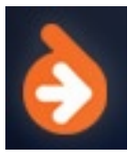
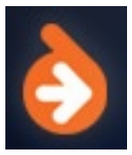


ORMs, why should I care?

Introduction to the Doctrine Object Relational Mapper

Lukas Smith (lukas@liip.ch)

IPC 08 Spring Edition in Karlsruhe



Err, and who are you?

I have been doing PHP since 2000

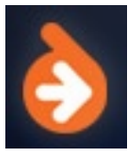
Joined PEAR in 2002 (MDB2, LiveUser, Group and QA)

Heavily involved in the PHP release process and information flow facilitation since several years

However my passion is Ultimate Frisbee

So where do you all come from?

What do you work on?



Those darn objects

PHP applications increasingly depend on OO code

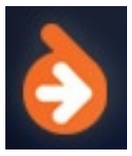
But how do I get objects out of my database?

Why do I even want objects to come out of my database?

OO does not really map naturally to RDBMS: identity, state, behavior and encapsulation vs. relation, attribute, tuple, relation value and relation variable.

All the pain is summed up by the term “impedance mismatch”

No real interitance, polymorphism
No private, protected, public (well sort of via views)
No pointers, semantic differences (collation is part of the type)
RDBMS do not like tree structures, which naturally lend themselves in OO structures (by reference)
RDBMS have a simple set of operators
OO has no sense of transactions in the RDBMS sense



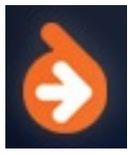
Buzz word bingo

ActiveRecord pattern maps objects to tables

“Object-relational mapping is a [..] technique for converting data between [..] relational databases and object-oriented programming languages. This creates, in effect, a "virtual object database"”

In other words: ORM work much harder at hiding the underlying tables, SQL etc. inside the RDBMS

Some smart people say its not even achievable or feasible



Simple facts of life!?!

ORMs require a lot of code, this adds overhead and potential for bugs ..

Everybody knows SQL, but who knows ORM XYZ?

ORMs make simple things hard and complex things impossible

ORMs generate inefficient SQL

ORMs force their style onto my code

What else sucks about ORMs?

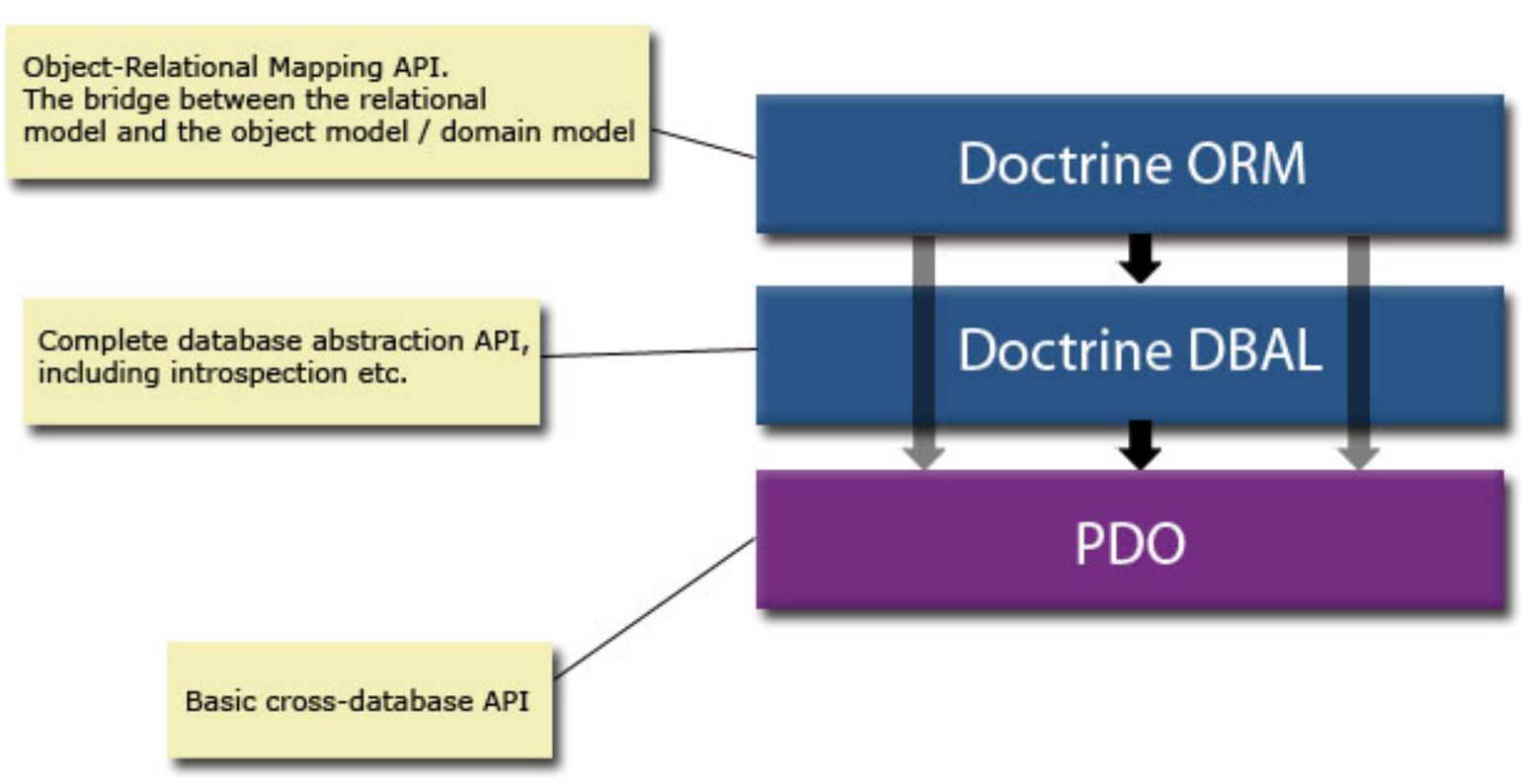
bugs certainly are true for Doctrine, test suite still needs a lot of work
DQL is fairly SQLish

