

2013

# Rapport de stage

Maintenabilité et évolution d'une  
solution informatique



Erik Aouizerate

05/09/2013





# Remerciements

Je tiens tout d'abord à remercier mon maître de stage, Julien Cognet, pour sa présence et les savoirs qu'il m'a transmis. L'équipe de la TMA, pour son accueil, son expérience et ses conseils avisés.

Je remercie ensuite la société CGI pour m'avoir pris en tant que stagiaire et offert une opportunité de mettre en pratique mes connaissances.

Je remercie la faculté Pierre Mendès France de m'avoir accueilli pendant 4 ans et plus particulièrement tous les professeurs que j'ai eu et qui m'ont transmis l'envie d'apprendre et la curiosité scientifique.

Je remercie aussi tout particulièrement mes camarades de classe pour les échanges et le partage naturel de nos connaissances.

Enfin, je remercie également mes proches qui m'ont aidé à mener à bien ces différents projets.

## Sommaire

Remerciements.....	3
A. Introduction.....	5
B. Cadre de travail.....	6
1. CGI - ST Microelectronics.....	6
2. L'équipe.....	7
3. La Tierce Maintenance Applicative (TMA) .....	8
4. Fonctionnement et gestion de projet.....	8
a. Méthode agile .....	8
b. Gestionnaire de sources – svn.....	9
5. EWip .....	9
a. Principe.....	9
b. Architecture.....	10
6. Paramétrages.....	12
C. Mon stage au sein de la TMA .....	13
1. Problématique .....	13
2. Planning .....	13
3. Les tests et l'évolution des applications.....	15
a. Les tests et la méthode TDD.....	15
b. Plateforme d'intégration continue.....	17
4. Conceptions et évolutions .....	18
a. Architecture.....	18
b. Langages.....	20
5. Développements et réalisations.....	22
a. Smart test .....	22
b. Report admin .....	26
D. Conclusion .....	30
E. Glossaire.....	31

## A. Introduction

Dans le cadre de mon Master 2 à l'UPMF, j'ai effectué mon stage de fin d'études dans la société CGI pendant une période de 5 mois.

Ce rapport a pour objectif de vous faire découvrir les problématiques auxquelles j'ai été confronté et la manière dont j'y ai répondu.

J'ai essayé de contextualiser de la façon la plus pertinente les différents développements et conceptions que j'ai dû réaliser afin d'offrir une vue globale et des connaissances réutilisables.

Le fil rouge de mon stage est la généricité des programmes et des algorithmes. Ce thème est assez récurrent en informatique car c'est en réalité un des principes de base. La difficulté est d'arriver à étendre cette généricité afin de couvrir le plus grand nombre de configurations.

Un des paradoxes qui réside dans ce type de conception est que plus le modèle est généraliste, plus il est difficile d'intégrer de nouveaux cas d'utilisations. Il est tout de même possible d'ajouter des fonctionnalités mais elles ont souvent pour conséquence d'alourdir et de complexifier le paradigme mis en place.

J'ai donc été amené à réfléchir sur les fondements de la généricité et mon travail s'est axé vers des recherches au niveau des modèles de données, de l'architecture matérielle et de l'architecture logicielle d'une solution informatique.

J'espère que ce rapport vous permettra de comprendre un choix de structure permettant de faciliter la maintenance et l'évolutivité d'un programme.

## B. Cadre de travail

### 1. CGI - ST Microelectronics

J'ai effectué mon stage de fin d'études chez CGI. Cette entreprise est une Société de Services en Ingénierie Informatique (SSII) et j'ai effectué ma mission pour le client ST Microelectronics (ST) sur le site de Crolles.

CGI est un groupe québécois spécialisé dans le domaine des Technologies de l'Information et de la Communication (TIC). Il a été créé en 1976 à l'initiative de Serge Godin. En août 2012, CGI achète la SSII Logica et accroît ainsi son implémentation en France et en Europe de manière générale. L'agence CGI de Grenoble cherche à travailler en priorité avec des acteurs locaux en offrant divers pôles de compétences. Les types de prestations offertes sont des Tierces-Maintenances Applicatives (TMA), Machin To Machin (M2M), Automation ou encore le développement de solutions mobiles.

La société STMicroelectronics est née en juin 1987 du regroupement de Thomson semi-conducteurs (France) et de SGS Microelectronica (Italie).

STMicroelectronics est un fabricant mondial de semi-conducteurs. La société conçoit, développe, fabrique et commercialise une vaste gamme de circuits intégrés et de composants pour des clients grands comptes tel que IBM, Western Digital, Nokia, Philips, Sony, Samsung, Alcatel, ou Motorola. Aujourd'hui au 5ème rang mondial des fabricants de semi-conducteurs avec un chiffre d'affaire de plus de 8 000 millions de dollars, STMicroelectronics se positionne comme leader européen sur le marché des semi-conducteurs.

## 2. L'équipe

L'équipe de TMA est composée de 10 personnes.

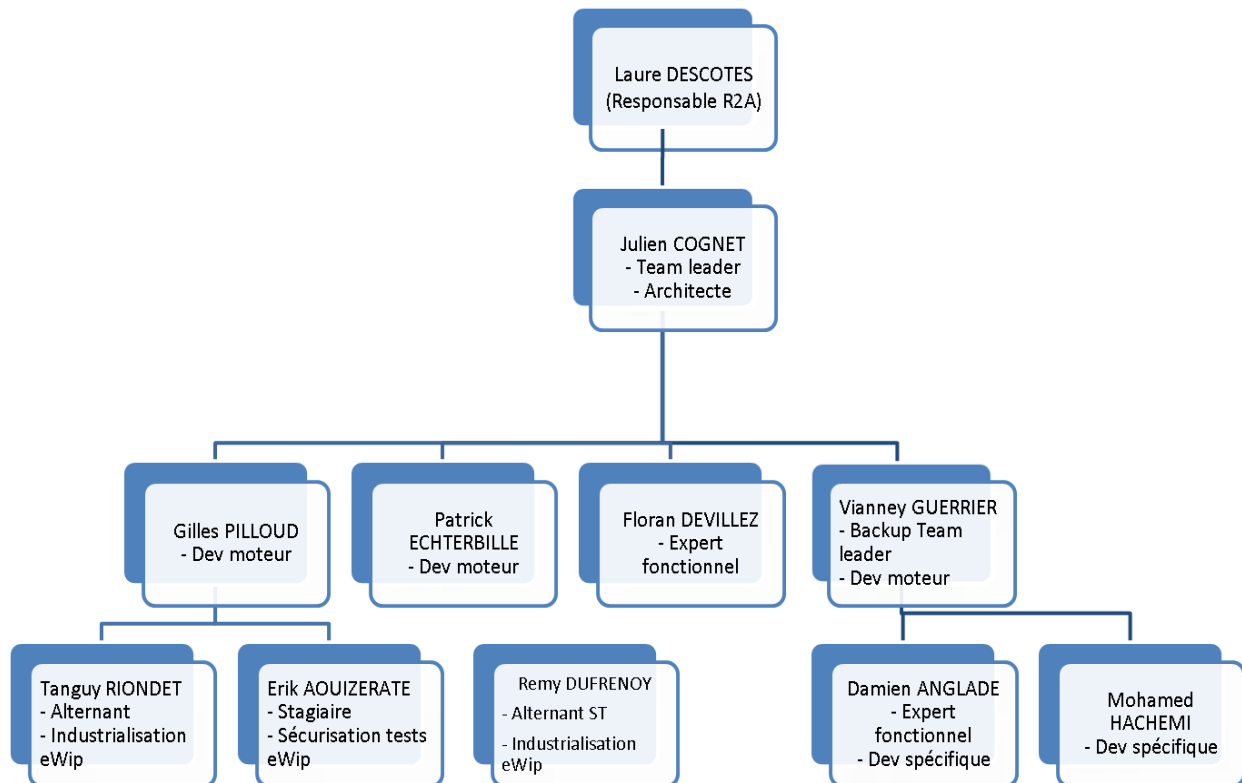


Figure 1: Organigramme de l'équipe

Le rôle de chacun et le fonctionnement de l'équipe est assez modulaire. En effet, en fonction des besoins toute l'équipe peut être affectée à une seule tâche. On peut cependant distinguer les profils techniques et de ceux qui sont fonctionnels.

Les informaticiens fonctionnels vont d'avantage interagir avec les utilisateurs pour recueillir leurs besoins et rédiger les documents de spécifications.

Les informaticiens techniques vont s'occuper des développements et de la programmation.

Au cours de mon stage, il y a eu des mouvements dans l'équipe : Patrick Echterbille a quitté la TMA et Tanguy Riondet a terminé son alternance.

### 3. La Tierce Maintenance Applicative (TMA)

La TMA a été mise en place pour développer et assurer la maintenance du logiciel *eWip*. Celui-ci permet de faciliter le bon fonctionnement des laboratoires.

Le contrat entre CGI et ST est valable pour 3 ans. A l'issue de cette période, un nouvel appel d'offre est créé. Enfin, tout ce qui est créé dans le cadre de l'appel d'offre est la propriété de ST.

La TMA est chargée de s'occuper des laboratoires Manufactures et Recherche & Développement (R&D).

