PHP avec documentation

Toute la difficulté est maintenant de traduire directement les propriétés UML en variables et méthodes de classe.

Ce qui est proposé dans cette traduction est le respect de l'encapsulation en déclarant tous les attributs privés et en déclarant des méthodes d'accès aux attributs (ou accesseurs).

De plus, la traduction ajoute automatiquement un constructeur de classe PHP permettant la création des objets avec les valeurs des attributs en paramètre.

```
<?php
/**
* Classe Utilisateur
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Utilisateur {
    /**
    * @access private
    * @var string
    */
    private $login;

    /**
    * @access private
    * @var integrer
    */
    private $id;
```

```
/**
* @access private
* @var string
*/
private $nom;

/**
* @access private
* @var string
*/
private $prenom;

/**
* @access private
* @var string
*/
private $motDePasse;



/**
* @access public
* @param none
* @return string
*/
public function getLogin() {
    return $this->login;
}
```

```
/**
* @access public
* @param none
* @return integer
*/
public function getId() {
    return $this->id;
}
/**
* @access public
* @param none
* @return string
*/
public function getPrenom() {
    return $this->login;
}
/**
* @access public
* @param none
* @return string
*/
public function getNom() {
    return $this->nom;
}
/**
* @access public
```

```
    * @param string
    * @return boolean
    */
    public function valideMotDePasse ($mdp) {
        $retour=FALSE;
        # Code à réaliser
        return $retour;
    }

    /**
    * Contructeur
    * @access public
    */
    public function __construct(     $login="",
                                     $id=0,
                                     $nom="",
                                     $prenom="",
                                     $motDePasse="")
    {
        $this->login=$login;
        $this->id=$id;
        $this->nom=$id;
        $this->prenom=$prenom;
        $this->motDePasse=$motDePasse;
    }
}
?>
```

b. Traduction en classe PHP de test PHPUnit

Le code produit les mêmes tests de non-nullité et ajoute les tests d'appel à l'ensemble des méthodes de l'objet.

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'src/base/Utilisateur.class.php';

/**
* Classe TestUtilisateur
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class TestUtilisateur extends PHPUnit_Framework_TestSuite
{
/**
* @var objet Utilisateur
* @access private
**/
private $util1;
```

```
/**
* @var objet Utilisateur
* @access private
**/
private $util2;

/**
* initialise l'objet Utilisateur
* @access protected
**/
protected function setUp()
    {
        $this->util1=new Utilisateur();
    }

/**
* libère l'objet Utilisateur
* @access protected
**/
protected function tearDown()
    {
        $this->util1 = NULL;
    }
/**
* teste la bonne création de l'utilisateur
* @access public
**/
```

```
public function testCreationUtilisateur()
    {
    $this->assertNotNull( $this->util1);
    }
/**
* teste le type de l'utilisateur
* @access public
**/
public function testObjetUtilisateur()
    {

    $this->assertType('object', $this->util1);
    }
/**
* teste la récupération de Login de la classe Utilisateur
* @access public
**/
public function testGetLoginUtilisateur()
    {

    $this->assertNotNull( $this->util1->getLogin());
    }



/**
* teste la récupération de Id de la classe Utilisateur
* @access public
```

```
**/
public function testGetIdUtilisateur()
    {

    $this->assertNotNull( $this->util1->getId());
    }

/**
* teste la récupération de Nom de la classe Utilisateur
* @access public
**/
public function testGetNomUtilisateur()
    {

    $this->assertNotNull( $this->util1->getNom());
    }

/**
* teste la récupération de Prenom de la classe Utilisateur
* @access public
**/
public function testGetPrenomUtilisateur()
    {

    $this->assertNotNull( $this->util1->getPrenom());
    }
```

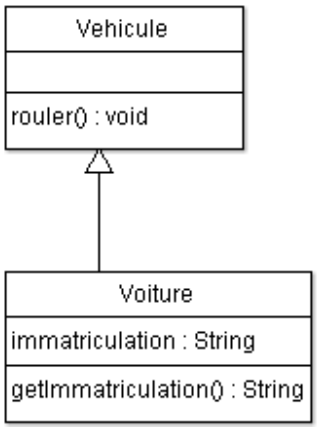
```
/**
* teste la méthode valideMotDePasse de la classe Utilisateur
* @access public
**/
public function testValideMotDePasseUtilisateur()
    {

