



Qualité du code source et intégration continue

XP DAY 2009 – 26 mai

*Erwan Alliaume
Nicolas Le Coz*



Sommaire de la présentation

- ✓ *La démarche*
- ✓ *Intégration continue*
- ✓ *Comment faire du code de qualité ?*
- ✓ *Démonstration*
- ✓ *Guide pratique*

Vos interlocuteurs ...

- **Erwan Alliaume (Xebia)**

- ▶ Architecte Junior - Expert Java / JEE
- ▶ Référent technique multi-projets
- ▶ Actuellement sur un projet à plusieurs 10aines de développeurs
- ▶ Blogueur à ses heures perdues

- **Nicolas Le Coz (Xebia)**

- ▶ Expert technique polyvalent
- ▶ Intervient sur des projets Agiles (Scrum & XP)
- ▶ Mise en place de bonnes pratiques et automatisation de l'assurance qualité des développements

<http://blog.xebia.fr>





La démarche

www.xebia.fr / blog.xebia.fr

www.xebia.fr / blog.xebia.fr

Problématique : une démarche d'amélioration

■ **La démarche**

- ▶ *Pourquoi écrire du code de qualité ?*
- ▶ *Qu'est ce que du code de qualité ?*
- ▶ *Comment écrire du code de qualité ?*
- ▶ *Comment mesurer la qualité ?*
- ▶ *Comment identifier les actions d'amélioration ?*
- ▶ *Comment valider ces actions ?*
- ▶ *Comment itérer ?*
- ▶ *Jusqu'où itérer ?*
- ▶ *Comment s'organiser pour faire de la qualité ?*

Pourquoi écrire du code de qualité

- **Code de mauvaise qualité**

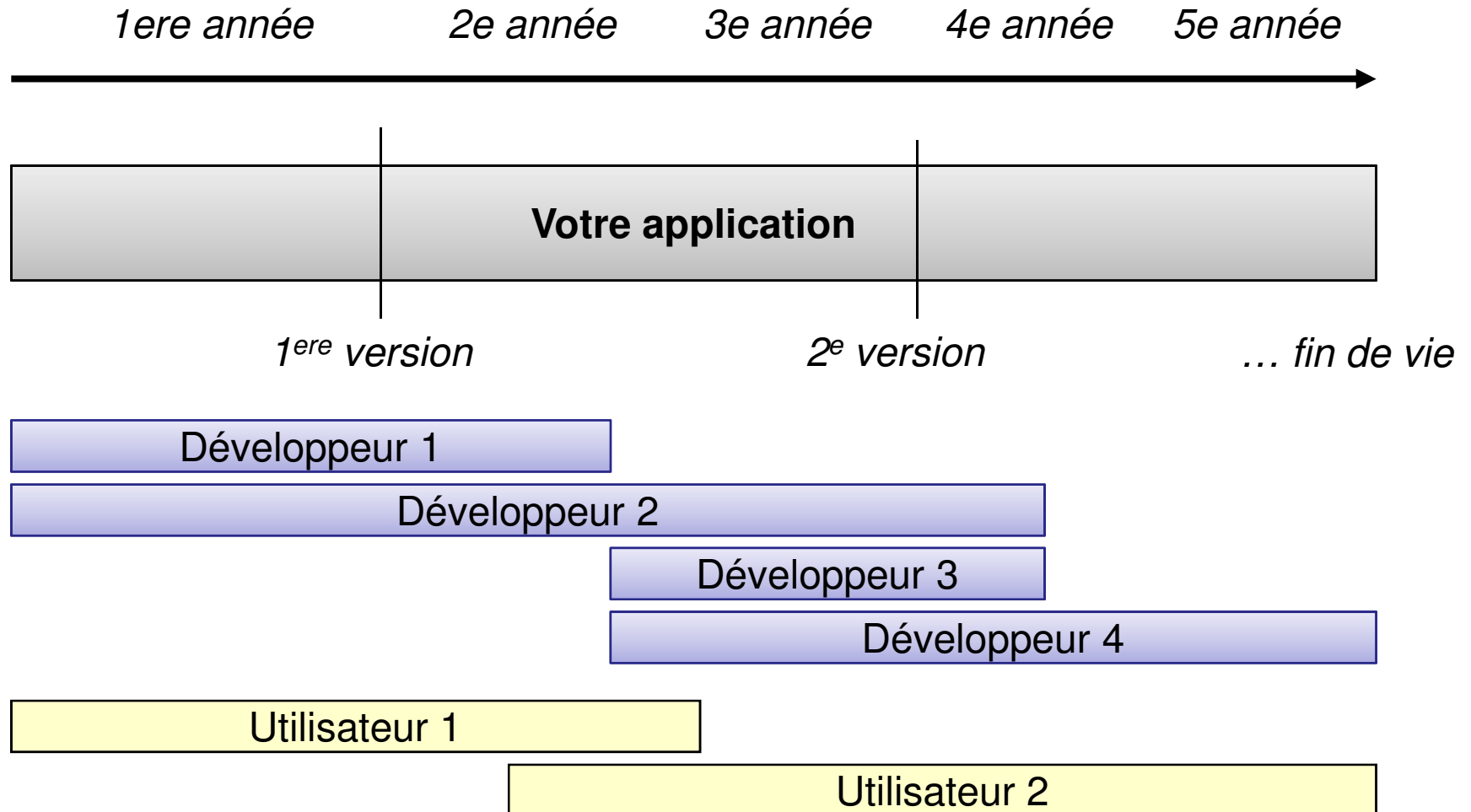
- ▶ *Coute plus cher à maintenir*
- ▶ *Augmente la dette technique*
 - ▶ *Peut conduire à la réécriture d'une application*

