CFTL
Journée Française
des Tests Logiciels 2016
Comité Français
des Tests Logiciels
ISTQB

Développement d’un projet billettique piloté par les tests métiers (BDD)

Laurent Py

@py_laurent

laurent.py@hiptest.net

http://hiptest.net

Raphaël Citeau

rciteau@parkeon.com

http://www.parkeon.com

Plan

- Contexte projet
- Pourquoi avoir choisi l'approche BDD
- Le déploiement du BDD étape par étape
- Les pratiques clés
- Conclusion

Contexte projet

Le projet : Helsinki

- donneur d'ordre finlandais (HSL), Intégrateur système Tieto (fournisseurs rang 1), plusieurs partenaires dont Parkeon
- un planning sur 3 ans
- un projet d'intégration multi-culturel
- 4000 équipements connectés
- 360 millions de passagers par an
- une migration de système existant

Contexte projet

Les phases de projet:

- Analyse fonctionnelle (1 an)
- Architecture (6 mois en parallèle)
- Cycles de développement et release (de 3 mois à 2 semaines en mode BDD)

Contexte projet

Le système :

Paramétrage et supervision
(back office node JS)

Vente - Distribution
(Web service)

Validation, Vente,
Contrôle
(Android)

Contexte projet

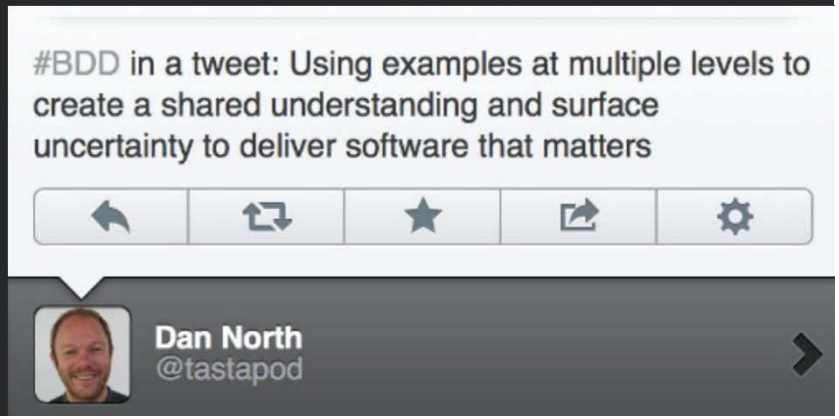
Organisation Parkeon:

- Management
  - 1 Directeur de Projet
  - 5 Chefs de Projet
- 1 x Design Authority
  - 2 Architectes
  - 4 Product Owners
- 4 x Équipe de Développement :
  - 1 Team Leader / Kanban Master
  - 1 Testeur
  - 5-6 Développeurs
- 2 x Équipe de test
  - 5-6 Testeurs

Au fait c’est quoi BDD?

Behavior Driven Development en un clin d'oeil

- Créer une compréhension partagée du système avec des exemples
- Basé sur un langage métier commun
- Définition du stop
- Lorsque que les tests/exemples sont automatisés, ils deviennent la spec vivante.

