



After work Agilité

Stéphane Malbéqui

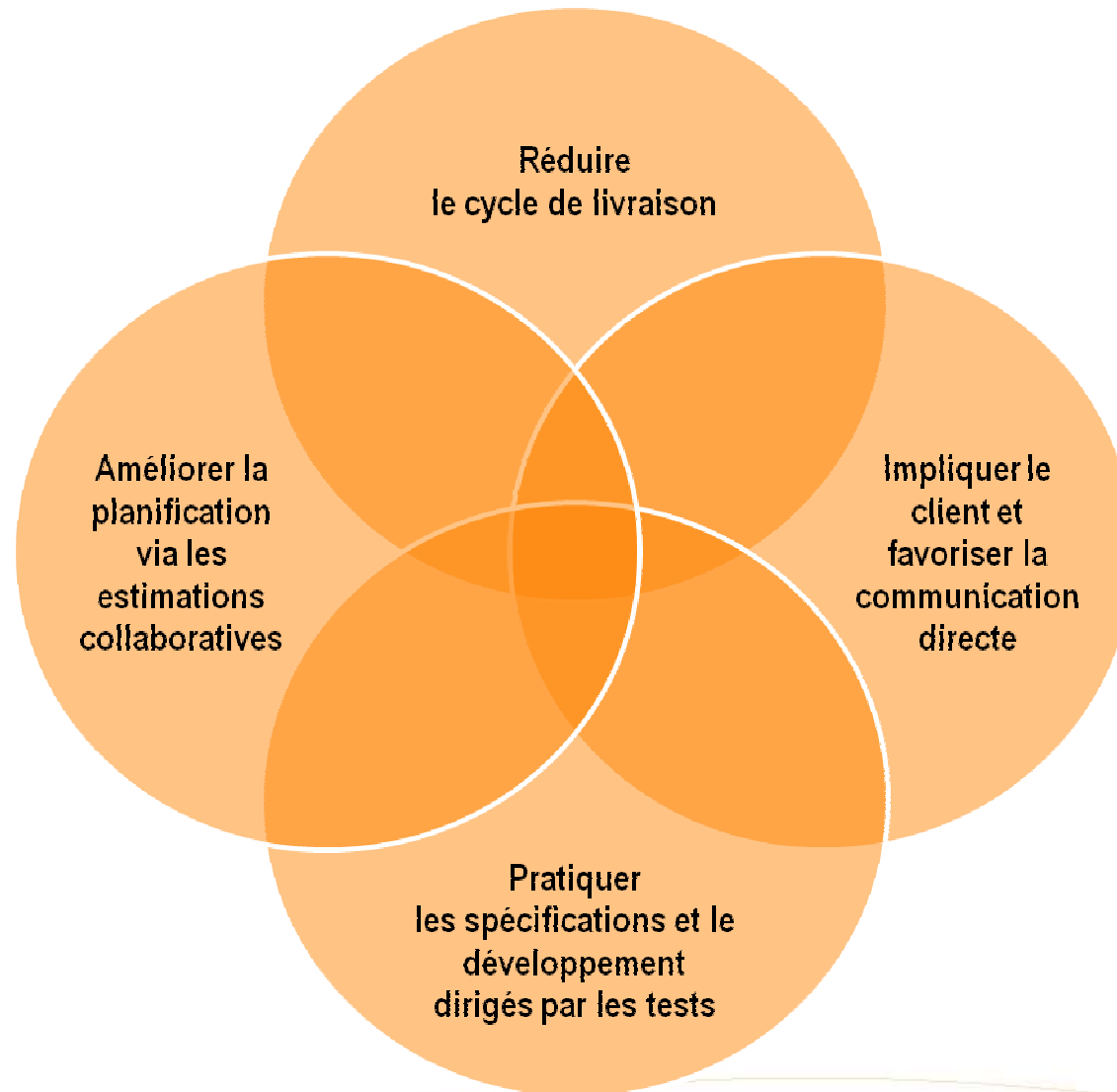
Consultant Senior Valtech Technology

stephane.malbequi@valtech.fr

delivering business agility

18 février 2009

Les 4 pratiques clés d'un projet agile



Les 4 valeurs du manifeste Agile

▪ Les pratiques clés d'un projet agile

- Réduire le cycle de livraison
- Améliorer la planification via les estimations collaboratives
- Impliquer le client et favoriser la communication directe
- Pratiquer les spécifications et le développement dirigés par les tests

▪ Zoom par l'exemple sur les pratiques TDR & TDD

- Les tests unitaires : la pierre angulaire
- Les frameworks JUnit4, Easymocks & Fit
- Démonstration interactive

Les 4 valeurs du manifeste Agile



Priorité aux personnes et aux interactions,
plutôt que sur les processus et les outils.

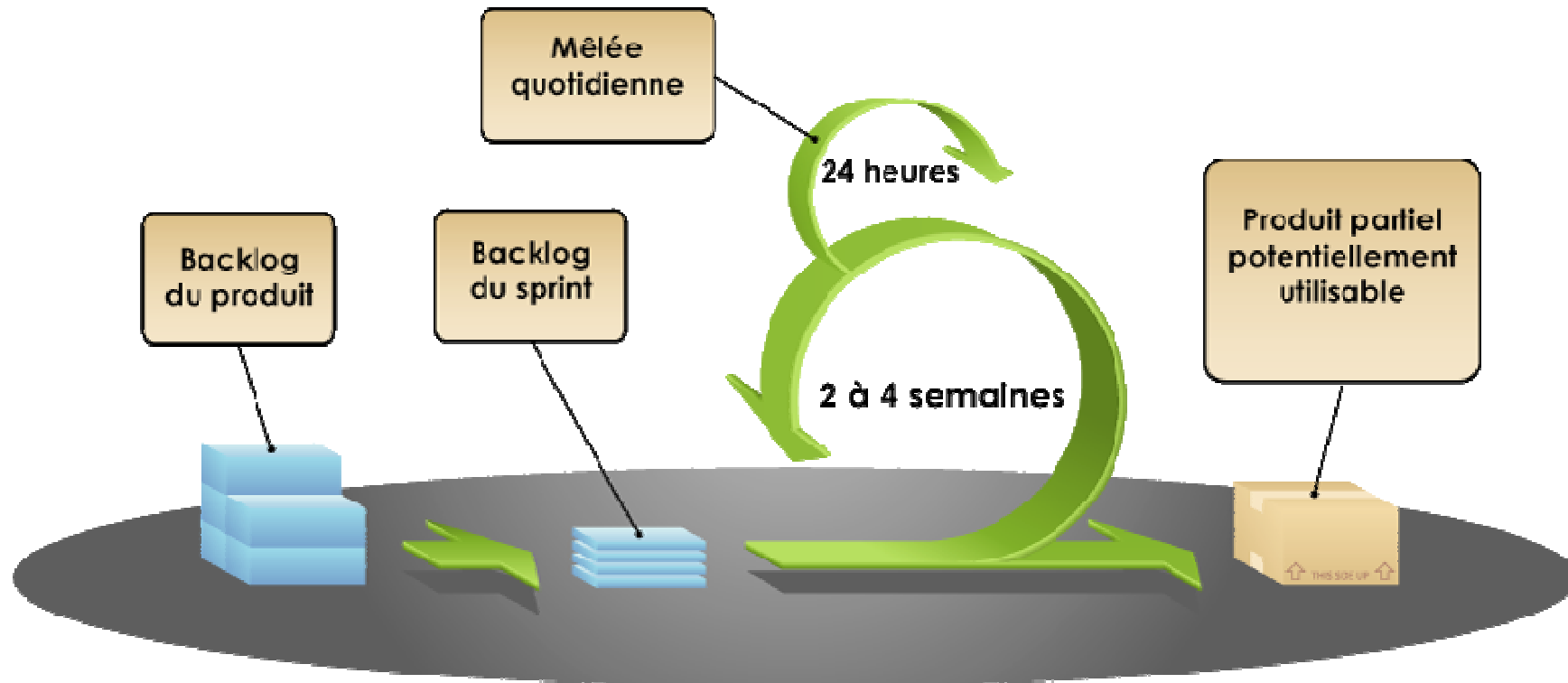
Des applications fonctionnelles opérationnelles,
plutôt qu'une documentation pléthorique.

