femto-st SCIENCES & TECHNOLOGIES

INRA SCIENCE & IMPACT

FP

PEPI
Ingénierie Développement Logiciel

Agilité, Test et Intégration Continue

Fabrice AMBERT – fabrice.ambert@femto-st.fr
Fabrice BOUQUET – fabrice.bouquet@femto-st.fr
Fabien PEUREUX – fabien.peureux@femto-st.fr
Ivan ENDERLIN, Jean-Marie GAUTHIER
Cédric JOFFROY, Alexandre VERNOTTE

UFC Université de Franche-Comté, cnrs, ensmm, utbm université de technologie Belfort-Montbéliard, Inria INVENTEURS DU MONDE NUMÉRIQUE

Ecole IDL Test – Ecully
le 11 décembre 2013

Plan

- **Rappel des pratiques agiles (XP)**
- **Pratique du test unitaire**
- **Pratique du test d'acceptation**
- **Pratique de l'intégration continue**
- **Mise en œuvre du test en intégration continue**
- **Bilan**

Agilité
4 règles d'or

- **Les individus et leurs interactions**

  plus que les processus et les outils

- **Les logiciels opérationnels**

  plus qu'une documentation exhaustive

- **La collaboration avec les clients**

  plus que la négociation contractuelle

- **L'adaptation au changement**

  plus que le suivi d'un plan

Méthodes Agiles
eXtreme Programming (XP)

- Méthode de développement basée sur :
  - 4 valeurs (agiles)
  - 12 principes
  - 13 pratiques (projet, équipe, développement)

- Pratiques centrées majoritairement sur les enjeux « développement » et « équipe » :
  - Collaboration étroite entre le client et les développeurs
  - Optimisation de la productivité en se concentrant sur le code
  - Cadre rigoureux des pratiques à appliquer

eXtreme Programming
Pratiques d'équipe

- **Responsabilité collective du code**
  - Rendre les développeurs plus polyvalents
- **Travail en binôme**
  - Assembler/partager les compétences
  - Prévenir les erreurs
  - Créer une motivation mutuelle
- **Langage commun (métaphore)**
  - Faciliter la compréhension et l'adhésion au groupe
- **Rythme régulier**
  - Atténuer les effets « rush » aux effets incontrôlables

eXtreme Programming
Pratiques de gestion de projet

- **Client sur site**
  - Accélérer les prises de décisions (et les bonnes !)
- **Tests d'acceptation (recette)**
  - Garantir la conformité par rapport aux attentes du client
- **Livraisons fréquentes**
  - Démontrer la valeur ajoutée de façon continue
- **Planification itérative et incrémentale des tâches**
  - Organisation par itérations de développement

eXtreme Programming
Pratiques de développement

- **Restructuration du code (refactoring)**
  - Investir pour le futur en maitrisant la dette technique
- **Conception simple**
  - Faciliter la reprise du code et l'ajout de fonctionnalités
- **Tests unitaires**
  - Détecter au plus tôt les erreurs et la régression
  - Test first (TDD) pour améliorer la testabilité et simplifier le code
- **Règles de codage**
  - Améliorer la lisibilité et la reprise du code
- **Intégration continue**
  - Ajouter continuellement de la valeur et accélérer la détection de bugs

