Les frameworks de développement web : comment enrichir rapidement et efficacement le système d'information de votre établissement

Benjamin Ninassi, Simon Panay, Frédéric Saint-Marcel

Les frameworks de développement web

Comment enrichir rapidement et efficacement votre système d'information

INTRODUCTION

Outils métiers du système d'information :

Question posée : développements internes ?

Craintes vis-à-vis des développements internes :

- Chronophages
- Difficultés de maintenance – pérennité de la solution

Contexte : beaucoup de turn-over dans les environnements de recherche :

- Peu de réutilisation des développements précédents
- Pas de garantie d'homogénéité des applications en terme de technologies et d'interfaces

⇒ Nécessité d'avoir un cadre régissant les développements

PLAN

1. Présentation des frameworks web MVC
2. Retour d'expérience sur **Ruby on Rails** / OSC
3. Retour d'expérience sur **Symfony** / CONFACT
4. Retour d'expérience sur **Django** / REMI
5. Conclusion

1

Présentation des frameworks web

Framework Web MVC

- Un framework web = **CADRE DE TRAVAIL** avec des **BONNES PRATIQUES**
- Ne pas ré-inventer la roue (DRY : « Don't Repeat Yourself »)
- Boîte à outils/composants logiciels génériques
- Rapidité et qualité du développement
- Structuration du développement (Architecture Modèle/Vue/Contrôleur)
- Des fonctionnalités par défaut pour le développement web

Ruby on Rails

Django

Symfony

Architecture MVC

Design Pattern

- Le **M**odèle de données : abstraction de la base de données, en charge de l'accès et des requêtes
- La **V**ue : présentation des données (HTML, XML, JSON, RSS)
- Le **C**ontrôleur : implémentation de la logique métier, liant entre le **M**odèle et la **V**ue

« Structuration MVC d 'une application »

```
M = /app/models/*
V = /app/views/*
C= /app/controllers/*
```