Collaborer avec le client,
plutôt que négocier un contrat.

Réactivité au changement,
plutôt que suivre un plan.

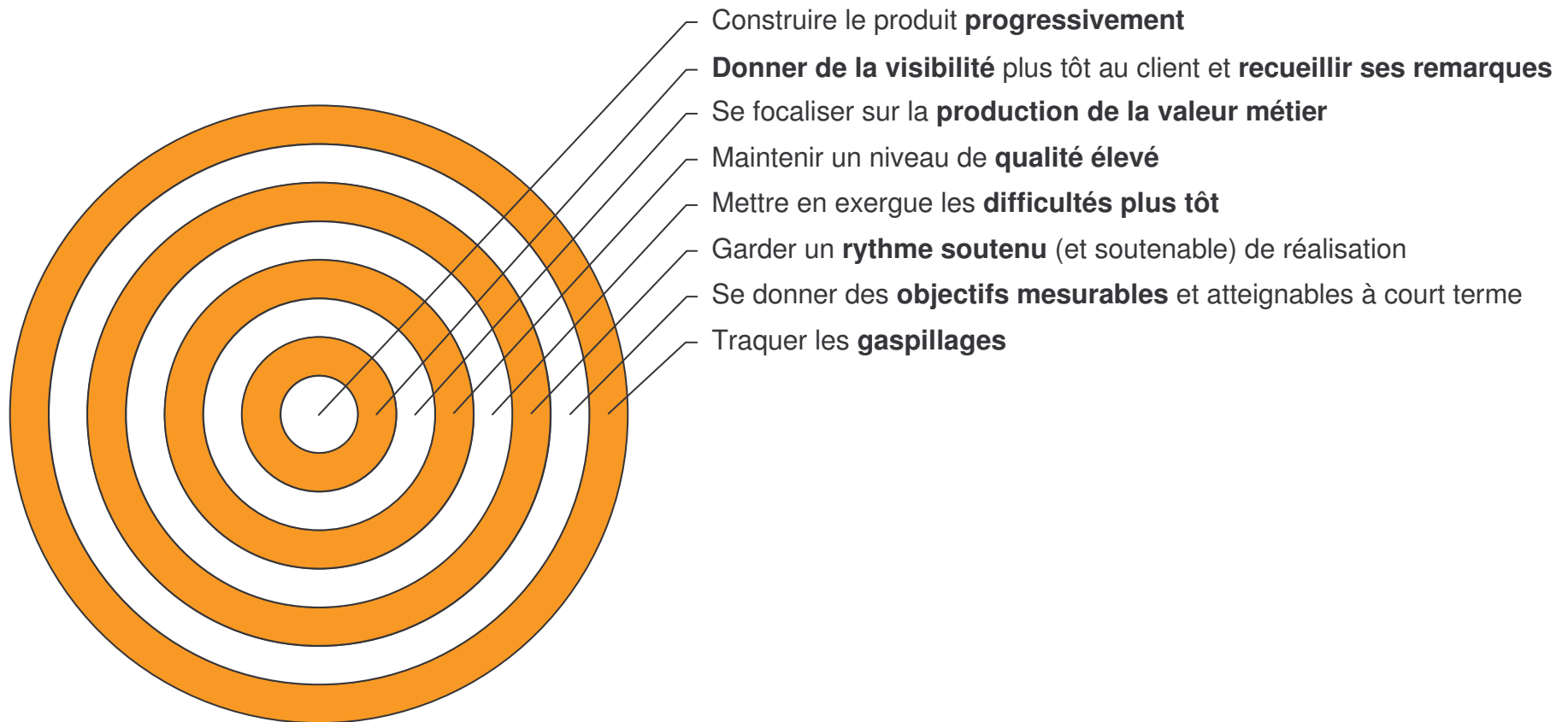
- **Les 4 valeurs du manifeste Agile**
- **Les pratiques clés d'un projet agile**
 - Réduire le cycle de livraison
 - Améliorer la planification via les estimations collaboratives
 - Impliquer le client et favoriser la communication directe
 - Pratiquer les spécifications et le développement dirigés par les tests
 - **Zoom par l'exemple sur les pratiques TDR & TDD**
 - Les tests unitaires : la pierre angulaire
 - Les frameworks JUnit4, Easymocks & Fit
 - Démonstration interactive