- ▶ **Ecrire du code de qualité ...**

- ▶ *Ce n'est pas une tâche spécifique, ça en fait partie*
 - ▶ *Fixer une étagère droite*
- ▶ *Ne doit pas prendre plus de temps que du mauvais code*
 - ▶ *Si c'est le cas, revoyez vos règles*
- ▶ *C'est l'affaire de tous !*
- ▶ *Ce n'est pas tout le temps possible*
 - ▶ *C'est l'exception qui confirme la règle*
 - ▶ *Mais le manque de temps n'est pas un excuse valable !*



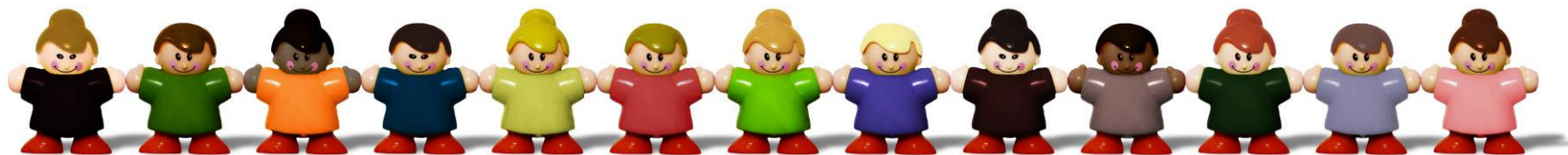
Durée de vie de votre application





Intégration continue

Un outil « indispensable » pour



... travailler ensemble !

Intégration continue

Hé Bob t'en est ou ?

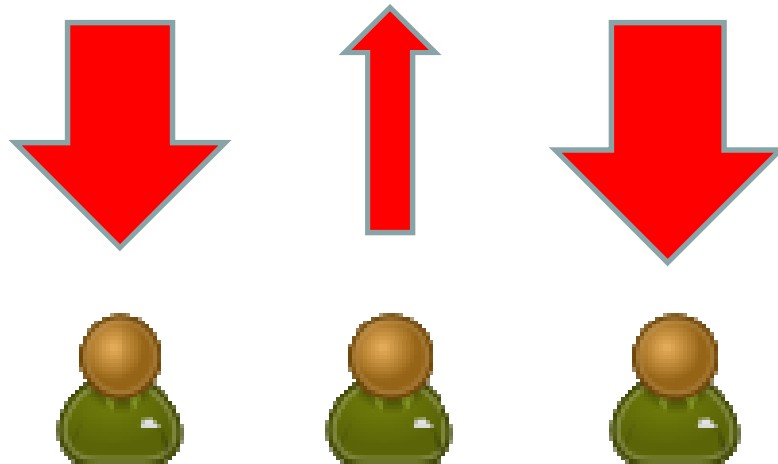
Attends, attends j'ai presque terminé?

Grouille toi !

