Intégration Continue

MICROPOLE
Driving Distinction

SOMMAIRE

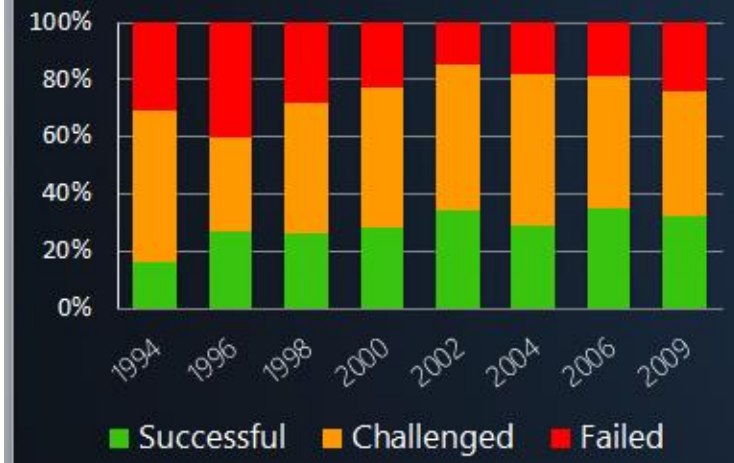
Intégration Continue

Introduction Générale

Introduction Générale

Situation des projets informatiques

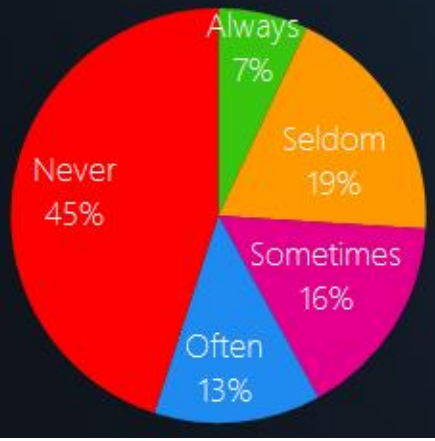
Standish Group Chaos Report



Principales Causes D'échecs

1. Time to Market toujours plus agressif
2. Métier du client et technologies utilisées toujours plus complexes
3. Manque de réactivité face au changement

Functionalities Usage Statistics



- Gaspillage de Ressources
- Forte part de livrables sans valeur métier au détriment de livrables à haute valeur ajoutée

Introduction Générale

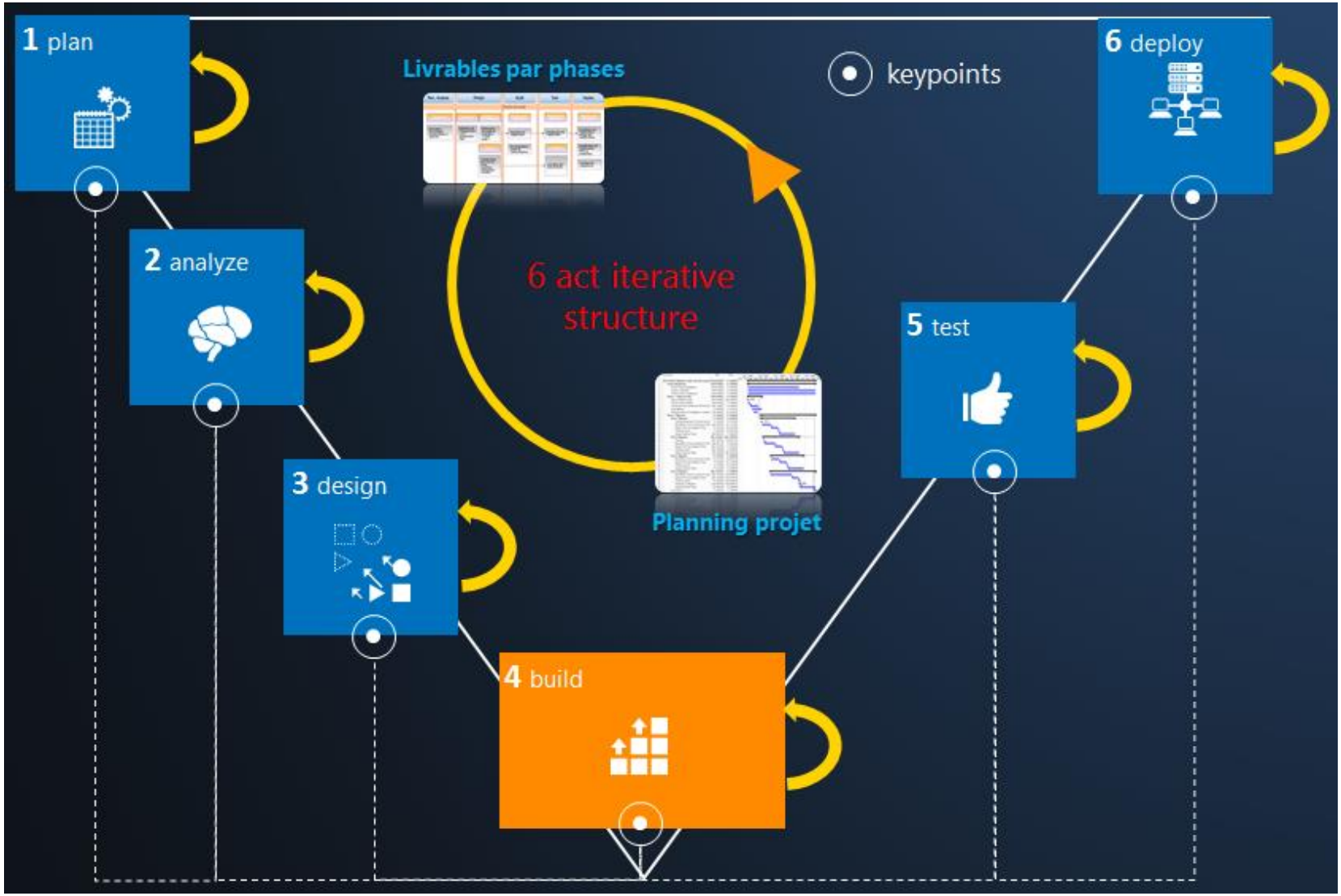
Approche CMMI / PMI

- CMMI (Capability Maturity Model Integration)
  - Livrables par phase
  - Planning projet
  - Référentiel de bonnes pratiques