Réduire le cycle de livraison



COPYRIGHT © 2005, MOUNTAIN GOAT SOFTWARE

Les objectifs de la démarche itérative et la valeur ajoutée pour le client

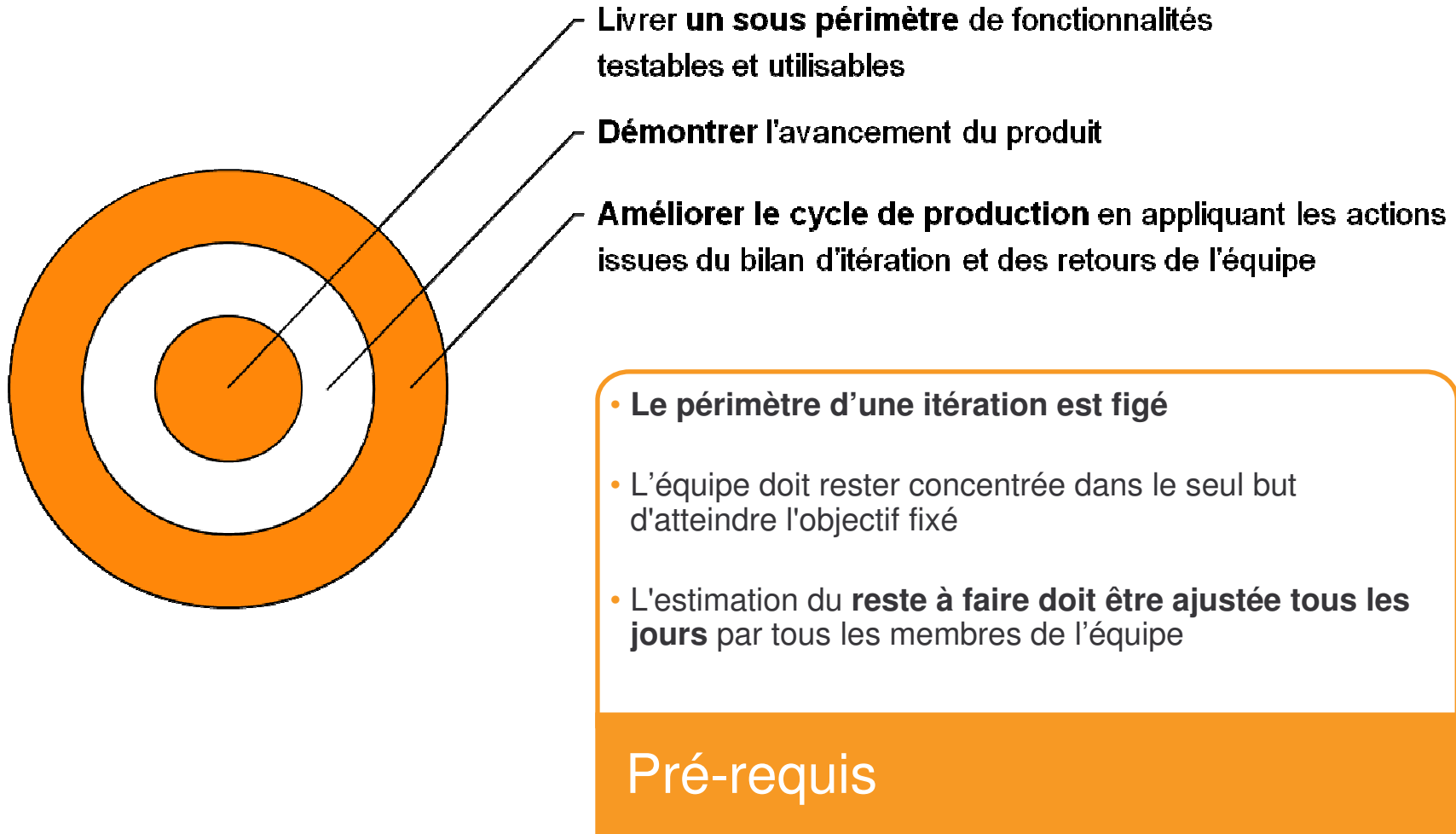


Pré-requis à la démarche itérative



- **Découper les applications en fonctionnalités métier, afin que chacune d'elles soit adoptable et démontrable dans une itération**
- **Prioriser les fonctionnalités en fonction de leur valeur métier**
- **Fixer la durée des itérations et les dates de fin d'itération**
- **Favoriser un travail à plein temps des membres de l'équipe sur le projet**
- **Etre en mesure de tester les fonctionnalités livrées**
 - Dédier des environnements de développement, d'homologation, d'intégration et de certification
 - Avoir un jeu de données de tests cohérent mais pas forcément exhaustif

Objectifs et pré-requis de l'itération



Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**

- Réduire le cycle de livraison

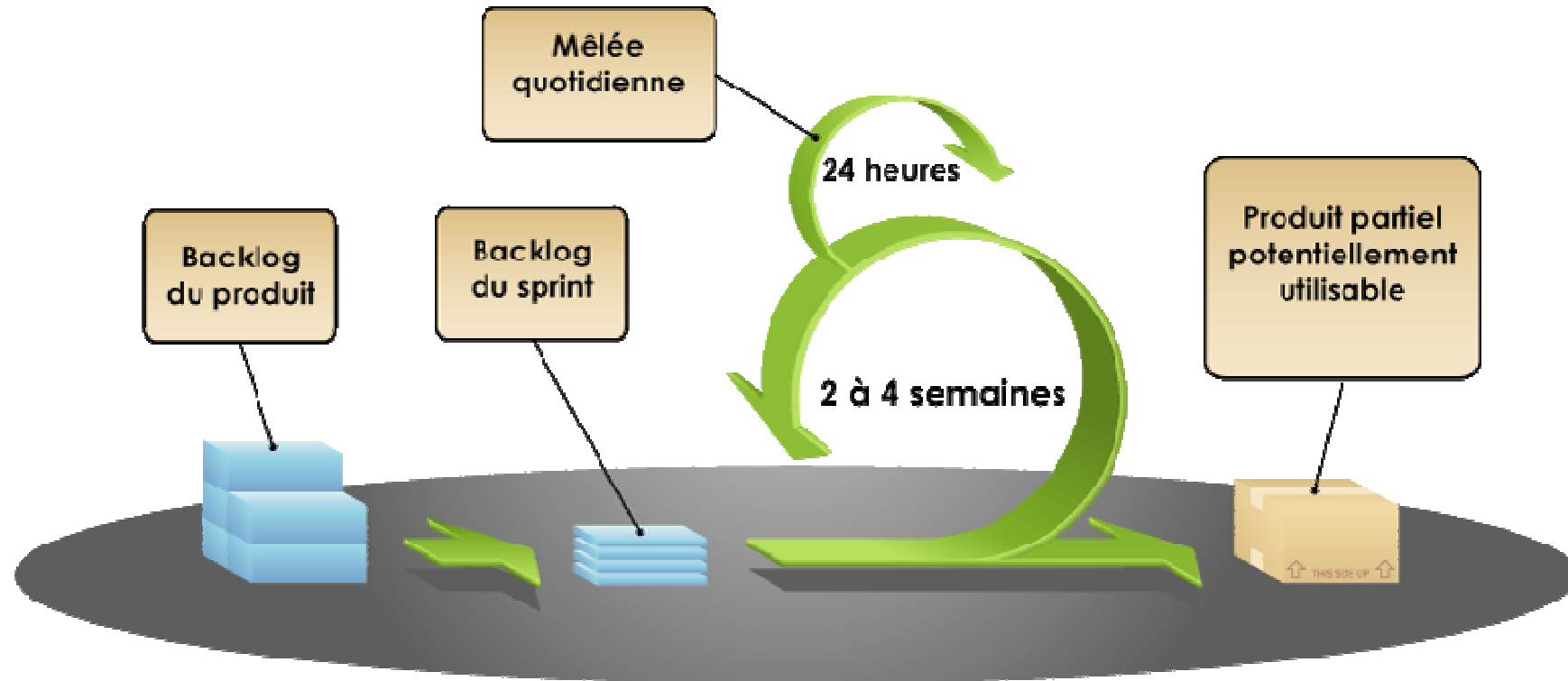
- Améliorer la planification via les estimations collaboratives

- Impliquer le client et favoriser la communication directe
- Pratiquer les spécifications et le développement dirigés par les tests

- **Zoom par l'exemple sur les pratiques TDR & TDD**

- Les tests unitaires : la pierre angulaire
- Les frameworks JUnit4, Easymocks & Fit
- Démonstration interactive

La planification et les estimations dans le processus Scrum



Copyright © 2005. MOUNTAIN GUY SOFTWARE

Améliorer la planification via les estimations collaboratives



- Les estimations collaboratives
 - Objectifs
 - Obtenir des estimations réalistes et pertinentes
 - Obtenir l'engagement de l'équipe
 - Gains
 - Meilleure prédictibilité, donc une meilleure visibilité, itération après itération
 - Planification réaliste
- Les outils de planification
 - Le « Product Backlog » et le « Product Burndown Chart »
 - [Product Backlog.xls](#)
 - Le « Sprint Backlog » et le « Sprint Burndown Chart »
 - [Sprint Backlog IT4 \(pdf\)](#)
 - [Sprint Burndown Chart IT4 \(pgn\)](#)

Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**

- Réduire le cycle de livraison
- Améliorer la planification via les estimations collaboratives