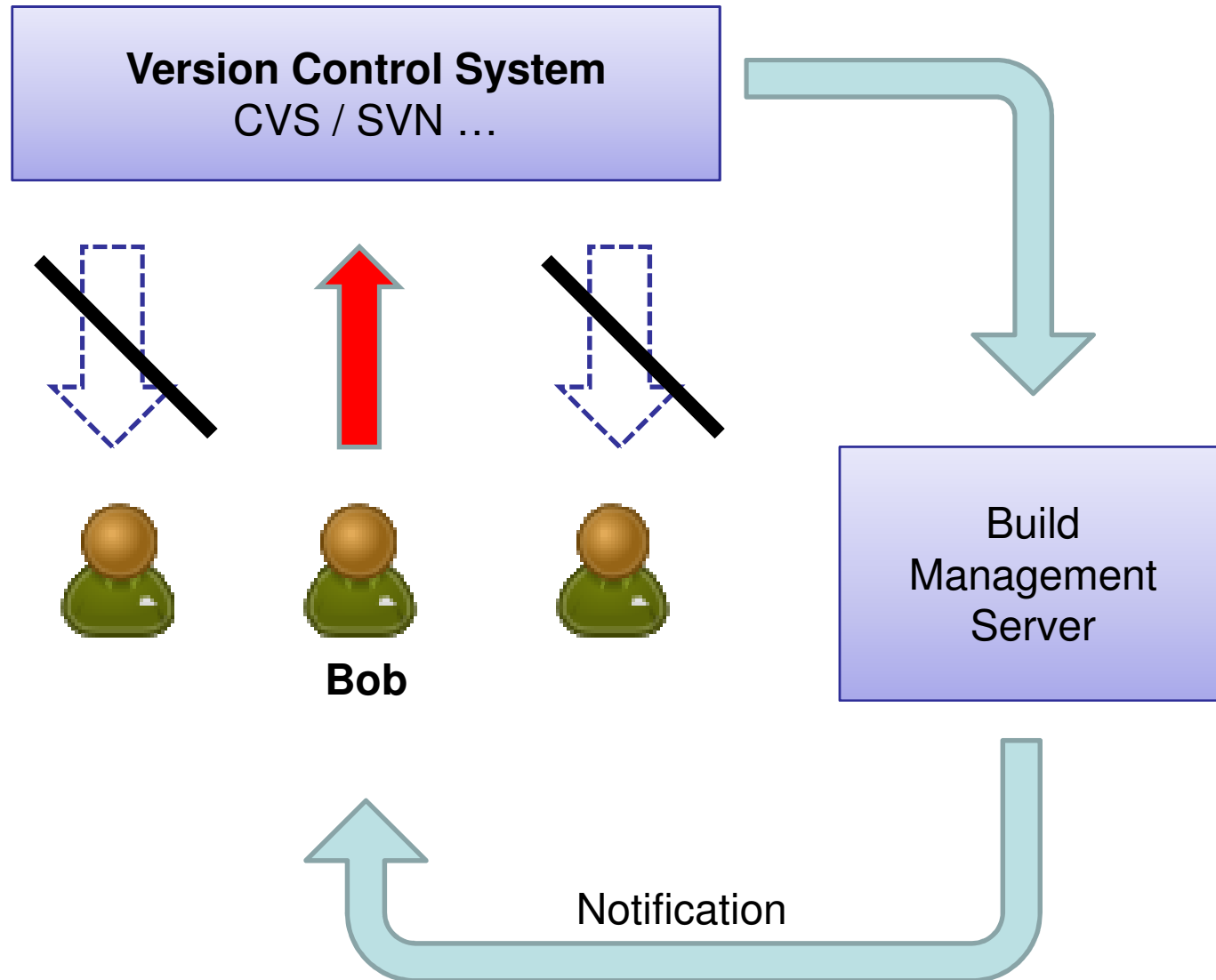
Ok, c'est bon je commit ...

QUI A TOUT ENCORE TOUT CASSE ?

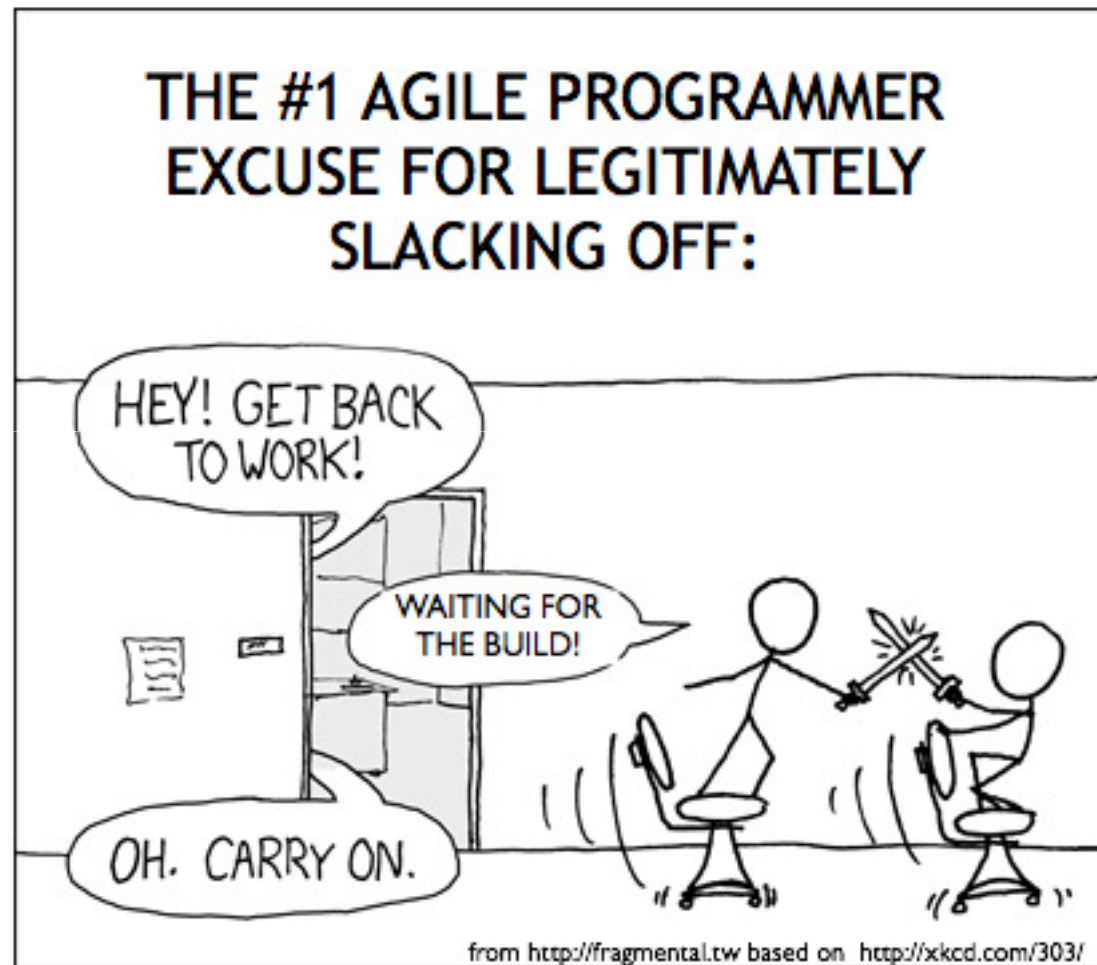
Version Control System
CVS / SVN ...