    $this->assertTrue ( $this->util1->valideMotDePasse(""));
    # ou bien
    $this->assertFalse ( $this->util1->valideMotDePasse(""));
    }

}

?>
```

c. Traduction en code SQL

Le processus de traduction ici épure grandement la sémantique UML car il est impossible de traduire en SQL des fonctions telles que les accesseurs.

```
CREATE TABLE 'base'.'UTILISATEUR' (
'ID_UTILISATEUR' INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
'LOGIN' VARCHAR(255),
'NOM' VARCHAR(255),
'PRENOM' VARCHAR(255),
'MOTDEPASSE' VARCHAR(255)
);
```

4. Traduction de l'héritage en classe



a. Traduction en classe PHP Véhicule

```
<?php
/**
* Classe Vehicule
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Vehicule {
    /**
    * @access public
    * @param none
```

```
    * @return void
    */
    public function rouler() {
    }

    /**
    * Constructeur
    * @access public
    */
    public function __construct()
    {
    }
}
?>
```

b. Traduction en classe de test PHPUnit pour Véhicule

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'src/base/Utilisateur.class.php';

/**
* Classe TestVehicule
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
```

```
* @version 0.1
* @package base
*/
class TestVehicule extends PHPUnit_Framework_TestSuite
{
/**
* @var objet Vehicule
* @access private
**/
private $vehcule1;

/**
* initialise l'objet Vehicule
* @access protected
**/
protected function setUp()
    {
        $this->vehicule1=new Vehicule();
    }

/**
* libère l'objet Vehicule
* @access protected
**/
protected function tearDown()
    {
        $this-> vehicule1 = NULL;
```

```
    }


/**
* teste la bonne création de vehicule
* @access public
**/
public function testCreationVehicule()
    {

    $this->assertNotNull( $this-> vehicule1);
    }

/**
* teste le type de Vehicule
* @access public
**/
public function testObjetVehicule()
    {

    $this->assertType('object', $this->vehicule1);
    }
}
?>
```

c. Traduction en classe PHP Voiture

```php
<?php
/**
* Classe Voiture
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Voiture extends Vehicule {
    /**
    * @access private
    * @var string
    */
    private $immatriculation;

    /**
    * @access public
    * @param none
    * @return string
    */
    public function getImmatriculation() {
        return $this->immatriculation;
    }

    /**
```

```
    * Constructeur
    * @access public
    */
    public function __construct($immatriculation="" )
    {
        $this->immatriculation =$immatriculation;
    }
}
?>
```

d. Traduction en classe de test PHPUnit pour Voiture

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'src/base/Utilisateur.class.php';

/**
* Classe TestVoiture
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class TestVoiture extends PHPUnit_Framework_TestSuite
{
```

```
/**
* @var objet Voiture
* @access private
**/
private $voiture;

/**
* @var objet Voiture
* @access private
**/
private $voiture;

/**
* initialise l'objet Voiture
* @access protected
**/
protected function setUp()
    {
        $this->voiture=new Voiture ();
    }

/**
* libère l'objet Voiture
* @access protected
**/
protected function tearDown()
    {
```

```
        $this->voiture = NULL;
    }


/**
* teste la bonne création de voiture
* @access public
**/
public function testCreationVoiture()
    {

    $this->assertNotNull( $this->voiture);
    }

/**
* teste le type de voiture
* @access public
**/
public function testObjetVoiture ()
    {

