OCTO Technology
There is a better way

Désendettement de code Javascript : le guide anti-crise

Michael AKBARALY
@Chehcheucheh
makbaraly@octo.com

François PETITIT
@francoispetitit
fpetitit@octo.com

PARIS WEB

Atelier Paris Web 2015

© OCTO 2015

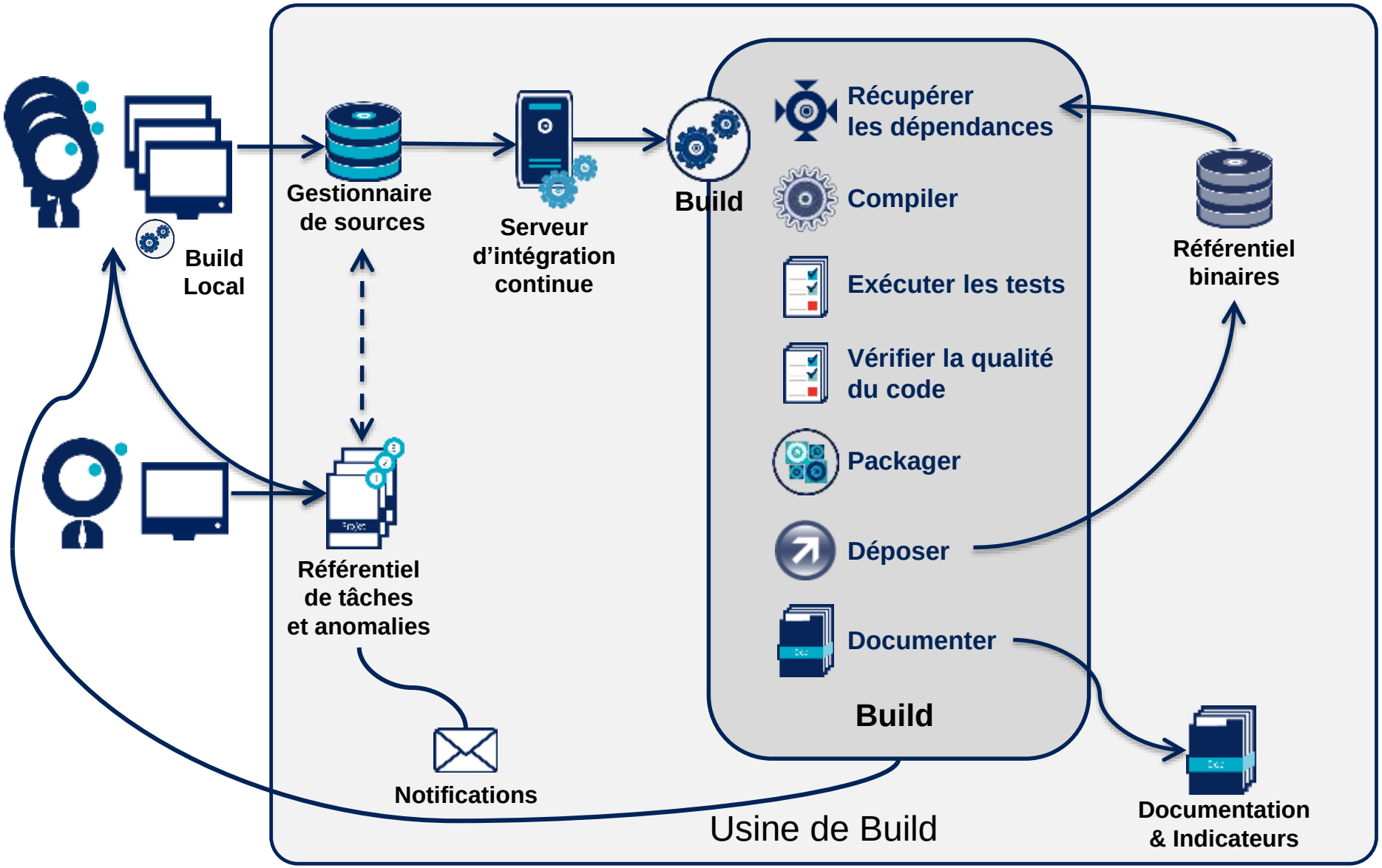
50, avenue des Champs-Elysées
75008 Paris - FRANCE