eXtreme Programming
Synthèse des pratiques

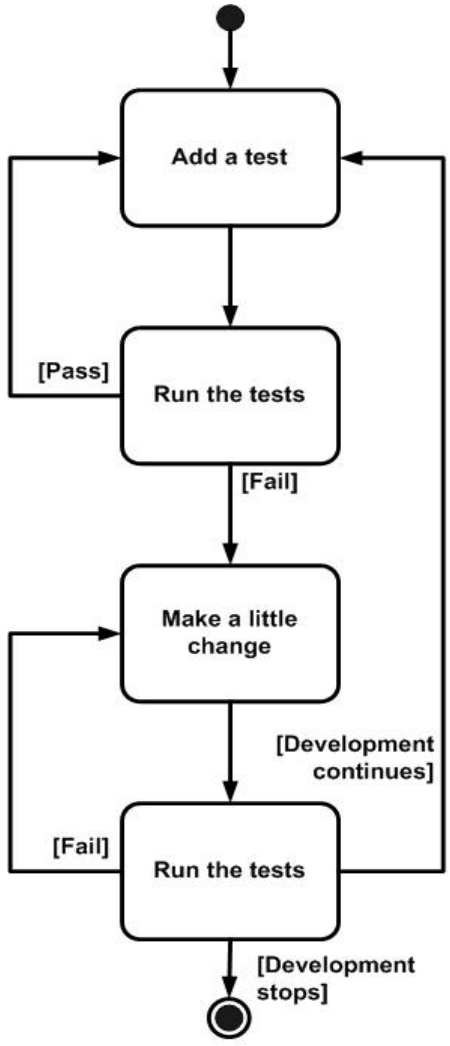
PROJET	CODE	EQUIPE
Livraisons fréquentes : l'équipe vise la mise en production rapide d'une version minimale du logiciel, puis elle fournit ensuite régulièrement de nouvelles livraisons en tenant compte des retours du client.	**Conception simple** : on ne développe rien qui ne soit utile tout de suite.	**Programmation en binômes** : les développeurs travaillent en binômes, ces binômes étant renouvelés fréquemment.
Planification itérative : un plan de développement est préparé au début du projet, puis il est revu et remanié tout au long du développement pour tenir compte de l'expérience acquise par le client et l'équipe de développement.	**Tests unitaires** : les développeurs mettent en place une batterie de tests structurels qui leur permettent de valider leur développement et de faire des modifications sans crainte (garantie de non régression).	**Responsabilité collective du code** : chaque développeur est susceptible de travailler sur n'importe quelle partie de l'application.
Client sur site : le client est intégré à l'équipe de développement pour répondre aux questions des développeurs et définir les tests fonctionnels.	**Restructuration (refactoring)** : le code est en permanence réorganisé pour rester aussi clair et simple que possible.	**Rythme régulier** : l'équipe adopte un rythme de travail qui lui permet de fournir un travail de qualité constant tout au long du projet.
Tests de recette : les testeurs mettent en place des tests fonctionnels qui vérifient que le logiciel répond aux exigences du client. Ces tests permettent une validation du cahier des charges par l'application livrée.	**Intégration continue** : l'intégration des nouveaux développements est faite de façon continue de manière à délivrer de la valeur de façon incrémentale.	**Métaphore** : les développeurs s'appuient sur une description commune du design.
	Règles de codage : les développeurs se plient à des règles de codage strictes définies par l'équipe elle-même.	