    $this->assertType('object', $this->voiture);
    }


/**
* teste la récupération de Immatriculation de la classe Voiture
```

```
* @access public
**/
public function testGetImmatriculationVoiture ()
    {

    $this->assertNotNull( $this->voiture->getImmatriculation());
    }
}

?>
```

e. Traduction en code SQL

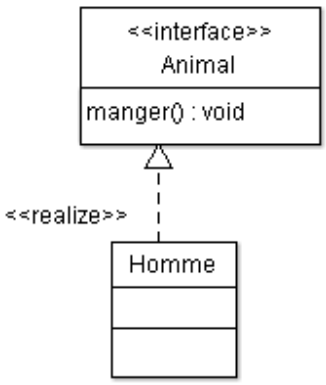
Traduction en code SQL classe mère :

```
CREATE TABLE 'base'.'VEHICULE' (
'ID_VEHICULE' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

Traduction en code SQL classe fille :

```
CREATE TABLE 'base'.'VOITURE' (
'ID_VOITURE' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'ID_VEHICULE' INT NOT NULL,
'IMMATRICULATION'VARCHAR(255),
FOREIGN KEY ('ID_VEHICULE') REFERENCES 'base'.'VEHICULE'
('ID_VEHICULE') ON DELETE CASCADE
);
```

5. Traduction de l'héritage d'interface



a. Traduction en interface PHP avec documentation

```
<?php
/**
* Interface Animal
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
interface Animal {
    public function manger() ;
}
?>
```

b. Traduction en classe PHP Homme

```
<?php
/**
* Classe Homme
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Homme implements Animal {
    /**
    * @access public
    * @param none
    * @return string
    */
    public function manger(){
    }

    /**
    * Constructeur
    * @access public
    */
    public function __construct()
```

```
    {
    }
}
?>
```

c. Traduction en classe PHP de test PHPUnit

```
<?php
require_once 'PHPUnit/Framework.php';
require_once 'src/base/Homme.class.php';

/**
* Classe TestHomme
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class TestHomme extends PHPUnit_Framework_TestSuite
{

/**
* @var objet Homme
* @access private
**/
```

```
private $homme;

/**
* initialise l'objet Homme
* @access protected
**/
protected function setUp()
    {
        $this->homme=new Homme ();
    }

/**
* libère l'objet Homme
* @access protected
**/
protected function tearDown()
    {
        $this->homme = NULL;
    }


/**
* teste la bonne création de homme
* @access public
**/
public function testCreationHomme ()
    {
```

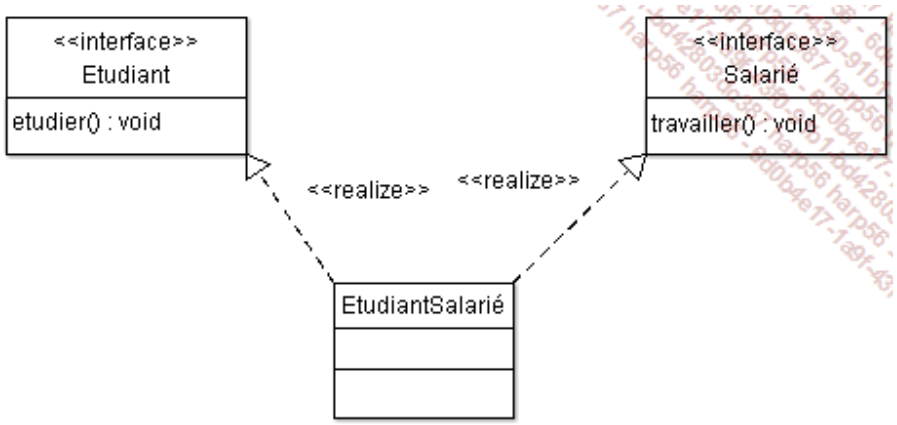
```
    $this->assertNotNull( $this->homme);
    }

/**
* teste le type de homme
* @access public
**/
public function testObjetHomme ()
    {

    $this->assertType('object', $this->homme);
    }



/**
* teste la méthode manger de la classe Homme
* @access public
**/
public function testMangerHomme ()
    {
    $this->homme->manger();
    }
}

?>
```

d. Traduction en code SQL classe fille

Il n'est pas possible de traduire en SQL le résultat d'une interface car une interface n'a pas d'identité propre. La solution consiste donc à ajouter un champ indiquant les interfaces implémentées sans créer de table pour l'interface.

```
CREATE TABLE 'base'.'HOMME' (
'ID_HOMME' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'INTERFACES' VARCHAR(255) NOT NULL
);
```