Pour chaque laboratoire, on dispose de 3 plateformes :

- Développement. Elle est utilisée pour créer les applications et les tester. On effectue aussi des essais avec les utilisateurs finaux pour vérifier que le développement correspond aux besoins.
- Intégration. Elle est utilisée pour simuler une migration de logiciel afin de faire une mise en production la plus efficace possible.
- Production. C'est la plateforme qui est utilisée 7 jours sur 7 et 24 heures sur 24 pour faire fonctionner l'entreprise.

Dans le cadre *d'eWip*, de fortes connaissances « métiers » sont nécessaires afin de comprendre les besoins des utilisateurs. Ce domaine requiert donc un temps d'adaptation pour l'appréhender au mieux et en saisir ses contraintes.

### 4. Fonctionnement et gestion de projet

#### a. Méthode agile

Au sein de la TMA, nous utilisons la méthode agile afin d'offrir plus de souplesse au fonctionnement de l'équipe. Pour s'assurer du bon déroulement du projet, nous organisons de façon quotidienne un *stand-up*. C'est une réunion de 5 minutes où chaque personne de l'équipe expose succinctement son travail de la veille et annonce son objectif pour la journée. On décide aussi des affectations de tâches en fonctions des besoins et des difficultés que chacun a pu rencontrer.

Cette méthode permet une plus grande modularité et améliore l'efficacité du groupe. Cependant, le cadre de notre équipe nécessite d'avoir un chef de projet qui a, à la fois une



vision globale du travail de l'équipe et qui est en même temps capable de comprendre les problèmes techniques.

### **b. Gestionnaire de sources – svn**

Afin de gérer au mieux nos projets et nos développements, nous utilisons un gestionnaire de sources. Celui-ci est très utile mais n'est parfois pas adapté à notre environnement de travail. En effet, une partie de notre développement consiste à créer des tables en base de données et ce type d'opération n'est pas sauvegardé directement dans le gestionnaire. Afin de pouvoir recréer ces tables nous sauvegardons les scripts SQL et nous les exécutons lors des migrations. De plus, nous travaillons sur 6 environnements différents ce qui oblige à une certaine rigueur.

## **5. EWip**

*EWip* est une solution permettant d'implémenter une application web sans ou très peu de programmation. Grâce à des écrans de paramétrages, le développeur peut configurer son application et se soucier uniquement des contraintes métiers. Toute la partie technique est déjà programmée et est générée à la volée.

### **a. Principe**

Ce type de solutions est particulièrement adapté aux différentes contraintes de la TMA. En effet, pour CGI elle permet de répondre aux besoins du client à moindre coût et pour ST, les durées de projet sont très courtes et permettent une grande réactivité. De plus, les applications ont toutes la même charte graphique et permettent d'avoir un rendu homogène et ergonomiquement correct.

EWip permet d'implémenter facilement :

- Un modèle de données relationnel d'application : objets avec attributs et relations entre ces objets
- Des *workflows* permettant de gérer des cas d'utilisations d'une application
- Des groupes hiérarchiques sécurisés, des rôles d'utilisateurs et de privilèges pour ceux-ci
- Une bibliothèque de fonctionnalités enrichie au fil des releases
- Des rapports personnalisables
- Des statistiques personnalisables
- Des outils de recherches avancées
- Une interface de programmations SOAP (protocole d'appel de procédure distante)

## b. Architecture

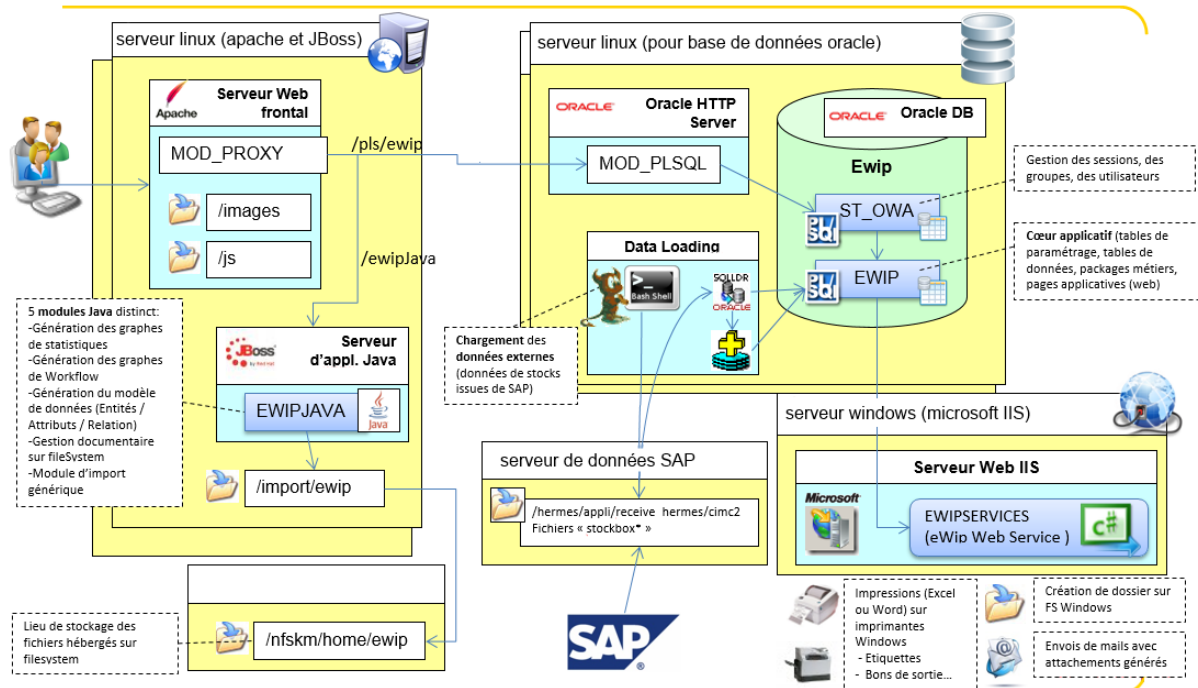


Figure 2 : Architecture globale d'eWip

Le schéma ci-dessus représente l'architecture globale d'eWip. Elle est basée sur plusieurs technologies rendant son fonctionnement complexe.

On peut distinguer les grandes entités suivantes :

### Le serveur Oracle

Ce serveur physique *linux* héberge deux serveurs logiciels : *oracle DB* et *oracle HTTP server* et un module de *data loading* (chargement de données) pour des données provenant de l'ERP SAP.

*Oracle HTTP Server* fournit un écouteur HTTP et sert pour l'hébergement des pages web statiques, dynamiques et des applications sur le Web.

*Oracle DB* est le serveur de base de données relationnel où sont traitées et stockées les données des différentes applications d'eWip. Le serveur héberge également une base de données appelée *ST\_OWA* qui permet de gérer les sessions, les groupes et les utilisateurs d'eWip.

### Le serveur IIS

Ce serveur physique *Windows* héberge un serveur logiciel : *Microsoft IIS*.

Le serveur Internet Information Services, communément appelé IIS, est le logiciel de serveurs services Web (ou FTP, SMTP, HTTP etc.) de la plateforme Windows NT. Il héberge des *webservices* développés en *C#* qui permettent de gérer les impressions *Excel* et *Word*, l'enregistrement de documents sur un système de fichiers *Windows* ou l'envoi de mails avec

des attachements générés. Le choix de *C#* plutôt que l'utilisation de *Java* est purement pratique. En effet, les bibliothèques disponibles en *C#* sont plus performantes dans les cas de traitements à effectuer dans les environnements *Microsoft* que *Java* et permettent une meilleure maintenance des applications.

### *Le serveur apache*

Ce serveur physique *Linux* héberge deux serveurs logiciels : *Apache* et *JBoss*. *Apache HTTP Server (Apache)* est un serveur HTTP libre créé et maintenu au sein de la fondation *Apache*. Il est utilisé pour afficher les pages d'*eWip* dans le navigateur des utilisateurs et pour stocker les images ainsi que les scripts *JavaScript*. *JBoss Application Server* est un serveur d'applications *Java EE* Libre entièrement écrit en *Java*, publié sous licence GNU GPL. Il utilise la machine virtuelle *Java (JVM)* pour exécuter les applications *Java* déployées dans son architecture. Ce serveur contient plusieurs applications essentielles à *eWip* pour afficher les graphiques statistiques, les graphiques des *workflows* ainsi que le modèle de données. Plus important encore, il gère le stockage documentaire sur le système de fichier *Linux*.

### *Le serveur SAP et le data loading*

SAP (Systems, Applications and Products for data processing) est par abus de langage le nom utilisé pour désigner un ERP développé et commercialisé par l'éditeur de ce produit (SAP AG).

Un ERP est un progiciel qui intègre les principales composantes fonctionnelles de l'entreprise: gestion de production, gestion commerciale, logistique, ressources humaines, comptabilité, contrôle de gestion. À l'aide de ce système unifié, les utilisateurs de différents métiers travaillent dans un environnement applicatif identique qui repose sur une base de données unique. Ce modèle permet d'assurer l'intégrité des données, la non-redondance de l'information, ainsi que la réduction des temps de traitement.