Intégration continue



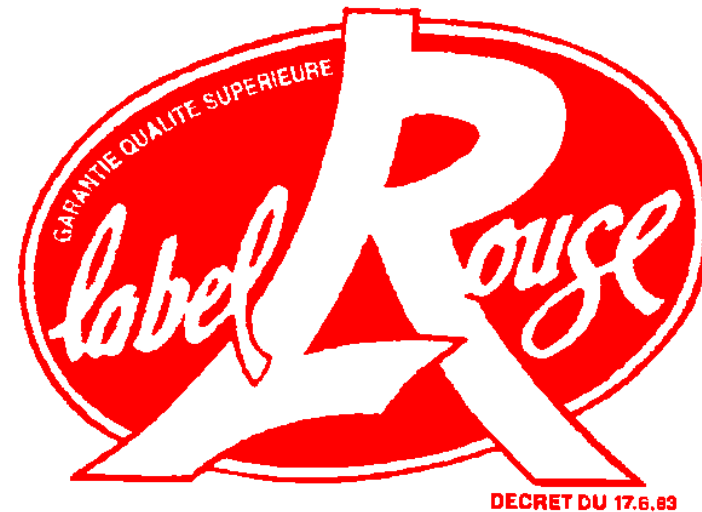
Intégration continue





La qualité du code source

*Un artisanat
qui possède ses
contrôles qualités*





Comment faire du code de qualité

- *Pourquoi ?*
- *Problématique*
- *Solutions*
- *Mise en place*



La qualité du code source



- ***Pourquoi ? ... continuons à être agile***
 - ▶ *Réactif au changement*
 - ▶ *A long terme le cout de développement*
 - ▶ *Le bien être des équipes (motivation, communication)*
 - ▶ *Un séance code review doit se focaliser sur des problématiques de conception plutôt que sur des violations de normes ...*

Problématique : Qu'est ce que du code de qualité (1/2)

- ***Le code doit révéler les intentions d'un développeur (Kent Beck « Implementation Pattern »)***
- ***Maintenabilité***
 - ▶ *Granularité (classes ou méthodes trop longues)*
 - ▶ *Duplication de code*
 - ▶ *Code mort*
 - ▶ *Dépendances entre artefacts*
 - ▶ *Pattern de conception tel que IoC (Inversion of Control)*
 - ▶ Solution : Outillage et configuration de son IDE (Eclipse)
- ***Performance***
 - ▶ *Respect des API & Frameworks*
 - ▶ Solution : compétence technique



Problématique : Qu'est ce que du code de qualité (2/2)

- ***Lisibilité du code***
 - ▶ *Formatage*
 - ▶ Solution : Automatiser sous Eclipse avec le `format.xml`
 - ▶ *Convention de nommage*
 - ▶ Solution : S'appuyer sur les conventions existantes
 - ▶ *Artefacts avec des noms fonctionnellement explicites*
 - ▶ *Documentation, nécessaire et suffisante*
 - ▶ Solution : Revue de code

- ***... Il existe de nombreux autres aspects de qualité, appliquons le principe KISS (Kept It Stupid Simple)***
 - ▶ *Fixons-nous quelques critères prioritaires de qualité*



Problématique : Comment s'organiser autour de la qualité

- ***Faire de la qualité part souvent de bonnes intentions, mais attention :***
 - ▶ *Le code parfait qui ne sera jamais modifié n'existe pas*
 - ▶ *Cela peut mobiliser beaucoup d'énergie et de personnes*
 - ▶ *Cela peut paraître un frein à la productivité et avoir l'effet inverse souhaité*



Solutions, les processus (1/3)