6. Traduction de l'héritage d'interface multiple



a. Traduction en interface PHP Etudiant

```
<?php
/**
* Interface Etudiant
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
```

```
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
interface Etudiant {
    public function etudier() ;
}
?>
```

b. Traduction en interface PHP Salarie

```
<?php
/**
* Interface Salarie
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
interface Salarie {
     public function travailler() ;
}
?>
```

c. Traduction en interface PHP EtudiantSalarie

```
<?php
/**
* Classe EtudiantSalarie
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class EtudiantSalarie implements Etudiant, Salarie {
    /**
    * @access public
    * @param none
    * @return none
    */
    public function etudier(){
    }

    /**
    * @access public
    * @param none
    * @return none
    */
    public function travailler(){
```

```
    }

    /**
    * Constructeur
    * @access public
    */
    public function __construct()
    {
    }
}
?>
```

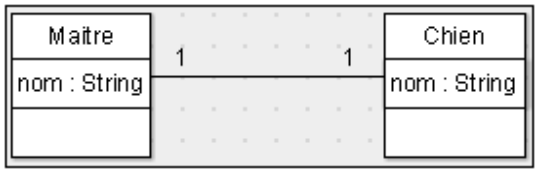
d. Traduction en code SQL classe fille

Comme dans l'exemple précédent, un champ avec le nom des interfaces séparées par une virgule permet l'implémentation de la notion d'héritage.

```
CREATE TABLE 'base'.'ETUDIANTSALARIE' (
'ID_ ETUDIANTSALARIE ' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'INTERFACES' VARCHAR(255) NOT NULL
);
```

Pour des raisons de non-existence d'une interface, nous n'avons pas créé de table définissant les interfaces. Il s'agit d'un choix conceptuel. Cependant, l'implémentation des interfaces sous forme de tables ou de vues SQL est totalement possible.

7. Traduction d'une association 1-1



a. Traduction en classe PHP Maître

```
<?php
/**
* Classe Maitre
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Maitre {
    /**
    * @access private
    * @var objet Chien
    */
    private $monChien;
    /**
    * @access private
    * @var string
    */
    private $nom;
```

```
    /**
    * @access public
    */
    public function __construct($nom, $chien=null) {
        $this->nom=$nom;
        if (isset($chien)) {
            $this->monChien=$chien;
            $chien->setMaitre($this);
        }
    }

    /**
    * @access public
    */
    public function __toString() {
        #var_dump($this);

        $str="Maitre [ Nom: ".$this->nom."]";
        if (isset($this->monChien)) {
            $str."\nMon chien : ".$this->monChien->getNom();
        } else { $str."\nPas de chien, je ne suis pas un maître-
chien"; }
        $str."\n\n";
        return $str;
    }
    /**
```

```
    * @access public
    */
    public function setChien($c) {
        $this->monChien=$c;
    }

    /**
    * @access public
    */
    public function getNom() { return $this->nom; }
}
?>
```

b. Traduction en classe PHP Chien

```
<?php

/**
* Classe chien
* Date : 27/01/2010 23 :05
* @author Jean-Marie Renouard
* @ copyright ENI 2010
* @licence GNU GPL 3.0
* @version 0.1
* @package base
*/
class Chien {
```

```
/**
* @access private
* @var objet Maitre
*/
private $monMaitre;
/**
* @access private
* @var string
*/
private $nom;

/**
* @access public
*/
public function __construct($nom, $Maitre=null) {
    $this->nom=$nom;
    if (isset($Maitre)) {
        $this->monMaitre=$Maitre;
        $Maitre->setChien($this);
    }
}

/**
* @access public
*/
public function __toString() {
    $str="Chien [ Nom: ".$this->nom."]";
```

```
        if (isset($this->monMaitre)) {
            $str.="\nMon Maitre : ".$this->monMaitre-
>getNom();
        } else { $str.="\nPas de maître, Chien errant."; }
        $str.="\n";
        return $str;
    }

    /**
    * @access public
    */
    public function setMaitre($m) {
        $this->monMaitre=$m;
    }
    public function getNom() { return $this->nom; }
}
?>
```

