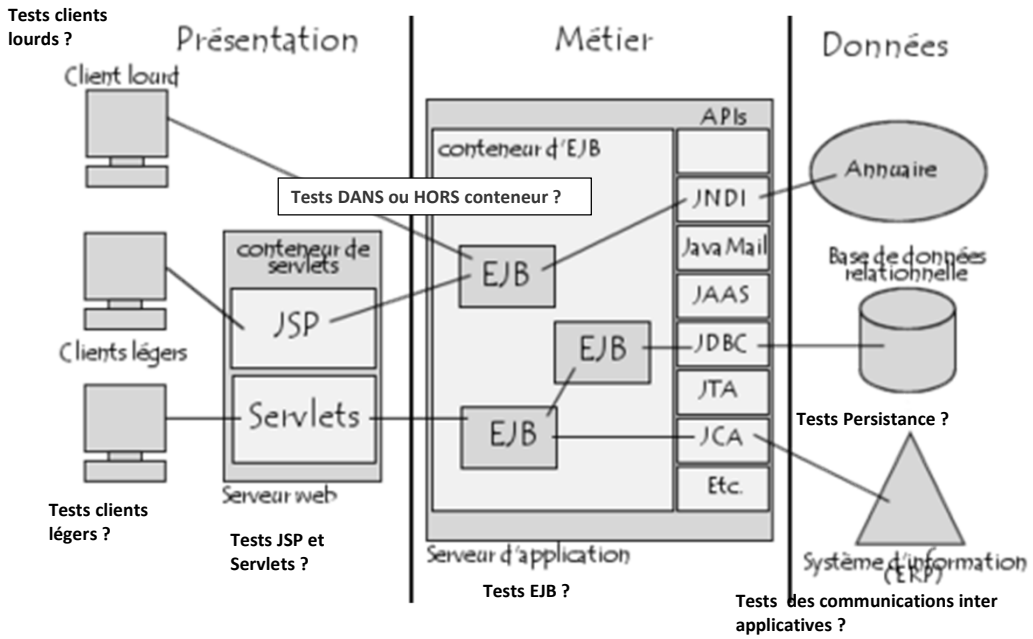
Introduction aux tests pour les applications web

Révision : Septembre 2016

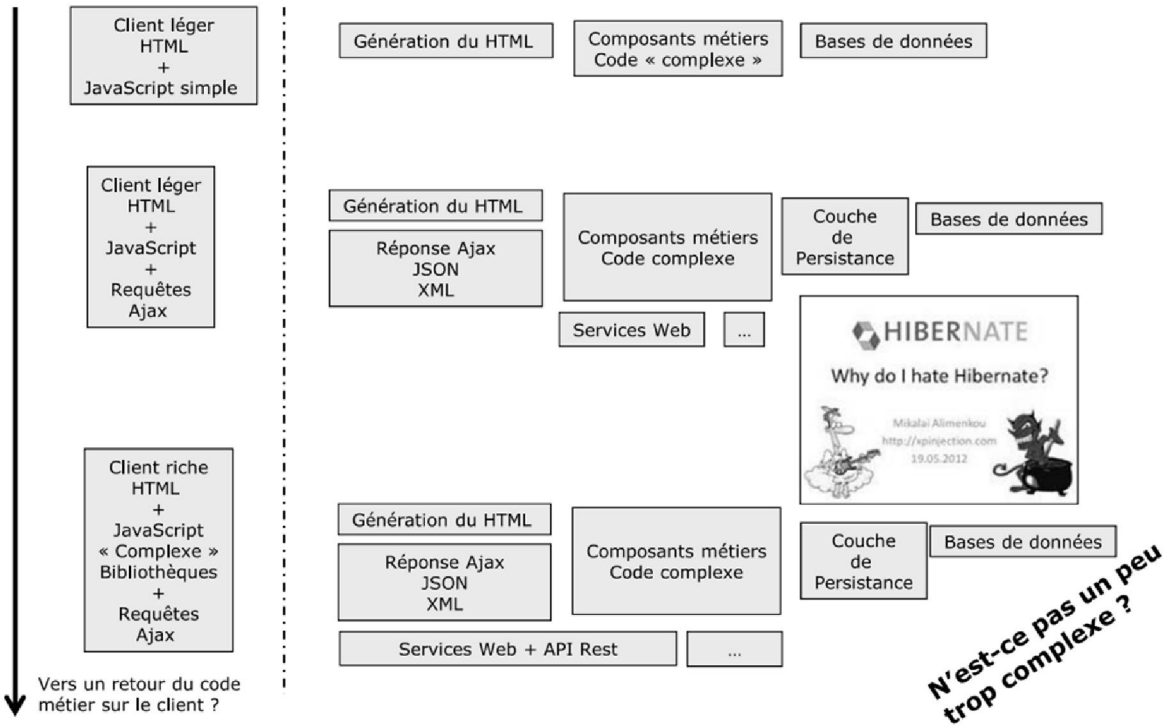
Application web ?