- PMI (Project Management Institute)
  - Fournit un Framework de gestion de projet reconnu partout dans le monde entier
  - Permet de contrôler les éléments de la triple contrainte : Budget, Délai, Scope et Qualité
  - Utilisation d'un WBS Projet : Définition et contrôle du Scope
  - Utilisation d'indicateurs de performance (KPIs) pour piloter
    - CPI : Cost Performance Index
    - SPI : Schedule Performance Index

Introduction Générale

Cycle en V



Introduction Générale

Agilité

- L'Agilité a gagné en momentum sur le marché du développement
- "Agile" est un terme parapluie qui englobe un grand nombre de méthodologies qui promeuvent un travail effectué sur une ***base itérative*** ou ***les exigences fonctionnelles et les solutions évoluent***
- Il existe beaucoup de Mythes autour de l'agilité tel que :
  - Pas de planification
  - Pas de documentation
  - Pas d'engagement
  - Les équipes Agiles ne sont pas disciplinées
  - Peu ou Pas de Gestion de Projet ni de QA

Le Développement agile confère une approche itérative et flexible du développement logiciel en s'appuyant sur 12 Principes qui constituent le Manifest AGILE

Introduction Générale

Les 12 principes de l'agilité

Satisfaire le Client

Accepter favorablement le changement

Livrer fréquemment

Travailler en collaboration

Motiver les membres de l'équipe

Maintenir un rythme durable

En délivrant tôt et de manière continue des fonctionnalités opérationnelles qui apportent de la valeur au client

Introduction Générale

Les 12 principes de l'agilité

Satisfaire le Client

Accepter favorablement le changement

Livrer fréquemment

Travailler en collaboration

Motiver les membres de l'équipe

Maintenir un rythme durable

Le changement est accueilli comme avantage compétitif pour le client y compris tard dans le développement

Introduction Générale

Les 12 principes de l'agilité

Satisfaire le Client

Accepter favorablement le changement

Livrer fréquemment

Travailler en collaboration

Motiver les membres de l'équipe

Maintenir un rythme durable

Livrer une solution qui fonctionne sur des cycles allant de quelques semaines à quelques mois avec une préférence pour la période la plus courte

Introduction Générale

Les 12 principes de l'agilité

Introduction Générale

Les 12 principes de l'agilité

Introduction Générale

Les 12 principes de l'agilité

Satisfaire le Client

Accepter favorablement le changement

Livrer fréquemment

Travailler en collaboration

Motiver les membres de l'équipe

Maintenir un rythme durable

Les processus Agile promeuvent un rythme de développement durable, les sponsors, développeurs et autres utilisateurs doivent être capable de soutenir ce rythme indéfiniment