New kid on the block?

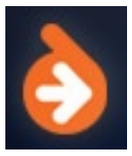
Started back in 2006, by Konsta from Finland

Today there are about a handful of active core devs

September 1st 2008 planned date for the big 1.0



- initially the focus was on adding “cool” features
- today the focus is more around stability and consistency
- requires PHP 5.2.3



C[R]UDiling with an ORM

Automatic Create, Read, Update, Delete

```php
<?php require_once('../setup.php');

$user = new User();
$user->name = 'Foo';
$user->username = $user->password = 'yummy';
$user->Phonenumber[0]->phonenumber = '007';

// create a user in the db
$user->save();

// read a user from the db
$user2 = Doctrine::getTable('User')->find($user->id);

// update the user in the db
$user2->set('name', 'Bar')->save();

// delete the user from the db
$user2->delete();
```

```php
<?php

class User extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('name', 'string',30);
        $this->hasColumn('username', 'string',20);
        $this->hasColumn('password', 'string',16);
    }

    public function setUp()
    {
        $this->actAs('Timestampable');
        $this->hasMany('Phonenumber', array('local' => 'id', 'foreign' =>
    }
}
```

```
INSERT INTO user (name, password, username, created_at, updated_at) VALUES (?, ?, ?, ?, ?)
    (Foo, yummy, yummy, 2008-05-18 09:59:01, 2008-05-18 09:59:01)
INSERT INTO phonenumber (user_id, phonenumber) VALUES (?, ?)
    (1, 007)
SELECT u.id AS u__id, u.name AS u__name, u.username AS u__username, u.password AS u__password, u.created_at
    (1)
UPDATE user SET name = ? WHERE id = ?
    (Bar, 1)
DELETE FROM user WHERE id = ?
```


- insert (including relations), select, update and delete is supported

To delete or not to delete?

Mark deleted phone numbers as deleted

```php
<?php require_once('../setup.php');

$user = new User();
$user->name = 'Foo';
$user->username = $user->password = 'yummy';
$user->Phonenumber[0]->phonenumber = '007';

$user->save();

// instead of a DELETE
// the row in the database is updated
$user->Phonenumber[0]->delete();
```

```php
<?php

class Phonenumber extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('phonenumber', 'string', 20);
        $this->hasColumn('user_id', 'integer');
        $this->hasColumn('deleted', 'boolean', null, array('default' => true));
    }
    public function preDelete($event)
    {
        $event->skipOperation();
    }
    public function postDelete($event)
    {
        $this->deleted = false;
        $this->save();
    }
}
```