- **Convention de programmation**
 - ▶ *Simple et courte style un cheat-sheet (une page)*
- **Mise en place de l'environnement de développement**
 - ▶ *Automatiser toutes les activités répétitives (vérification & rapports, intégration dans l'IC)*
- **Revue de code**
 - ▶ *Doit se focaliser sur le métier et la conception*
 - ▶ *Pas de « bruit » style la règle 3 de qualité est en échec*
- **Post mortem de mise en production & Test (unitaire, fonctionnel, concurrence, performance)**
 - ▶ *Des règles doivent être posées pour bloquer des bugs*
 - ▶ *Par exemple des connections en base de données non libérés*



Solutions, les outils d'analyse de code (2/3)

- **PMD** ★★ ★
 - ▶ *Nombreuses règles (Mc Cabe) à prioriser*
 - ▶ *Détection de copier/coller 😊*
 - ▶ *Personnalisation de règle*
- **CheckStyle** ★★ ★
 - ▶ *Idem, sauf pas de détection de copier/coller*
- **Simian** ★
 - ▶ *Détection de copier/coller*
 - ▶ *Intégration à CheckStyle*
- **JavaNCSS** ★
 - ▶ *Métrique de Mc Cabe et commentaires*
- **JDepends** ★★
 - ▶ *Dépendance entre module selon votre règle de nommage (package)*



Solutions, les outils de consolidation (3/3)

- **Findbugs** ★★☆☆
 - *Autre outils d'analyse statique de code*
 - *Basé sur le byte code*
- **Sonar** ★★★★★
 - ▶ *Excellent rapport, « zoomable » (lisible pour les décideurs & dev)*
 - ▶ *Evolution de la qualité au fil des versions de l'application*
- **XRadar** ★
 - ▶ *Précurseur*
 - ▶ *Agrège le meilleur des outils open source*
- **Crucible** ★★☆☆
 - ▶ *Outil Atlassian, parfaitement intégrer à JIRA, Confluence*
 - ▶ *Assiste les activités de code review*
 - ▶ *Payant (licence Atlassian)*



Mise en place de la demarche dans votre projet

1. Pré-requis

- *Avoir un build d'intégration continue rodé pour la compilation et les tests unitaires avec rapports*

2. Si c'est la première fois, choisir le moment opportun

- *ie. Ce n'est pas forcément en début de projet*

3. Choisir vos règles

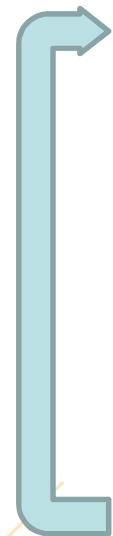
- *Etre modeste, choisir quelques règles les + importantes*

4. Mise en place de l'outillage dans d'IC puis dans l'IDE

5. Refactorer le code de mauvaises qualités

6. Motiver les équipes à travailler sur ce refactoring

7. Réitérer à partir du point 3



Choisir les règles (1/2)

■ **Philosophie**

- ▶ *Etre modeste, choisir quelques règles (une vingtaine)*
- ▶ *Ne pas effrayer les développeurs*
- ▶ *Ne pas affoler les compteurs pour être pris au sérieux*
 - ▶ *Mettre des seuils lâches au départ, les resserrer ensuite*

■ **Nos règles préférées**

- ▶ *Complexité cyclomatique (PMD & CheckStyle)*
- ▶ *Méthode et classe trop longue (PMD & CheckStyle)*
- ▶ *Duplication de code (PMD CPD Copy Paste Detection)*
- ▶ *Code mort (peu d'outil, idéal revu de code & refactoring)*



Choisir les règles (2/2)

- **Autres règles intéressantes**

- ▶ *Efferent coupling : trop d'import*
- ▶ *Trop de méthodes dans une classe*
- ▶ *Trop de paramètres dans une classe*
 - ▶ *Problèmes de conception?*
- ▶ *Close resource (Connection, Statement, ResultSet)*
- ▶ *Pas de new Thread (WebApp)*
- ▶ *Bloc catch vide*
- ▶ *JSP : pas de scriptlet*

- **N'oubliez pas de régler les seuils et les priorités !**



Mise en place dans l'intégration continue

- ***Intégration au build facilité***
 - *Maven, Hudson*
- ***Prévoir une machine dédiée***
 - *La plus puissance possible*
 - *Si possible, séparer le serveur IC du serveur de métriques qualités*
- ***Configuration du poste de développement***
 - *Intégration des outils dans l'IDE*
 - *Formation des développeurs sur les Best Practices*
 - *Ne pas recompiler tout le projet / DL les SNAPSHOTS*
 - *Ne pas désactiver les outils / communiquer sur les problèmes*



Refactorer

- ***Trouver des patterns de résolution par violation***

- ***Exemple 1***

- *Violation : Conditional complexity*
 - *Pattern : faire du polymorphisme*

- ***Exemple 2***

- *Violation : Méthode trop longue*
 - *Pattern : extract method (assisté par l'IDE)*

- ***Exemple 3***

- *Violation : Trop de méthodes*
 - *Pattern : extract classes (assisté par l'IDE)*

- ***Exemple 4***

- *Violation : fort couplage (trop d'imports)*
 - *Pattern : re-conception en plusieurs classes*





Démonstration



Guide pratique

Motiver les équipes de développement (1/3)