Introduction Générale

Les 12 principes de l'agilité

Mesurer l'avancement par un logiciel qui fonctionne

Privilégier la Communication Face à Face

Privilégier Excellence Technique et Qualité

Keep it Simple

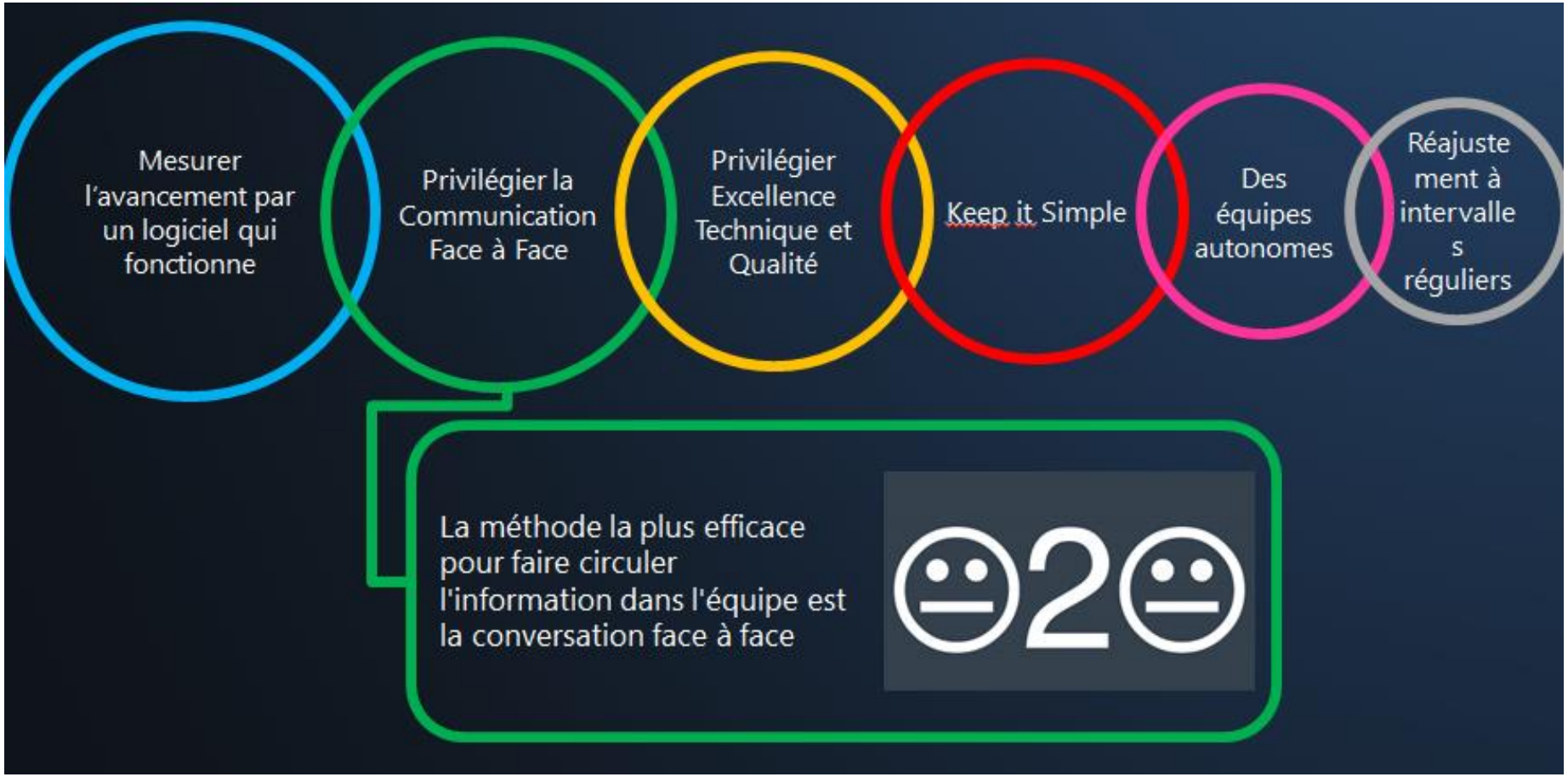
Des équipes autonomes

Réajustement à intervalles réguliers

Un logiciel qui fonctionne est la première source de mesure de la progression du projet

Introduction Générale

Les 12 principes de l'agilité



Introduction Générale

Les 12 principes de l'agilité

Mesurer l'avancement par un logiciel qui fonctionne

Privilégier la Communication Face à Face

Privilégier Excellence Technique et Qualité

Keep it Simple

Des équipes autonomes

Réajustement à intervalles réguliers

Une Attention continue à l'Excellence technique et à la Qualité du Design améliorent l'Agilité et les résultats du projet

Introduction Générale

Les 12 principes de l'agilité

Mesurer l'avancement par un logiciel qui fonctionne

Privilégier la Communication Face à Face

Privilégier Excellence Technique et Qualité

Keep it Simple

Des équipes autonomes

Réajustement à intervalles réguliers

La simplicité est essentielle, y voir également l'art de maximiser le travail à ne pas faire