- Partie serveur
  - SGBDR ou SGBD **NoSQL**
  - Couche de persistance des données
  - Appels de **Web Services** et autres protocoles de communication interprocessus
  - Couches métiers
  - Couche d'interprétation des requêtes des clients et de génération des réponses (HTTP + CSS + JSON + …)
- Parties clientes
  - HTML et **HTML 5.0**
  - Code **Javascript** utilisateur et bibliothèques
  - Interactions de type AJAX avec le serveur
  - Code dans un langage propriétaire de type RIA (Flash, Flex, ActionScript)
  - **Responsive Design** (Code adaptable à tous les types de clients)
  - Différents navigateurs

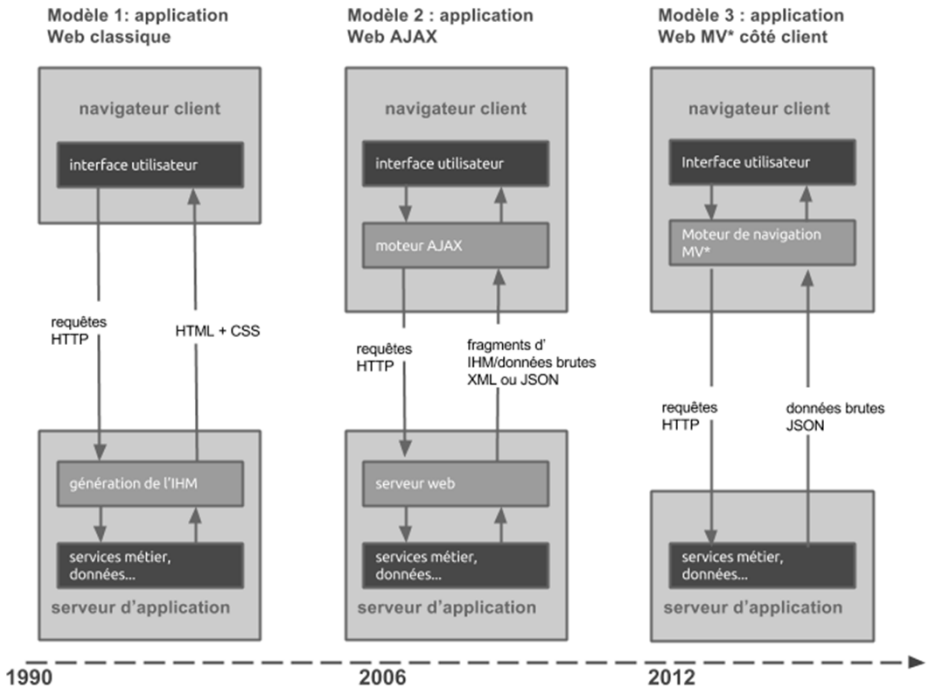
Exemple avec Java



Complexification !



Les 3 modèles principaux



Les nouvelles architectures front Web et leur impact sur les DSI – Partie 1 Posté le 29/10/2013par François Petitit

Single Page Web Application

- Face à la complexité, la mauvaise maintenabilité et parfois les piètres performances de certaines applications web une nouvelle voie semble très prometteuse :

1) La généralisation du concept AJAX et de HTML 5 pour ne développer que sur **une seule page HTML** avec des rafraichissements de zones par des requêtes sur le serveur

2) Retour du **code métier du côté client !**

Davantage de tests du code client!