- ***Ne pas noyer les équipes***

- ▶ *Faire comprendre une règle avant de la mettre en place*
 - ▶ *Commencez simple, customisez vos règles !*
 - ▶ *Concentrez vous sur les 'hotspots' pour augmenter plus rapidement les stats*
- ▶ *Adopter une approche itérative*
 - ▶ *Augmentez le nombre de règles progressivement*
 - ▶ *Devenez de plus en plus strictes sur les règles passées*
 - ▶ *Warning -> Error / Hook Svn ...*
- ▶ *Motiver les équipes plutôt que réprimander*
 - ▶ *Féliciter plutôt que blâmer (médailles vs stats Excel)*



Motiver les équipes de développement (2/3)

▶ **Attention à ne pas vous décrédibiliser**

- ▶ *Trop de règles tuent les règles*
- ▶ *Faire attention à ne pas impacter la productivité*
 - ▶ *Limitez le nombre d'outils*
 - ▶ *Automatisez vos rapports*
 - ▶ *A la place, nommez un mr qualité (un homme de confiance) par équipe*
 - ▶ *Chargé de regarder les stats et de demander les corrections*

Restez cohérent : la qualité ne doit pas être un frein

- ▶ *Rester à l'écoute des grief des équipes de dev*
 - ▶ *Faites vivre vos règles au besoin*



Motiver les équipes de développement (3/3)

- ***Ne pas faire cavalier seul : trouvez des alliés***
 - ▶ *Faites comprendre votre démarche aux chefs d'équipes*
 - ▶ *C'est à eux de manier le bâton si nécessaire*
 - ▶ *Impliquer directement les équipes*
 - ▶ *Choix des règles*
 - ▶ *Choix d'objectifs simple : la 'règle du jour'*
- ▶ ***Automatiser les mesures avec différents Zooms***
 - ▶ *2 ou 3 chiffres clés pour les décideurs*
 - ▶ *Comparaison inter projets pour les chefs d'équipes*
 - ▶ *Listes précises des problèmes pour les développeurs*



Que corriger en premier ?

- ***Si les métriques sont bonnes***
 - *Rien de particulier focalisez vous sur vos tâches*
 - *Assurez vous de maintenir ce niveau de qualité*
- ***Si les métriques sont mauvaises***
 - *Les bugs et règles de plus haute sévérité*
 - *Les 'hotspots' vous permettant d'augmenter rapidement vos statistiques (et donc motiver les équipes)*
 - *Vos métriques les plus faibles*
 - *Les warnings ou autres règles moins prioritaires*
- ***Idée : instaurez des demies journées de corrections qualité si nécessaire***
 - *Choisir le moment de la semaine le plus calme possible*



Guide pratique : que faire ... (1/4)

- **... si vous démarrez un projet**
 - Mettez en place une intégration continue
 - Identifier les outils qualité correspondants à votre projet
 - Automatiser la récolte de métriques
 - Attendez un peu avant de mettre en place vos règles qualité

- **... si votre projet est en cours de développement**
 - Adoptez une démarche itérative

- **... si votre projet possède du code legacy**
 - Adoptez deux stratégies différentes :
 - ✓ L'une pour le nouveau code
 - ✓ L'autre pour le code legacy



Guide pratique : que faire ... (2/4)

- **... si vous êtes sur un petit projet**

- Rien de particulier, cela ne devrait pas poser trop de problème
- Prenez les reines si personnes ne veut les prendre

- **... si vous êtes sur un gros projet**

- Divisez pour régner
 - ✓ Identifier les équipes et leurs spécificité
- Regroupez les équipes par 'profil' d'exécution
- Comparez les équipes 'comparables', séparez les autres métriques

- **... si vous êtes sur un multi-site (non géolocalisé)**

- Jouez sur ce découpage pour booster la motivation
- Adaptez les règles le moins possible



Guide pratique : que faire ... (3/4)

- **... si vos équipes ne respectent pas les règles**
 - *Identifier les freins, réexpliquer la démarches*
 - *Réduire vos attentes et repartir sur de bonnes bases*
 - *Récompensez les bon élèves*

- **... si vos équipes désactivent les outils qualité 'on-fly'**
 - *Réduire le nombre d'outils pour être le moins intrusif possible*
 - *Réduire le nombre de règles et alertes*
 - *Extrême : forcer leur utilisation avec des Hook SVN*



Guide pratique : que faire ... (4/4)

- ***... si les délais et le stress prennent le pas sur la qualité***
 - *Axez votre stratégie sur les décideurs*
 - *Continuer la récolte de métriques et les notifications*
 - *Estimez au possible le temps de correction des violations*

- ***... si les décideurs vous demandent de laisser tomber***
 - ~~*Rentrez chez vous en jurant contre le monde entier*~~
 - ~~*Passez vos journées au café*~~
 - *Essayez de trouver des appuis ailleurs*
 - *Réexpliquer la démarche de manière globale*
 - *Trouver des compromis*
 - *Profitez de cette période pour enrichir vos usines*