INSERT INTO user (name, password, username, created_at, updated_at) VALUES (?, ?, ?,
(Foo, yummy, yummy, 2008-05-23 20:20:01, 2008-05-23 20:20:01)
INSERT INTO phonenumber (deleted, user_id, phonenumber) VALUES (?, ?, ?)
(1, 1, 007)
UPDATE phonenumber SET deleted = ? WHERE id = ?
(0, 1)

- Instead of deleting, we will set the deleted field to false (this is actually a typo that i intentionally left in to show case how easy it is to have a typo in your code after recatoring .. see slide 15)

Making it hard to forget

Automatically modify DQL queries

```php
<?php $filter = '/^SE/'; require_once('../setup.php');

$user = new User();
$user->name = 'Foo';
$user->username = $user->password = 'yummy';
$user->Phonenumber[0]->phonenumber = '007';
$user->Phonenumber[1]->phonenumber = '911';
// save the user, and delete one of the relations
$user->save(); $user->Phonenumber[0]->delete();

// fetch all users ("deleted" check is automatically added)
$dql = 'SELECT u.id, p.id FROM User u LEFT JOIN u.Phonenumber p';
$query = Doctrine_Query::create();
$user = $query->parseQuery($dql)->execute()->getFirst();
var_dump($user->toArray());
```

```
SELECT u.id AS u__id, p.id AS p__id FROM user u LEFT JOIN
phonenumber p ON u.id = p.user_id WHERE p.deleted = ?
(0)
```

```
array
  'id' => string '1' (length=1)
  'name' => string 'Foo' (length=3)
  'username' => string 'yummy' (length=5)
  'password' => string 'yummy' (length=5)
  'Phonenumber' =>
    array
      0 =>
        array
          'id' => string '2' (length=1)
          'phonenumber' => string '911' (length=3)
          'user_id' => string '1' (length=1)
          'deleted' => boolean false
```

```php
<?php

class Phonenumber extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('phonenumber', 'string', 20);
        $this->hasColumn('user_id', 'integer');
        $options = array(
            'default' => false,
            'notnull' => true,
        );
        $this->hasColumn('deleted', 'boolean', 1, $options);
    }
    public function preDqlSelect($query, $component, $alias)
    {
        $field = $alias.'.deleted';
        if (!$query->contains($field)) {
            $query->addWhere($field.' = ?', array(false));
        }
    }
public function preDqlDelete($query, $component, $alias)
{
    $field = $alias.'.deleted';
    if (!$query->contains($field)) {
        $query->from('')->update($component.' '.$alias);
        $query->set($field, '?', array(false));
    }
}
```


- preDqlSelect() hook to ensure that “deleted” is dealt with
- notice that the entire data of the dumped entity is available, even though we only fetched the id’s. that is because the object is already fetched in memory

Trusted vs. Trustworthy

Primitive and more complex data type validation

```php
<?php require_once('../setup.php');

$conn->setAttribute(
    Doctrine::ATTR_VALIDATE,
    Doctrine::VALIDATE_ALL
);

$user = new User();
$user->name = 'Foo';
// username may not be blank
$user->username = '';
$user->password = 'yummy';
// email must be a valid email
$user->email = 'foo';

try {
    $user->save();
} catch (Doctrine_Validator_Exception $e) {
    // get all errors raised
    $userErrors = $user->getErrorStack();

    // iterate over all errors
    foreach($userErrors as $fieldName => $errorCodes)
        $message = 'Error Code for field "%s": ';
        switch ($fieldName) {
        default:
            $msg = sprintf($message, $fieldName);
            echo $msg.implode(', ', $errorCodes);
            echo "<br />\n";
            break;
        }
    }
}
```

```php
<?php

class User extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('name', 'string',30);
        $this->hasColumn('username', 'string',20, array('notblank' => true));
        $this->hasColumn('password', 'string',16);
        $this->hasColumn('email', 'string', '200', array('email' => true));
    }
}
```

```
Error Code for field "username": notblank
Error Code for field "email": email
```


- Added validation rule “notblank” to username
- Added a new column “email” with the validation rules “email”