Introduction Générale

Les 12 principes de l'agilité

Mesurer l'avancement par un logiciel qui fonctionne

Privilégier la Communication Face à Face

Privilégier Excellence Technique et Qualité

Keep it Simple

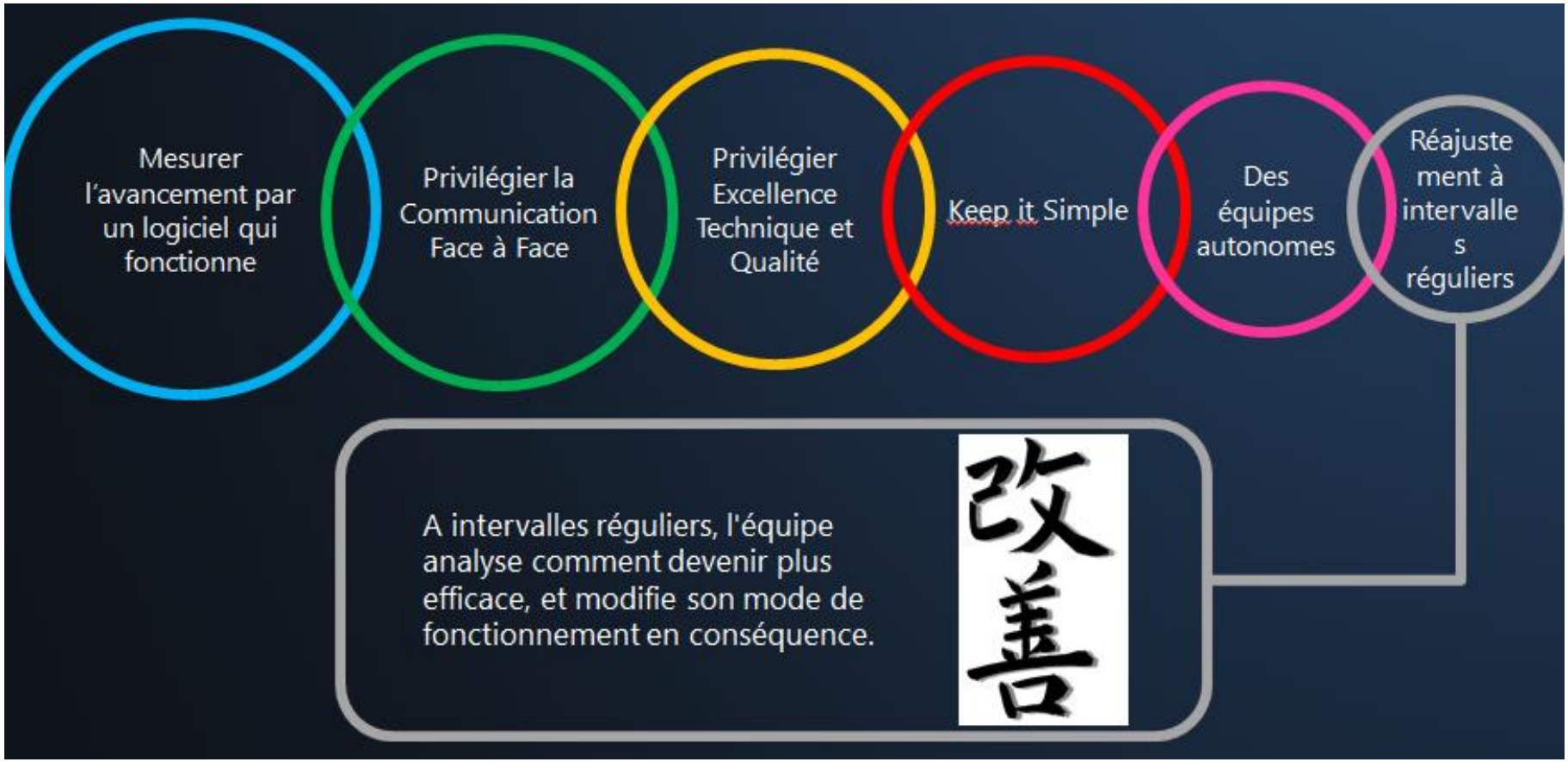
Des équipes autonomes

Réajustement à intervalles réguliers

Les meilleures architectures, design et exigences sont le fruit d'équipes qui s'auto organisent

Introduction Générale

Les 12 principes de l'agilité



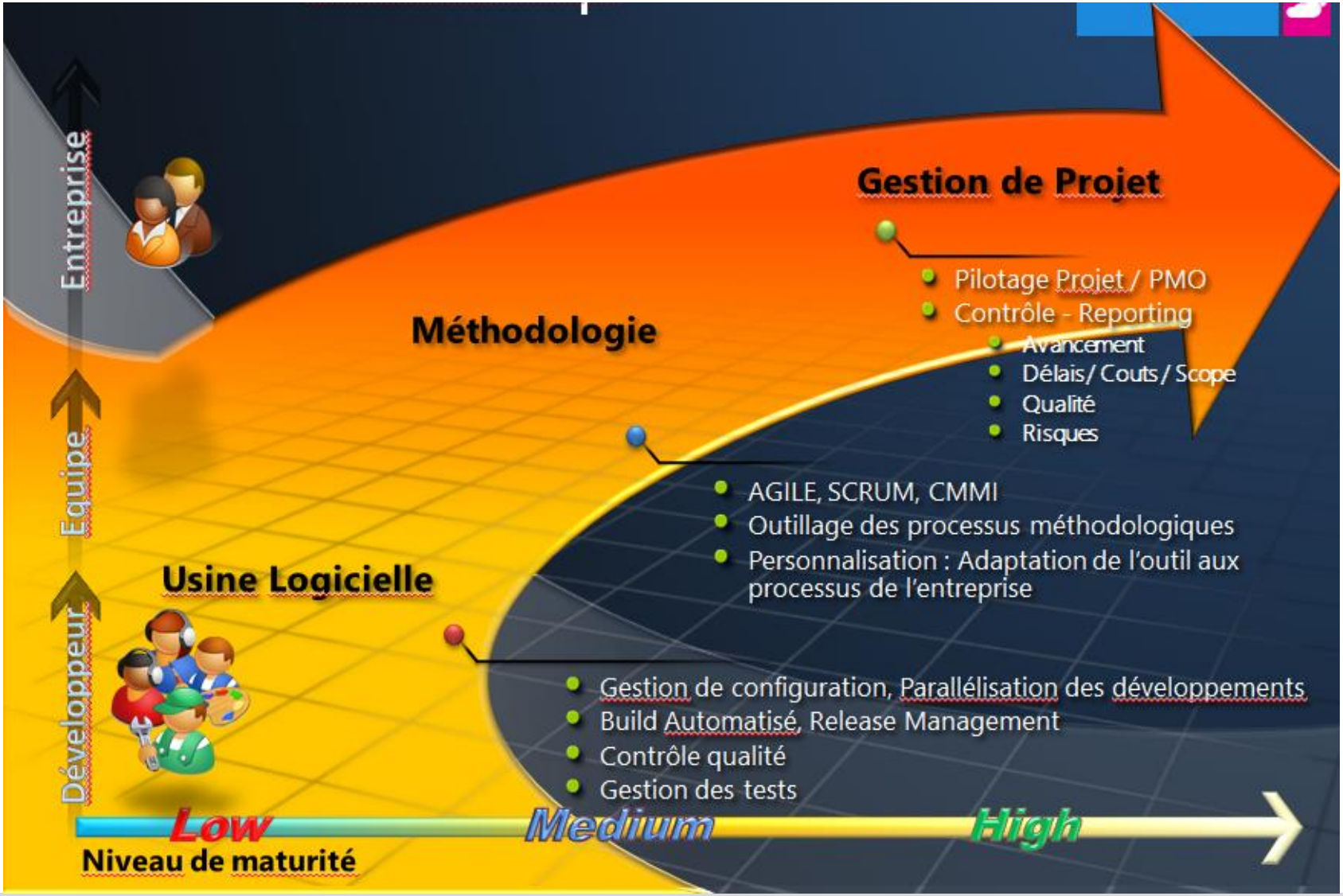
Introduction Générale

ALM ?