Tél : +33 (0)1 58 56 10 00
Fax : +33 (0)1 58 56 10 01
www.octo.com

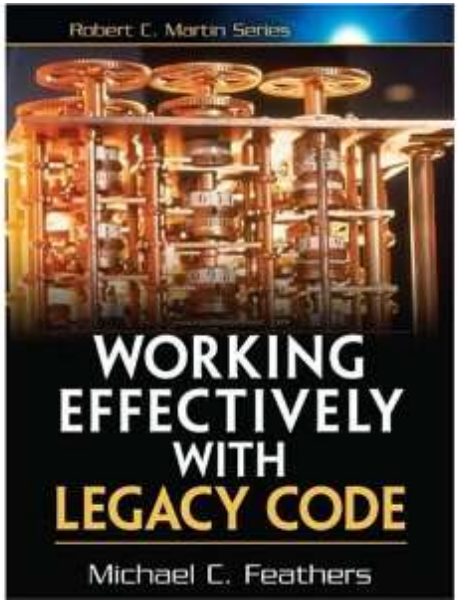
- Michael AKBARALY
- François PETITIT

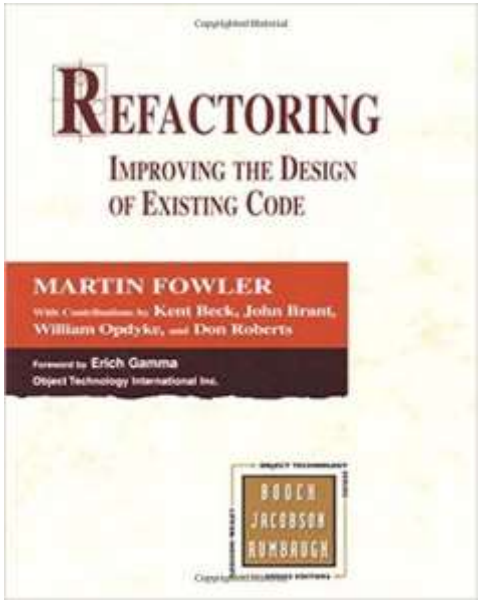
- OCTO Technology
  - Consultants chez Canal+, FranceConnect…
  - Ref card « Tests sur tous les fronts »
  - http://blog.octo.com