Being lazy is an art

Minimize overhead by loading properties when needed

```php
<?php $filter = '/^SE/'; require_once('../setup.php');

$user = new User();
$user->name = 'Foo';
$user->username = $user->password = 'yummy';
$user->Phonenumber[0]->phonenumber = '007';
$user->save();
$user2 = $user->copy();
$user2->Phonenumber[0]->phonenumber = '007';
$user2->Phonenumber[1]->phonenumber = '911';
$user2->save();

// make sure that no in memory references exist
$user->free(); $user2->free();

// fetch only the user ids
$users = Doctrine_Query::create()
    ->select('u.id')->from('User u')
    ->execute();

// already fetched
var_dump($users[0]->id);

// lazy fetch all properties of the first user
var_dump($users[0]->name);
// lazy fetch the Phonebumbers of the first user
var_dump($users[0]->Phonenumber->count());
// fetch all Phonenumbers for all users
$users->loadRelated('Phonenumber');
var_dump($users[1]->Phonenumber->count());
```

```
SELECT u.id AS u__id FROM user u

string '1' (length=1)

SELECT u.id AS u__id, u.name AS u__name,
u.username AS u__username, u.password AS
u__password, u.created_at AS u__created_at,
u.updated_at AS u__updated_at FROM user u
WHERE u.id = ? LIMIT 1
(1)

string 'Foo' (length=3)

SELECT p.id AS p__id, p.phonenumber AS
p__phonenumber, p.user_id AS p__user_id FROM
phonenumber p WHERE p.user_id IN (?)
(1)

int 1

SELECT p.id AS p__id, p.phonenumber AS
p__phonenumber, p.user_id AS p__user_id FROM
phonenumber p WHERE p.user_id IN (?, ?)
(1, 2)

int 2
```


- load all properties you want via DQL joins
- load others on demand
- or load a specific property for all objects in a collection via loadRelated()

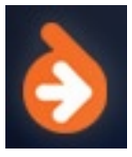
Yeah, we got that

DQL supports a good chunk of standard SQL features

```php
<?php $separator = "<br /><br />\n\n";
require_once('../setup.php');

Doctrine_Query::create()
    ->delete()
    ->from('User u')
    ->execute();

Doctrine_Query::create()
    ->update('User')
    ->set('name', '?', array('foobar'))
    ->where('username LIKE ?', array('_me%'))
    ->execute();

Doctrine_Query::create()
    ->select('u.id, ug.id, ug.name')
    ->from('User u LEFT JOIN u.Usergroup ug')
    ->where('MD5(password) = ?', array('foo'))
    ->execute(array(), Doctrine::HYDRATE_ARRAY);

Doctrine_Query::create()
    ->select('u.id, COUNT(*)')
    ->from('User u')
    ->groupBy('u.name')
    ->execute();

Doctrine_Query::create()
    ->select('u.*')
    ->from('User u')
    ->where('u.id NOT IN
        (SELECT u2.id FROM User u2
            INNER JOIN u2.Usergroup ug)')
    ->limit(10)
    ->execute();
```

```
DELETE FROM user

UPDATE user SET name = ? WHERE username
LIKE ?
(foobar, _me%)

SELECT u.id AS u__id, u2.id AS u2__id, u2.name
AS u2__name FROM user u LEFT JOIN usergroup
u2 ON u.id = u2.user_id WHERE MD5(u.password)
= ?
(foo)

SELECT u.id AS u__id, COUNT(*) AS u__0 FROM
user u GROUP BY u.name

SELECT u.id AS u__id, u.name AS u__name,
u.username AS u__username, u.password AS
u__password, u.created_at AS u__created_at,
u.updated_at AS u__updated_at FROM user u
WHERE u.id NOT IN (SELECT u2.id AS u2__id
FROM user u2 INNER JOIN usergroup u3 ON u2.id
= u3.user_id) LIMIT 10
```


- DQL is not really ideal for all too fancy reporting type queries
- However for these you often do not need hydration into objects anyways, so just fallback to plain SQL



Its always greener on ..