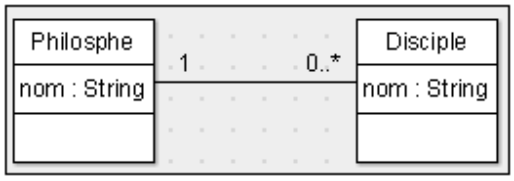
c. Traduction de la classe Maître en code SQL

```
CREATE TABLE 'base'.'MAITRE' (
'ID_MAITRE' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'ID_CHIEN' INT NOT NULL,
FOREIGN KEY ('ID_CHIEN') REFERENCES 'base'.'CHIEN'
('ID_CHIEN') ON DELETE CASCADE
);
```

d. Traduction de la classe Chien en code SQL

```
CREATE TABLE 'base'.'CHIEN' (
'ID_CHIEN' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'ID_MAITRE' INT NOT NULL,
FOREIGN KEY ('ID_MAITRE') REFERENCES 'base'.'MAITRE'
('ID_MAITRE') ON DELETE CASCADE
);
```

8. Traduction d'une association 1-N



a. Traduction en classe PHP Philosophe

```
<?php

class Philosophe {
    private $disciples;
    private $nom;

    public function __construct($nom, $etuds=null) {
        $this->nom=$nom;
```

```
        $this->disciples=array();

        if (isset($etuds)) {
            if ( is_array($etuds) ) {
                    $this->disciples=array_unique($etuds);
            } else {
                    array_push($this->disciples, $etuds);
            }
        }

        foreach ($this->disciples as $etu) { $etu-
>setPhilosophe($this); }
    }

    public function __toString() {
        $str="Philosophe [ Nom: ".$this->nom."]";
        if (count($this->disciples) > 0) {
            $str."\n\ts'occupe des disciples :";
            foreach ($this->disciples as $etu) {
$str."\n\t\t * ". $etu->getNom(); }
        } else { $str."\n\tPas de disciple."; }
        $str."\n";
        return $str;
    }

    public function addDisciple($etu, $propage=true) {
        if (!in_array($etu, $this->disciples)) array_push($this-
```

```
>disciples, $etu);
        if ($propage) $etu->addPhilosophe($this, false);
    }

    public function delDisciple($etu, $propage=true) {
        $this->disciples= array_diff($this->disciples,
array($etu));
        if ($propage) $etu->delPhilosophe($this, false);
    }
    public function getNom() { return $this->nom; }
}
?>
```

b. Traduction en classe PHP Disciple

```
<?php

class Disciple {
    private $philosophe;
    private $nom;

    public function __construct($nom, $phil=null) {
        $this->nom=$nom;

        $this->philosophe=$phil;

    }
```

```
    public function __toString() {
        $str="Disciple [ Nom: ".$this->nom."\n\tPhilosophe:
".$this->philosophe->getNom()."]";
        $str.="\n";
        return $str;
    }

    public function setPhilosophe($phil) {
        $this->philosophe=$phil;
    }

    public function getPhilosophe() {
        return $this->philosophe;
    }
}
?>
```

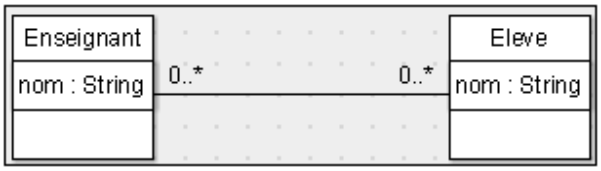
c. Traduction de classe Philosophe en code SQL

```
CREATE TABLE 'base'.'PHILOSOPHE' (
'ID_PHILOSOPHE' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

d. Traduction de classe Disciple en code SQL