Impliquer le client et favoriser la communication directe

- Pratiquer les spécifications et le développement dirigés par les tests

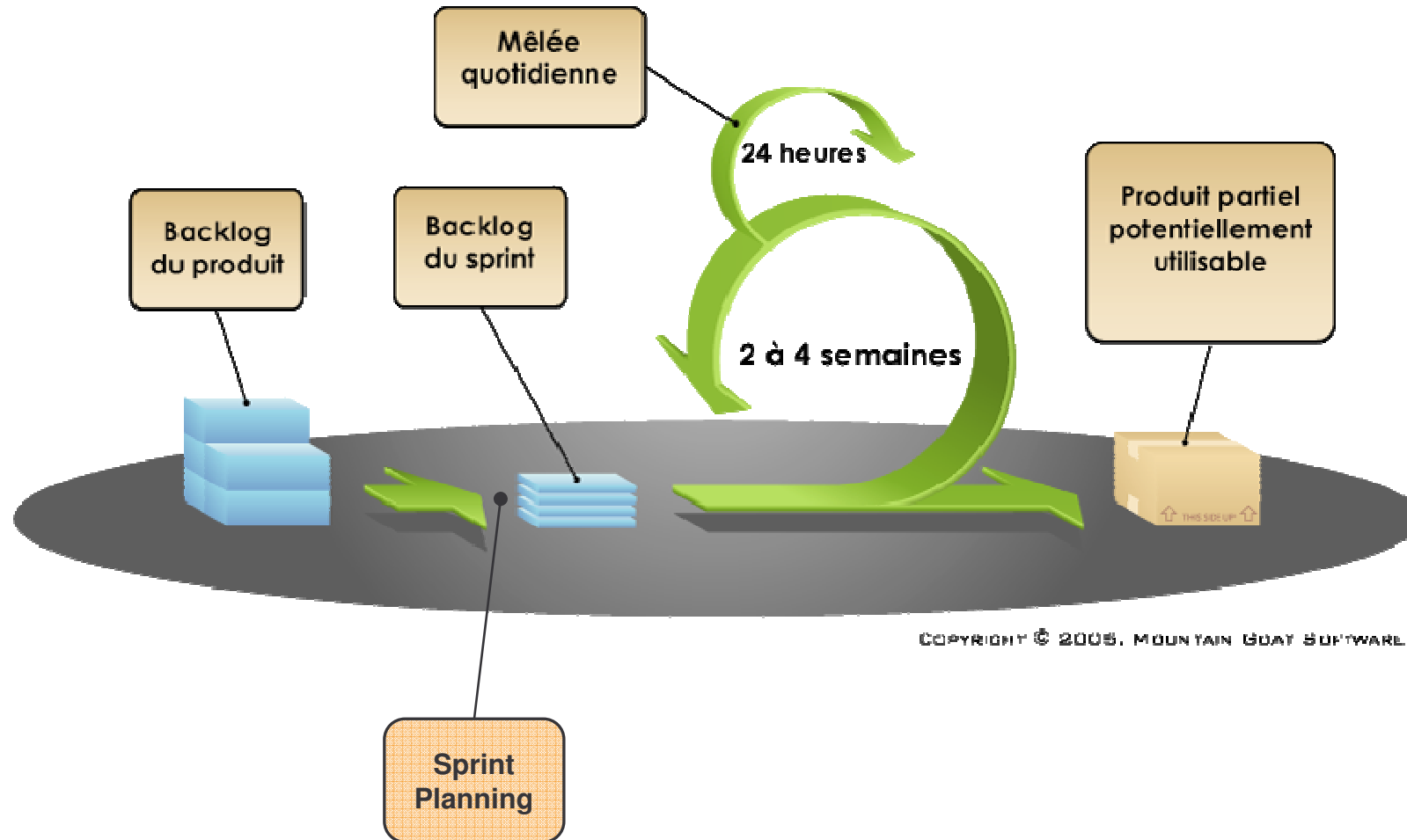
- **Zoom par l'exemple sur les pratiques TDR & TDD**

- Les tests unitaires : la pierre angulaire
- Les frameworks JUnit4, Easymocks & Fit
- Démonstration interactive

Rôles et responsabilités

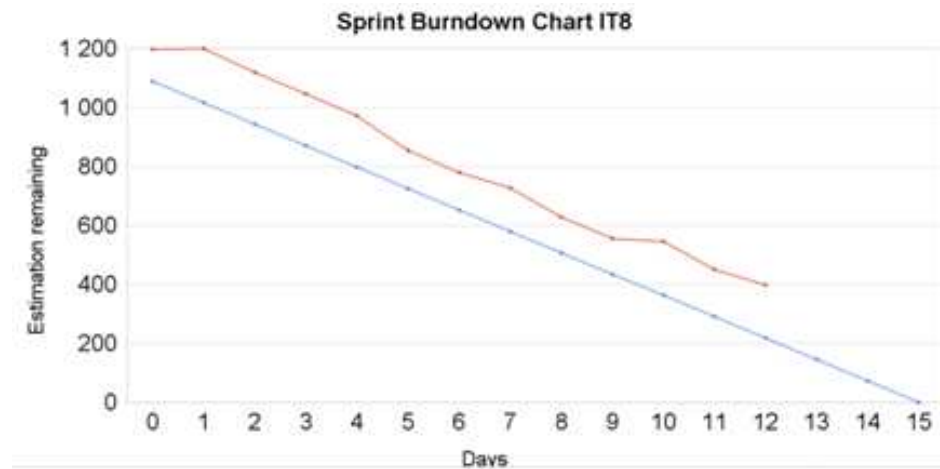
 <p>Product Owner</p>	<ul style="list-style-type: none">• Définit les fonctionnalités du produit• Priorise les fonctionnalités en fonction de leur valeur métier et de leur ROI• Détermine les dates et le contenu des différentes versions du produit• Ajuste les fonctionnalités et les priorités à chaque itération si nécessaire• Accepte ou rejette les livraisons de chaque itération
 <p>Scrum Master</p>	<ul style="list-style-type: none">• Est garant de l'application du processus• Est le garant de la qualité des livraisons• Elimine les obstacles et collabore étroitement avec le product owner• Protège l'équipe des interférences extérieures
 <p>Le reste de l'équipe</p>	<ul style="list-style-type: none">• De 5 à 10 personnes• Regroupe tous les autres rôles (analyste, développeur, testeur, ergonomes,...)• De préférence à plein temps sur le projet• La composition de l'équipe ne doit pas changer en cours d'itération