Ce serveur est un serveur de données SAP mis en place dans le but de donner à *eWip* la capacité de charger des données à partir de l'ERP pour remplir la base de données.

## 6. Paramétrages

Voici un schéma représentant les principales instances d'un paramétrage d'eWip.

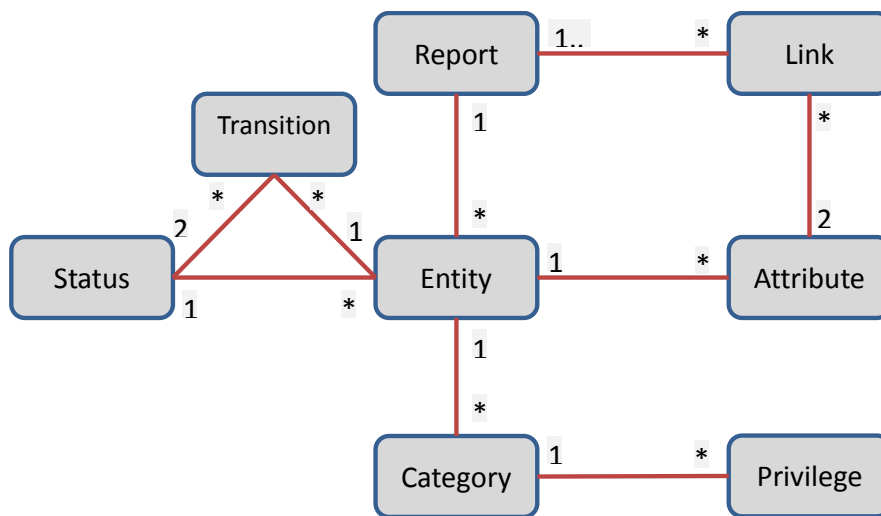


Figure 3 : diagramme de classes d'eWip

### Exemple de paramétrage

Les applications sont créées sur un modèle nommé entité. Chaque entité a plusieurs attributs qui la caractérisent.

Les données créées en suivant le modèle de l'entité sont nommées analyses. Si l'on prend l'exemple d'un *workflow* qui gère le processus de réparation d'une voiture, l'objet voiture sera une entité et ses attributs seront son numéro d'immatriculation, son nombre de chevaux, sa couleur, etc. Une Renault bleu de 120 chevaux est une analyse.

Ensuite nous allons attribuer à cette entité un statut (nouveau, à traiter, etc...) pour la faire entrer dans le *workflow*. Ainsi, une voiture arrivant dans le garage se verra attribuer un statut particulier : « en panne », par exemple.

Maintenant que notre analyse est créée et qu'un statut lui a été attribué, des acteurs vont devoir intervenir sur cette analyse afin de changer son statut *via* une transition. Ces *users* ont un rôle et des privilèges afin de limiter leurs actions.

Enfin pour visualiser les entités on dispose des catégories qui permettent d'afficher tous les attributs d'une entité.

Comme décrit précédemment, eWip se configure grâce à des écrans de paramétrages. La combinatoire possible pour le paramétrage d'une application eWip est assez grande. Par conséquent, les écrans de configuration se complexifient au fil des *releases*.

## C. Mon stage au sein de la TMA

### 1. Problématique

La solution mise en place par la TMA propose un grand nombre de possibilités pour paramétrer une application. Comme dit précédemment, la combinatoire possible augmente au fil des améliorations ce qui a pour conséquence de complexifier le logiciel *eWip*.

Afin de toujours assurer une qualité de rendu et de services, il est nécessaire de tester le moteur *eWip* à chaque *release*. Il est donc de plus en plus long d'effectuer ces tests ce qui impacte le coût du projet.

D'autre part l'architecture actuelle ne permet de distinguer la vue du modèle, car celle-ci est complètement encapsulée dans les procédures et les packages de l'application. Ce manque de découpage rend la maintenabilité du code encore plus compliquée.

Ces deux problématiques sont à replacer dans le contexte de la TMA. En effet, l'équipe est composée de seulement 8 personnes et les compétences sont réparties distinctement. Il est donc encore plus difficile de faire évoluer certains modules de manière simple et de nombreux échanges sont nécessaires pour s'assurer de la non-régression de l'application.

Le but de mon stage est donc double : il m'est demandé d'explorer les possibilités offertes par d'autres langages préalablement choisis, en offrant une interface de test générique pour *eWip*.

La structure de programmation que je vais implémenter va donc servir de modèle et de base pour les développements futurs et ce, en vue d'une possible migration. Enfin, afin de tester cette nouvelle architecture, j'ai développé un deuxième programme permettant de configurer les *reportings* de statistiques.

### 2. Planning

Etant donné que j'ai d'avantage effectué un travail de recherche, je n'ai pas eu de planning strict à respecter. Cependant, je peux fournir une vue d'ensemble de mon travail.

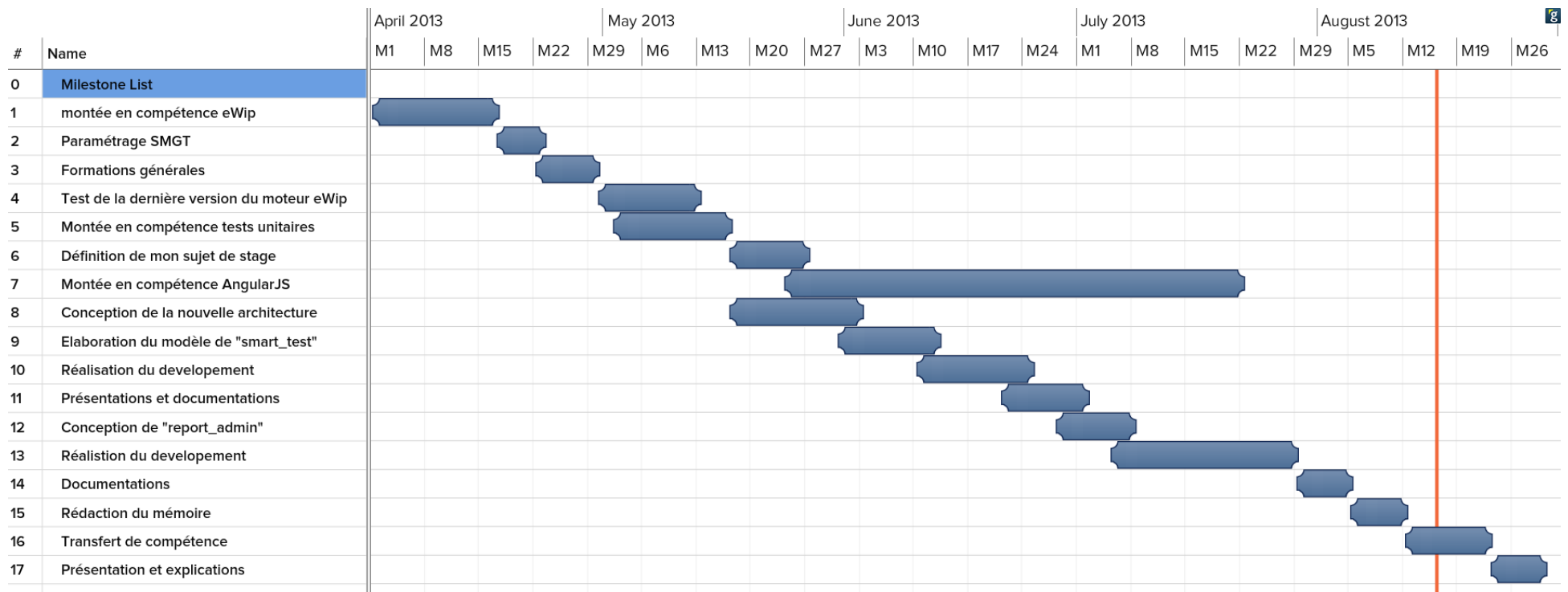


Figure 4 : Planning du stage

### 3. Les tests et l'évolution des applications

Le sujet de mon stage a été élaboré suite aux différentes problématiques que l'équipe TMA a pu rencontrer au cours des divers projets. Les applications créées sont le fruit d'un travail collaboratif important faisant intervenir de nombreux acteurs. Il est donc fréquent que ces acteurs soient affectés à d'autres tâches et que leurs compétences ne soient plus disponibles. Il faut donc permettre d'assurer un suivi et que n'importe quel développeur de l'équipe puisse modifier et améliorer le code aisément. De la même manière, face au volume des applications il est nécessaire d'avoir une organisation claire et simple.

Ces constats correspondent, entre autres, aux problématiques de test et de maintenabilité du code. Aujourd'hui, lorsqu'un projet est vendu au client, la moitié du temps est consacrée aux tests. Ce coût est important pour les SSII et elles souhaitent donc le réduire.

L'équipe TMA a ainsi lancé un programme d'amélioration et de maintenabilité du code. Il se décompose en 3 grandes parties :

- Mise en place de la méthode *Test Driven Development*.
- Mise en place d'une plateforme d'intégration continue.
- Amélioration de l'architecture et des technologies utilisées.