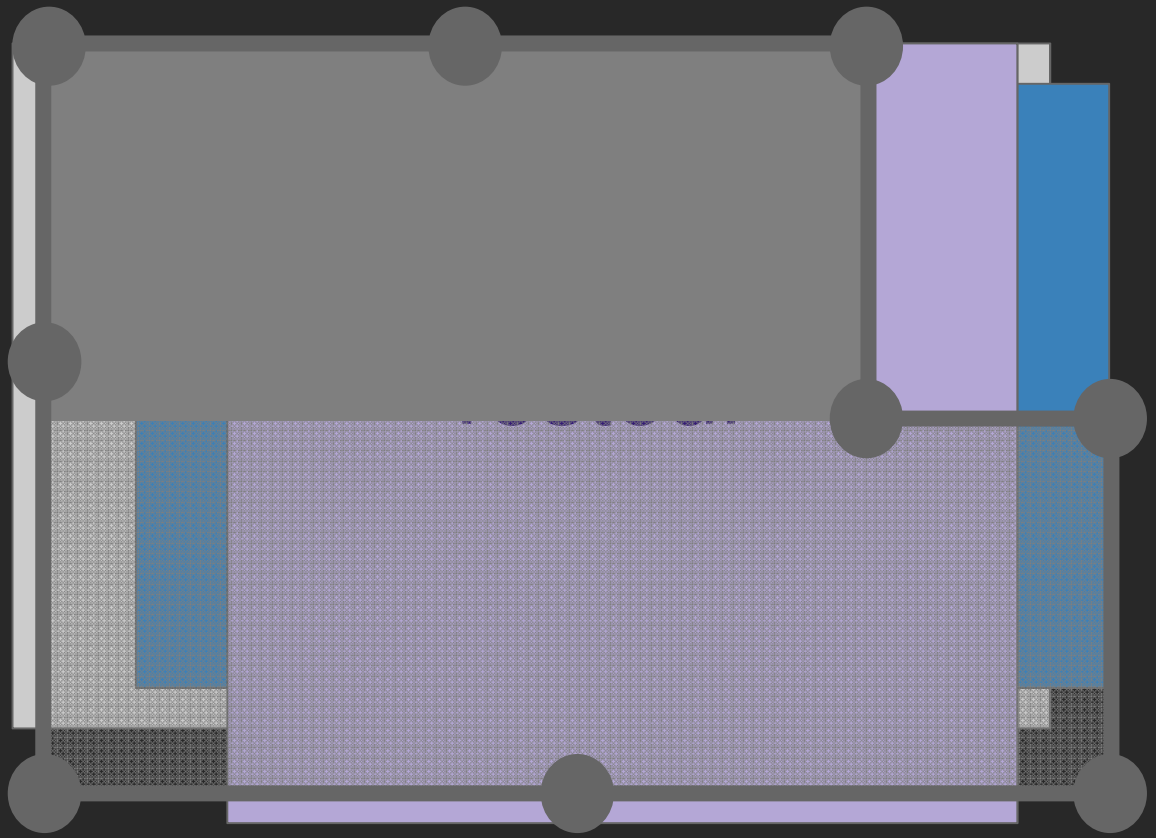


Story vue par le métier

Story vue par le développeur

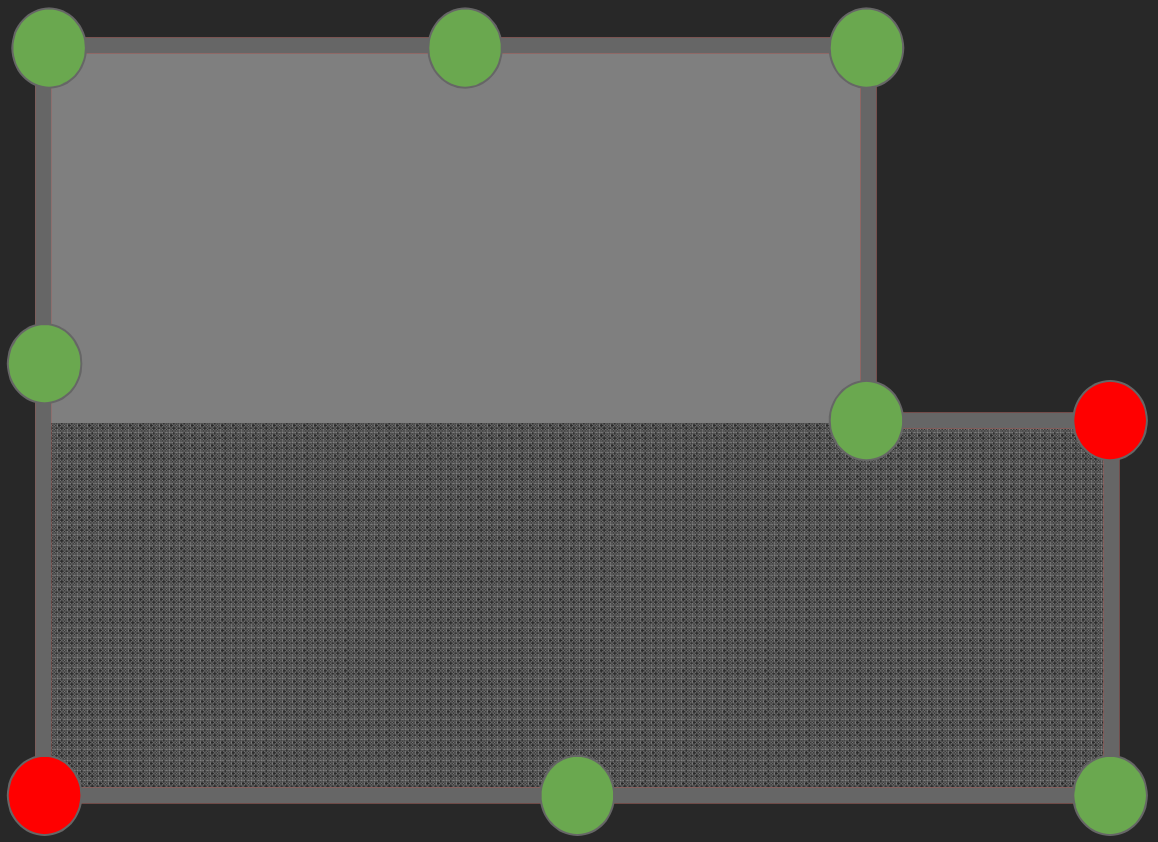
Story vue par le testeur

Ce que l’on va réellement faire



Un contrat basé sur des exemples

Une fois les exemples/tests exécutés



Spécification vivante

Pourquoi avoir choisi l'approche BDD?

Enjeux de Management de projet

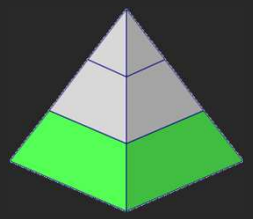
- Définir une méthode applicable à tous les archétypes d'équipes
- Favoriser un découpage fonctionnel basé sur la valeur et projetable sur un planning
- Maximiser l'automatisation des tests

Enjeux de Maintenance

- Assurer une qualité de service continue avec une équipe (très) réduite
- Faciliter la transmission des connaissances
- Fournir un framework pour développer des nouvelles fonctionnalités

Le déploiement du BDD étape par étape

Test des sous systèmes de façon autonome



- La Design Authority définit les comportements en relation avec le client
- Les PO et Développeurs les formalisent en BDD et développent ces comportements
- Les équipes de Testeurs implémentent les étapes de test et enrichissent la librairie de l'automate
- Une fois implémentés, les tests sont planifiés dans l'intégration continue

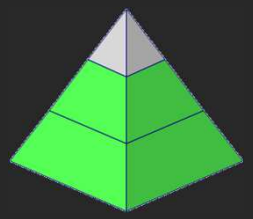