Doctrine DBAL (fork of PEAR::MDB2), to send native SQL and to execute DDL statements

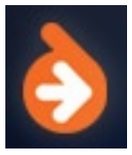
Different portability modes define what should be done

ORM can generate portable code and leverage more RDBMS specific features (ex. SUBQUERY with LIMIT, AS keyword, RDBMS specific identifier quoting)

A lot of SQL functions are abstracted (SUBSTRING() etc.), if not disable function portability (DAYOFWEEK())

100% portability is a myth

- Doctrine_Manager::connection()->execute($query, array())
- someone needs to step up and port all the recent improvements from PEAR::MDB2 to Doctrine
- different RDBMS require different syntax for LIMIT, MySQL does not support LIMIT in a subquery
- some RDBMS require/expect the AS keyword in other places (Oracle did not allow AS for column alias)
- many common functions just work, if the function is not yet abstracted, just disable portability mode
- some issues are simply due to the fact that error conditions are handled differently by each RDBMS



Cache the caching caches

ORMs always add some fixed overhead due to additional code, listeners and query generation

Cache DQL queries to remove DQL parsing overhead

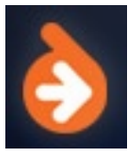
Cache query result sets to not have to query the RDBMS

Flexible container approach (SQLite, Memcache, APC)

Caching can be activated globally or per query

As a result the drawbacks of the ORM can be mitigated while still keeping all the convenience

– even when caching DQL there is still overhead and some parsing in order support callbacks



Deja vue, deja veu?

ORM allow centralizing key aspects of the data model

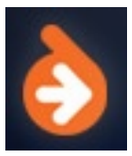
Listener hooks are one aspect of this (pre* methods)

Plugins/Templates easily add functionality to a given model class: I18n, Versionable, SoftDelete, Searchable Timestampable, NestedSet etc.

Other possible plugins: PessimisticLocking, other tree and inheritance algorithms etc.

Alternatively one can create new Doctrine_Record/ Doctrine_Query base classes to extend from

- There is no multiple inheritance in PHP
- Eventually a more sophisticated listener framework will make it possible to also handle DQL
- Alternatively PHP 5.4/6 might feature traits



A tree is a tReE is a TREE!?

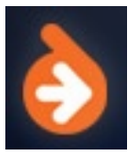
Using NestedSet behavior (other tree algos possible)

```php
<?php require_once('../setup.php');

$tree = Doctrine::getTable('User')->getTree();
$root = new User(); $root->name = 'root';
$tree->createRoot($root); // calls $root->save()

$record = new User();
$record->name = 'somenode';
// calls $record->save()
$record->getNode()->insertAsLastChildOf($root);

$isLeaf = $record->getNode()->isLeaf(); // isRoot()

foreach ($tree->fetchTree() as $node) {
    echo str_repeat('  ', $node['level']);
    echo $node['name'].'<br />';
}

// Use getNode()->delete() or tree may corrupt
$record->getNode()->delete();
```

```php
<?php

class User extends Doctrine_Record
{
    public function setTableDefinition()
    {
        $this->hasColumn('name', 'string',30);
    }

    public function setUp()
    {
        $this->actAs('NestedSet');
    }
}
```

```
INSERT INTO user (name, lft, rgt, level) VALUES (?, ?, ?, ?)
(root, 1, 2, 0)
UPDATE user SET lft = lft + ? WHERE lft >= ?
(2, 2)
UPDATE user SET rgt = rgt + ? WHERE rgt >= ?
(2, 2)
INSERT INTO user (name, level, lft, rgt) VALUES (?, ?, ?, ?)
(somenode, 1, 2, 3)
SELECT u.id AS u__id, u.name AS u__name, u.lft AS u__lft,
u.rgt AS u__rgt, u.level AS u__level FROM user u WHERE
u.lft >= ? ORDER BY u.lft ASC
(1)
root
  somenode
SELECT u.id AS u__id, u.name AS u__name, u.lft AS u__lft,
u.rgt AS u__rgt, u.level AS u__level FROM user u WHERE
(u.lft >= ? AND u.rgt <= ?)
(2, 3)
DELETE FROM user WHERE id = ?
(2)
UPDATE user SET lft = lft + ? WHERE lft >= ?
(-2, 4)
UPDATE user SET rgt = rgt + ? WHERE rgt >= ?
(-2, 4)
```