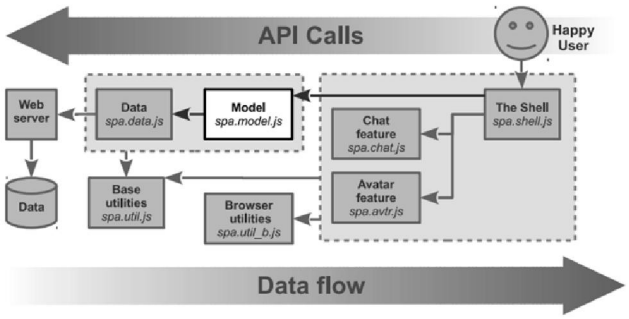
Unification des techniques de tests !

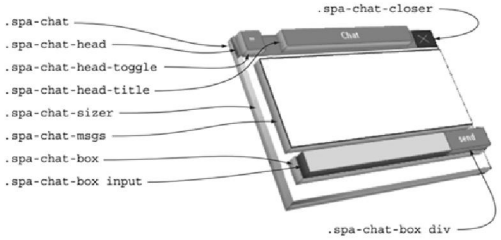
3) La généralisation de **javascript** pour développer **côté serveur !**

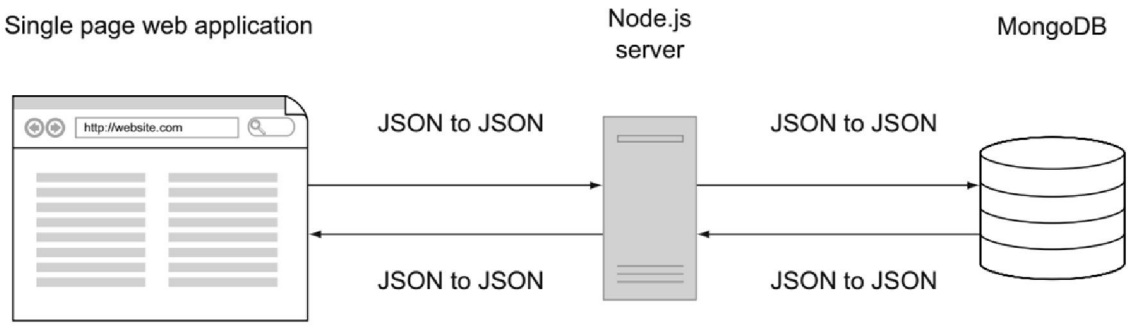
4) **Serveurs web javascript de nouvelle génération**
   - Programmation non synchrone non bloquante avec fonction callback pour gérer des milliers de sessions plus facilement.
   - Multitude d'API écrites en Javascript côté serveur pour
     - Services Web, Base de données SQL, NoSQL, Mail, ftp …
     - Communication avec le client (JSON, XML, Format d'outils)