L'ALM peut être définie comme une collection de disciplines et d'outils qui vont permettre de transformer un besoin métier en une solution informatique opérationnelle

Introduction Générale

ALM : Roadmap fonctionnelle



Introduction Générale

Préambule

- ✓ Les précurseurs
- ✓ Qu'est-ce que l'intégration continue ?
- ✓ Pourquoi automatiser ?

Introduction Générale

Les précurseurs

- ✓ **Outils de versionning**
- ✓ **Tests**
  - Unitaires
  - D'intégration
  - Performances
  - Qualité
  - Fonctionnels
  - Montée en charge
- ✓ **Outils de compilation**
- ✓ **Inspection de code**

Introduction Générale

Qu'est ce que l'intégration continue ?

- ✓ **Définition**
  - Technique puissante permettant dans le cadre du développement d'un logiciel en équipes de:
    - Garder en phase les équipes de dév
    - Limiter risques de dérive
    - Limiter la complexité
    - Évite le rush de l'intégration avant la livraison : tous les jours une version stable du logiciel est disponible.

- ✓ **Choix de l'intervalle**
  - A intervalles réguliers, vous allez construire (build) et tester la dernière version de votre logiciel.

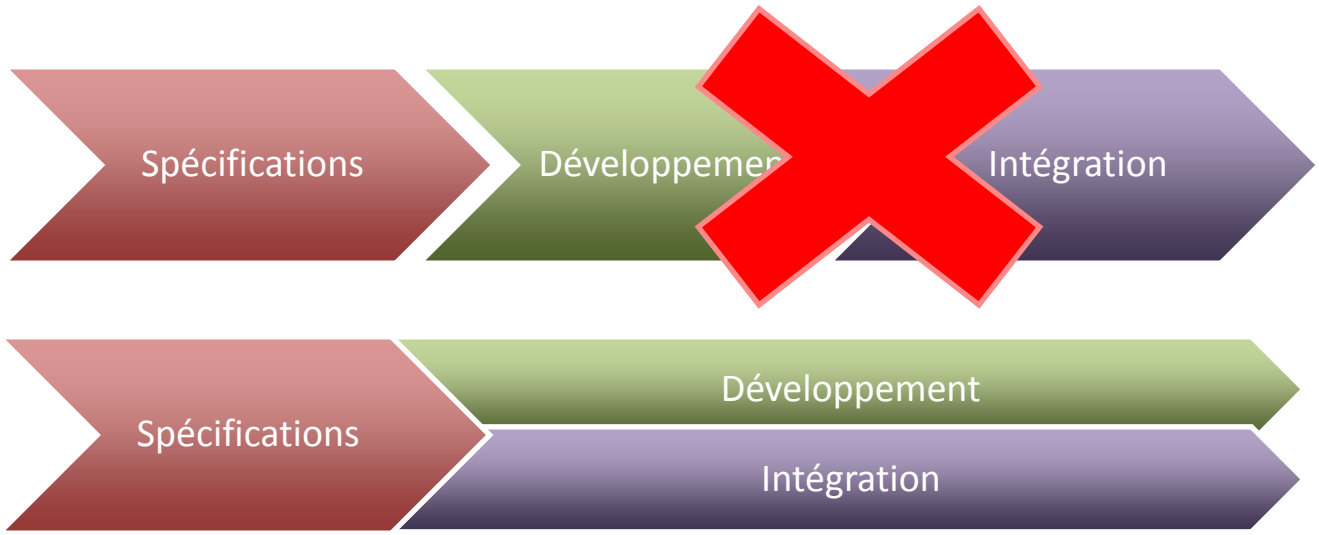
- ✓ **Parallèlement, chaque développeur teste et valide (commit) son travail en ajoutant son code dans un lieu de stockage unique.**

Introduction Générale

Pourquoi automatiser ? (1/2)

- ✓ **Gagner du temps**
  - Vous ne faîtes pas de taches répétitives
  - Réactivité face aux changements
- ✓ **Gagner en confiance**
  - Indépendant de votre efficacité du moment
  - Procédures répétables
- ✓ **Maitrise d'œuvre contrôlé**
  - Coordination des équipes
  - Abstraction de la technique

✓ Le schéma de développement « classique »