Tests automatisés

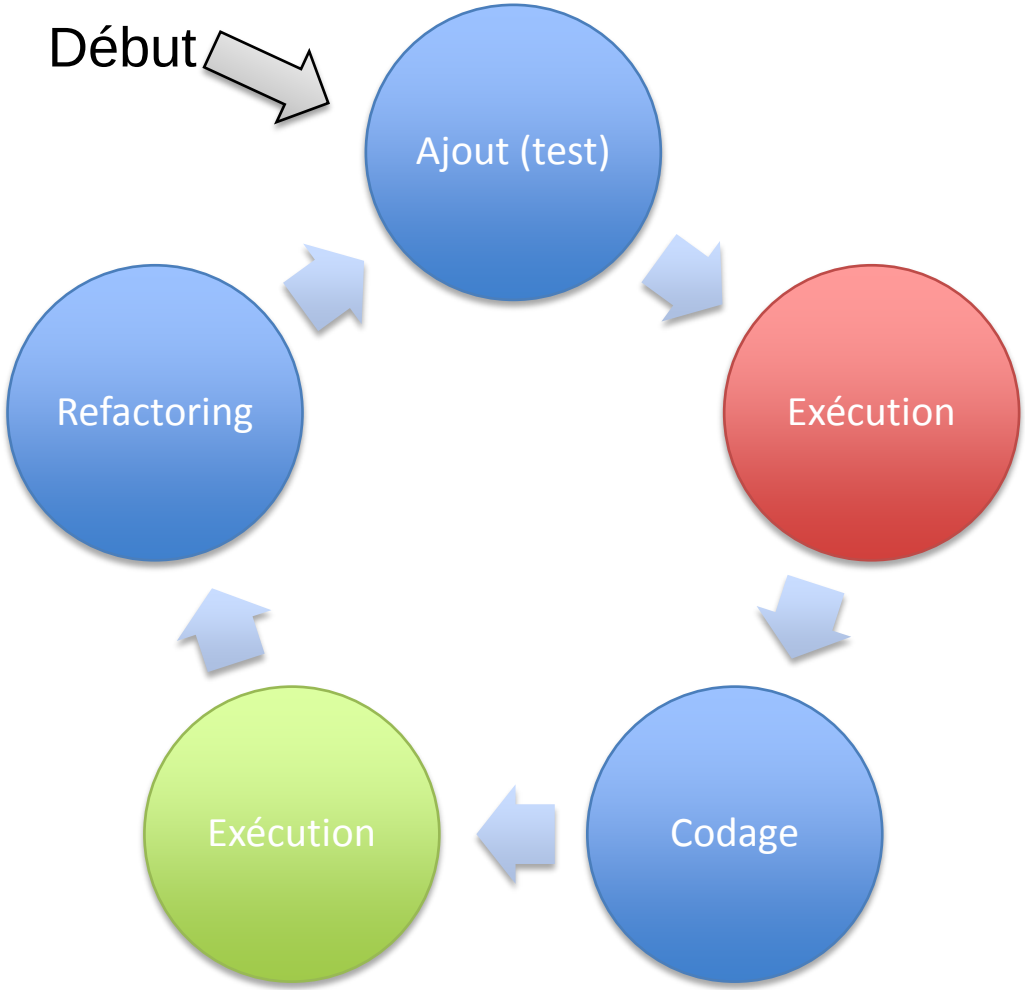
- Les tests automatisés doivent être :
  - Concis : écrits aussi simplement que possible
  - Clairs : faciles à comprendre
  - Spécifiques : chaque test cible un objectif précis et particulier
  - Suffisants : ils doivent couvrir tous les besoins du système
  - Nécessaires : pas de redondances dans les tests
  - Robustes : un test doit toujours produire le même résultat
  - Indépendants : ils ne dépendent pas de leur ordre d'exécution
  - Auto-vérifiables : aucune intervention humaine ne doit être nécessaire
  - Répétables : toujours sans intervention humaine
  - Maintenables : ils doivent pouvoir être aisément modifiés
  - Traçables : on doit pouvoir les (re)jouer au pas à pas