RAILS

```
M = /app/models.py
V = /app/template/*
C = /app/views.py
```

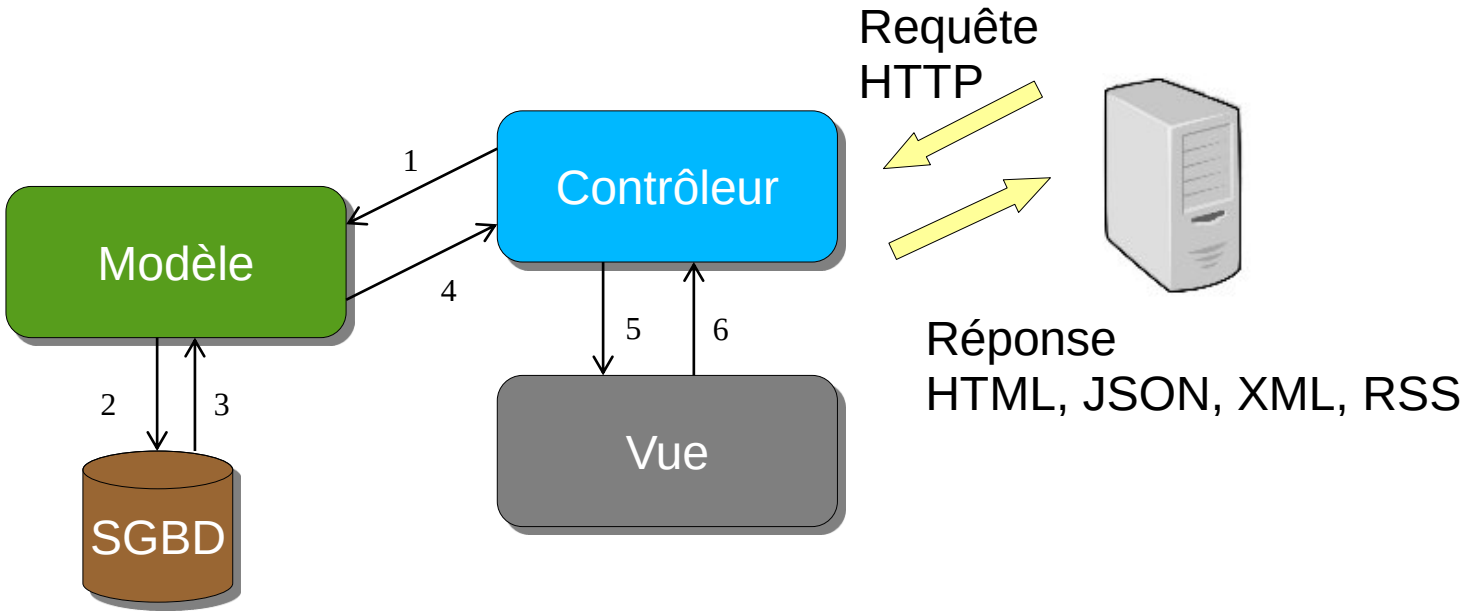
django

appelé aussi MTV framework ☺

```
M = /app/*Bundle/Entity/*
V = /app/*Bundle/Ressources/view/*
C = /app/*Bundle/Controller/*
```

Symfony

Architecture MVC



Modèle de données

Définition

- ORM
- Couche d'abstraction Objet-Relationnel
- Génération automatique de la base de donnée

Une classe objet ⇔une table de la base de données avec :

- Les types de données
- Les contraintes d'intégrité (ex :unicité d'un champ)
- Les relations entre les classes et leur cardinalité (ex : relation 1-n)

« Création d'un modèle de donnée Article pour les JRES avec une relation n-n avec le modèle Auteur »

RAILS

```
Class Article < ActiveRecord::Base
    has_many :authors
end
```

django

```
Class Article(models.Model):
  authors = models.ManyToManyField(Autthor)
```

Symfony

```
Class Article {
     /**
     * @ORM\ManyToMany(targetEntity="Author")
     */
    private $authors;
}
```

Modèle de données

Manipulation

L'ORM fournit des instructions de haut-niveau pour la manipulation des objets.

- Abstraction complète du SQL
- Utilisation de CRUD pour la persistance des données (Create, Read, Update, Delete)

« Création d'un Article depuis le contrôleur et instruction de recherche dans la base de donnée »

```
article = Article.new
article.name = "Framework"
article.save

article = Article.find_by_nom("Framework")
```

```
article = Article(name="Framework")
article.save()

Article = Article.objects.filter(name="Framework")
```

```
$article = new Articlet();
$article->setName('Framework');
$em = $this->getDoctrine()->getEntityManager();
$em->persist($article);
$em->flush();

$article = $em->getRepository('ArticleBundle:
  Article')->findByName('Framework');
```

Vue

Système de « template » Web :

- Variables interprétées dynamiquement
- Utilisation des instructions d'itération et de condition des langages de programmation
- Possibilité aussi de présenter des vues sous différents formats : JSON, XML, RSS, PDF.

« Affichage de la liste des Articles JRES avec le lien hypertexte correspondant dans une page HTML »

RAILS

```
<% for article in @articles %>
<%= link_to h(article.name),ligne_path(article) %>
<% end %>
```

django

```
{% for article in articles %}
  <a href="{{ article.get_absolute_url }}">
    {{ article.name }}
  </a>
{% endfor %}
```

Symfony

```
{% for article in articles %}
  <a href="{{ article.href }}">
    {{ article.name }}
  </a>
{% endfor %}
```

Gestion des URL(s)

« Une URL de la forme /article/id est envoyé vers la méthode show du contrôleur Article»

Système de « routage » :

- Correspondance d'une URL à une méthode ou action d'un contrôleur
- Basé sur des expressions régulières

RAILS

```
match "article/:id", :to => "article#show"
```

django

```
(r'^article/(P<id>\d+)/$', article.show')
```

Symfony

```
pattern: /article/{id}
  defaults: { _controller: ArticleBundle:Article:show }}
```

Contrôleur

- Découpage du contrôleur en méthodes (ou actions)
- Liant entre le modèle et la vue en fonction des évènements
- Connaissance du contexte HTTP (environnement, session, etc.)

« Le méthode show récupère l'article avec son id en paramètre et envoie le résultat à la vue »

```ruby
def show
    @article = Article.find(params[:id])
    respond_to do |format|
        format.html  # show.html.erb
    end
end
```

```python
def show(request,id):
  article = Article.objects.filter(id=id)
  return render_to_response('show.html', {'article':
article})
```

```php
public function show($id)  {
  $article = $this->getArticle($id);
  return $this->render('ArticleBundle:Article:
    show.html.twig', array('article' => $article));
}
```

Des fonctionnalités supplémentaires

Internationalisation

Gestion des formulaires

Gestion des tests unitaires

Gestion des sessions

API (REST,SOAP.etc.)