Artefacts :

mockups
simulateurs
sondes

Difficultés:

uniformiser le dictionnaire de donnée (vocabulaire)
limiter le nombre de *steps*
limiter le nombre d'implémentations pour 1 *step*
ne pas perdre le fil avec l'exigence initiale (traçabilité et sémantique)

Test d'intégration et de bout en bout



- Les Développeurs réutilisent les étapes de test définies précédemment pour chaque module afin de les intégrer 2 à 2 (avant livraison interne)
- Les équipes de test font de même, sur l'ensemble du système, pour réaliser une intégration bout en bout (après livraison interne)
- Ces tests sont planifiés dans une intégration continue transverse

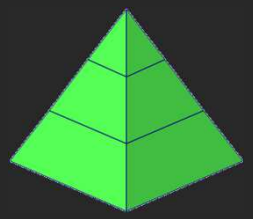
Artefacts :

simulateurs
sondes
robots

Difficultés:

limiter la dépendance avec un état initial du système
prendre en compte la volumétrie du système en production

Validation du système



- Les équipes de test et la DA définissent des scénarios qui vont mettre en jeu des cycles de vie “métiers” (je suis administrateur, je suis conducteur, ...)
- Ces tests sont exécutés pour la plupart manuellement (mais rapidement)
- Les tests “voyageurs”, les plus critiques, sont néanmoins tous automatisés
- Cette base de tests sert aussi aux démo réalisées avec le client