5) L'utilisation de bases de données NON RELATIONNELLES

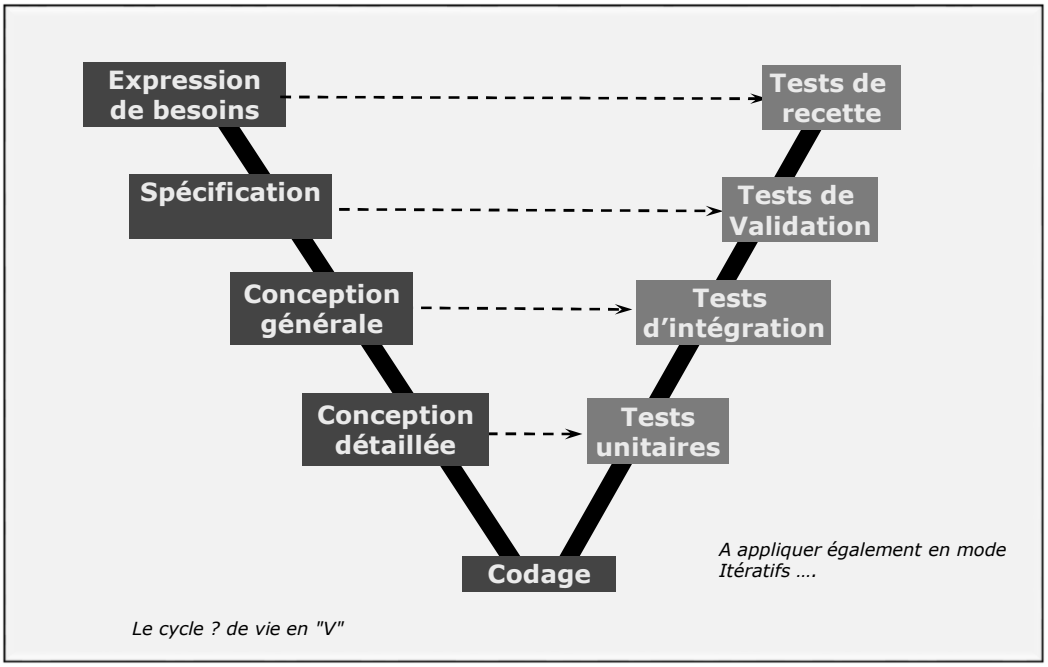
Exemple de SPWA







Les phases du test



Test : Validation ou Vérification (V&V) ?

- **Validation** : est-ce que le système fait ce qu'il doit faire ?
  - Tests fonctionnels de validation MOE
  - Tests de recette MOA

Are we doing the right thing ?

- **Vérification** : est-ce que les différentes phases de conceptions et de fabrications suivent les meilleures pratiques et recommandations ?

Are we doing the things right ?

  - Revues diverses (Exigences, Spécifications, Code, Tests …)
  - Tests d'analyse statique du code côtés serveur et client

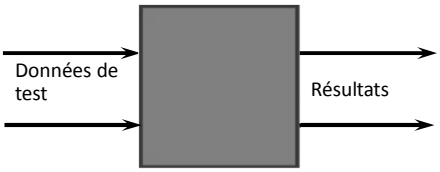
Intérêts de la V&V

- **Constat** : un logiciel peut être fonctionnellement correct mais conçu de manière complexe et non homogène
- La non qualité résultante entraînera une **explosion des temps de correction** des anomalies et rendra **l'évolution du logiciel plus difficile**.
- Il est donc souhaitable de mettre en place, en conjonction avec les tests de validation, des **tests de vérifications** qui vont mesurer la complexité, la maintenabilité du logiciel et son évolutivité.

Fonctionnel ou structurel ?

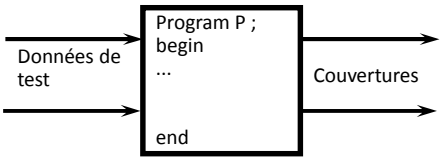
❖ Fonctionnel (ou boîte noire)

Dynamique : Exécution et test du comportement

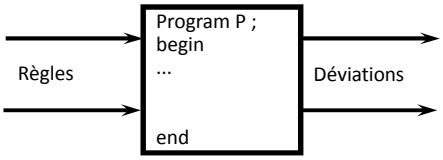


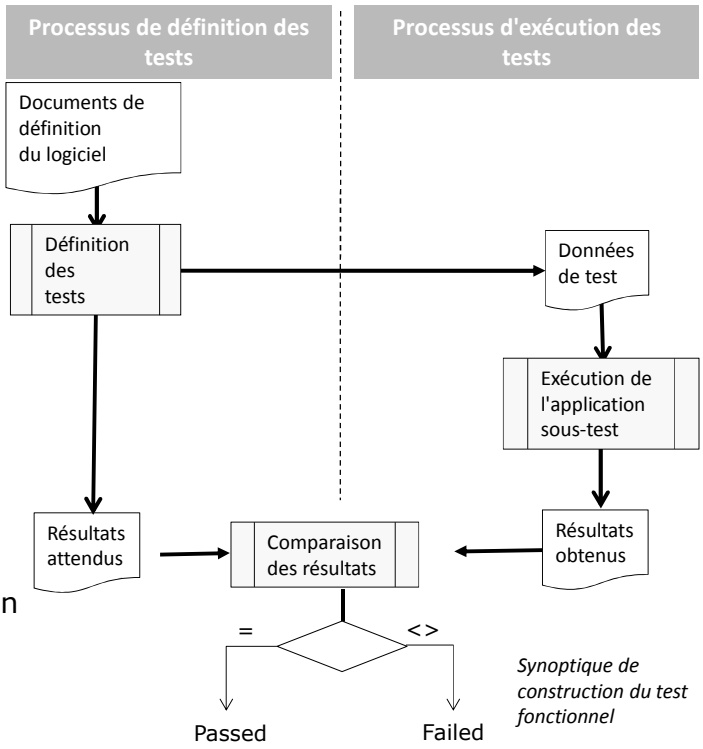
❖ Structurel (ou boîte blanche)