Mon stage porte donc sur le dernier point que je détaillerais d'avantage par la suite. Cependant, les autres solutions pour améliorer la maintenabilité répondent à la même problématique et je vais donc apporter un complément d'information à ce sujet.

#### a. Les tests et la méthode TDD

Les tests sont de plus en plus intégrés aux langages et aux *frameworks* afin de faciliter le travail du développeur. On peut citer l'exemple d'*AngularJS* qui inclut directement un mécanisme de test. L'objectif de cette approche est de tendre vers des applications « zéro bug ».

Les tests sont une autre manière de coder le programme en définissant un jeu de données et le résultat à obtenir pour chaque fonction. Afin d'avoir une approche optimale, la définition du jeu de tests est primordiale. En effet, il faut prévoir tous les cas de figures lorsque l'application sera en production.

La première étape des tests sont les tests unitaires. Ils servent à tester les fonctions de manière isolée. Il y a ensuite les tests d'intégrations qui permettent de vérifier le fonctionnement des classes entre elles puis les tests de performances.

Dans le cas des tests unitaires, afin de couvrir tous les cas de figure, il en existe 4 types :

- Test passant : le programme doit poursuivre son exécution et le résultat obtenu est celui attendu.
- Test non passant : le programme doit s'arrêter et fournir un message d'erreur clair qui doit faciliter la maintenance du programme.
- Les tests aux limites : on définit la valeur maximale que peut prendre une variable et on observe le comportement. Ce type de test est normalement passant.
- Les tests de non-régression : ceux-ci permettent de vérifier que l'application n'a pas régressé fonctionnellement parlant.

Afin d'assister le développeur dans sa tâche il existe de nombreux frameworks qui permettent de faciliter la mise en œuvre des tests. Une autre manière d'accompagner le programmeur est le *Test Driven Development*. Cette méthode de travail place le test au centre de la programmation.

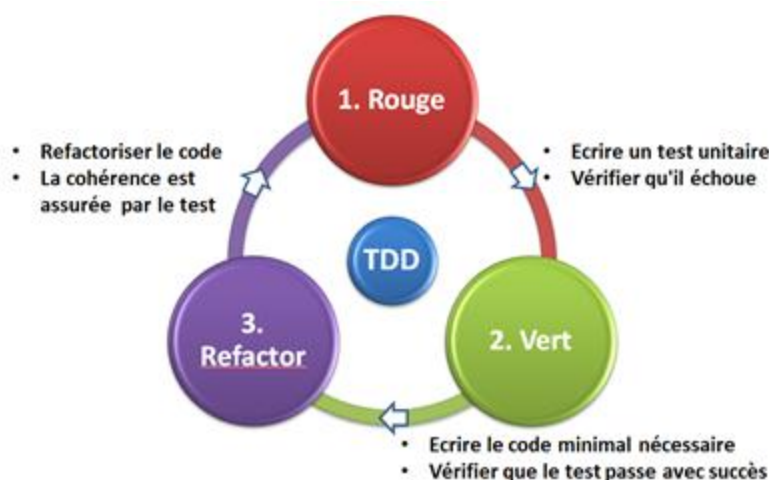


Figure 5 : Schéma de la méthodologie TDD

Comme on peut le voir sur le schéma, on commence par écrire le test et ensuite le code.

Cette méthode nécessite une modification de l'approche de la programmation. Cette évolution peut être longue à mettre en place dans une équipe ou une société.

On peut voir que les tests sont importants dans le développement mais ils nécessitent d'être maintenus afin d'assurer la pérennité de l'application.



## b. Plateforme d'intégration continue

Une plateforme d'intégration continue (PIC) est un système couplé avec un gestionnaire de versions qui permet d'automatiser les tests à chaque *commit* (procédé consistant à partager ses sources sur le dépôt central). Ensuite, la PIC est paramétrée pour effectuer les tests de manière récurrente et faire remonter les erreurs.

Ce type de plateforme permet un gain de temps car si les tests sont bien maintenus on limite la régression de l'application en signalant le *commit* qui l'a causé ; la correction du bug est donc facilitée.

Une PIC permet de limiter le laps de temps entre le développement et les tests et de raccourcir le cycle en V. De plus les possibilités de *rollback* (capacité à revenir dans un état antérieur) sont améliorées.

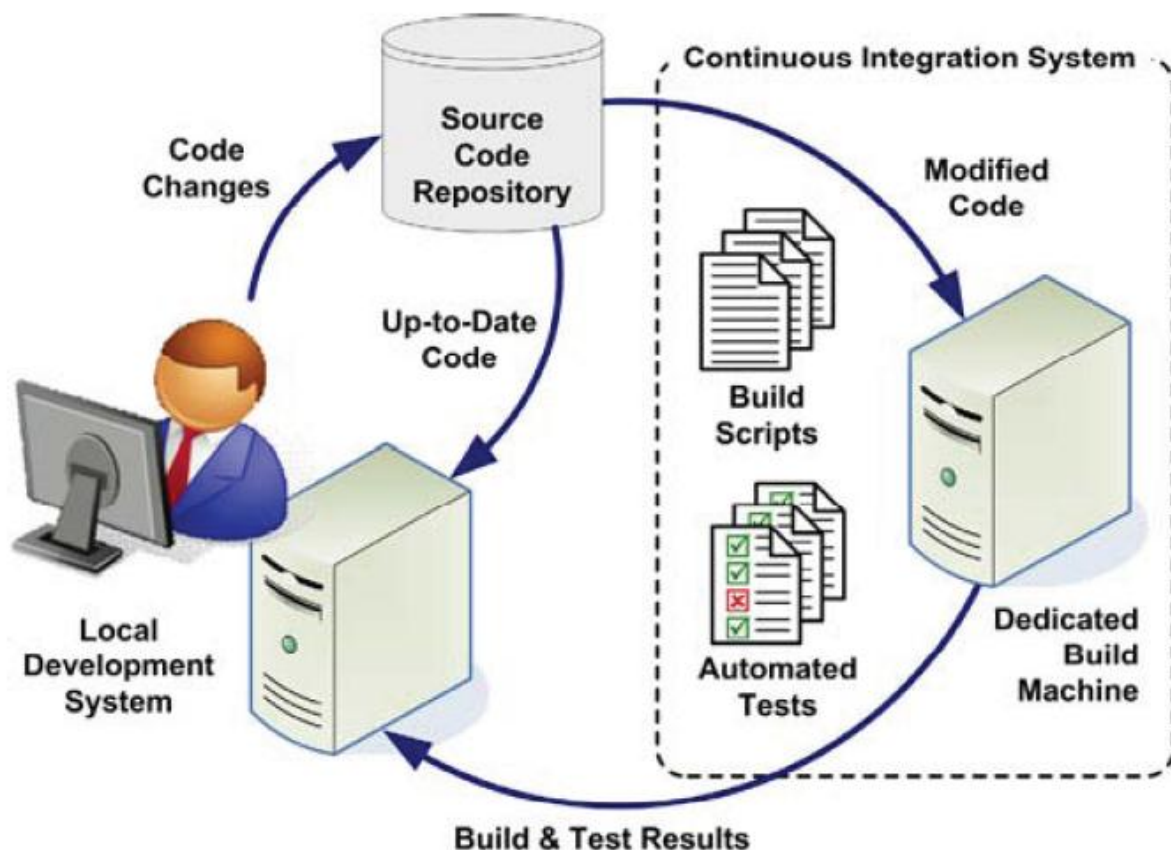


Figure 6 : Schéma de fonctionnement d'une PIC

<http://www.fil.univ->

[lille1.fr/~decomite/ue/ResumesStages/2010/resumes/guette/integration\\_continue.png](http://lille1.fr/~decomite/ue/ResumesStages/2010/resumes/guette/integration_continue.png)

## 4. Conceptions et évolutions

### a. Architecture

Voici le schéma de l'architecture au début de mon stage :