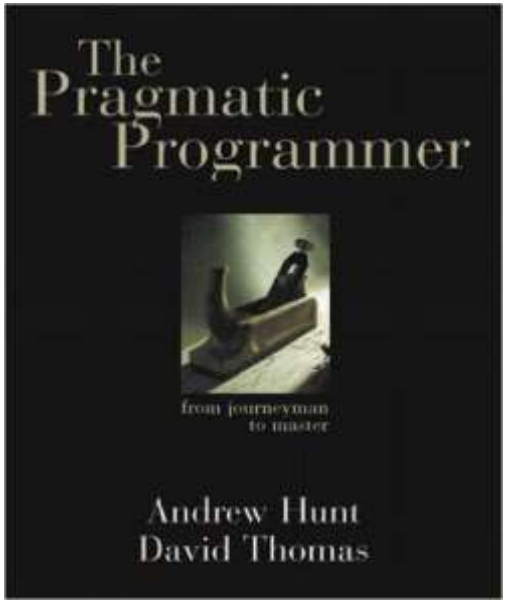
Quand on commence un projet de zéro…



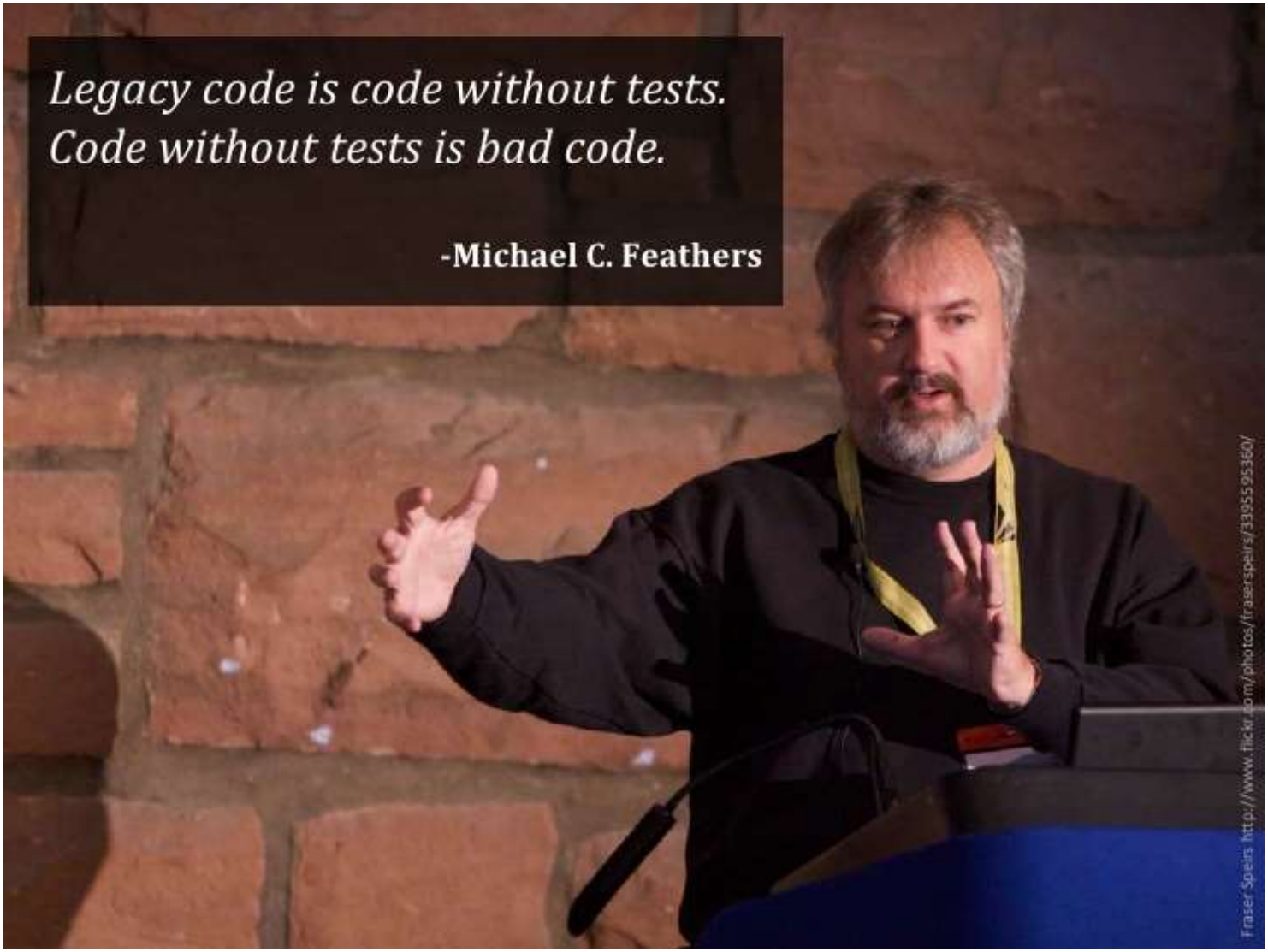
Travailler sur du legacy : les livres de référence







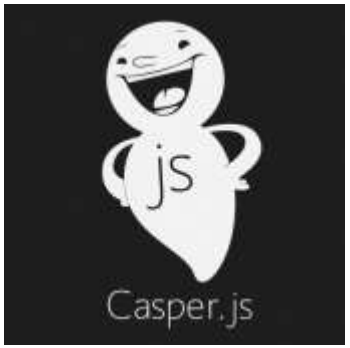
Du code ‘legacy’, c’est quoi?



Eco-système Javascript: vous n’avez pas d’excuse!