Dynamique : Mesure de couverture des tests fonctionnels



Statique : Examen et recherche de défauts par rapport à l'implémentation





Synoptique de construction du test fonctionnel

Tests unitaires

Caractéristiques
Sont réalisés en isolation sur chacun des composants de l'architecture du logiciel (classe, module, unité de traitement) selon la stratégie de test définie dans le plan de tests unitaires
Les tests unitaires peuvent mettre en œuvre conjointement plusieurs techniques : Test fonctionnel (boîte noire, Test structurel (boîte blanche : couverture du code, qualité de la conception et du code)
Demandent d'instrumenter le code du composant pour exécuter les tests
Ont pour référence les dossiers de conception détaillé en mode cycle en V
Produisent des résultats (preuves)

Tests d'intégration

Caractéristiques :
Le logiciel est intégré composant par composant à partir des livraisons (commit) des développeurs selon la **stratégie d'intégration définie** dans le plan d'intégration logiciel
Une **plate-forme d'intégration** est requise pour **produire les builds et converger vers la release**
Les contrôles sont réalisés dans un contexte où l'objet sous test est un assemblage de composants, mais pas encore le logiciel dans son intégralité
Les contrôles doivent porter sur **la conformité fonctionnelle des fonctions intégrées**, mais également sur la **qualité de l'architecture et du code**
Les documents de référence pour les tests sont les dossiers de conception générale et d'interface en mode cycle en V
Les tests d'intégration **produisent des résultats (preuves)**

Tests de validation

Caractéristiques
Sont réalisés selon la stratégie de validation définie dans le plan de test de validation
Démarrent avec le gel des fonctionnalités (feature freeze) sur la première beta et s'achèvent après plusieurs livraisons de builds correctifs
Ont pour référence les dossiers d'exigences et de spécifications. Liens de traçabilité.
Mettent en œuvre conjointement plusieurs techniques **: tests des features, scénarios d'utilisation, utilisabilité, performance, …**
Produisent des résultats (preuves) et des indicateurs de couverture
Conduisent à une décision de livraison (procès verbal)

Fonctionnel ou non fonctionnel ?

La norme nouvelle norme ISO/29119
La norme ISO/CEI 9126

- **Capacité fonctionnelle**
  Il s'agit d'un ensemble d'attributs permettant de vérifier si le logiciel répond aux besoins fonctionnels exprimés. On y retrouve notamment les notions de pertinence, d'exactitude, d'interopérabilité, de sécurité et de conformité.
- **Fiabilité**
  Il s'agit ici de décrire les attributs permettant d'évaluer l'aptitude d'un système à maintenir son niveau de service : tolérance aux pannes, conditions de remise en service, maturité…
- **Facilité d'utilisation**
  On retrouve dans cette caractéristique un ensemble d'attributs permettant de caractériser l'effort nécessaire à un utilisateur potentiel pour utiliser le système : facilité de compréhension, d'apprentissage, d'exploitation ou encore pouvoir d'attraction.
- **Rendement ou efficacité**
  Cette caractéristique permet de qualifier le rapport entre le service rendu par le logiciel et les efforts qu'il faut entreprendre pour le faire fonctionner ((quantité de ressources utilisées notamment).
- **Maintenabilité**
  Il s'agit ici d'évaluer les possibilités de faire évoluer le système dans le cas de nouveaux besoins : facilité d'analyse, de modification, stabilité et testabilité de la solution.
- **Portabilité**
  Cette caractéristique décrit la possibilité de transférer le logiciel d'une plateforme à une autre, et les efforts nécessaires pour le faire : facilité d'adaptation et d'installation, coexistance, interchangeabilité.

