Programmation Agile
–
Mise en oeuvre via Scrum et l'Extreme Programming (XP)

B. Mermet
2010

Plan

- La programmation Agile et L'artisanat du logiciel
- Mise en œuvre avec Scrum
- Mise en œuvre avec l'eXtreme Programming
- Programmation agile et intégration continue ; application avec Hudson

La notion de programmation Agile

Le Manifeste Agile (2001)
http://agilemanifesto.org/

Manifeste pour le développement de logiciel Agile

Nous découvrons de meilleures façons de développer des logiciels en le faisant et en aidant les autres à le faire. Grâce à ce travail, nous avons été amenés à privilégier :

- **Les individus et les interactions** aux processus et outils ;
- **Les logiciels qui fonctionnent** à une documentation exhausive ;
- **La collaboration avec le client** à la négociation de contrat ;
- **La réactivité aux changements** au respect d'une planification.

C'est pourquoi, même si les critères "à droite" ont leurs intérêts, nous privilégions les critères "à gauche".

Manifeste Agile
Principes sous-jacents (1)

Nous suivons les principes suivants :

1. Notre priorité est de satisfaire le client en lui fournissant rapidement et en continu des logiciels utiles.

2. Les modifications dans les besoins sont les bienvenues, même tardivement dans le développement. Les changements dans les processus agiles sont faits à l'avantage du client.

3. Fournir des logiciels fonctionnels fréquemment, de quelques semaines à quelques mois, avec une préférence pour la période la plus courte.

4. Les chargés d'affaires et les développeurs doivent travailler ensemble quotidiennement durant le projet.

Manifeste Agile
Principes sous-jacents (2)

5. Construire des projets grâce à des gens motivés. Leur donner l'environnement et le support dont ils ont besoin, et leur faire confiance pour que le travail soit réalisé.

6. La façon la plus efficace et la plus concrète de véhiculer de l'information vers et au sein d'une équipe de développement est la conversation face à face.

7. Un logiciel qui fonctionne est la première mesure de progrès.

8. Les processus agile prônent un développement soutenable. Les animateurs, développeurs, et utilisateurs doivent être en mesure de maintenir un rythme constant indéfiniment.

9. Une attention permanente portée à l'excellence technique et à une bonne conception améliore l'agilité.

Manifeste Agile
Principes sous-jacents (3)

10. La simplicité – l'art de maximiser la somme de travail non fait – est essentielle.

11. Les meilleurs architectures, besoins, conceptions émergent d'équipes auto-organisées.

12. A des intervalles réguliers, l'équipe réfléchit à comment devenir plus efficace, puis adapte et affine sont comportement en fonction.

Manifeste pour l'artisanat du logiciel
http://manifesto.softwarecraftsmanship.org/

En tant qu'aspirants "artisans logiciels", nous relevons la barre du développement de logiciel professionnel, en le pratiquant et en aidant les autres à apprendre le métier. Grâce à ce travail, nous avons été amenés à privilégier :

- Pas seulement des logiciels qui marchent, mais des logiciels bien conçus ;
- Pas seulement de la réactivité aux changements, mais un ajout régulier de valeur ;
- Pas seulement des individus et des interactions, mais aussi une communauté de professionnels ;
- Pas seulement une collaboration avec le client, mais aussi des partenariats productifs.

Ainsi, dans la continuité des items "à gauche", nous avons été amenés à considérer les items "à droite" comme indispensables.