Authentification

Gestion de profils/droits utilisateurs

Export de données (csv, pdf, etc.)

Gestion du cache

Gestion de la sécurité (injection SQL, XSS, etc.)

Intégration de Jquery, Ajax

Et bien d'autres développées par les communautés !

SIG

Cycle de développement

- Création du modèle de données
  - Génération automatique de la classe
  - Écriture manuelle
  - Reverse engineering

  ➡ Génération automatique de la base de données
- Implémentation des actions de contrôleurs associées
  - Génération automatique des actions de base (CRUD)
  - Ecriture des actions complexes
- Implémentation des vues
- **Ecriture des tests unitaires**

➡ Particulièrement bien adaptés au développement itératif

Développement itératif

- Nombreux déploiements
- Nombreuses migrations de base de données
- Risque accru de régression

Automatisation d'un maximum de tâches

- Outils de migration automatique de la base de données
  - Ruby on rails : **rake** (avec ActiveRecord) ✔
  - Symfony : **Doctrine** ✔
  - Django : **syncdb** ✗

Automatisation du Déploiement

- Pourquoi ?
  - Nombreuses tâches à effectuer à la main sur différentes machines (prod, qualif, dev, bdd distantes, etc.)
  - Risques d'oubli / erreur de séquencement
  - Tâches relevant plus de l'administration système que du développement
- Comment ?
  - Objectif : un déploiement = une ligne de commande !
  - Identification des tâches redondantes + Administration centralisée
  - **Utilisation d'outils de déploiement automatique d'applications :**
    - Ruby on rails : **Capistrano** ✔
    - Django : **Fabric** (réalisé par la communauté)
    - Symfony : **Capifony** (réalisé par la communauté)

Non régression

- Écriture de tests unitaires - test d'une brique logicielle élémentaire (fonction)
- Suites de tests unitaires fournies facilitant la rédaction et l'exécution des tests :
  - Ruby on rails : **Test::Unit** ✔
  - Django : **unittest** ✔
  - Symfony : **phpunit** ✔

- Allez plus loin **:**
  - Besoin d'une base de connaissances commune entre développeurs
  - Couverture de code
  - Respect des conventions de codage

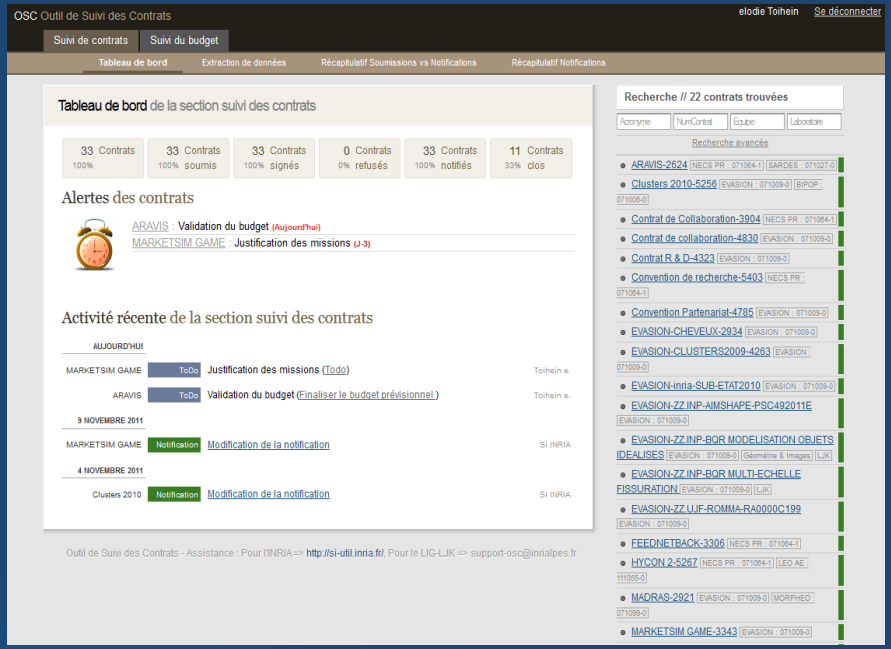
  ⟹ Mise en place d'une plateforme d'intégration continue + plugins