Test de performance

P <-- F (x1, x2, x3, …, xn)

Mesure Modèle Paramètres influents sur la performance

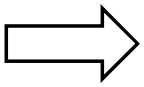
- Test de performance : ensemble de tâches visant principalement deux objectifs :
  - Prouver le niveau de performance réellement atteint par l'application en termes de capacité à monter en charge et de robustesse
  - Régler et optimiser la plateforme technique pour obtenir la performance souhaitée

Tests et application web

- **Tests du côté serveur**
  - Tests du code métiers
  - Tests de persistance des données
  - Tests de production du flux HTTP
  - Tests des services web internes et externes
  - Tests de la qualité du code
  - Tests de déploiement à chaud/froid
  - Tests de performance du serveur

- **Tests du côté client**
  - Tests utilisateurs (cas d'utilisation, scénario métier)
  - Tests du code javascript
  - Tests de la qualité du code javascript
  - Tests multi clients et multi navigateurs
  - Tests de robustesse de l'IHM (Interface Homme Machine)
  - …

Quels types de test ?
Avec quels efforts de test ?
Avec quels processus ?
Avec quelle organisation ?
Avec quels outils ?
Pour quel coût ?

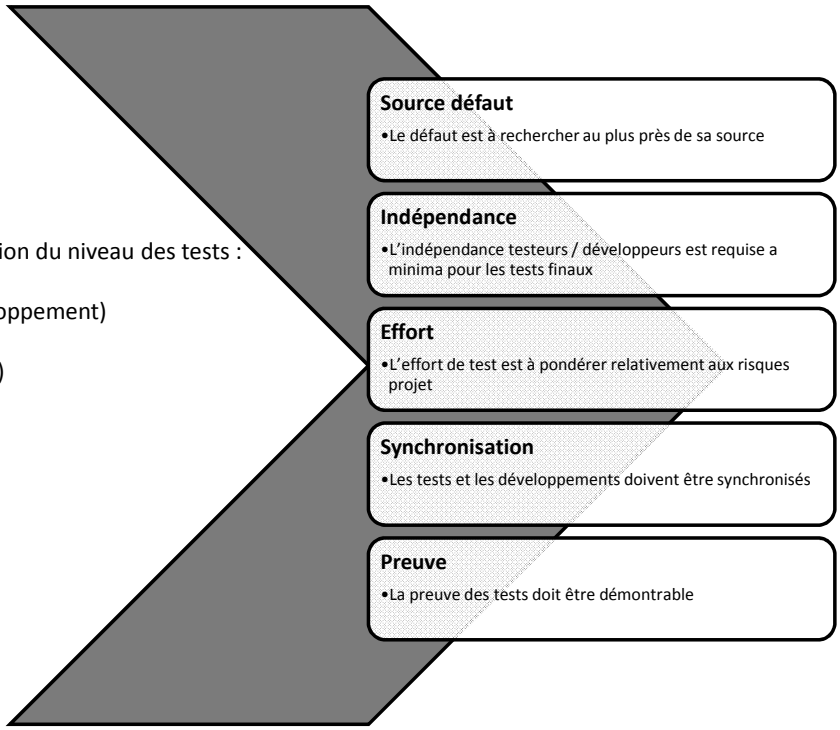


Stratégie de tests

Les 5 fondements de la stratégie

A ajuster en fonction du niveau des tests :

- Unitaires (Développement)
- Intégration
- Validation (MOE)
- Recette (MOA)



Réussite ou échec ?

- **Constat : plus de 50% des projets d'automatisation des tests de validation/recette sont des échecs !**

- **Pourquoi ?**

  – Pas de processus de tests existants
  – Outils inadaptés avec les processus de tests existants
  – Manque de compétences techniques en interne
  – Syndrome de l'outil miracle (Robot de tests) (A fool with a tool is still a fool !)
  – Pas de projet d'automatisation des tests parfaitement identifié et reconnu (Budget, Ressources, …)
  – Sous-estimation de la charge de travail (implémentation et maintenance), et du coût de l'investissement en qualité pour diminuer le coût de la non qualité
  – L'automatisation des tests de validation n'est pas vu comme un métier à part entière
  – Manque de considération pour les tests. Equipes de tests parfois vue comme un centre de coût, improductif …
  – Manque de réflexion sur les tests en amont