Artefacts :

sondes

robots

Difficultés:

prouver la couverture des exigences

limiter la dépendance avec un état initial du système

interactions «physiques» avec le système

Synthèse

Toutes les équipes ont participé à la réalisation d'un objectif commun : construire une bibliothèque et un automate de tests leur permettant de valider l'ensemble du système

Le BDD a donné un cadre au contrat qui les liait

La démarche a été perçue avec enthousiasme par les équipes, sans contrainte. Elle a l'avantage de mettre en avant la valeur humaine et de favoriser les échanges.

Les pratiques clés

Langage commun spécifique au métier

我希望我可以混合颜色

Expert métier

```
function(c1, c2) {
  if (c1==c2) {  return c1;
}
  else { … }
};
```

Développeur

Given the color red
  and the color green
when you mix the colors
then you obtain yellow

Testeur

Syntaxe Gherkin

Given : Étant donné un état

When : Lorsqu'une action est effectuée

Then : Une conséquence est constatée

Exemple d’un comportement

Edit | View tests

1 Given I prepare a device
- in_firmware_version = 4.1.1.svn7308_0
- in_sw_version = > 0.18.0
- out_service_state = open

2 When I prepare and deploy DM parameters
- in_firmware_package = ${in_firmware_package1}

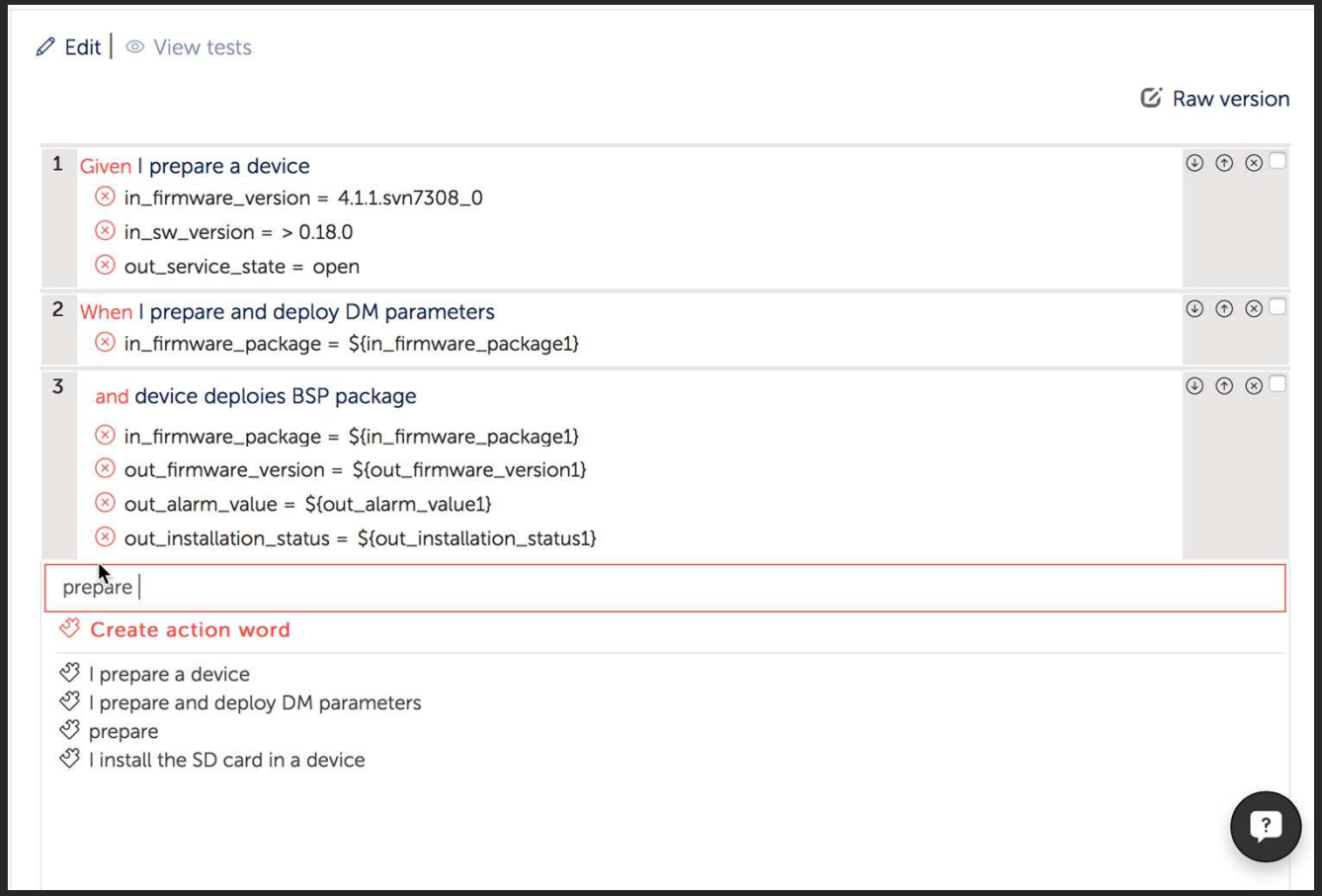
3 and device deploies BSP package
- in_firmware_package = ${in_firmware_package1}
- out_firmware_version = ${out_firmware_version1}
- out_alarm_value = ${out_alarm_value1}
- out_installation_status = ${out_installation_status1}