Test unitaire

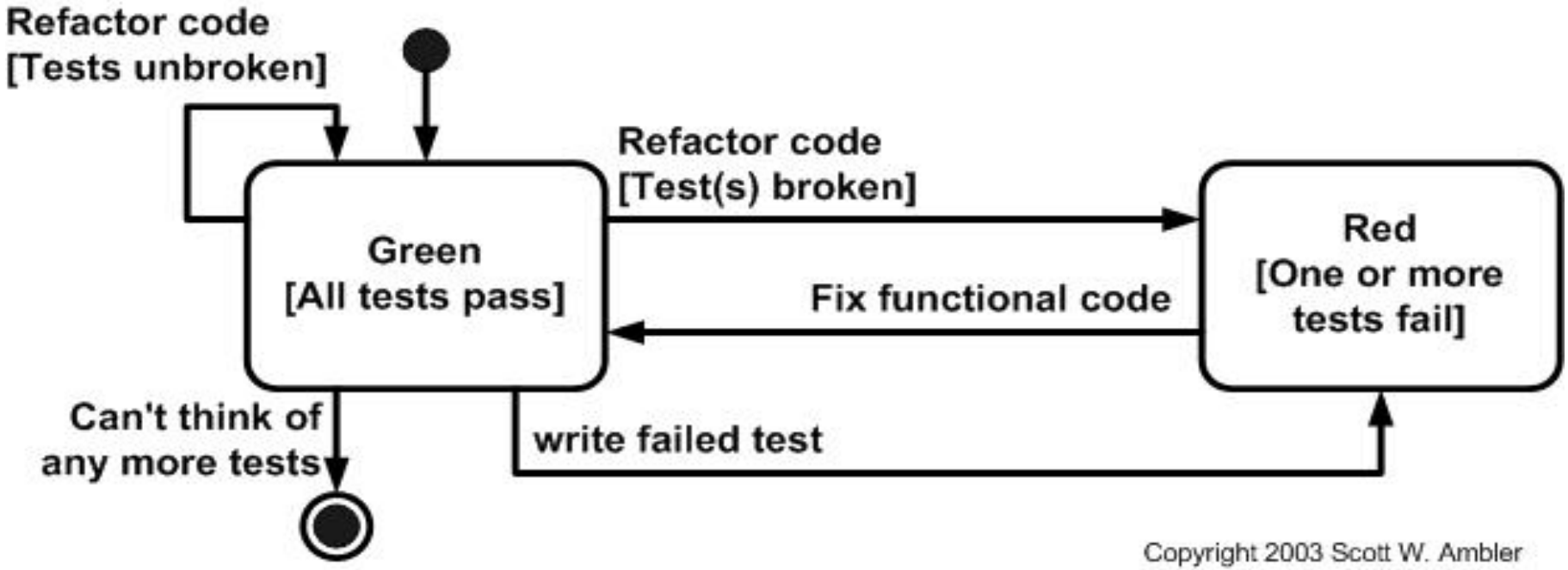
- Acteur : développeur
- Objectifs :
  - Validation structurelle du code
  - Le produit fait les choses « bien »
- Pratique (TDD) :
  - Développer les tests en premier
  - Développer le code correspondant
  - Refactoriser le code / compléter la couverture
- GAINS :
  - Détecter au plus tôt les erreurs
  - Améliorer la testabilité du code
  - Simplifier l'architecture
  - Simplifier le code source
  - Assurer la non-régression