  Ex : **Jenkins** + **django-jenkins**

2

Ruby on Rails

OSC/OSB : Outil de Suivi des Contrats / Outil de Suivi des Budgets d'une équipe de recherche



Outil de Suivi des Contrats (OSC)

- Application métier de gestion réalisée en **Ruby on Rails**
- Comprenant de nombreux processus fonctionnels :
  - Gestion des budgets
  - Gestion des contrats
- Plateforme commune utilisée par :
  - INRIA
  - Laboratoire Informatique de Grenoble (LIG)
  - Laboratoire Jean-Kuntzman (LJK)
- Interconnectée avec les différents systèmes d'informations :
  - Authentification multiple en fonction de l'établissement de rattachement
  - API REST d'extraction de données
  - Tâches de synchronisation nocturnes avec le système d'information d'INRIA
- Open source distribué sous licence LGPL

Pourquoi Ruby on Rails ?

- Framework web le plus mature à l'époque (2006)
- Reprise de la solution ✔ :
  - Absence de documentation technique contrebalancée par la documentation disponible en ligne
  - Langage Ruby : syntaxe « élégante »
- Développement collaboratif ✔ :
  - Structuration du code => segmentation des développements
  - Templating (ActiveView) => homogénéité des interfaces utilisateurs
- Rapidité des développements ✔ :
  - Mécanisme des « gems » pour l'utilisation des librairies supplémentaires (simplicité de gestion)
  - Puissance et maturité de l'outil de migration de bases de données (`rake db:migrate`)
  - Outils de création de tâches d'administration puissants : **rake**

Les difficultés rencontrées

- Compétences en Ruby rares ✗
- Disponibilité des versions de Ruby selon les OS ✗
  - Une solution existe maintenant : RVM (Ruby Version Manager)
- Hélas reprise d'une application sans tests unitaires ✗
  - Aucune visibilité sur les régressions possibles
  - Difficultés pour mettre à jour le framework (version utilisée 2.1 , version actuelle de RoR 3.1)
  - Compatibilité ascendante limitée de Ruby
  - Compatibilité des « gem » entre les versions, création de dépendances externes

3

Symfony – PHP

Confact : un outil de gestion des actes de conférences

Outil de gestion des actes de conférences

Développement local au centre INRIA de Grenoble Rhône Alpes :

- Simplification de la procédure d'archivage d'un ensemble de documents (compression, dépôt sur le système de stockage, etc.)
- Ajout de métadonnées (formulaires d'éditions)
- Interfaces de visualisation des informations (filtres, tris, etc.)
- Fonctionnalité d'export csv des informations
- A l'occasion d'un stage de licence, présent pour 2 mois

Pourquoi Symfony ?

- Framework en PHP ✔ :
  - Compétence très répandue

    Partagée par le stagiaire et l'équipe chargée de la reprise
- Communauté très développée ✔ :
  - Nombreux tutoriaux disponibles (autoformation possible)
  - Pérennité de la solution
  - Documentation très complète (documentation technique = documentation du framework)

Les difficultés rencontrées

- Mise en garde : Apprentissage du framework non négligeable
  - Chronophage pour une petite application ✗
  - Mais double capitalisation : montée en compétence + reprise de la solution ✔

- Reprise de la solution :
  - Structuration du code en Symfony 1.4 pas intuitive :
    - Plus simple en Ruby on rails ✗
    - Défaut corrigé dans Symfony 2 ✔
  - Hélas développée sans suivre les bonnes pratiques : code métier dans les vues ✗