4 and I prepare and deploy DM parameters
- in_firmware_package = ${in_firmware_package2}

5 Then device deploies BSP package
- in_firmware_package = ${in_firmware_package2}
- out_firmware_version = ${out_firmware_version2}
- out_alarm_value = ${out_alarm_value2}
- out_installation_status = ${out_installation_status2}

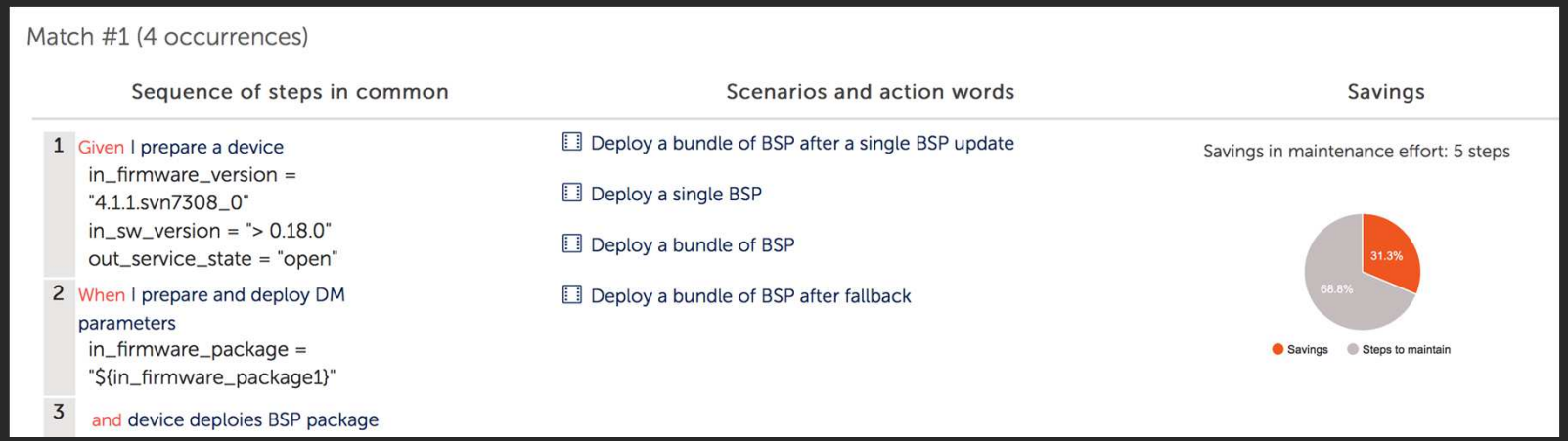