Test Driven Development



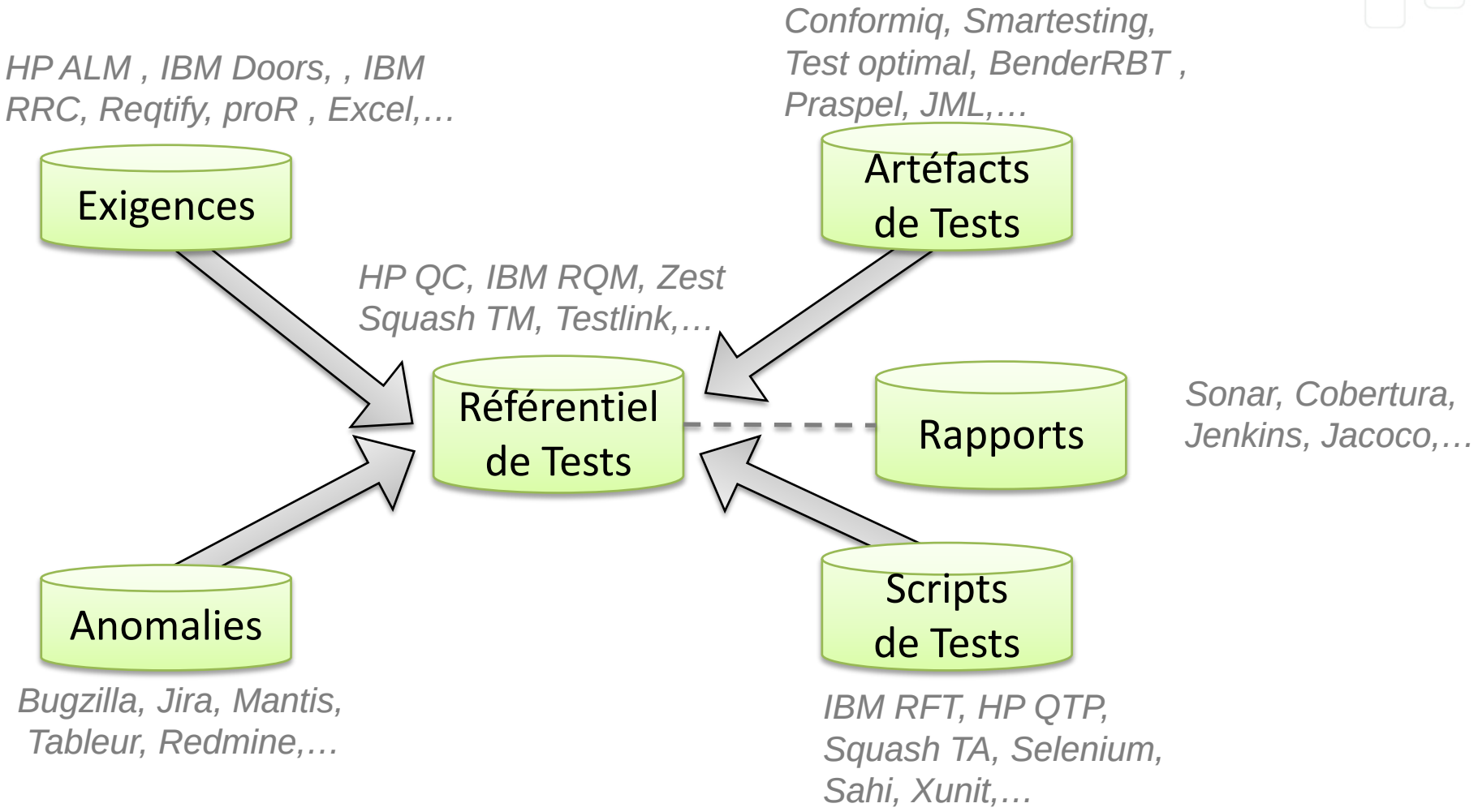
Test unitaire
Non-régression



Test d'acceptation

- Acteurs : client (définition) + développeur (automatisation)
- Objectifs :
  - validation fonctionnelle de l'application
  - Le produit fait les « bonnes » choses
- Pratique :
  - Définir textuel de scénarios d'usage par le « métier »
  - Développer le code correspondant par les développeurs
  - Valider par le développeur puis le client
- GAINS :
  - Spécification exécutable
  - Capturer les besoins réels
  - Garantir la conformité vis-à-vis des attentes du client
  - Documentation

Test - Outillage



Intégration continue
Principes

- Ensemble de pratiques utilisées en Génie Logiciel qui consiste à vérifier, à chaque modification de code source, que le résultat des modifications ne produit pas de régression de l'application en cours de développement.