4

Django – Python

REMI : REférentiel des Moyens Informatiques

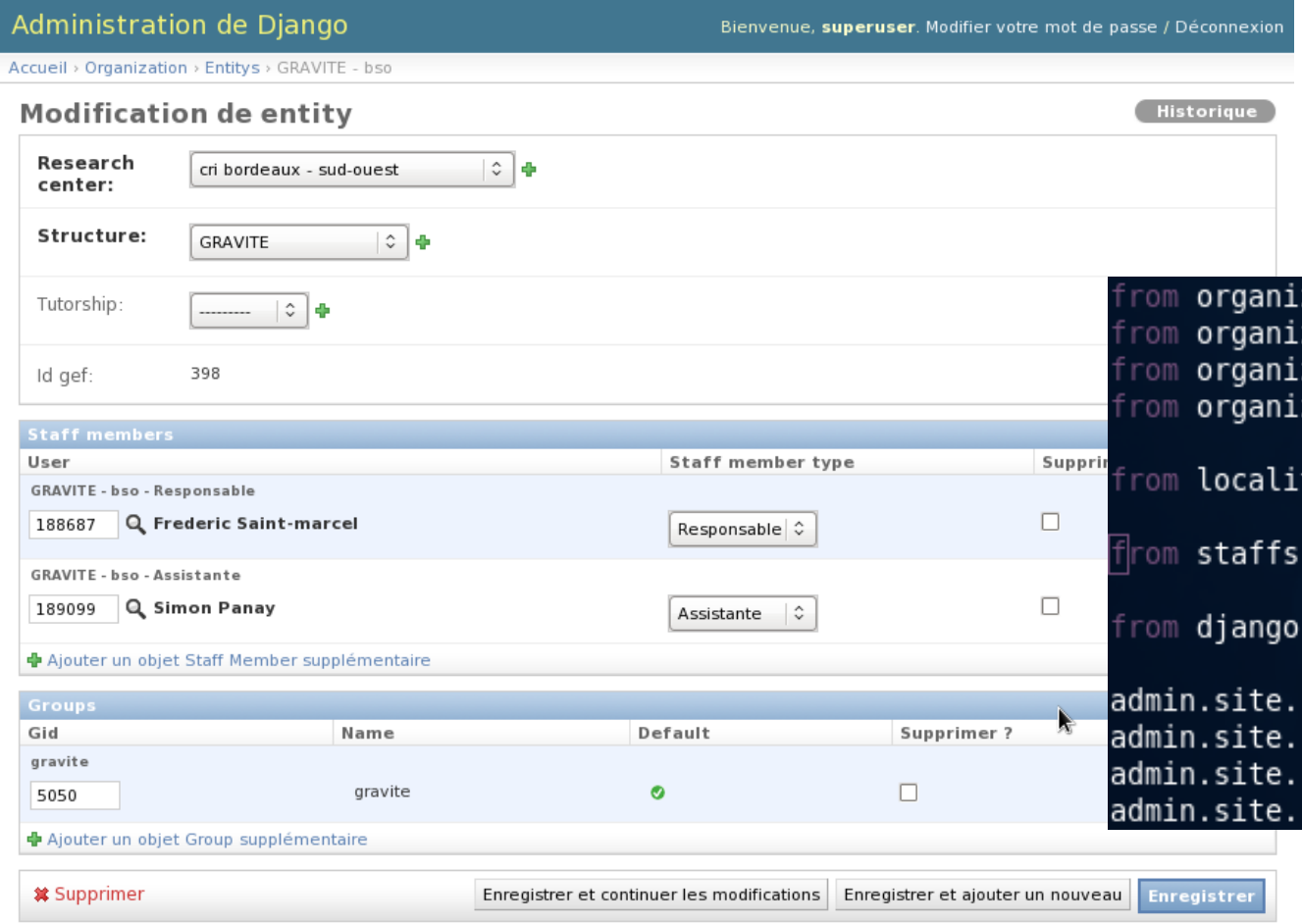
RÉferentiel utilisateur des Moyens Informatiques

- Application réalisée en **Django** (framework python)
- Gestion et uniformisation des comptes informatiques des 8 centres INRIA
- Interconnectée avec les différents systèmes d'informations :
  - Import d'informations du SI RH (via notifications XML et API REST)
  - Population des annuaires métiers de l'INRIA (ActiveDirectory, Ldap)
- Périmètre du projet appelé à s'élargir :
  - Connecteur suite collaborative Zimbra
  - Alimentation des listes de diffusion Sympa
  - Gestion des espaces de données

Pourquoi Django ?

