INF2050 – Outils et pratiques de développement logiciel

Intégration continue

Jacques Berger

Objectifs

Introduire une pratique de contrôle de la qualité

Prérequis

Gestion de sources

Construction automatisée

Qualité

L'intégration continue est une méthode de contrôle de la qualité

Souvent, on développe sans cadre de contrôle de la qualité et l'on effectue une grande quantité de tests juste avant de livrer au client

Qualité

Ces tests permettent de détecter beaucoup d'erreurs, ce qui est bien

Par contre, il arrive souvent que la quantité de bogues trouvés soit trop grande pour le temps restant avant la livraison du projet

Qualité

Cette méthode est largement répandue mais ne permet pas de bien gérer les risques et la qualité du logiciel

Peu d'effort de qualité durant le développement et beaucoup d'effort à la fin du projet

Intégration continue

On met un peu d'effort dans la qualité tout au long du développement

Nécessite de changer un peu sa façon de travailler

On veut intégrer son travail avec le travail des autres le plus rapidement possible et le plus souvent possible

Intégration continue

Plusieurs problèmes surviennent lorsqu'on intègre le travail de plusieurs personnes ensemble

L'intégration continue tente de diminuer ce problème

Concepts

On veut faire un build de l'application aussi tôt que possible après un changement

- Idéalement, un build pour chaque commit

Le build doit être complètement automatisé

On exécute les tests lors de chaque build

Concepts

On veut détecter les problèmes d'intégration le plus rapidement possible, c'est l'objectif!

Tous les développeurs vont faire leurs modifications dans le même dépôt pour faire ressortir rapidement les problèmes d'intégration

Concepts

On divise les développements en plusieurs très petits changements

Pour chaque petit changement, on fait un commit, un build et on exécute les tests

On peut faire une dizaine de commit par jour si nécessaire, il faut éviter de s'isoler du travail des autres

Concepts

Chaque commit devrait laisser le logiciel dans un état stable qui pourrait être déployé en tout temps

Logiciel

On utilise un logiciel qui exécutera un build du système après chaque commit et qui exécutera les tests

Ce logiciel nous avisera si un problème survient

- Compilation
- Tests

Logiciel

Quelques logiciels connus

- CruiseControl
- Apache Continuum
- Jenkins (Hudson)
- TeamCity

Équipe

Les problèmes d'intégration vont arriver fréquemment mais ils vont être plus faciles à corriger puisque les modifications sont petites

Lors d'un problème d'intégration, les développeurs concernés doivent discuter de la solution au problème

Ceci facilite la communication dans l'équipe

Équipe

Avec un tel système, les développeurs sont avisés rapidement des problèmes qu'ils ont injectés dans le code

Ils peuvent donc plus facilement et plus rapidement les corriger (un problème récent est souvent plus facile à corriger qu'un vieux problème)

Pratique

Utiliser un gestionnaire de sources

On y place tout ce qui est nécessaire à la construction du livrable

Pratique

On évite les branches parallèles à la branche de développement, ou sinon elles doivent avoir une courte durée de vie

Ou encore, on pratique l'intégration continue sur toutes les branches simultanément

Pratique

Automatiser le build

- Avec un outil comme Ant, Maven, Gradle, etc.
- Une seule commande devrait construire le livrable

Pratique

Le build doit s'auto-tester

- Le build doit contenir tous les tests nécessaires pour s'assurer qu'il est valide
- Tests unitaires
- Tests fonctionnels

Pratique

Faire des commits le plus souvent possible

- Au moins un par jour (si possible)
- Pas de limite maximale
- Un plus petit commit qui introduit un bogue est facile à retirer

Pratique

Faire un build pour chaque commit

- Peut être difficile à faire avec beaucoup de développeurs
- C'est l'idéal mais un compromis peut être acceptable également

Pratique

Publier le résultat des builds

- Ainsi, on sait quelle version est utilisable et où il y a des corrections à faire

Pratique

Automatiser le déploiement

- Une fois que le build est terminé, on déploie l'application
- On peut ensuite retester si nécessaire

Avantages

Lorsqu'un bogue est détecté, on peut facilement revenir en arrière à un état stable

- Les changements sont petits

Conflits moins durs à corriger

Détection plus rapide des bogues

Détection des problèmes d'intégration

Inconvénients

Installation initiale

Moins utile sans tests

Pourrait nécessiter plusieurs machines de build

Nécessite de changer sa façon de travailler

Plus loin...

Jenkins
    http://jenkins-ci.org/

Apache Continuum
    http://continuum.apache.org/

CruiseControl
    http://cruisecontrol.sourceforge.net/

TeamCity
    https://www.jetbrains.com/teamcity/