```
CREATE TABLE 'base'.'DISCIPLE' (
'ID_DISCIPLE' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'ID_PHILOSOPHE' INT NOT NULL,
FOREIGN KEY ('ID_PHILOSOPHE') REFERENCES 'base'.'PHILOSOPHE'
('ID_PHILOSOPHE') ON DELETE CASCADE
);
```

9. Traduction d'une association N-N



a. Traduction en classe PHP Enseignant

```
<?php

class Enseignant {
    private $etudiants;
    private $nom;

    public function __construct($nom, $etuds=null) {
        $this->nom=$nom;

        $this->etudiants=array();
```

```
        if (isset($etuds)) {
            if ( is_array($etuds) ) {
                $this->etudiants=array_unique($etuds);
            } else {
                array_push($this->etudiants, $etuds);
            }
        }

        foreach ($this->etudiants as $etu) { $etu-
>addEnseignant($this); }
    }

    public function __destruct() {
        foreach ($this->etudiants as $etu) { $etu-
>delEnseignant($this); }
    }


    public function __toString() {
        $str="Enseignant [ Nom: ".$this->nom."]";
        if (count($this->etudiants) > 0) {
            $str."\n\ts'occupe des étudiants :";
            foreach ($this->etudiants as $etu) {
$str."\n\t\t * ". $etu->getNom(); }
        } else { $str."\n\tPas d etudiant."; }
        $str."\n";
        return $str;
```

```
    }

    public function addEtudiant($etu, $propage=true) {
        if (!in_array($etu, $this->etudiants)) array_push($this-
>etudiants, $etu);
        if ($propage) $etu->addEnseignant($this, false);
    }

    public function delEtudiant($etu, $propage=true) {
        $this->etudiants= array_diff($this->etudiants,
array($etu));
        if ($propage) $etu->delEnseignant($this, false);
    }
    public function getNom() { return $this->nom; }
}
?>
```

b. Traduction en classe PHP Etudiant

```
<?php

class Etudiant {
    private $enseignants;
    private $nom;

    public function __construct($nom, $enss=null) {
        $this->nom=$nom;
```

```
        $this->enseignants=array();

        if (isset($enss)) {
            if ( is_array($enss) ) {
                $this->enseignants=array_unique($enss);
            } else {
                array_push($this->enseignants, $enss);
            }
        }

        foreach ($this->enseignants as $ens) { $ens-
>addEtudiant($this); }
    }

    public function __destruct() {
        foreach ($this->enseignants as $ens) { $ens-
>delEtudiant($this); }
    }


    public function __toString() {
        $str="Etudiant [ Nom: ".$this->nom."]";
        if (count($this->enseignants) > 0) {
            $str.="\n\tsuit les cours de :";
            foreach ($this->enseignants as $ens) {
$str.="\n\t\t * ". $ens->getNom(); }
```

```
        } else { $str.="\n\tPas d enseignant."; }
        $str.="\n";
        return $str;
    }

    public function addEnseignant($ens, $propage=true) {
        if (!in_array($ens, $this->enseignants))
array_push($this->enseignants, $ens);
        if ($propage) $ens->addEtudiant($this, false);
    }

    public function delEnseignant($ens, $propage=true) {
        $this->enseignants= array_diff($this->enseignants,
array($ens));
        if ($propage) $ens->delEtudiant($this, false);
    }
    public function getNom() { return $this->nom; }
}
?>
```

c. Traduction de la classe Enseignant en code SQL

```
CREATE TABLE 'base'.'ENSEIGNANT' (
'ID_ENSEIGNANT' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