- Fait suite à une application du centre de Grenoble en Django ✔
- Langage python : Réutilisation de scripts écrits par des ingénieurs systèmes ✔
- Interface d'administration auto-générée ✔ :
  - Pas une ligne de code écrite pour les Vues et les Contrôleurs :

    ➡ pas de html, css, javascript, validateur, etc.
  - Focalisation sur la partie Modèle (de données)



Interface d'administration auto-générée par Django

```
from organization.models import Tutorship
from organization.models import Structure
from organization.models import Entity
from organization.models import Group

from localities.models import ResearchCenter

from staffs.admin import StaffMemberInline

from django.contrib import admin

admin.site.register(Tutorship)
admin.site.register(Structure)
admin.site.register(Entity)
admin.site.register(Group)
```

Difficultés rencontrées

- Migrations de bases de données :
  - Utilitaire fourni de base avec Django pas assez puissant (pas de migrations de données) ✗

  ➡ Utilisation de **South** (http://south.aeracode.org/)
- Déploiements :
  - Pas d'outils de déploiement livré par défaut ✗

  ➡ Utilisation de **Fabric** (http://docs.fabfile.org)
    - Outil python en ligne de commande pour des tâches d'administration à travers SSH
    - Nécessite néanmoins l'écriture de procédures de déploiements

5
En Conclusion

Utilité des frameworks web

- Les frameworks apportent ✔ :
  - Rapidité des développements
  - Moins de documentation technique nécessaire
  - Focalisation sur le code métier
  - Facilité de reprise des projets
  - Mécanismes de sécurité
  - Correctifs réguliers
- Mais ils contraignent ✗ :
  - Temps d'apprentissage du framework
  - Chaine de production encore pas complètement intégrée
  - Perte d'indépendance
  - Mises à jour nécessaires

Quel framework choisir ?

- Similitude des principaux frameworks :
  - Fonctionnalités (paradigme MVC, ORM, templating, etc...)
  - Communauté (pérennité, correction des failles de sécurité, documentation)
- Nos autres critères de choix : Langage, contexte et compétences de l'équipe de développement

Choix de l'équipe DSI - SEISM : Symfony 2

- Recommandations :
  - Suivre les bonnes pratiques de développement (tests unitaires, normalisation de la base de données)
  - Développement itératif (Abandon du cycle en V, mise en place d'une chaîne de production, etc.)
  - Mutualiser les efforts en choisissant un seul framework pour tous les développements

Merci

JRES 2011
TOULOUSE

Utilisation de South pour un projet Django

- Modification du modèle de données :

  > python manage.py schemamigration <application>

- Génération d'un fichier décrivant les changements à effectuer dans la base de données :

  > python manage.py migrate <application>

- Applique la migration sur la base de données
- Historisation des migrations (rollback possible)
- Détection et application automatique des migrations sur les autres plateformes à l'aide de la commande *migrate*