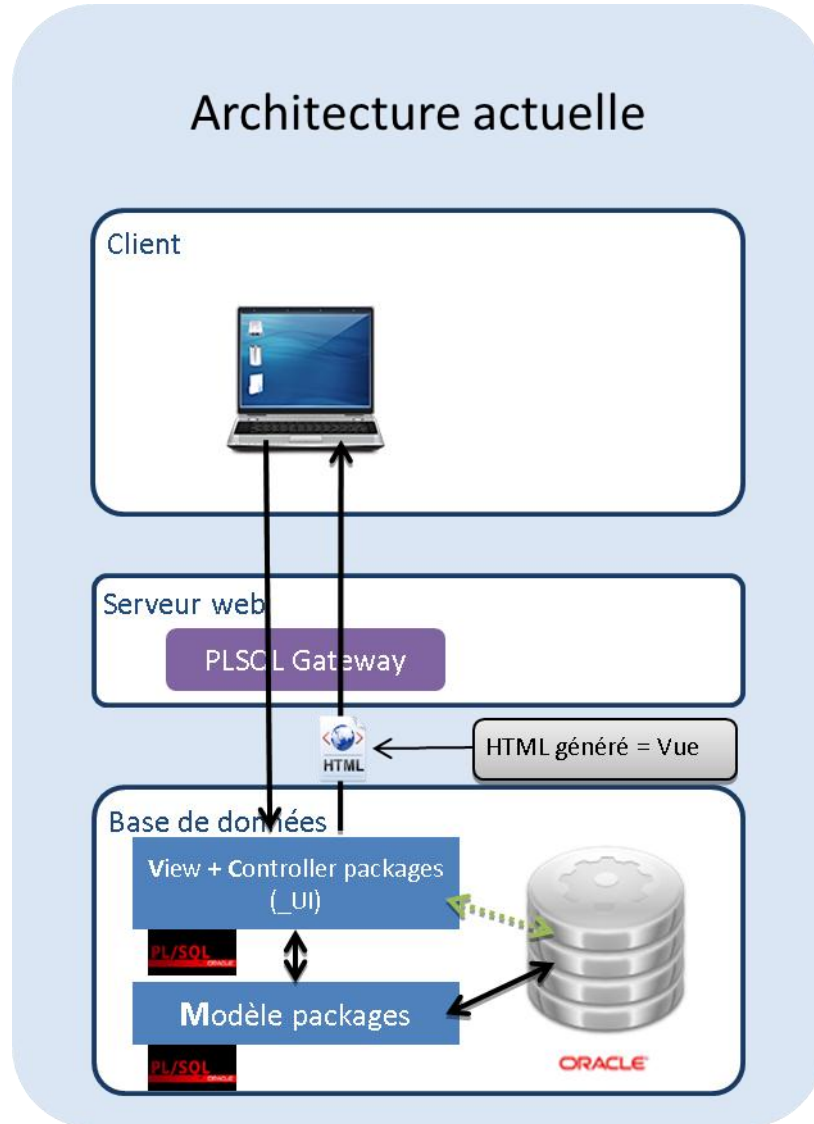


Figure 7 : Schéma de l'architecture initiale

Comme décrit précédemment, il n'y a aucune distinction MVC et tout le code se trouve dans la base de données Oracle.

Celle-ci génère du HTML, qui est envoyé au client via la PL/SQL gateway.

Voici le schéma de l'architecture cible :

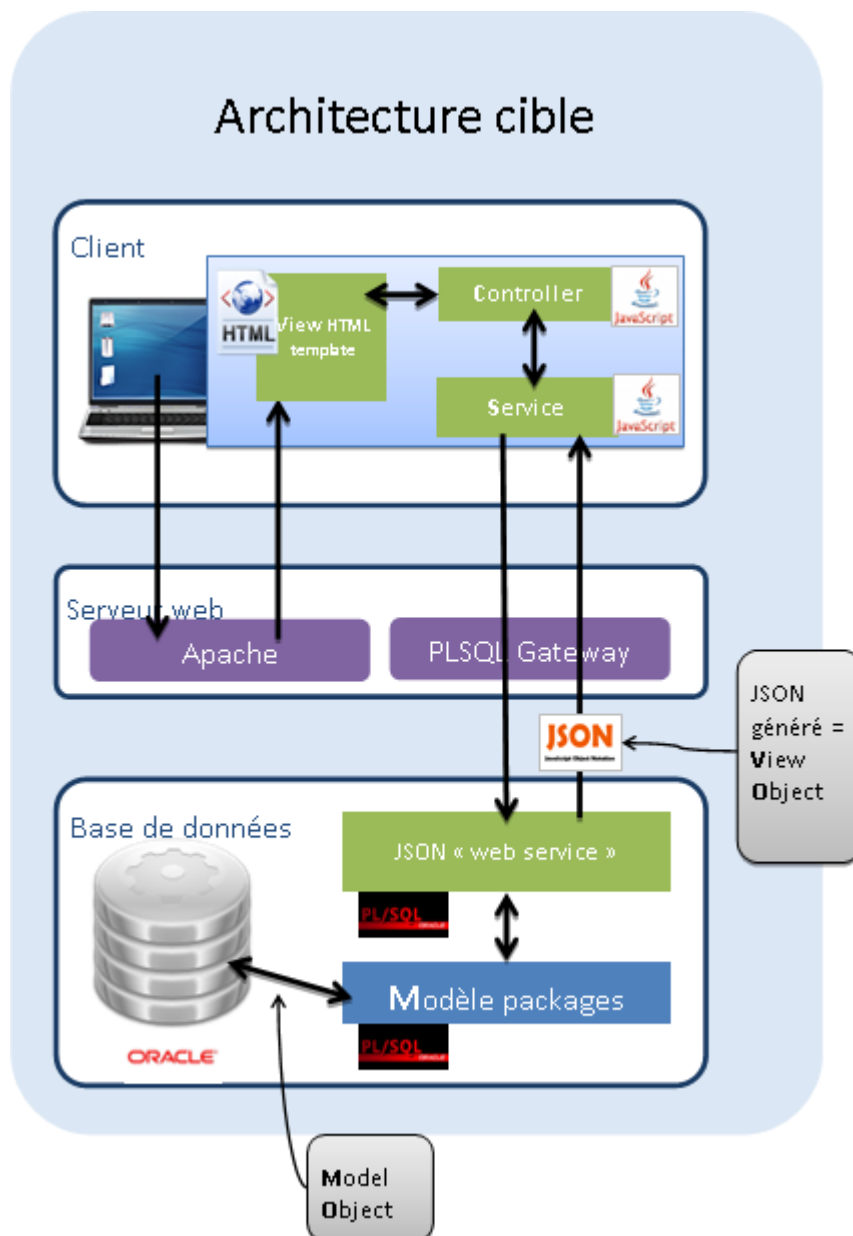


Figure 8 : Schéma de l'architecture cible

Le but de cette nouvelle architecture est d'offrir une meilleure distinction des couches MVC.

- Les accès à la base se font toujours en PL/SQL car le but de cette évolution n'est pas de modifier complètement *eWip* mais juste de faciliter sa maintenabilité en limitant au plus possible les coûts.
- Les données récupérées en base sont envoyées au client sous le format *Json*. Cet envoi est effectué via la *PL/SQL gateway*.
- Le client récupère donc un fichier *Json* qui est interprété en *JavaScript* et permet de générer la page HTML grâce à un *template*.

On peut voir sur le schéma qu'un autre modèle MVC est présent chez le client. C'est à ce niveau qu'interviennent les nouveaux langages précédemment cités et que je vais détailler par la suite.

Il est intéressant de noter que dans la nouvelle architecture, nous disposons de 2 objets génériques :

Le « *model object* » et le « *view object* ». Ils sont définis en fonction des performances de chaque environnement et facilitent les manipulations de données.

### *Model object*

C'est un format générique de données en PL/SQL qui contient les résultats des requêtes SQL. Cet objet est un tableau à 3 dimensions :

- La 1<sup>ère</sup> correspond à l'index des résultats de la requête.
- La 2<sup>ème</sup> contient les noms des colonnes de la requête (*i.e.* la clause SELECT)
- La 3<sup>ème</sup> contient la valeur enregistrée en base.

Ce format permet de traiter toutes les requêtes SQL de la même manière. Nous utiliserons une fonction qui prendra en paramètre la requête SQL et retournera un *model object*.

### *View object*

Ce format de données est du *Json* et permet de communiquer entre le modèle et la vue. En effet, une fonction (écrite en PL/SQL) nous permet de convertir le *model object* en tableau associatif *Json*. Cet objet va nous permettre de configurer la vue en fonction des données contenues en base.

## **b. Langages**

Dans le but d'utiliser un modèle MVC coté client et d'avoir une interface ergonomique dans des délais brefs, nous avons utilisé le *framework AngularJS* et la librairie CSS *Twitter bootstrap*.

### *AngularJS*

C'est un *framework* open-source développé par *Google* en 2009 et la version actuelle est la 1.0.7.

Ce *framework* est particulièrement adapté pour concevoir des applications mono-page. En effet, grâce à un double *binding* entre la vue et le modèle, les changements sont instantanés et permettent d'avoir une interface interactive.