Être consistant dans la terminologie

Autocomplete



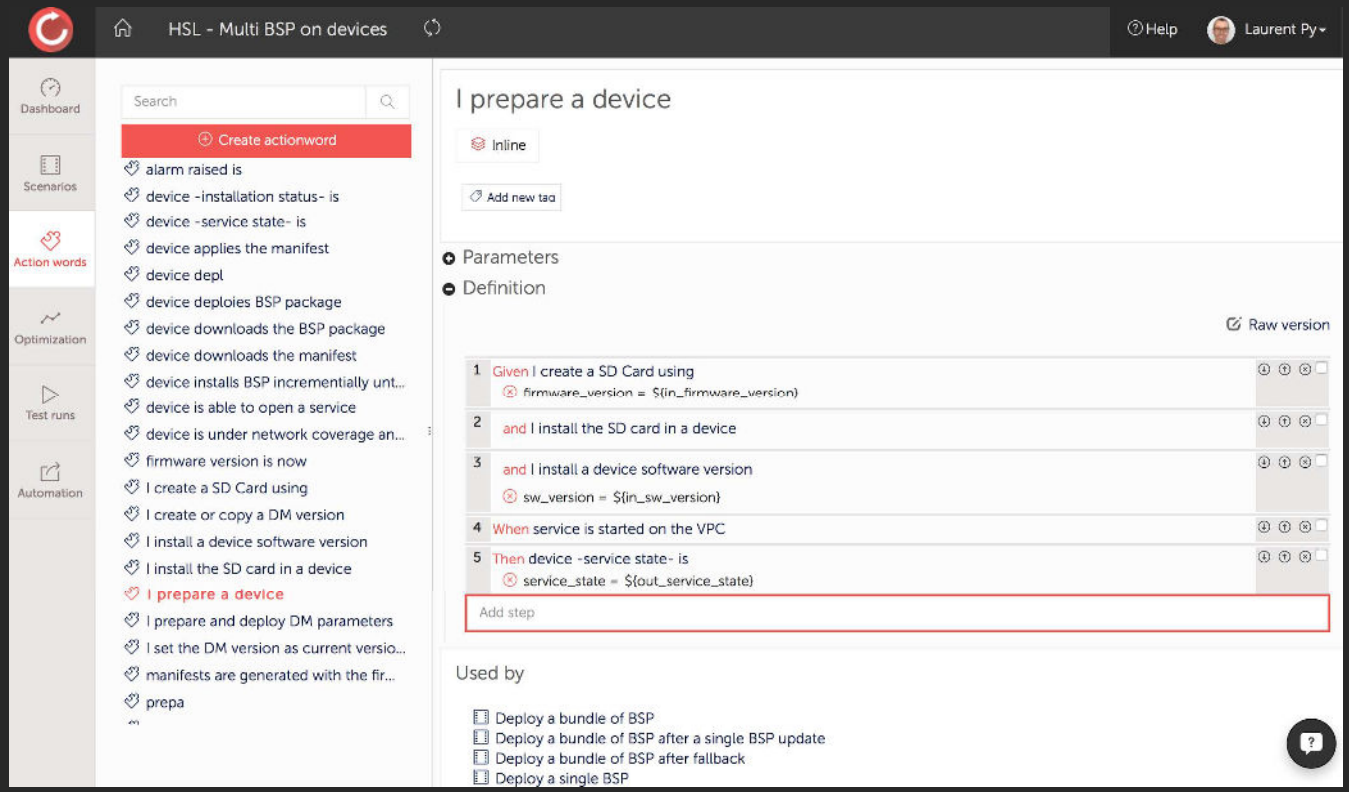
Refactoring continu des tests

Détection des duplications pour créer un seul point de maintenance



Refactoring continu des tests

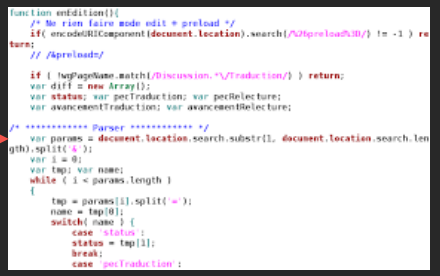
Propagation automatique des modifications sur le langage métier