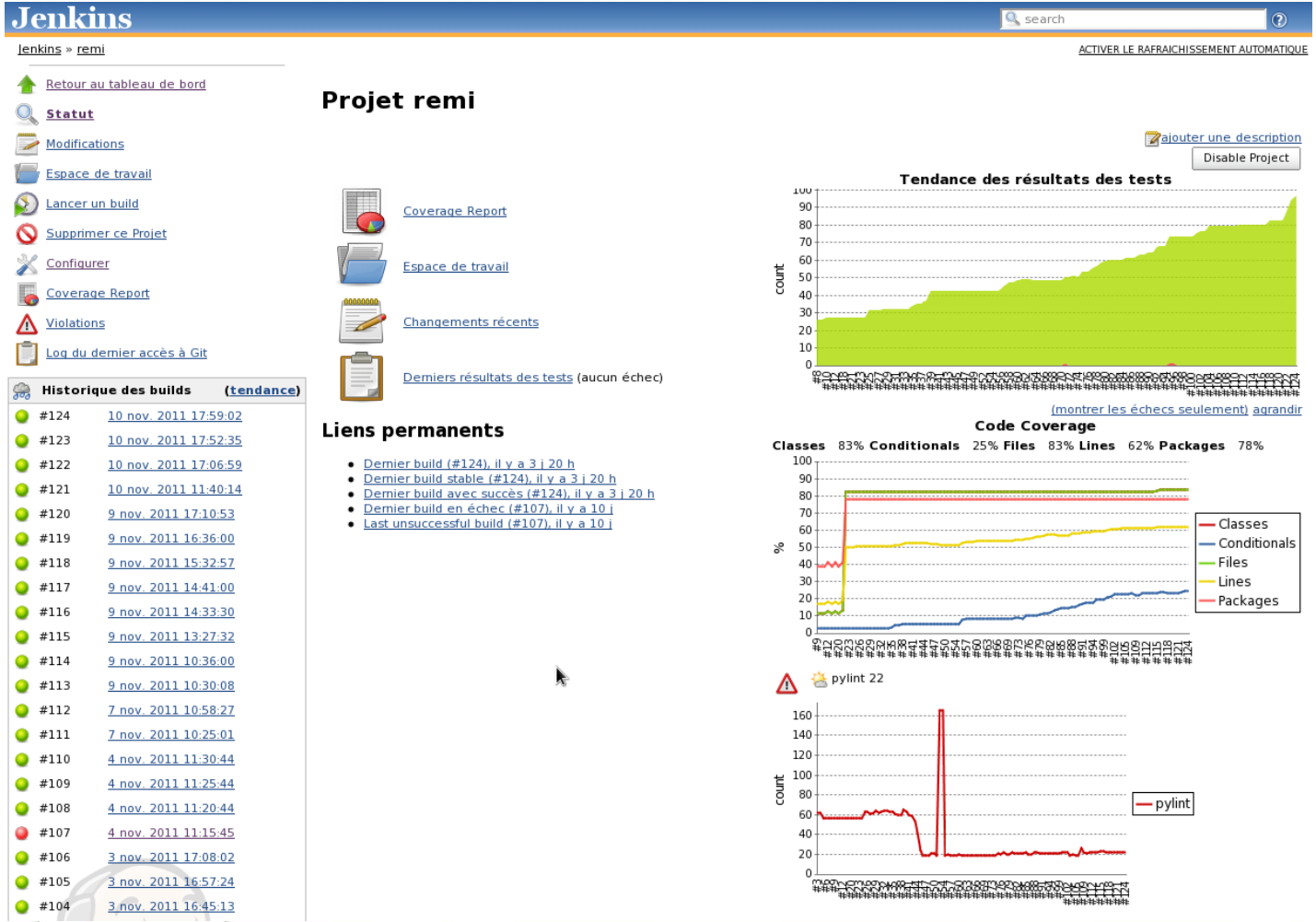
Gestion des régressions dans REMI

- Identifier les régressions ! (« On a le droit de tout péter, mais il faut le savoir !»)
- Écriture de tests unitaires
- Django possède une suite de tests unitaires, mais :
  - Pas de visibilité sur leur couverture
  - Pas de vision partagée de la qualité du code entre développeurs
- Besoin d'une base de savoir commune

Jenkins (Plateforme d'intégration continue) + **django-jenkins**
(mise en forme de résultats des tests unitaires Django pour Jenkins)

Annexe

Visible par tous les développeurs



Annexe

Couverture incomplète des tests unitaires

```
152  |
153 1|    def _compare(self, new_group, old_group, excluded_keys):
154 0|        first_dict, second_dict = new_group.__dict__, old_group.__dict__
155 0|        comparison_result = {}
156 0|        for key, value in first_dict.items():
157 0|            if key in excluded_keys:
158 0|                continue
159 0|            try:
160 0|                if value != second_dict[key]:
161 0|                    field_diff = {}
162 0|                    field_diff.update(old=second_dict[key], new=value)
163 0|                    comparison_result.update({key: field_diff})
164 0|            except KeyError:
165 0|                continue
166 0|        return comparison_result
167  |
168 1|    def get_related_directories(self, visibility):
169 1|        if visibility == 'nationale':
170 1|            return Directory.objects.filter(directory_range__id=self.entity.research_center.id)
171  |        else:
172 1|            return Directory.objects.annotate(ranges=Count('directory_range')).filter(directory_range__id=self.entity.research_center.id, ranges
173  |
174 1|    def get_directories(self):
175 0|        return Directory.objects.filter(directory_range__id=self.entity.research_center.id)
176  |
177 1|    def generate_new_gid(self):
178 1|        return Group.objects.order_by('-gid')[0].gid + 1
179  |
180 1|    def create_subscription(self, directory):
181 1|        if directory.is_ldap():
182 1|            entry = LdapDirectoryGroupEntry(ldap_directory=directory.ldapdirectory, group=self)
183  |        else:
184 1|            entry = ActiveDirectoryGroupEntry(active_directory=directory.activedirectory, group=self)
185 1|        entry.save()
186  |
187 1|    def create_subscriptions(self):
188 0|        for directory in self.get_directories():
189 0|            self.create_subscription(directory)
190  |
191 1|    def add_subscriptions(self, visibility):
192 1|        for directory in self.get_related_directories(visibility):
193 1|            self.create_subscription(directory)
194  |
195 1|    def get_ldap_subscriptions(self):
196 1|        return LdapDirectoryGroupEntry.objects.filter(group=self)
197  |
198 1|    def get_ad_subscriptions(self):
199 1|        return ActiveDirectoryGroupEntry.objects.filter(group=self)
200  |
201 1|class Membership(models.Model):
202 1|    user = models.ForeignKey('persons.User')
203 1|    group = models.ForeignKey(Group)
204 1|    primary = models.BooleanField()
205  |
206 1|    class Meta:
207 1|        unique_together = ('user','group')
```