Le principe clef d'*AngularJS* est de permettre de rajouter des balises ou des attributs directement dans le code HTML et d'avoir un code dynamique. Voici un exemple de code disponible sur <http://jsfiddle.net/z9cGm/45/>:

Figure 9 : Exemple de code AngularJS

Balise  
dynamique

```

1 <div ng-app ng-controller="myController">
2   <table>
3     <tr>
4       <th>column_1</th>
5       <th>column_2</th>
6     </tr>
7     <tr ng-repeat="row in rows">
8       <td>{{row.data_1}}</td>
9       <td>{{row.data_2}}</td>
10    </tr>
11  </table>
12 </div>

```

HTML

```

1 function myController($scope) {
2   $scope.rows = [{data_1 : '(1) data 1', data_2 : '(1) data 2'},
3                 {data_1 : '(2) data 1', data_2 : '(2) data 2'}];
4 }
5

```

JavaScript

1

column\_1 column\_2  
 (1) data 1 (1) data 2  
 (2) data 1 (2) data 2

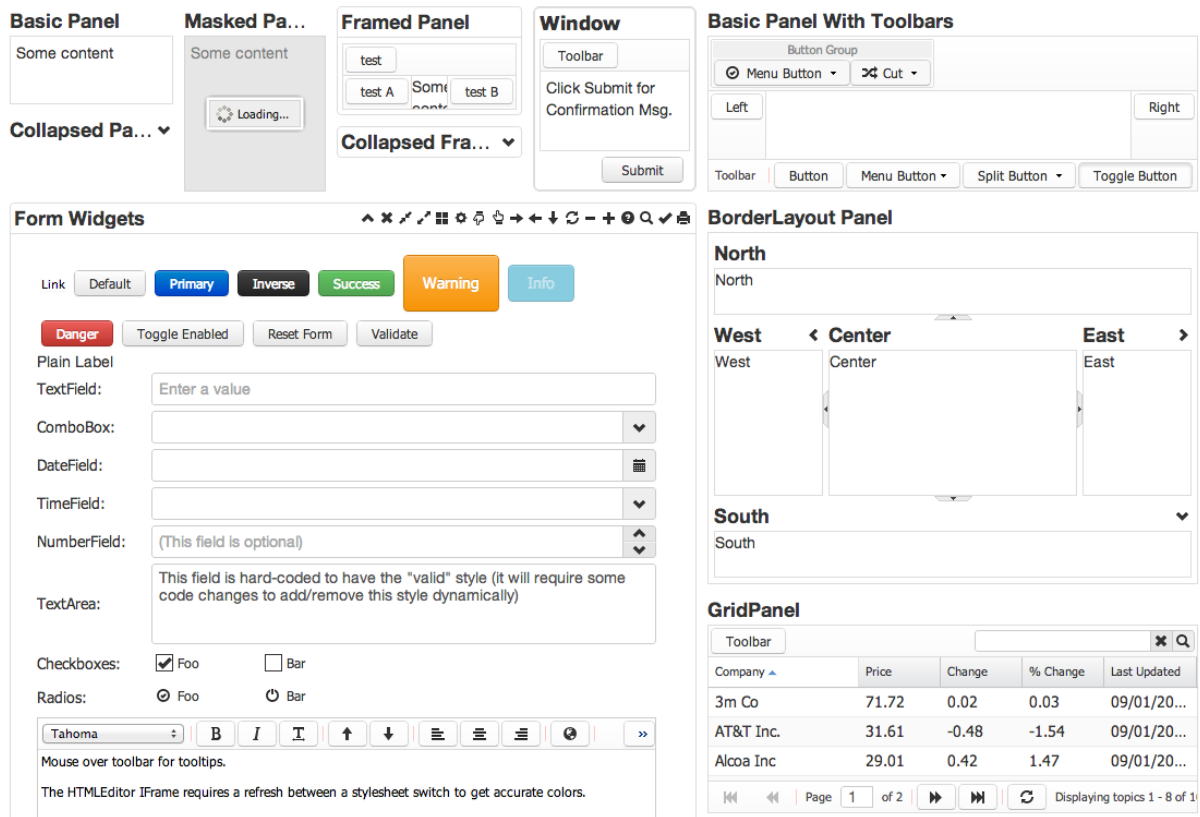
Résultat dans le

Dans cet exemple, on peut voir que la balise `<tr ng-repeat="row in rows">` permet de répéter cette ligne du `<table>` autant de fois que `$scope.rows` contient de données. La vue HTML est donc dynamique en fonction du modèle et peut être mise à jour sans recharger la page, le code est donc plus simple à produire et à maintenir.

Une autre force d'*AngularJS* est de pouvoir créer ses propres balises et attributs, appelées *directive*. On peut donc rendre son code complètement générique et centralisé.

### Twitter bootstrap

C'est une librairie développée par Mark Otto et Jacob Thornton permettant d'avoir un ensemble de classes pour obtenir une interface ergonomique et standard. La dernière version est la 2.3.2. Voici un aperçu des différents items disponibles :



### Figure 10 : Exemple d'utilisation de Twitter Bootstrap

<http://bootstrap.newbridgegreen.com/images/screenshot.png>

Cette librairie se couple assez bien avec *AngularJS*, cependant elle est statique et directement implémentée dans le code HTML. Afin de pouvoir utiliser *Twitter bootstrap* de manière dynamique, on peut utiliser *UI bootstrap* qui est une extension d'*AngularJS* avec l'interface *bootstrap*. C'est en réalité un ensemble de *directives* qui permettent de créer à la volée les différents éléments de la page.

## 5. Développements et réalisations

### a. Smart test

*Smart test* est la première application que l'on m'a demandé de développer. Elle a pour but d'offrir une interface générique permettant de visualiser l'avancement d'une campagne de tests et de pouvoir ainsi estimer quel ratio d'une application ou du moteur est fiable.

Un autre aspect de *smart test* est de quantifier quelles fonctionnalités du moteur sont les plus utilisées et de ne pas les tester plusieurs fois.

Afin de réaliser correctement cette application, il faut bien intégrer le fonctionnement *d'eWip*.

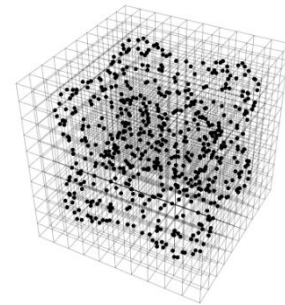
Une fonctionnalité générique *eWip* peut être configurée avec de nombreuses combinaisons (type d'attribut représenté, caractère obligatoire ou non, etc...).

Le nombre de cas de tests potentiels se calcule donc selon la formule:

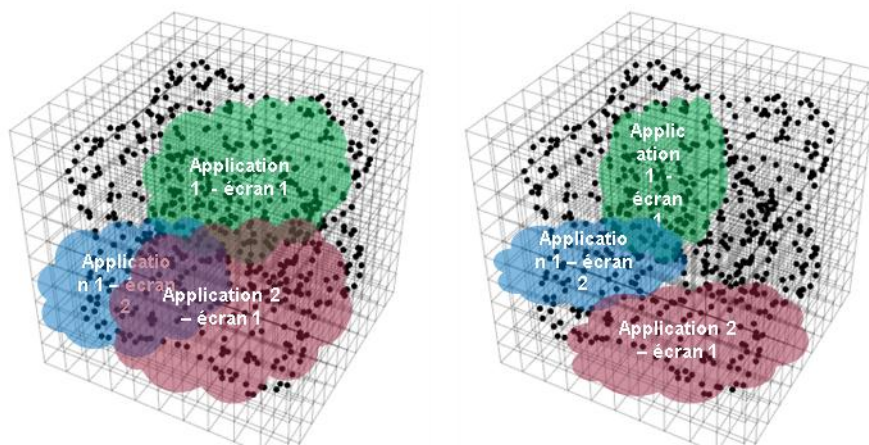
*nb cas de tests =*

*nb combinaisons axe paramétrage 1 × nb combinaisons axe paramétrage 2 ×  
nb combinaisons axe paramétrage 3 × ...*

Ce que l'on peut représenter sous la forme d'un cube (s'il y a seulement 3 dimensions d'axes de paramétrages) où chaque point représente un paramétrage possible.



Si on implémente 3 applications sur ce moteur utilisant différentes fonctionnalités on peut voir que toutes les possibilités de paramétrages ne sont pas utilisées et que certaines sont utilisées par plusieurs applications.



**Figure 11 : Schéma d'utilisation des fonctionnalités**

Grâce à des requêtes SQL sur les tables de paramétrages on peut déterminer quelles fonctionnalités sont critiques et optimiser les tests.



Voici le modèle de données de *smart test* :

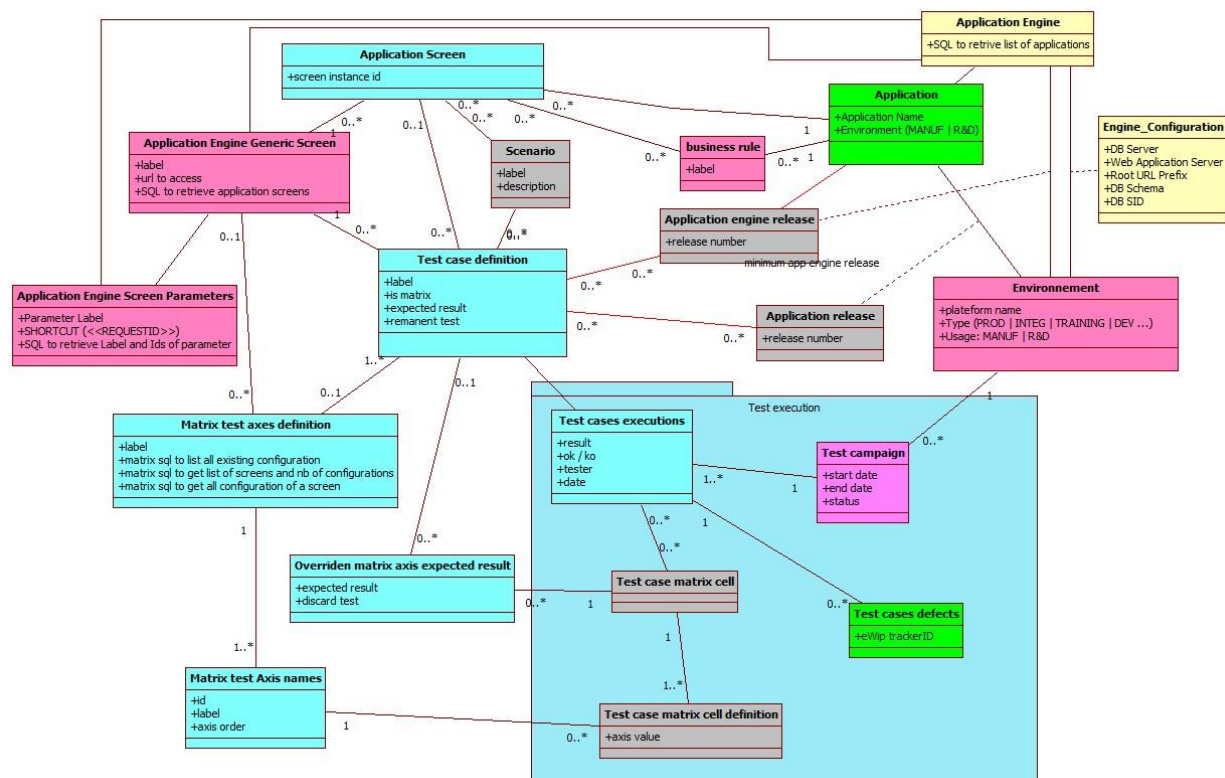


Figure 12 : Modèle de données de Smart Test

Il y a deux aspects assez intéressants dans ce modèle :