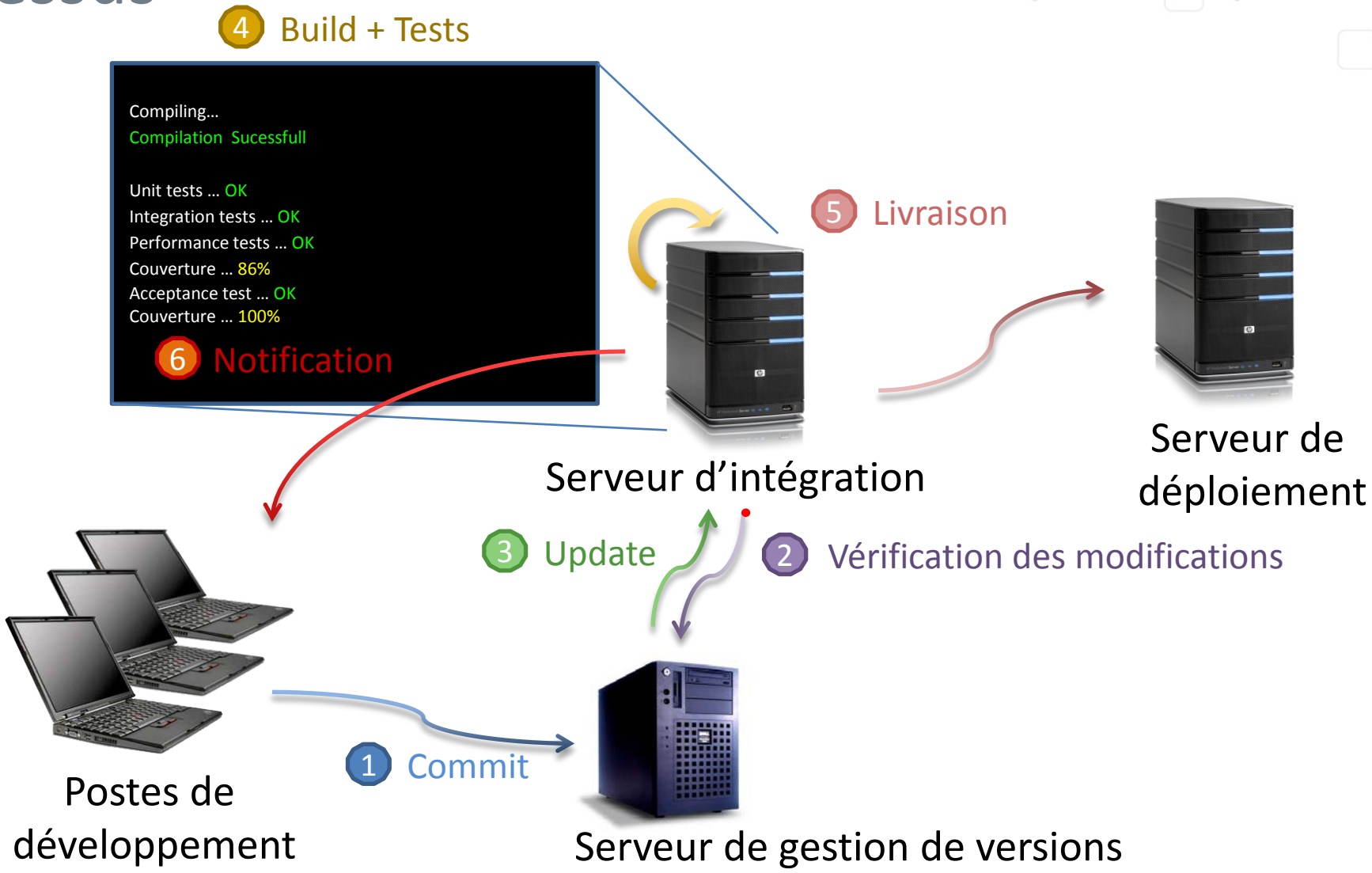
- Bénéfices de cette approche :
  - Capacité de reporting
  - Capacité à livrer un produit « utilisable » à tout instant
  - Coordination des équipes
  - Maîtrise d'œuvre contrôlée

Intégration Continue
Composants

Intégration Continue
Outillage

- Utilisation des outils d'exécution, d'automatisation et de contrôle :
  - SVN, Git, … (version)
  - XUnit, TestNG, …. (test unitaire)
  - Concordion, Cucumber, … (test d'acceptation)
  - ANT, MAVEN, … (Build)
  - Hudson, Jenkins, … (intégration continue)

Intégration Continue
Processus



Bilan (1)

