



Projet d'application d'entreprise

Infrastructure

Approche naïve de la qualité

- « Qualité » est interprétée :
 - Client final : programme qui fonctionne sans planter, facile à utiliser, qui remplit la tâche souhaitée, ...
 - Marketing : satisfaction du client, liste des fonctionnalités, ...
 - Gestionnaire de projet : deadlines respectées, documentation complète, ...
 - Développeur X : code optimal, ...
 - Développeur Y : code sans bug, extensible, ...
 - Mainteneur : code compréhensible, documenté, ...

Comment augmenter la qualité ?

- Réponse naïve : on essaie de faire de son mieux !
- Réponse professionnelle : on met en place des processus qui sont reconnus pour favoriser la qualité, au sens de la satisfaction du client.

Exemple de processus qualité

- Suivi d'une méthodologie de développement, méthodologies que vous verrez au cours d'organisation des entreprises !
- Suivi des bonnes pratiques qui vous seront données dans ce cours :
 - Dépôt de code
 - Règles de style
 - Intégration continue
 - Checkstyle
 - CPD
 - ...

Autre exemple de processus favorisant la qualité

- Dépôt de code source (source repository)
 - Outil facilitant le travail à plusieurs sur le même code source.
 - Centralise le code de tout le monde.
 - Gestion des versions : historique des fichiers, retour à une version antérieure, etc.
 - Résolution des éditions conflictuelles.

Outil de dépôt de code : GIT

- Support natif dans la plupart des IDE.
- Principe :
 - Maître : copie du code de référence.
 - Existe physiquement sur un serveur (Gitlab).
 - Esclaves : vos copies personnelles locales.
 - Copie sur votre compte IPL.
 - Copie sur votre machine personnelle à la maison.

GIT

- Chacun modifie sa copie personnelle.
- A la demande : prise de photo des modifications.
 - Commit : emballe toutes les modifications (ajouts de fichiers/répertoires, modifications de fichiers, suppression de fichiers/répertoires) dans une unité appelée commit et qui contient un commentaire. Ceci reste local à la machine.
- A la demande : synchronisation de sa copie personnelle avec le maître.
 - Pull : réception des commits appliqués au serveur et qui n'ont pas encore été appliqués localement.
 - Push : envoi des commits locaux qui n'ont pas encore été appliqués au serveur et application de ceux-ci.

GIT : conflit

- Un commit est une opération locale qui réussit toujours.
- Un pull est automatique quand :
 - Les dernières modifications locales ont toutes été commitées.
 - Les changements effectués au maître portent sur du code qui est resté inchangé à l'esclave.
- Un push est automatique quand :
 - Les changements effectués à l'esclave portent sur du code qui est resté inchangé au maître.
- Dans tous les autres cas : conflit à résoudre manuellement.

Git : Règles de bonne collaboration

- Commit est une opération locale que vous pouvez effectuer à tout instant.
- Mais il est interdit de push du code qui ne compile pas.
- De plus, vos commits peuvent ne pas être compatibles avec d'autres commits que vous n'avez pas encore pull. Il faut donc toujours pull avant de push et s'assurer que tout continue à fonctionner.

SI LE CODE NE COMPILE PAS ET QUE QUELQU'UN D'AUTRE SE SYNCHRONISE AVEC LE SERVEUR, ALORS SA VERSION A SON TOUR NE COMPILERA PLUS ET IL SERA DANS L'INCAPACITE DE TRAVAILLER.

Usage de git

- Un repository avec le projet backend à la racine, et un dossier webapp pour le front-end indépendant.
- Jenkins ne vérifie que le back-end.
- Workflow tel que vu en devops :
 - master/main = version déployable (qui sera évaluée)
 - Feature-branches = une branche par fonctionnalité
- Merge requests obligatoires avec :
 - Validation par au minimum 1 personne.
 - Passage de Jenkins automatique.


Git : Règles de bonne collaboration

- Conventions de nommage pour les feature-branches :
 - feature-register-user
 - bug-logout-crash
 - refactor-add-multithread-dal
- Commentaires pour les commits :
 - feat: add first working DAO for user
 - fix: remove password when returning user
 - refactor: better exceptions handling

Autre exemple favorisant la qualité

- Fabrique et analyse de code :
 - Le code de référence est régulièrement compilé et analysé.
 - Diverses mesures associées à la qualité sont calculées.
 - But : faire évoluer le code et ses mesures vers toujours plus de qualité.