Intégration Continue

Les outils de l'intégration continue

Les outils de l'intégration continue

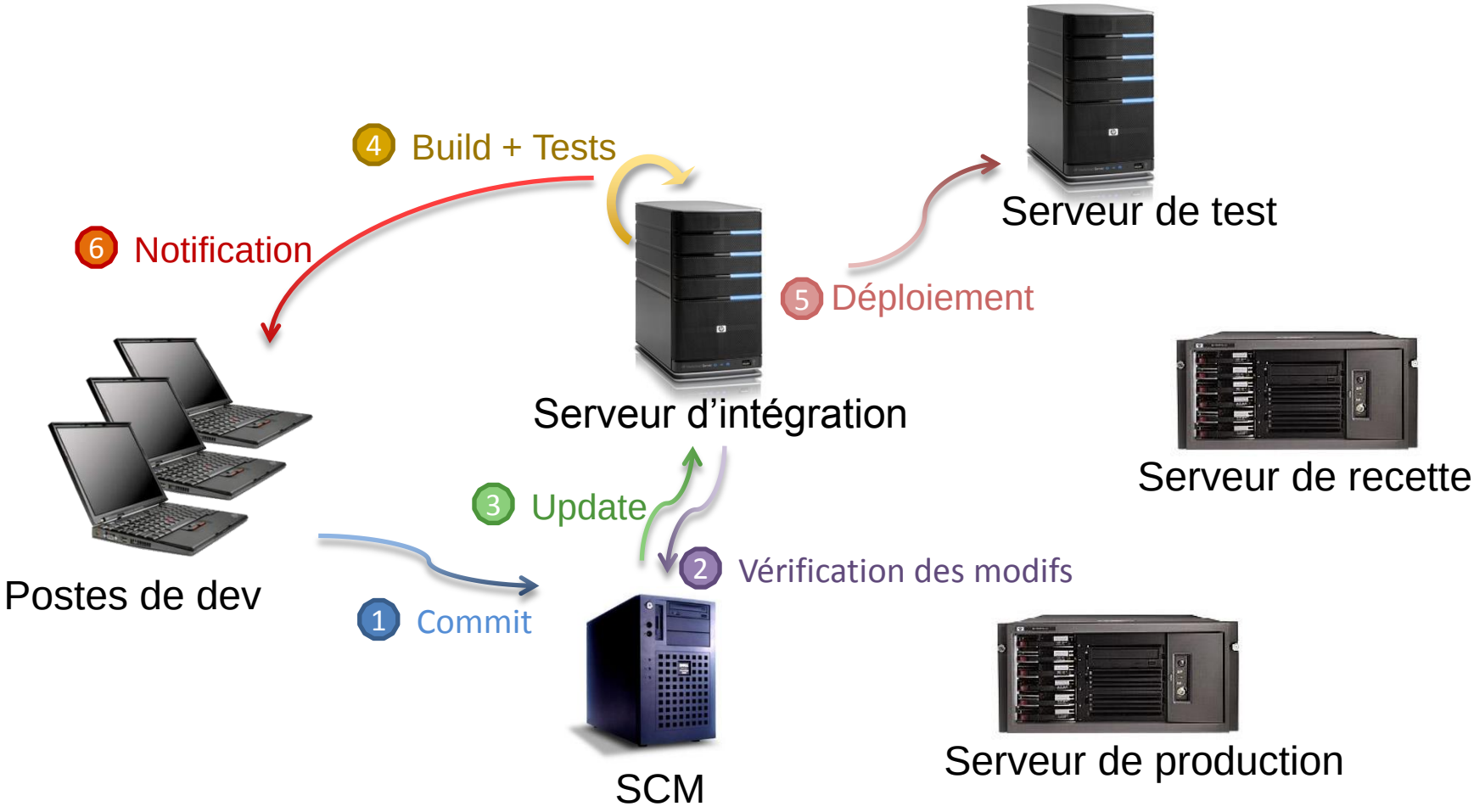
- ✓ **MsTest**
  - Depuis Visual Studio 2005
  - Ecriture des tests unitaires
  - Exécution des tests unitaires
- ✓ **Static Code Analysis**
  - Faire une analyse statique du code
  - Vérifier qu'un certain nombre de règles (paramétrables) ont bien été respectés
    - des conventions de nommage (une classe héritant de System.Exception doit être suffixée par Exception)
    - des principes de design (un événement doit avoir une signature acceptant deux paramètres : un System.Object et une classe héritant de System.EventArgs)
    - des performances (utiliser const plutôt que shared readonly)
    - de sécurité (un constructeur statique doit être privé)

Les outils de l'intégration continue

- ✓ **MsBuild**
  - Automatisation de tâches diverses
  - compilation des solutions et des projets
  - Exécuter un fichier de projet (ou fichier de build), traditionnellement d'extension « .proj » ou « .target ».
  - Ce fichier de projet génèrera une fois exécuté un fichier texte comportant tous les messages, avertissements ou erreurs relatifs à l'exécution.
- ✓ **TFSBuild**
  - Une surcouche au dessus de MsBuild
  - Réaliser plus facilement de l'intégration continue avec le Team Foundation Server:
    - Récupération des sources à partir du Source Control
    - Compilation des sources
    - Exécution des tests
    - Publication des résultats sur le Team Foundation Server