Annexe

Violation des conventions de codage

organization/management/commands/update_staffs.py

pylint	3 violations
18	Too many branches (14/12)
18	Too many local variables (19/15)
37	Unused variable 'created'

File: update_staffs.py Lines 9 to 47

```
 9
10 from persons.models import User
11
12 from organization.models import Entity
13 from organization.models import Structure
14
15 class Command(NoArgsCommand):
16     help = u'Update staff from GEF Notifications'
17
18     def handle_noargs(self, **options):
19         notifications_directory = settings.GEF_NOTIFICATIONS_DIR
20         notifications_list = os.listdir(notifications_directory)
21         notifications_list.sort()
22
23         for notification in notifications_list:
24             if re.match("NotificationStructureMI_", notification):
25                 dom = parse("%s%s" % (notifications_directory, notification))
26
27                 if dom.getElementsByTagName('idStructure'):
28                     inria_id = dom.getElementsByTagName("idStructure")[0].firstChild.nodeValue
29                     gef_id = int(dom.getElementsByTagName("idGef")[0].firstChild.nodeValue)
30                     name = sanitize_structure_name(dom.getElementsByTagName("nom")[0].firstChild.nodeValue)
31                     errors = 0
32
33                     try:
34                         entity = Entity.objects.get(id_gef=gef_id)
35                         structure = entity.structure
36                     except Entity.DoesNotExist:
37                         structure, created = Structure.objects.get_or_create(name=name, functionnal=False)
38                         structure.save()
39
40                         entity = Entity(structure=structure, id_gef=gef_id)
41                         entity.get_research_center_with_gef()
42                         entity.save()
```

Gestion des déploiements

- Phases critiques du projet :
  - Nombreuses tâches à effectuer manuellement sur plusieurs machines (perte de temps)
  - Risques d'oubli / d'hétérogénéité
  - Le chaos n'est jamais loin
- Automatiser les déploiements
  - Ne pas buter sur des tâches d'administrations simples
  - Identification des sources d'erreur et corrections en conséquence

  ➡ **Fabric** (outil python en ligne de commande pour effectuer des tâches d'administration à travers SSH)

Gestion des déploiements sous Django: Fabric

- Identification des plateformes à cibler
- Identification de toutes tâches redondantes lors d'un déploiement :
  - Installation et mise à jour des paquets nécessaires
  - Checkout du dépôt de gestion de sources
  - Copie des fichiers de configuration
  - Redémarrage du serveur web
  - ...
- Factorisation de ces tâches sous forme de fonctions simplistes
- Écriture de procédures de déploiements
- Un déploiement = UNE ligne de commande :

```
> fab production deploy_and_migrate
```

Contexte du développement d'OSC

- Développée en 1 an par un CDD entre 2006 et 2007
- Prototype devenu production en 2007
- Départ du développeur initial en 2008 :
  - Aucune documentation technique
  - Pas de transmission de compétences avec l'équipe d'exploitation actuelle
  - Pas de compétences interne en Ruby, encore moins en Ruby on Rails
- Reprise de la solution depuis 2008 par une collaboration INRIA/LIG :
  - Maintenance corrective (correction des bugs)
  - Maintenance évolutive (mise à jour du framework)
  - Evolutions itératives de l'application (ajouts de fonctionnalités)
- Maintenance et exploitation de la plateforme : INRIA

OSC en quelques chiffres

- 650+ utilisateurs
- 450+ structures
- 5000+ contrats
- Plusieurs centaines de milliers d'entrées budgétaires
- Plusieurs dizaines d'interfaces
- Plusieurs dizaines de milliers de lignes de codes réparties entre plusieurs centaines de fichiers
- Des modèles de données complexes avec 66 tables interconnectées pour 722 champs