Outil d'intégration continue : Jenkins

 **Jenkins**


1


rechercher


Laurent Leleux | se déconnecter


Jenkins


Rafraîchissement automatique


 Nouveau Item


 Utilisateurs


 Historique des constructions


 Relations entre les builds


 Vérifier les empreintes numériques

 Administrer Jenkins







 Mes vues

 Identifiants




 New View

 Ajouter une description

Tous +

S	M	Nom du projet ↓	Dernier succès	Dernier échec	Dernière durée	# Issues
		projet-ae-architecture	6 h 4 mn - #8	1 j 22 h - #4	2 mn 23 s	 334
		projet-ae-boilerplate	5 h 56 mn - #7	s. o.	21 s	 0

Icône: [S](#) [M](#) [L](#)

[Légende](#)  [RSS pour tout](#)  [RSS de tous les échecs](#)  [RSS juste pour les dernières compilations](#)

File d'attente des constructions

File d'attente des constructions vide

État du lanceur de compilations

1 Au repos
2 Au repos

Page générée: 15-févr.-2019 15:28:57 CET [REST API](#) [Jenkins ver. 2.150.2](#)

Analyse de code : Checkstyle

- Vérifie que le code respecte un sous-ensemble des règles de formatage identiques pour tout le monde.
- Se mettre d'accord sur un formatage commun entre les développeurs = faciliter l'interchangeabilité.

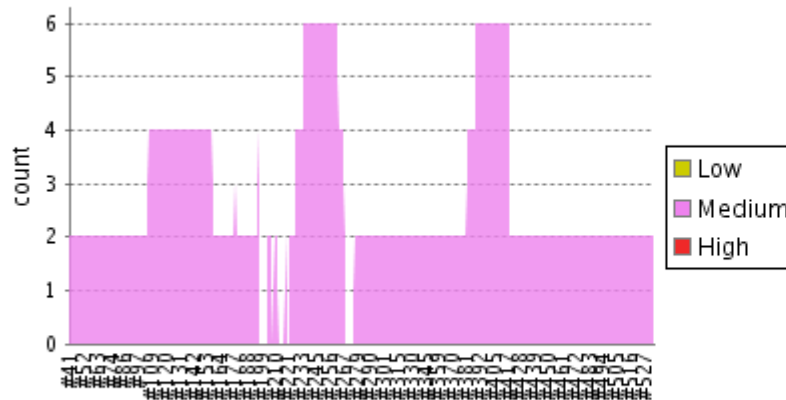
Règles de style Java : celles de Google

- Dans votre IDE :
 - CTRL + ALT + SHIFT + L formate selon les règles définies.
 - Format on save.
- Configurer votre IDE pour qu'il utilise les règles de Google :
 - Cfr. Vidéo sur Moodle.

Attention : votre IDE ne fera pas de miracle et vous devrez vous assurer que Jenkins valide bien vos règles. Par exemple : votre IDE n'écrira pas la Javadoc pour vous...

Analyse de code : Copy/Paste Detector

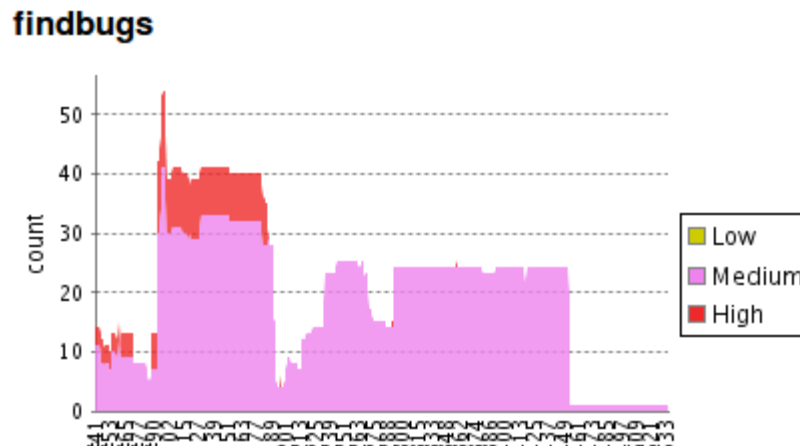
cpd



filename	l	m	h	number ↑
PAOO/src/ihm/FrameApplication.java	0	2	0	2

- Copier du code = copier des bugs = multiplier l'effort de maintenance = c'est mal.
- CPD détecte les copies automatiquement.
 - Changez l'architecture pour éviter les copies.

Analyse de code : PMD



- Trouve des erreurs classiques.
- Vérifiez bien chaque erreur signalée, s'il ne se trompe pas, corrigez-la !

Démo Jenkins

- Indicateurs (boule, météo).
- Builds.
- Métriques.
- Tests et Coverage.
- Javadoc.
- Console.