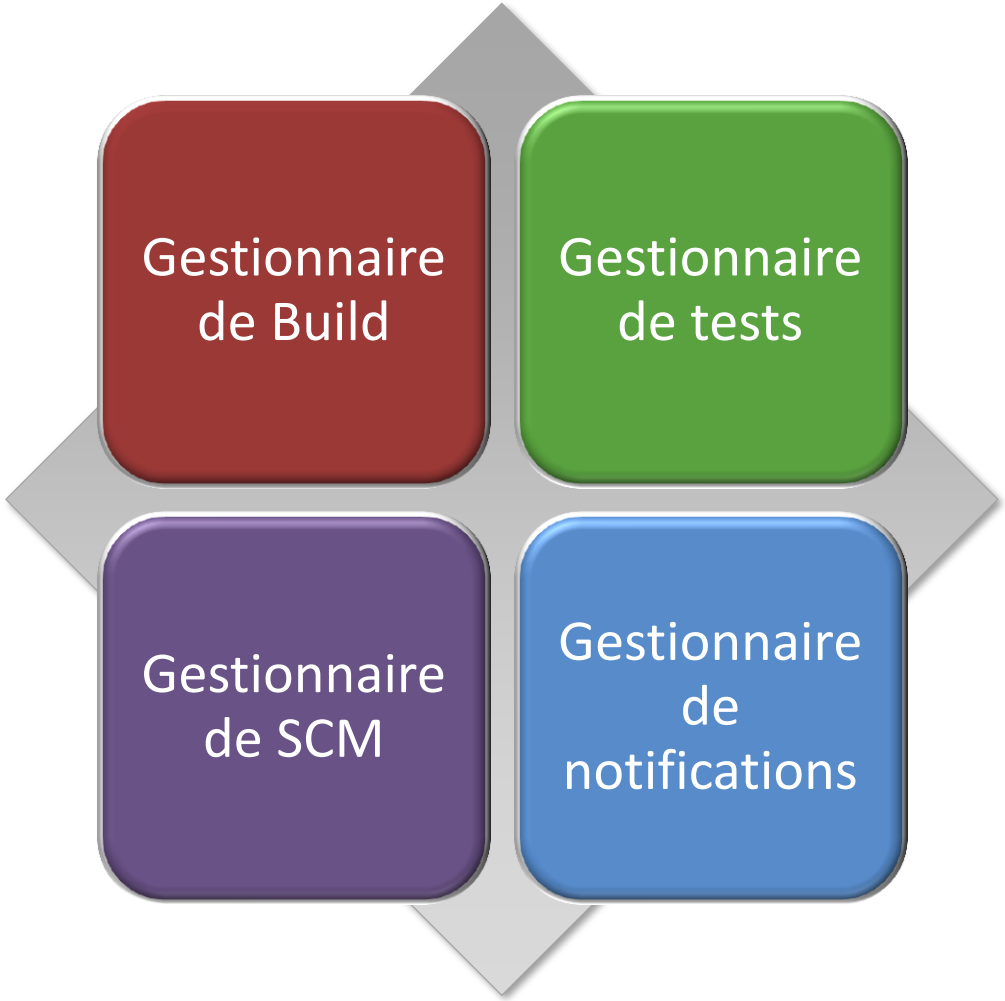
Les outils de l'intégration continue

Architecture globale



Les outils de l'intégration continue

Architecture d'un logiciel d'intégration



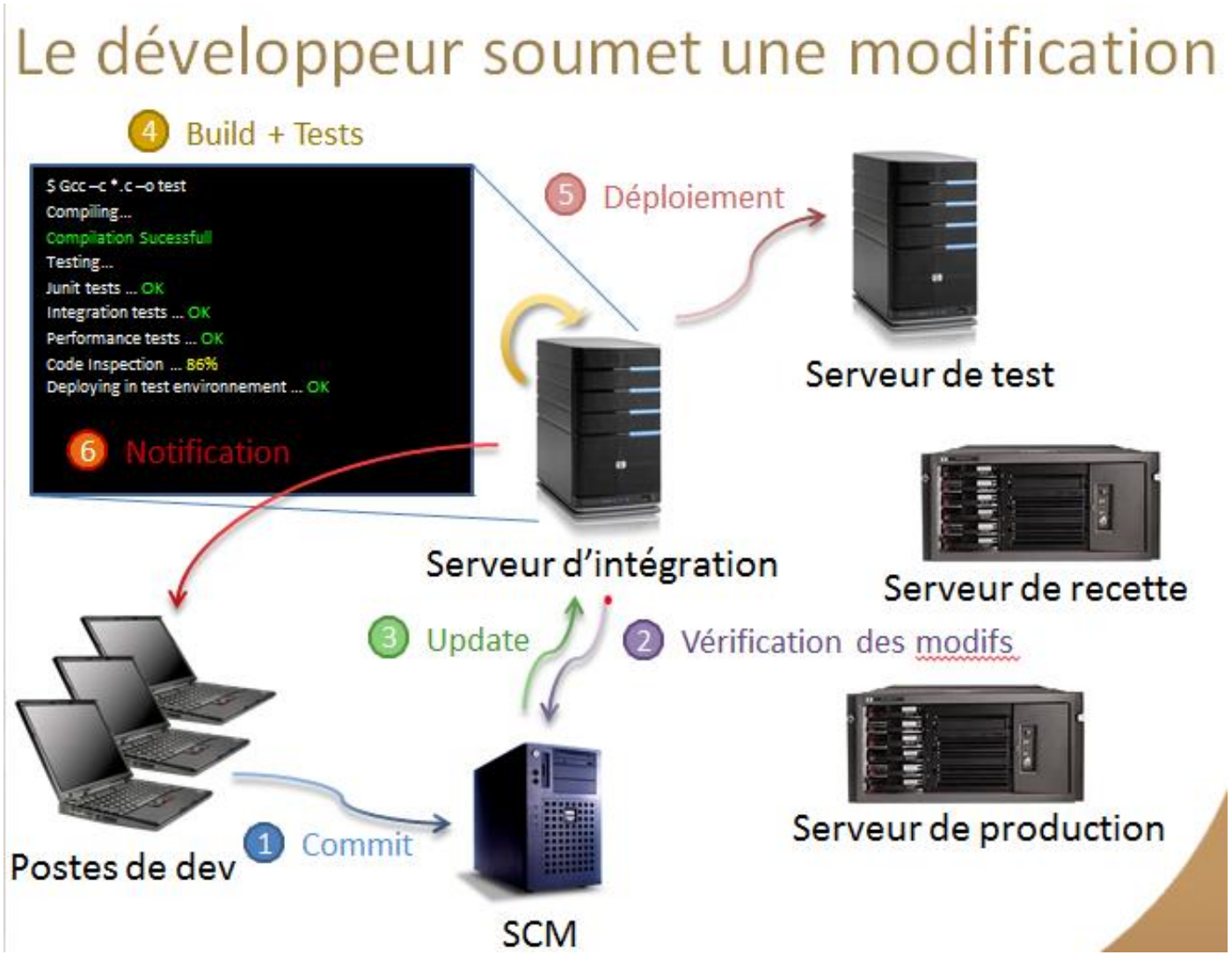
Les outils de l'intégration continue

Fonctionnement

- ✓ Les développeurs « committent »
- ✓ Le serveur d'intégration surveille le serveur SCM (gestion de la configuration logicielle)