- Le test est une activité aujourd'hui incontournable dans la qualité du logiciel (assurance qualité) – promue par l'agilité

- Définition des tests :
  - Plusieurs approches suivant les objectifs
  - Complémentarité des approches (structurelle / fonctionnelle)
  - Plusieurs techniques dédiées

- Validation par le test :
  - Gain de confiance dans le code produit (filet de sécurité face aux régressions)
  - Garantie de la « bonne » couverture du besoin (cahier des charges)
  - Capacité à mieux maîtriser / évaluer / démontrer la qualité du logiciel
  - Dans le contexte agile : expansion du rôle des tests
    - Spécifier le besoin (cas d'utilisation)
    - Guider le développement (TDD, ATDD)
    - Simplifier et minimiser le code développé

Bilan (2)

- Automatisation de l'exécution des tests :
  - Non nécessaire mais décuple le ROI des tests
  - Facilite souvent l'introduction du test auprès des équipes
  - Structuration et cadencement du processus de développement
  - Levier important pour améliorer la qualité du logiciel

- Freins et réticences :
  - Caractère destructif du test
  - Nouveaux langages à maîtriser
  - Difficulté de scripter les cas de test
  - Difficultés d'automatiser l'assignement du verdict
  - Plateformes IC perçues comme des « usines à gaz »

- Axes d'amélioration actuels :
  - Outillage croissant (langages visés, simplicité d'utilisation, …)
  - Technologie connexe pour la traçabilité (exigences, bugtracker, planificateur,…)
  - Solutions avancées d'automatisation (génération des cas de test / MBT)

Après, vous ne pourrez plus dire… (1)

Les 24 réponses les plus fréquentes à un sondage réalisé auprès de développeurs pour expliquer / justifier un problème rencontré avec leur développement :

24. "Il marche bien sur mon ordinateur.""

23. "Vous vous connectez comme quel utilisateur ?"

22. "C'est une fonctionnalité."

21. "C'est ce qu'il avait été demandé."

20. "C'est bizarre, étrange (une explosion solaire ?)."

19. "Il n'avait jamais fait cela avant."

18. "Ca marchait hier."

17. "Comment cela est-il possible ?"

16. "Ca doit être un problème matériel (hardware)."

15. "Vous avez fait n'importe quoi, ou vous ne savez pas vous en servir ?"

14. "Il y a un problème dans vos données."

13. "Je n'ai pas touché à ce module dernièrement !"

Après, vous ne pourrez plus dire…(2)

Les 24 réponses les plus fréquentes à un sondage réalisé auprès de développeurs pour expliquer / justifier un problème rencontré avec leur développement :

12. "Vous n'avez pas la bonne version."

11. "Ca doit venir d'une coïncidence (un morceau de dll resté en mémoire... redémarrer)"

10. "Je ne peux pas tout tester !"

9. "Ceci ne peut pas correspondre à mon/au code source."

8. "Ca marche, mais ça n'a pas été testé."

7. "Quelqu'un a changé mon code."

6. "Avez-vous vérifié qu'il n'y avait pas de virus dans le système ?"

5. "Bien que cela ne fonctionne pas, cela correspond-il à votre attente ?"

4. "Vous ne pouvez pas utiliser cette version sur votre système."

3. "Pourquoi voulez-vous faire cela de cette façon ?"

2. "Où (en) étiez-vous quand le programme a planté ?"

1. "Je pensais avoir corrigé ce problème."

Bibliographie

- “Manifesto for Agile Software Development”, Beck et al, 2001, http://agilemanifesto.org/

- “The Test Automation Manifesto”, Meszaros et al, Extreme Programming and Agile Methods – XP/Agile Universe 2003, LNCS 2753, Springer.

- Méthodes agiles : http://www.agilealliance.org/
- XP : http://www.extremeprogramming.org/
- Scrum : http://www.scrumalliance.org/
- Pour la détente : http://byatoo.com/la-rache/

Merci pour votre attention…