- Les tables en vertes sont des données déjà existantes auxquelles il faut se rattacher.
- Les tables en bleu sont des entités *eWip*.
- Certaines tables comportent des requêtes SQL ce qui rend le modèle dynamique.



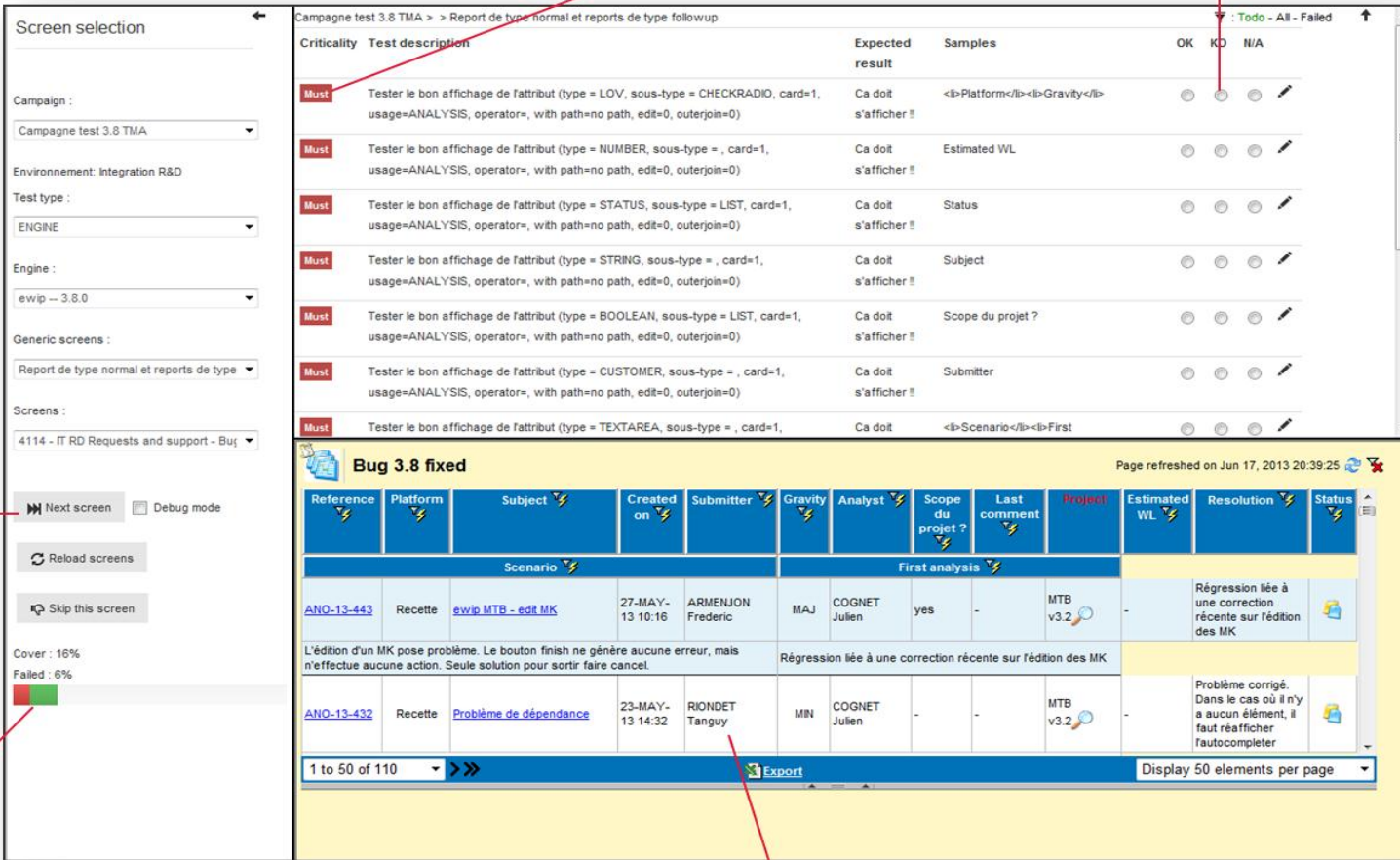
Intégration de la criticité MoSCoW

Cas de test KO = anomalie loguée dans eWip

Accès au prochain meilleur écran à tester (basé sur le nombre de cas de test \* pondération MOSCOW)

Etat d'avancement temps réel

Accès direct à l'écran à tester (paramétré au niveau de la plateforme associé à la campagne) = gain de temps substantiel



The screenshot displays the 'Smart Test' interface. On the left, a 'Screen selection' sidebar allows filtering by Campaign (e.g., 'Campagne test 3.8 TMA'), Environment (e.g., 'Integration R&D'), Test type (e.g., 'ENGINE'), Engine (e.g., 'ewip - 3.8.0'), and Generic screens (e.g., 'Report de type normal et reports de type'). It also shows 'Screens' (e.g., '4114 - IT RD Requests and support - Bug') and progress indicators for 'Cover' (16%) and 'Failed' (6%).

The main area shows a table of test results with columns: Criticality, Test description, Expected result, Samples, OK, KO, and N/A. Several tests are marked as 'Must' with a red icon. A red arrow points from the 'Intégration de la criticité MoSCoW' label to the 'Must' criticality column.

Below the test results, a 'Bug 3.8 fixed' section displays a table of bugs with columns: Reference, Platform, Subject, Created on, Submitter, Gravity, Analyst, Scope du projet, Last comment, Project, Estimated WL, Resolution, and Status. Two bugs are listed: ANO-13-443 and ANO-13-432. A red arrow points from the 'Cas de test KO = anomalie loguée dans eWip' label to the 'KO' column in the test results table.

At the bottom, a red box contains the text: 'Accès direct à l'écran à tester (paramétré au niveau de la plateforme associé à la campagne) = gain de temps substantiel'. A red arrow points from this box to the 'Next screen' button in the sidebar.

Figure 13 : Interface de Smart Test

Cette application permet un gain de temps pour les tests autant au niveau ergonomique qu'organisationnel. Cependant, il ne faut plus juste écrire le test en langage informel mais faire correspondre le cas de test à sa fonctionnalité *eWip* et écrire la requête SQL nécessaire pour retrouver les écrans des différentes applications utilisant cette partie du code.

La charge de travail est donc plus importante au début des tests mais permet une forte capitalisation.

### *Difficultés*

Une des principales difficultés a été d'intégrer la complexité du logiciel *eWip* et d'en saisir les rouages. En effet, l'existant est assez complexe et fait appel à de nombreuses notions et à des langages différents. D'autre part, l'application doit avoir des temps de réponses corrects. Pour cela, j'ai dû modifier l'architecture et l'algorithme afin d'obtenir les performances voulues.

Le moteur *eWip* étant générique et étant donné que l'on doit tester les 6 plateformes, j'ai donc dû réaliser une application générique sur un moteur générique. Il y a donc assez peu de code à écrire mais celui-ci est complexe techniquement.

Enfin, une amélioration que je n'ai pas réussi à mettre en place est la navigation dans *l'iFrame* (écran de visualisation de l'application à tester) et l'affichage des tests correspondants. En effet, pour des raisons de sécurité de *cross-scripting*, il est interdit par le navigateur d'obtenir des informations entre la page maîtresse et *l'iFrame* lorsque celles-ci ne sont pas sur le même domaine. Afin de contourner cette interdiction, nous avons essayé de faire de la réécriture d'URL sur le serveur *Apache* pour « tromper » le navigateur. Cette technique n'a pas fonctionné car seul l'URL était modifiée et non les *cookies* servant à l'authentification. Par conséquent, nous étions constamment déconnectés du serveur.

## **b. Report admin**

*Report admin* est la deuxième application que j'ai dû créer en respectant la nouvelle architecture. C'est une application permettant de configurer des écrans de statistiques en fonction des entités *eWip*. Ce module s'inspire des tableaux pivots de *microsoft excel*.