4 cérémonies qui favorisent l'implication du client et la communication directe

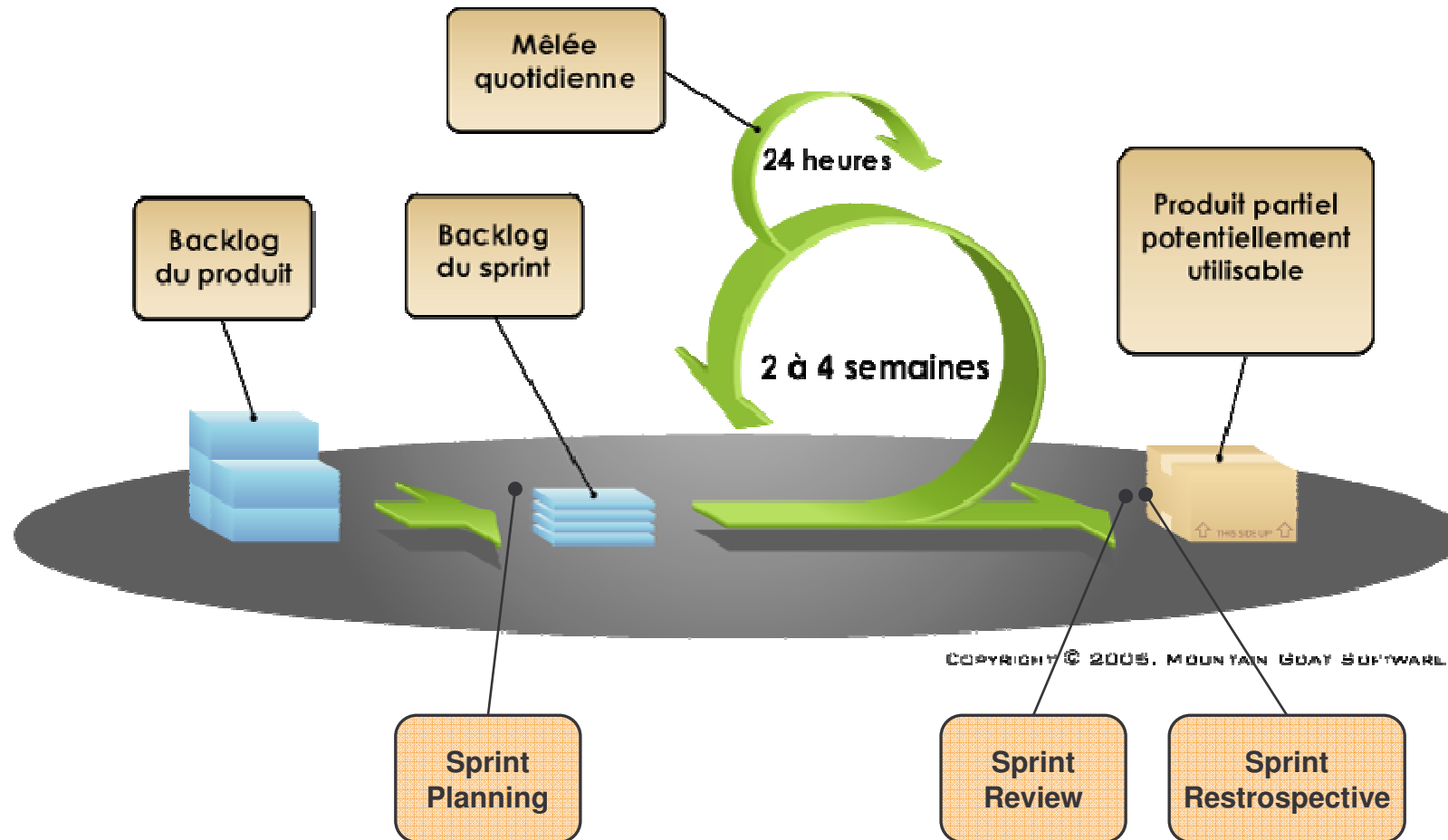


COPYRIGHT © 2005. MOUNTAIN GOAT SOFTWARE.

Daily Scrum Meeting



4 cérémonies qui favorisent l'implication du client et la communication directe



Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**
 - Réduire le cycle de livraison
 - Améliorer la planification via les estimations collaboratives
 - Impliquer le client et favoriser la communication directe

Pratiquer les spécifications et le développement dirigés par les tests

- **Zoom par l'exemple sur les pratiques TDR & TDD**
 - Les tests unitaires : la pierre angulaire
 - Les frameworks JUnit4, Easymocks & Fit
 - Démonstration interactive

Pourquoi mettre en œuvre le TDR et le TDD ?



■ Constat

- écart entre le besoin exprimé et le produit final
- qualité des livraisons non satisfaisante
- délai trop long entre l'expression du besoin et la livraison de la fonctionnalité
- développer les tests après avoir développé les fonctionnalités est une tâche ingrate, pénible et peu efficace

■ Objectifs du TDR et du TDD

- améliorer l'adéquation entre le besoin exprimé et le produit final
- améliorer la qualité globale de l'application
- raccourcir les délais de livraison

Qu'est-ce que le TDR ?

- Le Test Driven Requirement est une pratique permettant d'améliorer le recueil et la formalisation des exigences fonctionnelles
- Elle consiste à rédiger de manière collaborative, les tests en amont des développements, en s'appuyant sur des exemples qui facilitent l'exploration du besoin et la formalisation des spécifications qui deviennent des livrables vivants et exécutables
- Elle permet d'éliminer les gaspillages tels que
 - la multiplication des intermédiaires et des échanges formels entre ces intermédiaires
 - les fonctionnalités non nécessaires
 - les anomalies non détectées
 - les tests de non-régression manuels
- Exemples de spécifications actives

TDR : Rédiger les cas de test fonctionnels « avant » les spécifications

Qu'est-ce que le TDD ?

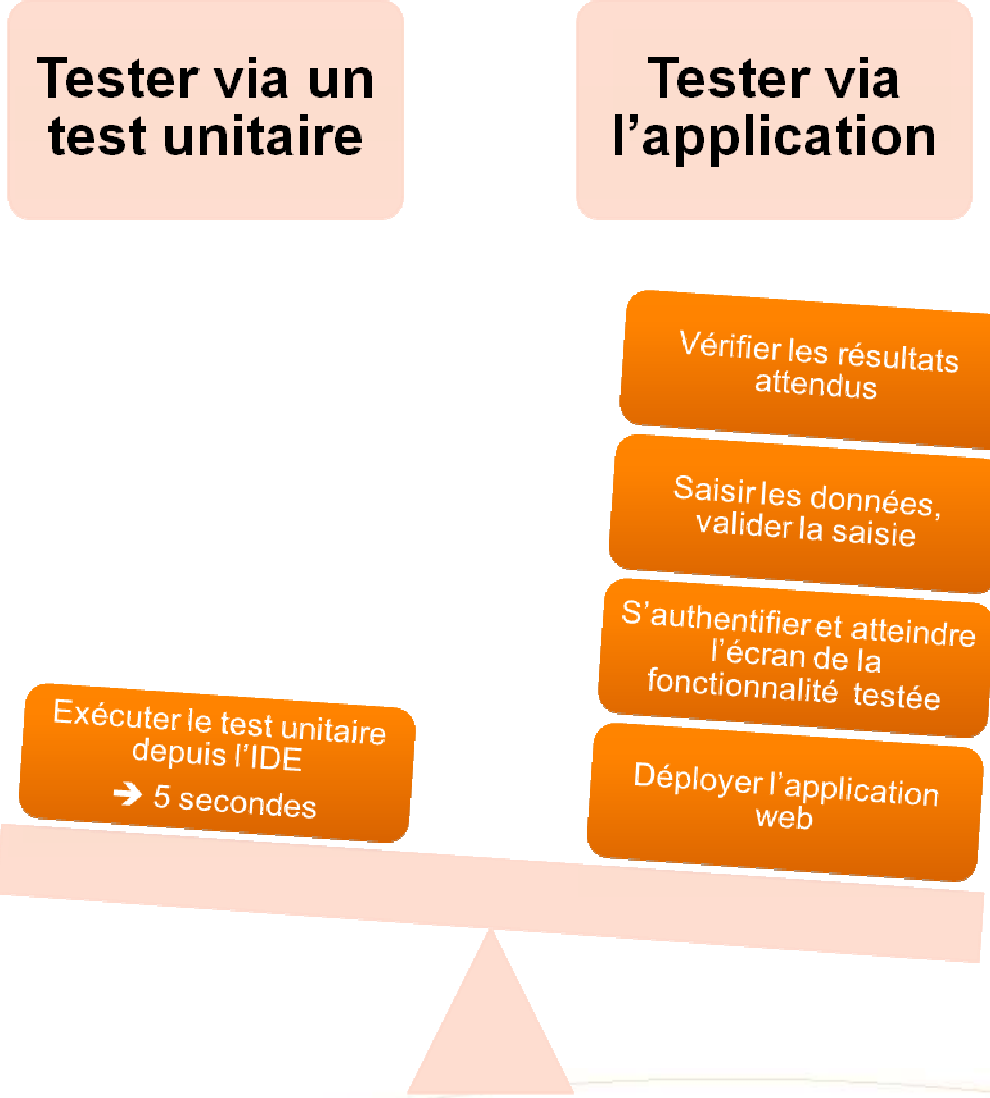
- Le Test Driven Development est une pratique qui consiste à développer les tests unitaires avant même de développer les fonctionnalités