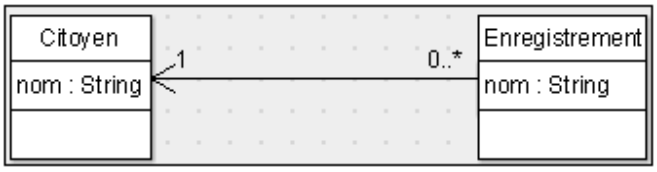
d. Traduction de la classe Etudiant en code SQL

```
CREATE TABLE 'base'.'ETUDIANT' (
'ID_ETUDIANT ' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
);
```

c. Traduction de l'association Etudiant/Enseignant en code SQL

```
CREATE TABLE 'base'.'ENSEIGNANT_ETUDIANT' (
'ID_ENSEIGNANT_ETUDIANT' INT NOT NULL AUTO_INCREMENT PRIMARY KEY
'ID_ENSEIGNANT' INT NOT NULL,
'ID_ETUDIANT' INT NOT NULL,
FOREIGN KEY ('ID_ENSEIGNANT') REFERENCES 'base'.'ENSEIGNANT'
('ID_ENSEIGNANT') ON DELETE CASCADE
'ID_PHILOSOPHE' INT NOT NULL,
FOREIGN KEY ('ID_ETUDIANT') REFERENCES 'base'.'ETUDIANT'
('ID_ETUDIANT') ON DELETE CASCADE
);
```

10. Traduction d'une association 1-N à navigation restreinte



Dans cet exemple le code est similaire au code du philosophe et des disciples. Cependant le code sera allégé côté de la classe Citoyen où aucune information sur les enregistrements ne sera stockée dans un tableau PHP (pas d'attribut enregistrements). Dans le cas d'une navigation restreinte, le code produit est plus simple.

Le fait de poser des navigations restreintes dans votre code permet de simplifier le code et donc d'offrir plus de performance en exécution car moins de code est nécessaire pour créer un objet. Le stockage en base de données ou sous forme sérialisé dans un fichier ou en mémoire, par exemple, prendra moins d'espace et donc fera un bon candidat pour la montée en charge.

a. Traduction en classe PHP Citoyen

```
<?php

class Citoyen {
    private $nom;

    public function __construct($nom) {
        $this->nom=$nom;


    public function __toString() {
        $str="Citoyen [ Nom: ".$this->nom."]";
        $str.="\n";
        return $str;
    }

    public function getNom() { return $this->nom; }
}
?>
```

b. Traduction en classe PHP Enregistrement

```
<?php
class Enregistrement {
    private $citoyen;
    private $nom;

    public function __construct($nom, $cito=null) {
        $this->nom=$nom;

        $this->citoyen=$cito;

    }

    public function __toString() {
        $str="Enregistrement [ Nom: ".$this->nom."\n\tCitoyen:
".$this->citoyeni->getNom()."]";
        $str.="\n";
        return $str;
    }

    public function setCitoyen($cito) {
        $this->citoyen=$cito;
    }

    public function getCitoyen() {
```

```
        return $this->citoyen;
    }
}
?>
```

c. Traduction en SQL

Le mécanisme de traduction d'une navigation restreinte reste similaire à celui d'une navigation non restreinte. Le code ne change pas, avec ou sans navigation restreinte, pour des raisons évoquées précédemment sur le maintien de la cohérence des liens sémantiques dans un modèle relationnel classique.

11. Traduction d'agrégations

Le langage PHP n'offre que très peu de moyens de traduire une distinction forte entre une association et une agrégation et nous nous contenterons de traduire ce lien d'association fort comme nous avons procédé avec les associations. La seule chose que nous pouvons réaliser est la mise en place de commentaires spécifiques dans le code indiquant le lien fort mis en évidence dans la modélisation du système.

Le langage SQL n'offre que très peu de moyens de traduire une distinction forte entre une association et une agrégation. Nous nous contenterons de traduire ce lien d'association fort comme nous avons déjà procédé, avec les associations par les contraintes d'intégrité et plus précisément par le dépôt de clé de référence étrangère. La seule chose que nous pouvons réaliser c'est la mise en place d'une contrainte d'intégrité spécifique indiquant que la suppression d'un tuple dans une table doit entraîner une suppression en cascade. Ce type de traduction permettra de modéliser assez finement la notion de cycle de vie lié.