Plusieurs aspects ont été intéressants sur cette application:

- Appliquer l'architecture précédemment définie.
- Délimiter la frontière entre vue et modèle.
- Approfondir l'aspect technique d'*AngularJS* et algorithmique en PL/SQL.

Ce projet s'est déroulé dans un sens assez particulier pour des raisons d'organisation et de gestion de projet. En effet, j'ai commencé par définir le fichier *Json* de configuration puis la vue et enfin le modèle.

### *Directives*

Afin d'offrir une interface agréable j'ai utilisé des librairies telle que *DhtmlX* offrant une multitude de *widgets*. Comme on peut le voir sur la figure 15, il y a un *color picker* permettant de sélectionner une couleur. Afin de pouvoir générer cet élément à la volée, j'ai dû créer une directive *AngularJS*. Autrement dit, j'ai pu ajouter une balise à mon code HTML afin d'assurer le bon fonctionnement de mon application et de *bind* automatiquement cet élément sur le modèle.

### *Frontière vue – modèle*

Il m'a été assez difficile de délimiter la frontière entre la vue et le modèle. En effet, j'ai essayé de rendre des éléments génériques alors que c'était finalement assez peu efficace. Je peux citer par exemple les dépendances d'affichages entre les éléments qui sont définies dans le fichier *Json* de configuration. Je suis parti du principe que la vue n'avait pas besoin de connaître ces dépendances et qu'elle devait être générique pour toutes configurations du modèle. Cette pratique a grandement alourdi mon code et n'a pas offert un réel gain car je n'ai pas eu besoin de modifier ces dépendances.

### *Transformation des données*

La dernière étape de ce projet a été la plus difficile à réaliser techniquement parlant. J'ai dû développer un algorithme qui permet de transférer des données entre une structure de type arborescence en *Json* et des tables SQL. Cette transformation devait s'opérer dans les deux sens mais la structure de ces deux modèles ne devait être enregistrée qu'à un seul endroit pour faciliter la maintenabilité. Enfin mon algorithme devait être assez générique pour implémenter d'autres conversions de données ou du moins servir de modèle.

Ecran d'interface :

### Edit report My report

Admin(repair center) > Edit reports > PN > My report

Global informations

Name:

My report

36001

Macro type:

AGGREGATED DATA

Report type:

LogSearch

Attributes

Attributes of entity PN

String Attributes

no dropable

numeric

**text**

date

date of status

Numeric values

Linked objects

Rows / Axis field

Attribute name	Setup	Move	Remove
text			
date			

+ add item

Filters

Attribute name	Mandatory	Mode	Setup	Move	Remove
text	<input checked="" type="checkbox"/>	Updatable			

+ add item

DataSets

Create new dataset:

Create

Default

an other

Columns / Legend fields

Attribute name	Column name	Setup	move	Remove
numeric				

+ add item

Computation

Attribute name	Computation label	Computation	Shown as	Setup	Remove
text					

+ add item

Roles authorized to access report

Choose authorized roles:

Administrator

**PN Viewer**

>

>>

<<

<

PN Editor

Figure 14 : Interface de Report Admin – sélection des attributs

### Edit report My report

Admin(repair center) > Edit reports > PH > My report

**Sort**

Selected	Type
<input type="radio"/>	Ascending
<input checked="" type="radio"/>	Descending
<input type="radio"/>	Custom

**Filters**

☐ Based on operator
 ☒ Select values
 ☐ No filters

value text2  
value text3

>  
>>  
<<  
<

value text1  
value text2  
value text3

**Grouping**

☐ No grouping
 ☒ Manual grouping

New name:  Add ID:  Reuse:  Temp: send options

Create new group

group 1 group 2 group 3

Group name: group 2

Group charting color:

☒ Based on operator  
☐ Select values  
☐ No filters

**Format options**

Column label:

Conditional formatting: ☐ Range si prioritaire  
☐ Range si commenté

Figure 15 : Interface de Report Admin - édition d'un attribut

### Edit report My report

Admin(repair center) > Edit reports > PH > My report

**Filters**

☒ Is mandatory Mode:

label: test1

☒ Based on operator
 ☐ Select values
 ☐ No filters

between

Figure 16 : Interface de Report Admin - édition d'un attribut

## D. Conclusion

Ce stage m'a permis d'avoir un premier aperçu du métier d'informaticien et de découvrir de nombreux aspects liés aux développements.

Cette expérience a été très bénéfique autant sur le plan de la conception que de la réalisation, car j'ai pu être confronté à des problèmes concrets et actuels.

J'ai pu m'apercevoir qu'une problématique assez récurrente dans les différentes équipes d'informatique était d'assurer la fiabilité et la pérennité de son programme ; la question des tests est donc centrale dans les développements.

Même si cette problématique est posée depuis longtemps, les avancées technologiques nous permettent aujourd'hui de concevoir des systèmes plus autonomes et plus viables. Cette évolution technologique entraîne, à mon sens, une évolution du métier d'informaticien. En effet, il est finalement assez trivial, en regard des réalisations actuelles, qu'un algorithme n'appartient pas à son créateur et que celui-ci doit faciliter sa modification, sa réutilisabilité et sa maintenabilité.

Cette mouvance donne lieu à de nouvelles méthodologies (*i.e.* méthode *TDD*) qui permet de placer le test au centre des priorités. Cette solution n'est pas parfaite mais elle nous permet d'imaginer d'autres manières de conceptualiser un problème et de produire du code.

Comme dans toutes les branches professionnelles, le métier d'informaticien est en perpétuelle mutation à l'image des besoins demandés. Cependant, les conceptions et les architectures n'évoluent pas aussi vite que les technologies ce qui crée parfois un décalage.

Par le biais de ce stage j'ai pu me rendre compte de l'importance de la veille professionnelle dans le but de pouvoir choisir les solutions les plus pertinentes par rapport aux besoins d'un projet.

## E. Glossaire

**AngularJS** : *framework JavaScript* qui implémente un modèle MVC.

**Apache** : serveur écrit en *Java*.

**Balise** : se dit d'un élément de la syntaxe XML compris entre < >.

**Binding** : procédé consistant à faire correspondre une donnée entre plusieurs sources.

**C#** : langage objet développé par *Microsoft*.

**Cas de tests** : définit un exemple d'utilisation d'un logiciel ou d'une fonction.

**Color picker** : élément visuel permettant de sélectionner une couleur.

**Commit** : Action consistant à partager ses fichiers sources sur le serveur *Svn*.

**Cookies** : Donnée stockée par un site web sur l'ordinateur du client.

**Cross scripting** : technique de hacking consistant à exécuter du code *JavaScript* malveillant.

**Cycle en V** : Cycle spécifique de réalisation d'un projet.

**Data loading** : procédé permettant de charger des données.

**Dhtmlx** : librairie *JavaScript* permettant d'implémenter des *widgets*.

**Directives** : ajout d'une balise html avec *AngularJS*.

**EWip** : logiciel de *workflow* générique développé par CGI.

**Framework** : outil offrant un cadre de travail pour développer.

**FTP** : protocole type de requête réseau.

**Gateway** : peut se traduire par une porte d'accès, un portail.

**Généricité** : procédé consistant à extraire le caractère commun de plusieurs entités distinctes.

**HTML** : langage XML interprété par les navigateurs.

**HTTP** : protocole de requête réseau.

**Iframe** : permet d'insérer une page web à l'intérieur d'une autre.

**Java EE** : langage *Java* appliqué aux applications n-tiers.

**Javascript** : langage permettant d'interagir avec le DOM XML.

**Jboss** : serveur d'application *Java*.

**Json** : JavaScript Object Notation. Syntaxe de tableaux associatifs.

**JVM** : machine virtuelle de *Java*.

**Licence gnu GPL** : Licence libre d'un logiciel permettant son utilisation et sa modification.

**Machin To Machin** : interaction définie entre 2 machines.

**MOSCOW** : criticité et importance d'un test (Must , Should, Could, Would).

**MVC** : structure particulière d'un programme en Modèle, Vue, Contrôleur.

**Open source** : type de logiciel dont le code est libre d'accès et de modification.

**Oracle** : c'est un gestionnaire de base de données.

**Packages** : regroupement de fonctions en PL/SQL.

**PIC** : plateforme d'intégration continue.

**PL/SQL** : procedure language SQL. Langage inclus dans le SGBD Oracle.

**Release** : version d'une application.

**Report admin** : nom d'une application que j'ai créé.

**Reporting** : traitement statistique et visualisation de données.

**Rollback** : capacité à revenir à un état antérieur.

**Script** : ensemble de commande.

**Smart test** : nom d'une application que j'ai créé.



**SMTP** : protocole de requête réseau de type mail.

**SQL** : langage de requête sur les bases de données.

**Svn** : gestionnaire de source centralisé.

**TDD** : *Test Driven Development*. Méthode de développement accès sur les tests.

**TMA** : Tierce Maintenance Applicative.

**Twitter bootstrap** : librairie CSS.

**UI bootstrap** : librairie de directive *AngularJS* couplé à *Twitter Bootstrap*.

**Webservices** : service disponible via le réseau.

**Workflows** : succession d'état d'une entité symbolisant un flux.