- Elle permet d'améliorer la qualité du code, du design et de sa maintenance
 - Un design simple
 - Une meilleure séparation des responsabilités
 - Un couplage faible entre les composants développés
 - Une meilleure couverture de code (> 80%)
 - Un code documenté

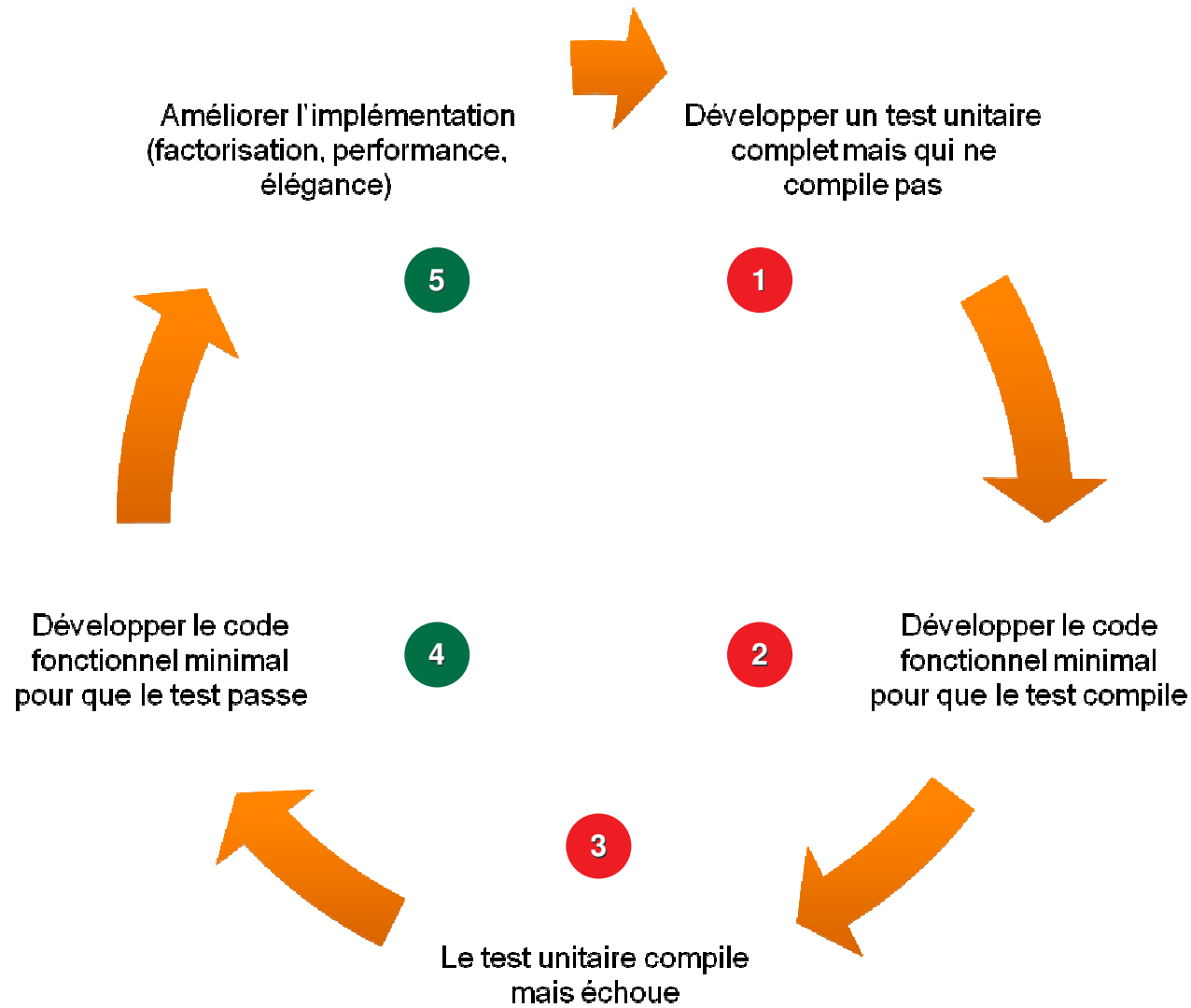
- Elle permet d'améliorer la productivité des équipes de développement
 - Permet de mieux appréhender le besoin et les exigences
 - Limite le gaspillage car seules les fonctionnalités nécessaires à l'exécution du test sont développées
 - Réduit l'effort de débogage
 - Réduit la durée du cycle de développement (code, test, code, test...)