Processus outillé

1- Détailler les *stories*

2 - Raffiner les cas de tests



3 - Implémenter les *stories* et tests

5 - Exécuter les tests

Résultats

Résultats

4 - Orchestrer les builds et tests runs

Automatisation et check (BDD)

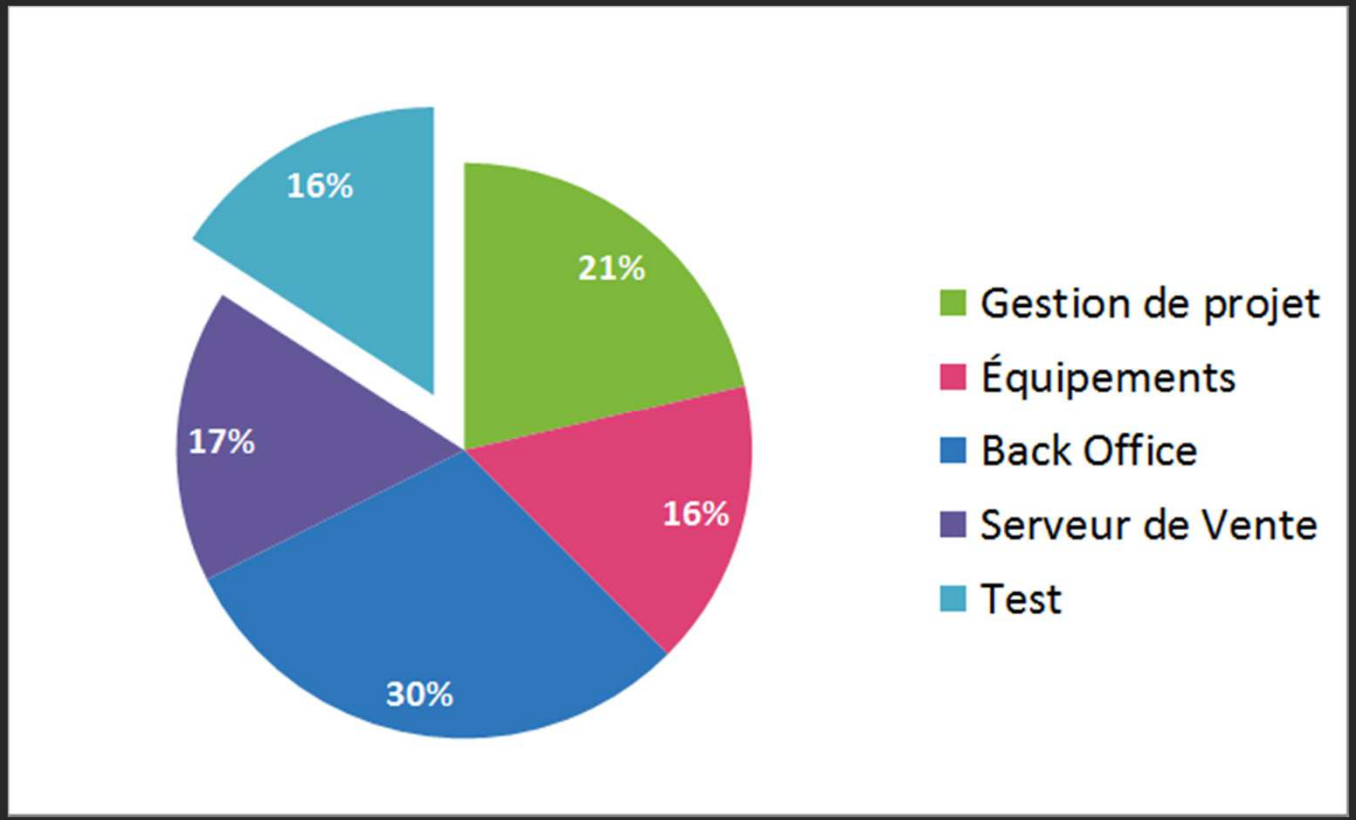
1635 tests fonctionnels unitaires
990 tests d'intégration sous systèmes
475 tests d'intégration système
200 tests de validation métiers