Les outils de l'intégration continue

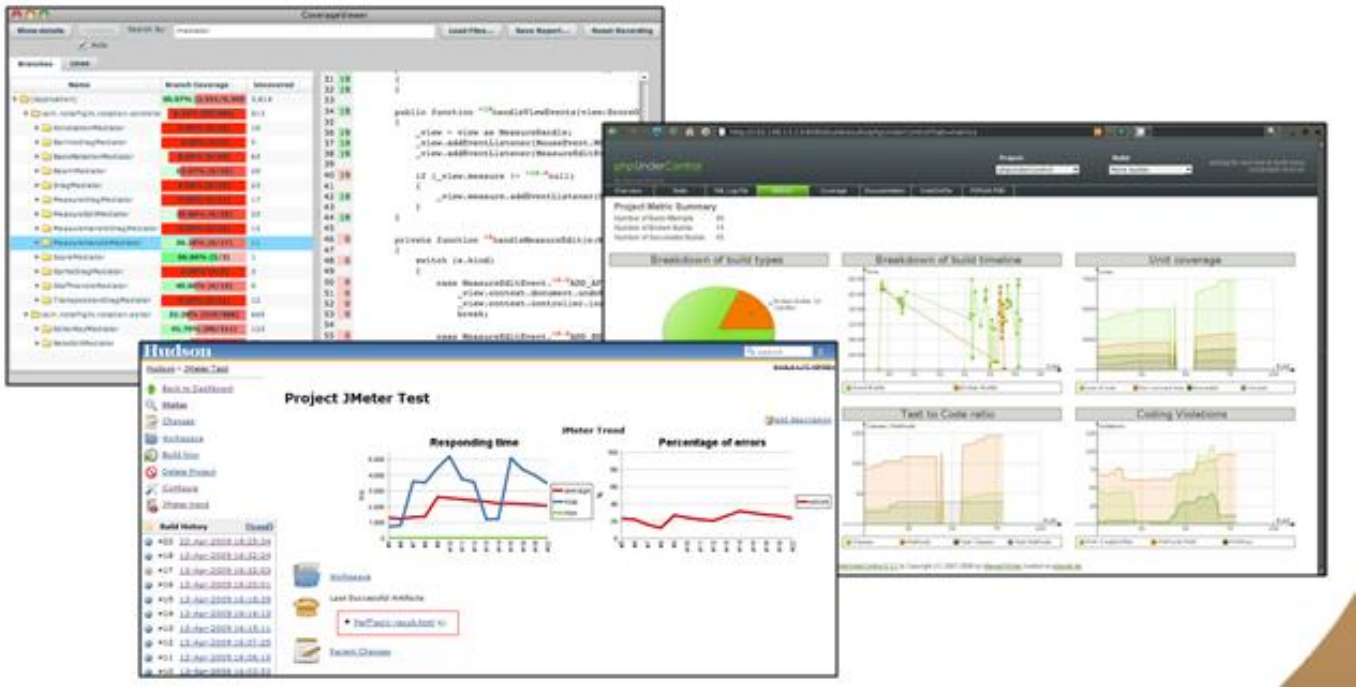
Cas d'utilisation



Les outils de l'intégration continue

Cas d'utilisation

Le chef de projet analyse le reporting



Intégration Continue

Test unitaires

Test unitaires

Présentation

- ✓ **Procédure permettant de vérifier le bon fonctionnement d'une partie précise d'un logiciel ou d'une portion d'un programme**
- ✓ **Confronter une réalisation à sa spécification**
- ✓ **Vérifier que la relation d'entrée / sortie donnée par la spécification est bel et bien réalisée.**

Test unitaires

Ecrire des tests unitaires.

- ✓ **Structure d'une classe de test**
- ✓ **Vérifier la justesse d'un test**
  - Vérifier la cohérence entre valeurs attendues et valeurs courantes
  - Vérifier qu'une exception a bien été lancée
  - Vérifier qu'une méthode s'exécute dans un temps raisonnable

Test unitaires

Ecrire des tests unitaires.

```
[TestClass]
public class MyClassTest
{
    [AssemblyInitialize]
    public static void MyAssemblyInitialize(TestContext context)
    {
        //1. Sera lancé une et une seule fois avant que le moindre test
        //   de l'assembly ne soit exécuté
    }

    [ClassInitialize]
    public static void MyClassInitialize(TestContext testContext)
    {
        //2. Sera lancé une et une seule fois avant que le moindre test
        //   de la classe de test ne soit exécuté
    }

    [TestInitialize]
    public void MyTestInitialize()
    {
        //3. Sera lancé avant chacun des tests de la classe de test
        //   Cette méthode est donc commune à tous les tests de la classe
    }

    [TestMethod]
    public void MyTest()
    {
        //4. Test à exécuter - Ne sera lancé qu'une seule fois
    }

    [TestCleanup]
    public void MyTestCleanup()
    {
        //5. Sera lancé après chacun des tests de la classe de test
        //   Cette méthode est donc commune à tous les tests de la classe
    }

    [ClassCleanup]
    public static void MyClassCleanup()
    {
        //6. Sera lancé une et une seule fois après que tous les tests de
        //   la classe de test aient été exécutés
    }

    [AssemblyCleanup]
    public static void MyAssemblyCleanup()
    {
        //7. Sera lancé une et une seule fois après que tous les tests de
        //   l'assembly aient été exécutés
    }
```

Test unitaires

Créer un projet de test unitaire

DEMO

EXPLICATION & CONTRAINTES

MERCI DE VOTRE ATTENTION

AVEZ-VOUS DES QUESTIONS ?