TDD : Développer les tests unitaires « avant » les fonctionnalités

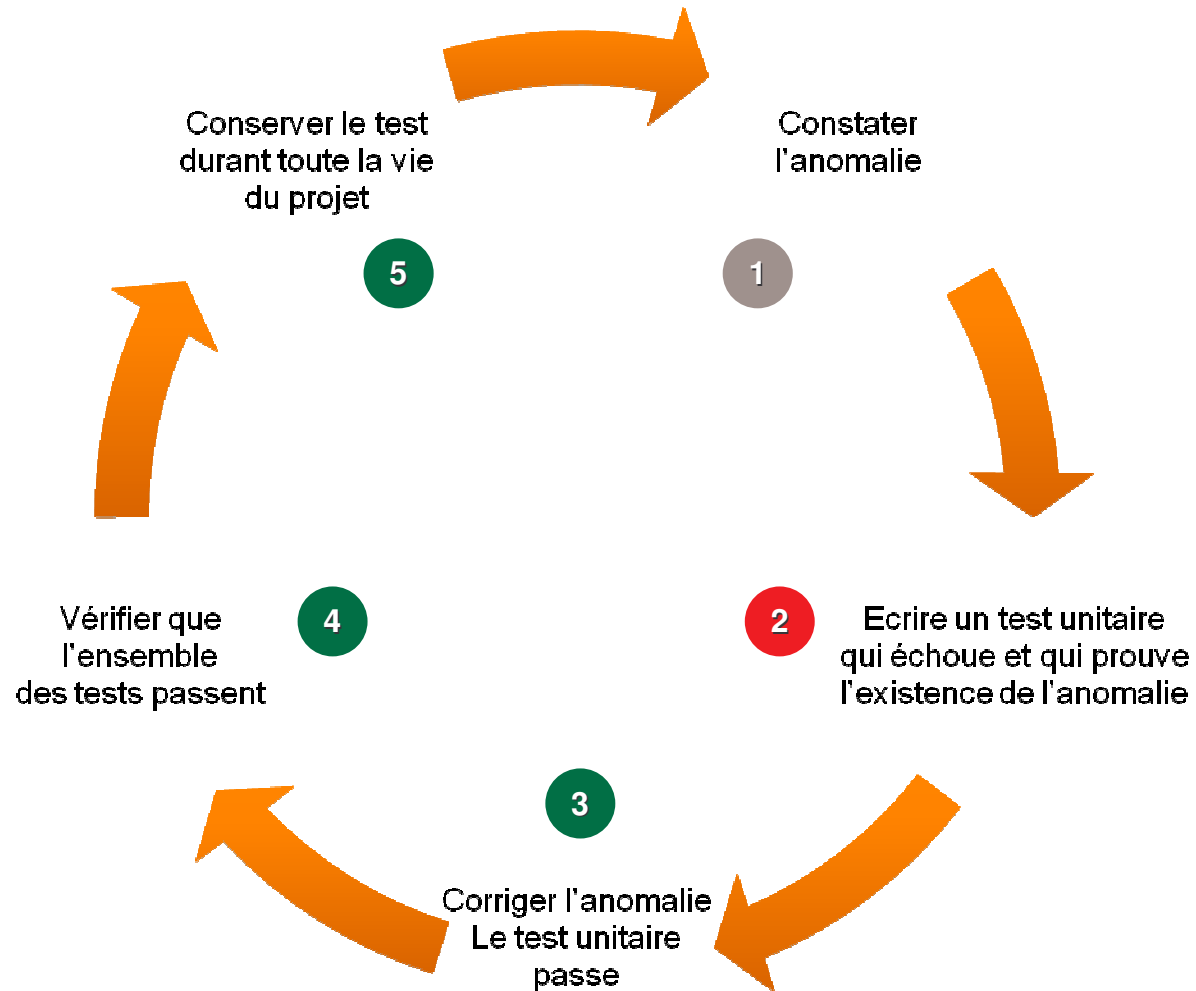
Gain de temps concret du TDD



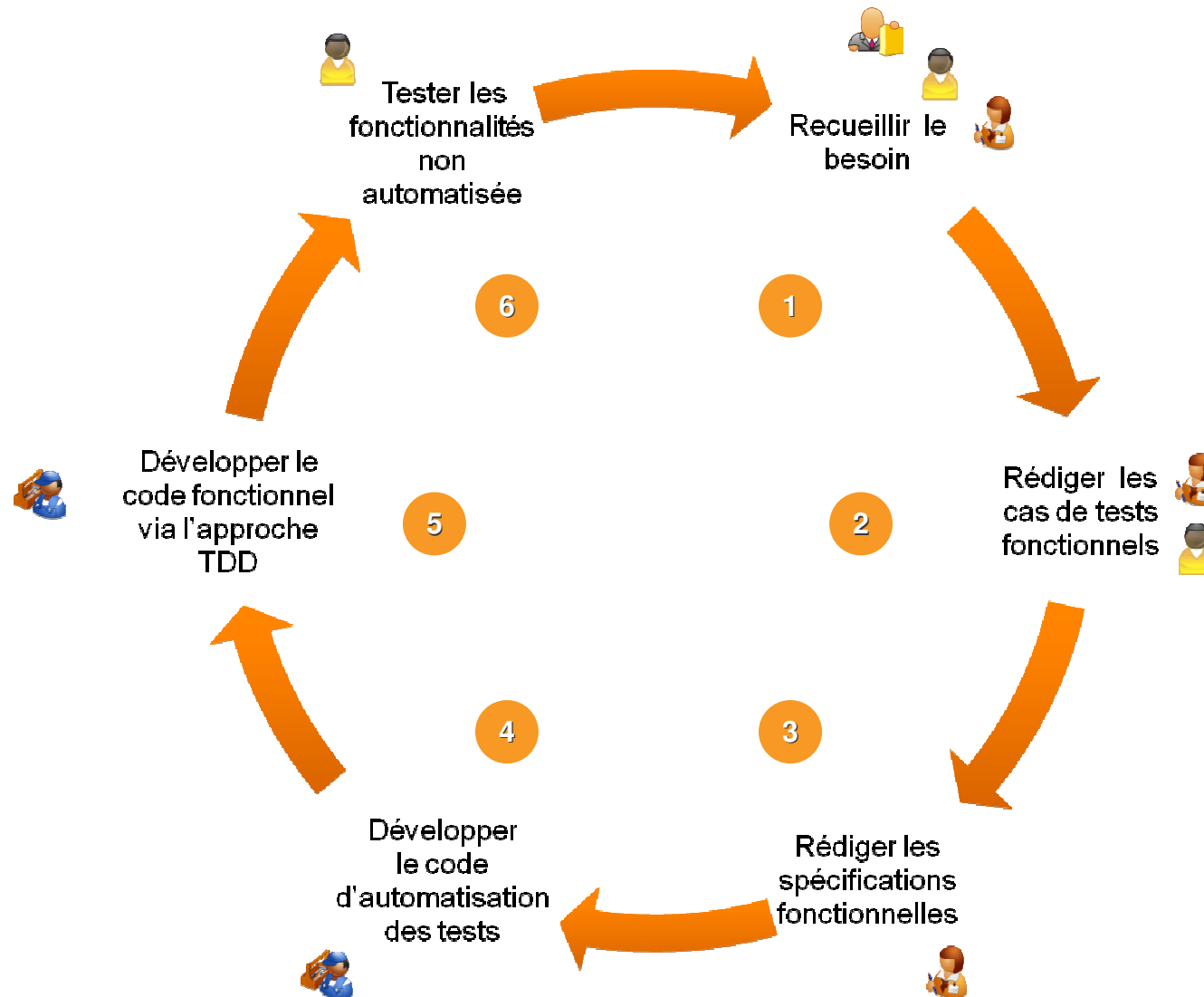
Cycle d'un développement dirigé par les tests (Red / Green / Refactor)



Correction des anomalies avec TDD



Pratiquer le TDR et le TDD



Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**
 - Réduire le cycle de livraison
 - Améliorer la planification via les estimations collaboratives
 - Impliquer le client et favoriser la communication directe
 - Pratiquer les spécifications et le développement dirigés par les tests