- adjacency list model (could use CONNECT BY/WITH RECURSIVE)
- materialized path



Setting the ground rules

Manage schema in yml files

```php
<?php $skipcreate = true; require_once('../setup.php'); cleandir('models');

$options = array(
    // Whether or not to generate abstract base models containing the
    // definition and a top level class which is empty extends the base
    'generateBaseClasses'    =>  true,
    // Whether or not to generate a table class for each model
    'generateTableClasses'   =>  false,
    // // Name of the folder to generate the base class definitions in
    'baseClassesDirectory'   =>  'generated',
    // // base Doctrine_Record class
    'baseClassName'          =>  'Doctrine_Record',
    // Extension for your generated models
    'suffix'                 =>  '.php'
);

Doctrine::generateModelsFromYaml('.', $models_path, $options);

Doctrine::createTablesFromModels($models_path);
```

```
CREATE TABLE phonenumber (id BIGINT AUTO_INCREMENT,
phonenumber TEXT, user_id BIGINT, INDEX user_id_idx
(user_id), PRIMARY KEY(id)) ENGINE = INNODB
CREATE TABLE user (id BIGINT AUTO_INCREMENT,
username TEXT, password TEXT, UNIQUE INDEX
name_index_idx (username(10) ASC), PRIMARY KEY(id))
DEFAULT CHARACTER SET utf8 COLLATE utf8_unicode_ci
ENGINE = INNODB
ALTER TABLE phonenumber ADD FOREIGN KEY (user_id)
REFERENCES user(id)
```

```yaml
---
detect_relations: true

User:
  columns:
    username: string
    password: string
  options:
    type: INNODB
    collate: utf8_unicode_ci
    charset: utf8
  indexes:
    name_index:
      fields:
        username:
          sorting: ASC
          length: 10
      type: unique

Phonenumber:
  columns:
    phonenumber: string
    user_id: integer
```


- Doctrine knows your FKs, so it will do all loading, dropping in the proper order
- Allows fancy stuff like Inheritance (well only one algo currently supported)

Where the wind blows

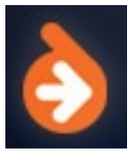
Migrations help in managing schema evolution

```php
<?php $skipcreate = true; require_once('../setup.php');

$migrations_dir = dirname(__FILE__).'/migrations';
$migration = new Doctrine_Migration($migrations_dir);

// Assume current version is 0
$migration->migrate(3); // takes you from 0 to 3
$migration->migrate(0); // takes you from 3 to 0

$migration->getCurrentVersion(); // 0
```

```php
<?php

// 001_add_table.class.php
class AddTable extends Doctrine_Migration
{
    public function up()
    {
        $params = array('field1' => array('type' => 'string'));
        $this->createTable('migration_test', $params);
    }

    public function down()
    {
        $this->dropTable('migration_test');
    }
}
```

```
CREATE TABLE migration_version (version INT) ENGINE = INNODB
SELECT version FROM migration_version
CREATE TABLE migration_test (field1 TEXT) ENGINE = INNODB
ALTER TABLE migration_test ADD field2 INT
ALTER TABLE migration_test CHANGE field1 field1 TEXT
SELECT version FROM migration_version
INSERT INTO migration_version (version) VALUES (3)
SELECT version FROM migration_version
ALTER TABLE migration_test CHANGE field1 field1 INT
ALTER TABLE migration_test DROP field2
DROP TABLE migration_test
SELECT version FROM migration_version
UPDATE migration_version SET version = 0
SELECT version FROM migration_version
```


- Obviously heavily inspired by Rails
- I think in most cases the down() method could be automated from the up()



Common wtf's

Really wants the id columns of all models and some times (not always) magically adds them to the query

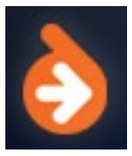
Due to hydration complex queries may need to be written in a specific order

Not all DB functions recognized in DQL

Some SQL constructs not supported in DQL

Significant OO overhead when not monitoring the generated queries

Fixtures do not work with multi column PKs



Not scared away yet?

Mailinglist for developers, users and svn commits

Trac with over 1000+ tickets (about 150 still open)

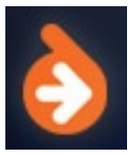
A defined roadmap (1.0 planned for September 1st)

Documentation for each minor version with examples and best practices (will get a major overhaul before 1.0)

Sandbox example project to get up and playing quickly

Integration with Symfony, Code Igniter, and ZF

Doctrine website already runs on Symfony 1.1RC1



Should you have cared?

Thank you for listening.

Lukas Smith (lukas@liip.ch)

http://pooteeweet.org/files/ipc08spring/doctrine_orm.pdf

http://pooteeweet.org/files/ipc08spring/doctrine_orm_code_examples.zip