Langage métier: 1690 steps

Conlusion

Investissement en test

Répartition des coûts de développement



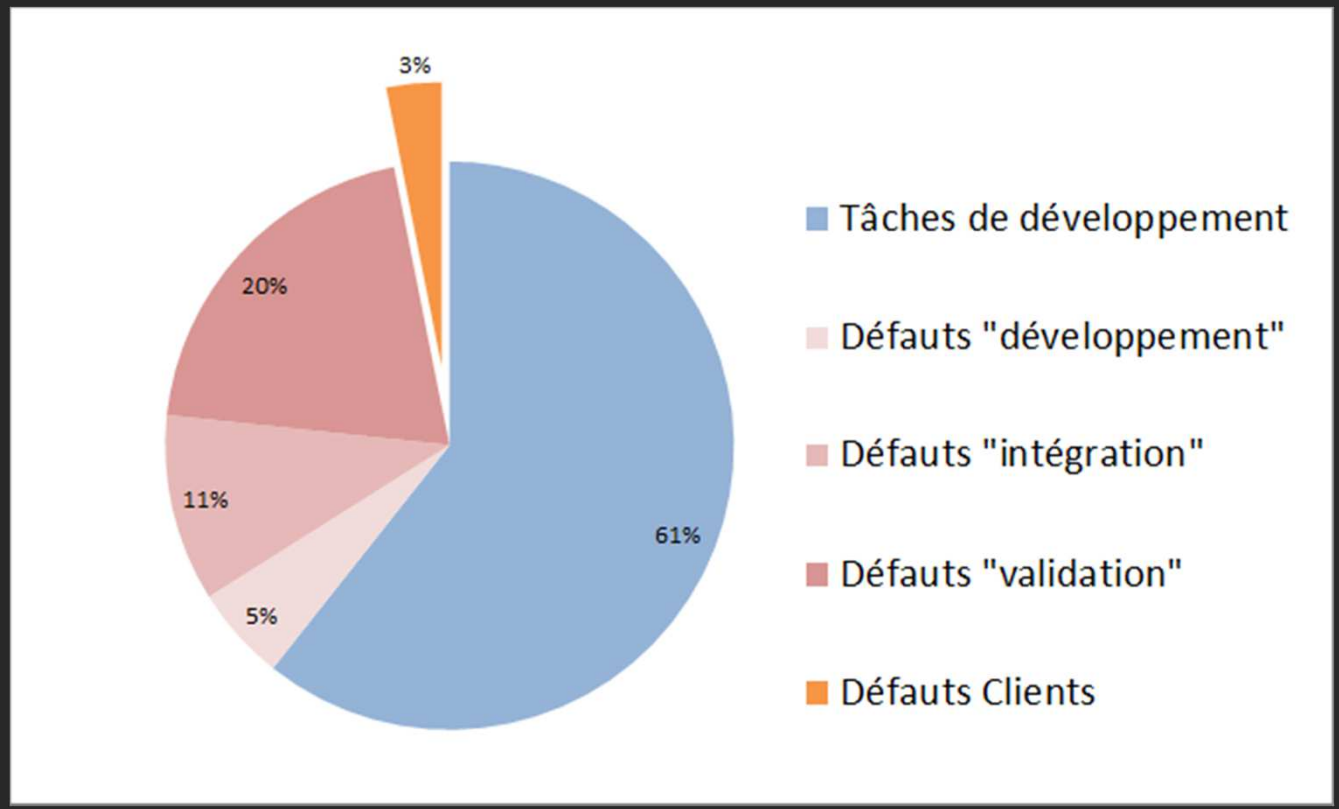
Respect du calendrier

Cohésion du planning:

- Par rapport aux engagements pris, le plus gros décalage de livraison a été d'une semaine
- Aucun *déscopage* fonctionnel n'a été réalisé pour tenir le planning
- Environ 10% de fonctions (sur 800) ont pu être modifiées/ajoutées en parallèle du développement initial.

Répartition des défauts par phase

Gain sur la qualité du système délivré



Questions ?

Laurent Py

@py_laurent

laurent.py@hiptest.net

http://hiptest.net

Raphaël Citeau

rciteau@parkeon.com

http://www.parkeon.com