Autres conseils

- ***N'hésitez pas à configurer plusieurs 'profils' qualité***
 - *Les règles des couches hautes ne sont pas forcément les même que pour les couches basses (Threads, Exceptions ...)*
 - *Selon les contraintes et spécificités de chaque équipe (géo localisation, code legacy ...)*



Réitérer / les limites

- **Application parfaite n'existe pas**
 - *Ne pas trop réitérer sans réel amélioration*
 - *Maitriser ses outils et donc ses règles*
- **Les outils ne peuvent pas trop contrôler**
 - *La conception*
 - *Les performances*
 - *L'exécution (il y a toujours des mauvaises surprises)*
 - *Le nommage venant des spécifications (Domain Driven Design)*
- **Les codes review et le pair programming**
 - **prennent le relais des outils**
 - *Augmente la communication dans l'équipe de développement*
 - *Vérification humaine (algorithme, intégration l'existant, ...)*

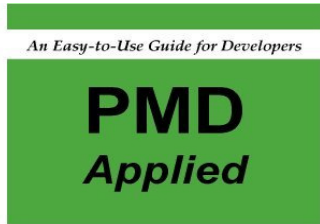


Ressources

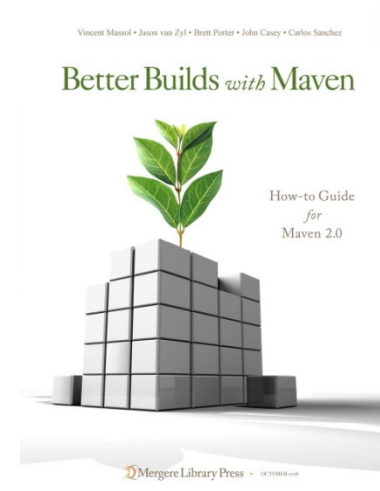
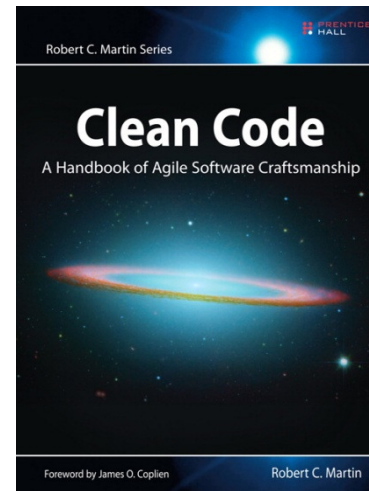
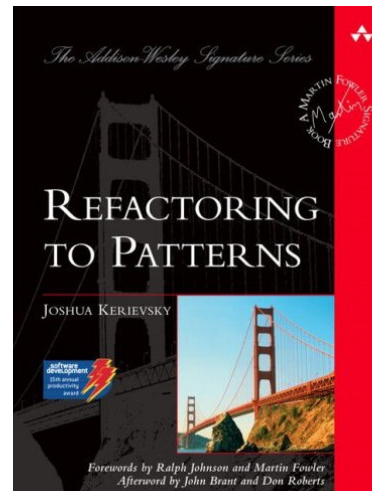
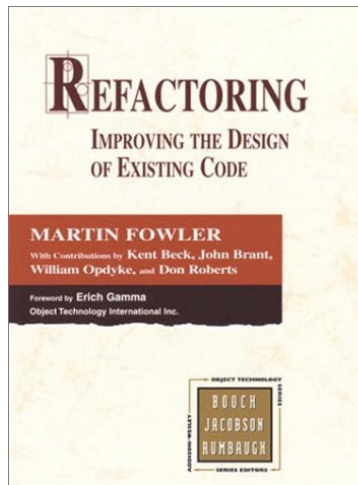
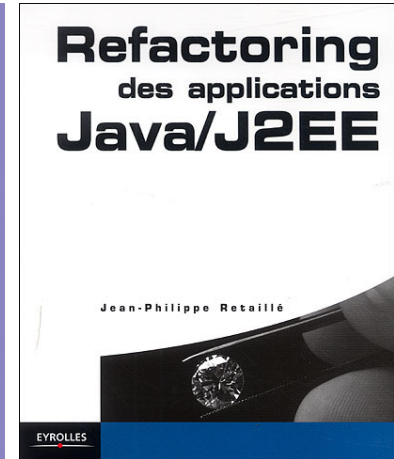
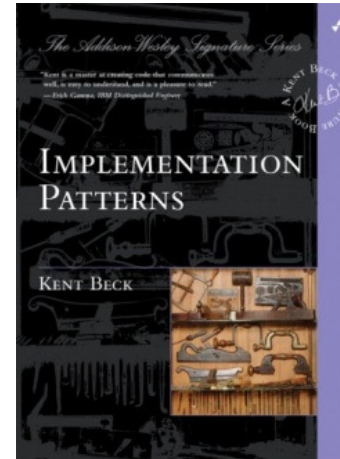
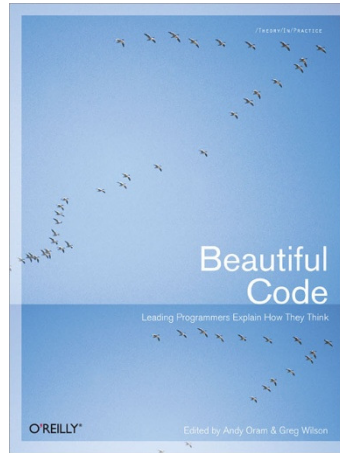
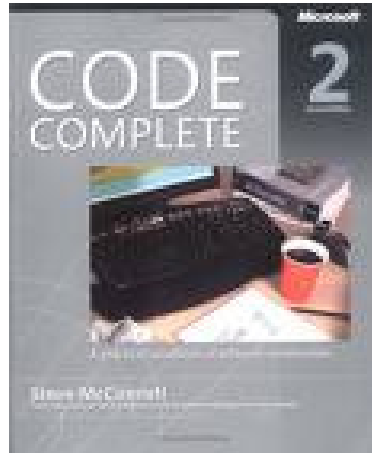
- ***Refactoring d'une méthode longue***
 - *<http://www.infoq.com/articles/RefactoringMyths>*
- ***Automation for the people : Continual refactoring (Paul Duvall)***
 - *<http://www.ibm.com/developerworks/library/j-ap07088/>*
- ***Continuous integration (Paul Duvall)***
 - *<http://www.amazon.com/gp/product/0321336380/?tag=in-tegratecom-20>*
- ***La dette technique (Martin Fowler) :***
 - *<http://www.martinfowler.com/bliki/TechnicalDebt.html>*