- **Zoom par l'exemple sur les pratiques TDR & TDD**

Les tests unitaires : la pierre angulaire

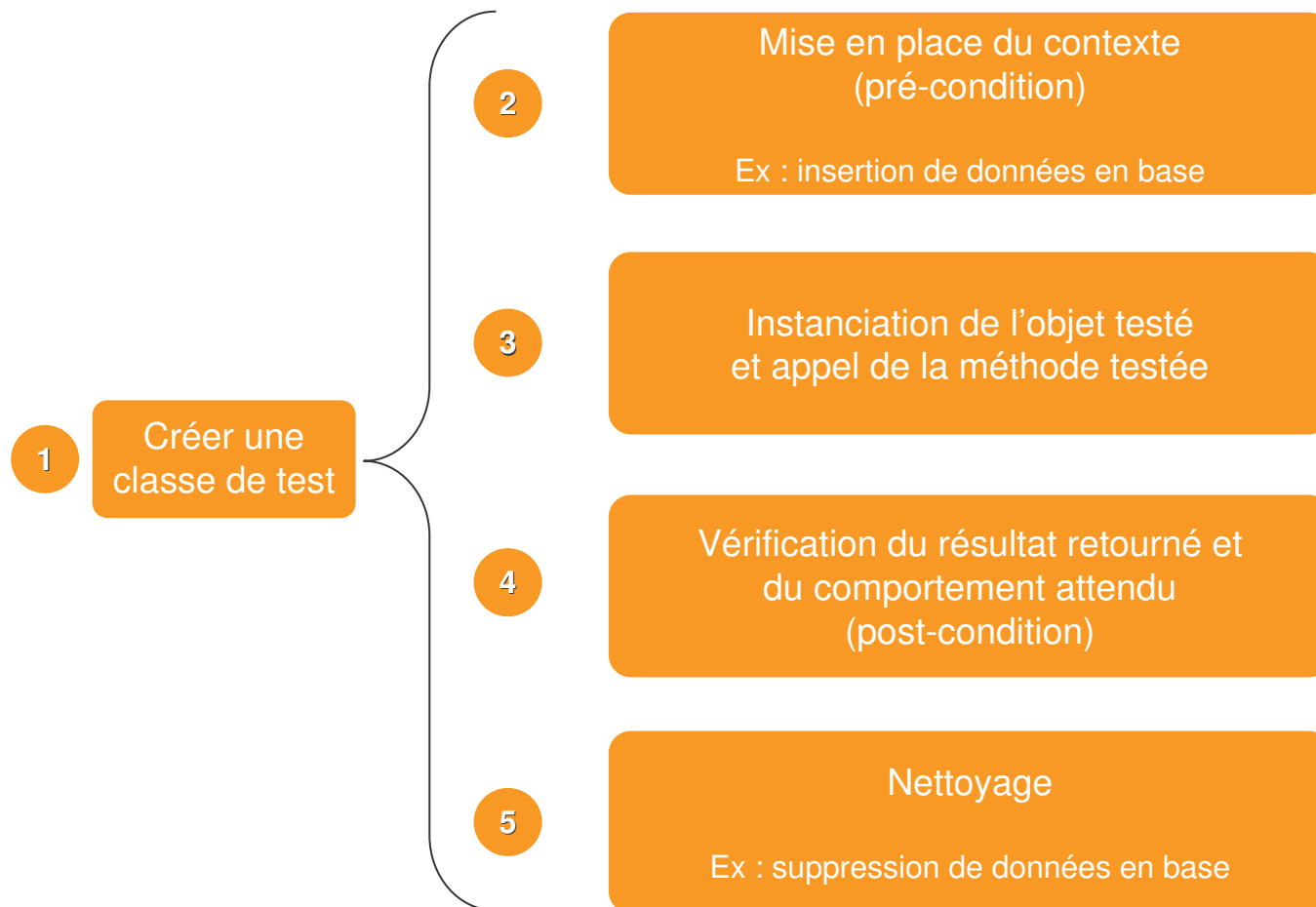
- Les frameworks JUnit4, Easymocks & Fit
- Démonstration interactive

Qu'est-ce qu'un test unitaire ?



- Un test unitaire est du code permettant de tester une petite partie de code fonctionnel telle qu'une classe ou une méthode.
- Un test unitaire permet de s'assurer que le code fait bien ce que l'on lui demande de faire dans un contexte bien défini par des pré et post conditions
- Un test unitaire doit être
 - indépendant
 - répétable
 - simple et rapide à développer et à exécuter
 - exécuté automatiquement
- Un test unitaire fait partie du code de l'application développée
 - il est versionné au même titre que le code fonctionnel (SCM)
 - il est disponible durant toute la vie du projet, il n'est jamais supprimé
 - il n'est pas déployé en production

Comment écrire un test unitaire ?



Exemple de test unitaire en C#



```
[TestFixture]
public class ServicesFinanciersTests
{
    [SetUp]
    public void SetUp()
    {
        SqlHelper.Execute("insert into Valeur(code,valeur) value('0004155885',20.12)");
    }

    [Test]
    public void RecupererValeur()
    {
        IServicesFinanciers servicesFinanciers = new ServicesFinanciers();
        double valeur = servicesFinanciers.RecupererValeur("0004155885");
        Assert.AreEqual(20.12,valeur);
    }

    [TearDown]
    public void TearDown()
    {
        SqlHelper.Execute("delete Valeur where code='0004155885'");
    }
}
```



Comment écrire un test unitaire efficace ?



- **Pour chaque test, se poser les questions suivantes**

- Les résultats correspondent-ils aux résultats attendus ?
- Est-ce que les conditions aux limites ont été testées ?
- Puis-je forcer des conditions d'erreur et tester le comportement de mon composant en conséquence ?
- Les performances de mon composant répondent-elles aux exigences ?
- Est-ce que le test documente la fonctionnalité implémentée ?

- **Ne tester qu'un seul composant à la fois**

- Tous les composants dont dépend le composant testé, doivent être remplacés par des « mocks »
- Les « mocks » sont des classes dédiées aux tests, non livrées en production, qui ont pour objectif de simuler l'implémentation réelles des dépendances

Exemple d'utilisation de « mocks »



```
[Test]
public void CalculerValeurTotale()
{
    MockRepository mocks = new MockRepository();
    IServicesFinanciers servicesFinanciers = mocks.CreateMock<IServicesFinanciers>();

    using (mocks.Record())
    {
        Expect.Call(servicesFinanciers.RecupererValeur("FR0004155885")).Return(20.37);
        Expect.Call(servicesFinanciers.RecupererValeur("FR0000047072")).Return(13.50);
    }

    using (mocks.Playback())
    {
        GestionCompteTitres gestionCompteTitres = new GestionCompteTitres();
        gestionCompteTitres.AjouterPosition("FR0004155885", "100", 20.10);
        gestionCompteTitres.AjouterPosition("FR0000047072", "2500", 11.30);

        gestionCompteTitres.ServicesFinanciers = servicesFinanciers;

        double total = gestionCompteTitres.CalculerValeurTotale();
        Assert.Equals(((20.37f*100) + (13.50f*2500)), total);
    }
}
```



Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**
 - Réduire le cycle de livraison
 - Améliorer la planification via les estimations collaboratives
 - Impliquer le client et favoriser la communication directe
 - Pratiquer les spécifications et le développement dirigés par les tests

 - **Zoom par l'exemple sur les pratiques TDR & TDD**
 - Les tests unitaires : la pierre angulaire
- Les frameworks JUnit4, Easymocks & Fit
- Démonstration interactive

- **JUnit fournit une API permettant de développer des tests unitaires java**

- Fournit depuis la version 4 des annotations qui facilitent l'écriture des tests unitaires
 - @Test, @Before, @After...
- Fournit des classes et des méthodes d'assertions
 - Assert.assertTrue(*condition*)
 - Assert.assertEquals (*resultatAttendu, resultat*)
 - Assert.assertNotNull(*objet*)

- Easymock fournit une API permettant de créer dynamiquement des mocks à partir d'interface java

- MonService service = Easymock.createMock(MonService.class);
- Easymock.except(service.doSomething()).andReturn(3);

- FIT fournit une API java permettant d'automatiser les tests fonctionnels rédigés sous forme de tableaux

- FitNesse est un wiki qui embarque le moteur FIT

Les 4 valeurs du manifeste Agile

- **Les pratiques clés d'un projet agile**
 - Réduire le cycle de livraison
 - Améliorer la planification via les estimations collaboratives
 - Impliquer le client et favoriser la communication directe
 - Pratiquer les spécifications et le développement dirigés par les tests

- **Zoom par l'exemple sur les pratiques TDR & TDD**
 - Les tests unitaires : la pierre angulaire
 - Les frameworks JUnit4, Easymocks & Fit

Démonstration interactive