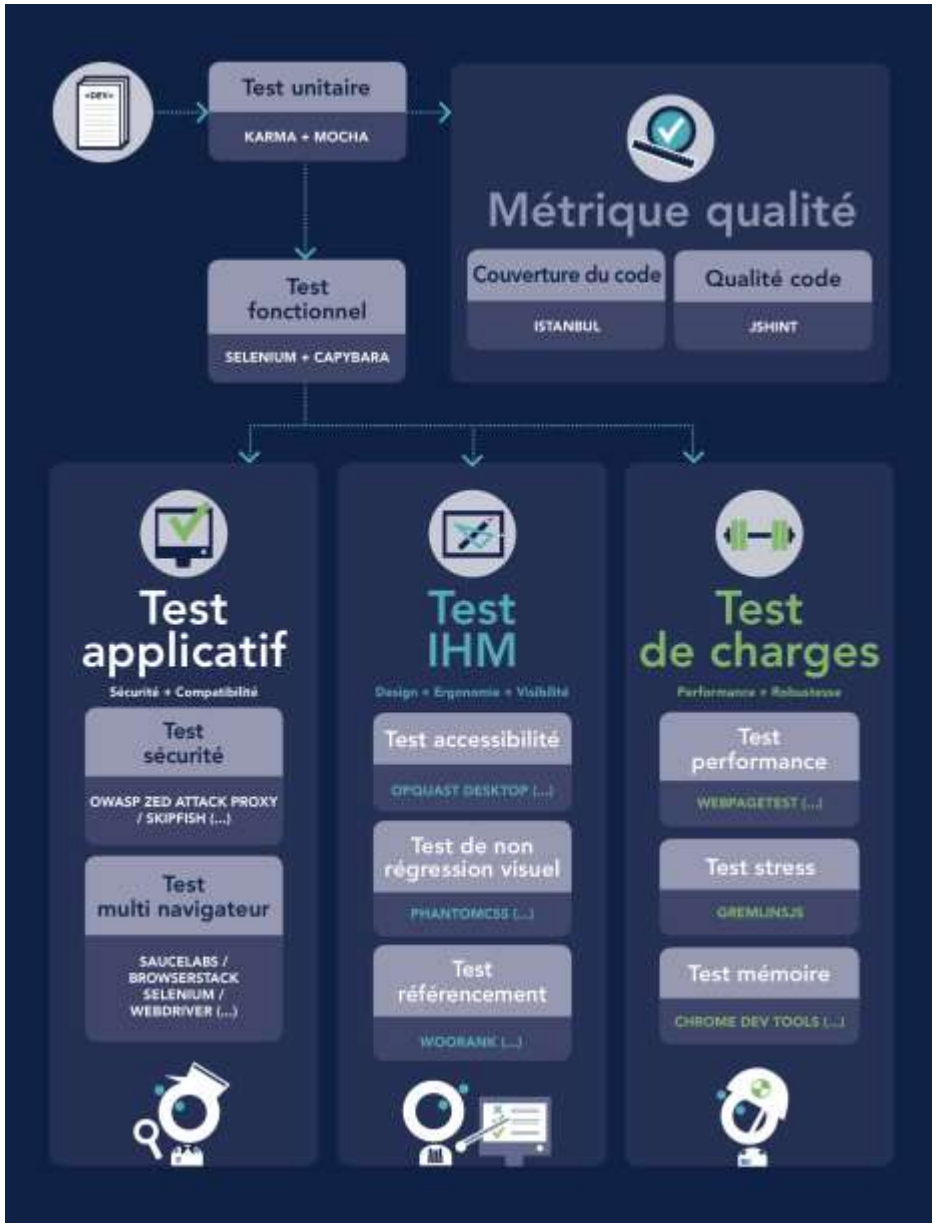
Istanbul

supertest



Travis CI

Types de tests



http://bit.ly/pw-legacy

https://github.com/octo-web-front-end-tribe/ka-ching

Démarche pour un audit express

- Les grandes étapes
  - Faire tourner l'application
  - Regarder le code à la main pour voir la tête du code
    - Code non formaté !
    - On n'a pas de test !
    - On n'a pas de gestion des ressources front-end !
    - …
  - Faire une analyse statique
    - Détection de code dupliqué
    - Calcul de la complexité cyclomatique
    - …

- Objectif
  - Visualiser certaines métriques sur le code : couverture par les tests, complexité cyclomatique, nombre de lignes de code…
- Comment
  - Avec Plato.js :
    - https://github.com/es-analysis/plato
  - Npm install –g plato
  - Côté back-end :
    - plato -d report -x node_modules/ index.js
  - Côté front-end :
    - plato -r -d report -x node_modules/ app/scripts