Livres

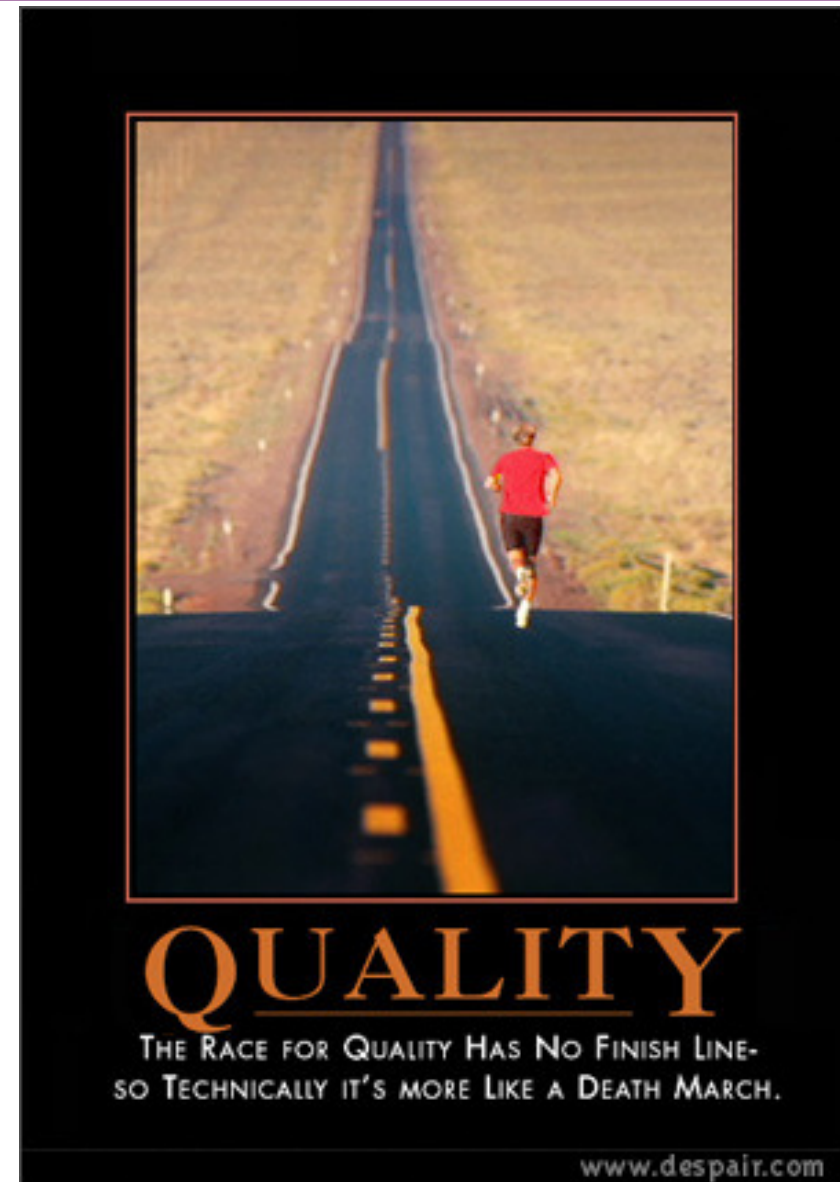


Tom Copeland



Conclusion

La qualité est un voyage, pas une destination.



Des questions ?



Merci.