Améliorations – axe qualité du code

Modification	Avantages	Criticité	Complexité
Améliorer la couverture de tests côté API puis front-end	• Améliorer la maintenabilité	+++	+++
Mettre en place des normes de formatage	• Améliorer la lisibilité du code • Permettre d'avoir des diffs Git utiles	+++	+
Mettre en place un outil de linting	• Eviter des erreurs de code détectables par des outils	+++	+
Supprimer les duplications de code	• Eviter des régressions en oubliant de faire évoluer chaque version	++	++
Supprimer le code mort	• Améliorer la lisibilité du code	+	+

Améliorations – axe build

Modification	Avantages	Criticité	Complexité
Mettre en place un build automatisé Gulp pour le front	• Possibilité d'utiliser des preprocesseurs CSS compilés (compass…) • Live-reload sur le poste de dev • Outillage JS dédié pour optimisation des ressources (plug-ins Grunt : https://github.com/yeoman/grunt-usemin,)	++	++
Automatiser les tests	• maintenabilité	+++	+

Améliorations – axe run

Modification	Avantages	Criticité	Complexité
Optimiser les ressources statiques front-end	• Améliorer les performances	++	++
Déployer les ressources statiques en-dehors du serveur NodeJS (serveur HTTP, CDN…)	• Améliorer les performances • Décharger le serveur d'application	+	++

Feuille de route

- Formater le code
- Mettre en place un outil de « linting » de code
- Mettre en place une couverture de tests minimale
  - D'abord côté API
  - Puis côté front-end
- Supprimer le code mort
- Supprimer les duplications de code
  - En vérifiant la non-régression via des tests unitaires

Formatage de code JavaScript

- Objectif
  - Rendre le code plus facile à lire
  - Avoir des diffs utiles

- Comment faire
  - Un outil en ligne de commande : JS-Beautify : https://www.npmjs.com/package/js-beautify
    - Npm install –g js-beautify
    - js-beautify -f ./app/modules/**/* -r

Linting

- Objectif
  - Détecter des erreurs de code potentielles (ex: inner-declaration,…)
  - Mettre en place des règles de code en plus du formatage

- Comment
  - Mettre en place ESLint
    - http://eslint.org/
  - Intégrer ESLint au build

Tester l'API en boîte noire

- Objectif
  - Avoir un harnais de tests de non-régressions sur l'API pour pouvoir la faire évoluer sereinement
- Comment
  - Mettre en place des tests de non-regression HTTP avec SuperTest (https://github.com/visionmedia/supertest)
  - Bouchonner les données de base
  - Ajouter ces tests au build (attention : nécessite de déployer l'application)
- Le code
  - Branche « api-integration-tests »
  - https://github.com/octo-web-front-end-tribe/ka-ching/tree/api-integration-tests

Analyse dynamique : détecter le code mort côté back

- Objectifs
  - Mieux visualiser comment fonctionne l'application au runtime
  - Supprimer le code non utilisé
  - Visualiser le code non testé

- Comment ?
  - Istanbul pour visualiser la couverture par les tests :
    - https://github.com/gotwarlost/istanbul
  - Istanbul-middleware pour instrumenter à l'exécution
    - https://github.com/gotwarlost/istanbul-middleware

Ajouter de la non régression sur l'IHM en mode boîte noire

- Objectifs
  - Poser un harnais de test pour pouvoir ensuite modifier le code front-end
- Comment
  - Avec PhantomCSS
    - https://github.com/Huddle/PhantomCSS
- Exemple
  - https://github.com/octo-web-front-end-tribe/ka-ching-phantomcss

Supprimer le code dupliqué

- Objectif
  - Détecter le code dupliqué
  - Supprimer sans risque les duplications

- Comment ?
  - JSInspect pour analyser le code et détecter les duplications
  - Mise en place de tests unitaires pour sécuriser les appels aux fonctions dupliqués
  - Refactoring pour supprimer les doublons
  - S'assurer que les tests fonctionnent toujours

MERCI

- Atelier Paris Web 2015