Scrum

Scrum
Présentation générale

- Méthodologie de gestion globale d'un projet agile
- Scrum = mêlée en rugby

  on fait progresser le ballon en travaillant ensemble
- Reprise des principes de base « agile »
  - Des itérations courtes, donnant lieu à des livrables
  - On travaille mieux ensemble
    - Pas de « spécialistes »
    - Pas de primes individuelles sur les performances
  - Le rythme de développement doit être soutenable
  - Une équipe qui s'auto-organise est plus efficace

Scrum
Les rôles

- Cochons (Pigs)
  - Product owner (propriétaire du logiciel)

    Client, représentant des utilisateurs (rôle proche de celui du MOA en développement classique)
  - ScrumMaster

    Vérifie la mise en œuvre de Scrum, au service de l'équipe, assurant la bonne collaboration entre l'équipe et le Product owner
  - Equipe (Team) d 'intervenants (Stackholders)

    Groupe de 5 à 10 personnes travaillant ensemble pour spécifier, coder, valider, documenter des fonctionnalités

- Poulets (Chickens)

  Personnes extérieures au développement mais intéressées (utilisateurs finaux, patron, etc.)

Initialisation du processus : Création du Backlog produit

- Backlog produit (Product Backlog) ; backlog ≈ restant dû
  - Définition

    liste ordonnée de points à développer pour le logiciel.
  - Caractéristiques d'un item
    - Définition
    - Effort estimé
    - Valeur
- Démarche
  - Le Product Owner génère la liste des items
  - L'équipe associe à chaque item un « effort estimé »
  - Le Product Owner, éventuellement assisté du Scrum Master, donne une valeur à chaque item
  - Le Product Owner associe une priorité à chaque item, par exemple pour maximiser le Retour Sur Investissement (Return On Investment)

Scrum
Estimation de l'effort

- Principe général

  Estimation **relative** entre les items d'un backlog et non absolue

- Éviter les ambiguïtés

  Utilisation recommandée des valeurs de la suite de Fibonacci : 1, 2, 3, 5, 8, 13, 21, 34, 55, …

- Favoriser une bonne estimation très rapide à une très bonne estimation potentiellement très fausse mais lentre

  Utilisation du *planning poker*

Planning Poker

1. Les membres de l'équipe disposent chacun d'un jeu de carte suivant la suite de Fibonacci (avec éventuellement une carte ∞)
2. Chacun joue une carte face cachée représentant son estimation de l'effort
    a) Si tout le monde est d'accord, ou presque, l'estimation est réalisée
    b) Sinon, le plus pessimiste et le plus optimiste expliquent leur vote, des discussions rapides ont lieu, puis on recommence en 2.

- Rôle de la carte ∞
  trop long ; à séparer en au moins 2 parties

Scrum
Phases du processus

- Release

  aboutit à une version délivrée à échéance moyenne (6-12 mois), finalisée

- Sprint

  aboutit à une version non délivrée mais délivrable (testée, documentée) à échéance courte (4-6 semaines)

- Relation entre les phases
  - Un **Release Backlog** est associé à chaque *release* (extrait du *product Backlog*)
  - Un **Sprint Backlog** est associé à chaque sprint (à partir d'un extrait du R*elease Backlog* de la *release* à laquelle le sprint est associé)

Scrum
Initialisation d'un Sprint

- 2 réunions
  - Sprint Planning Meeting Part One
  - Sprint Planning Meeting Part Two
- Première réunion
  - Choix par le Product Owner des items du Release Backlog qu'il aimerait voir implantés dans ce sprint (en fonction de la *vélocité* de l'équipe)
  - Discussion entre l'équipe et le Product Owner pour mieux appréhender ce qui est attendu pour un item
- Deuxième réunion
  - Sélection définitive des items à implanter
  - Sans présence obligatoire du Product Owner (mais souhaitable), qui doit rester joignable
  - Découpage des items du Release Backlog choisis en tâches ajoutées au Sprint Backlog ; association d'un effort estimé (en heures) à chaque tâche

Scrum
Planning Meeting : contraintes

- Durée raisonnable
  - Inférieure ou égale à 8 heures pour un Sprint d'un mois
  - Estimation du temps effectif de développement quotidien par développeur (4-6 heures) et du temps total pendant le Sprint
- Conception préliminaire
  - Effectuée collectivement, avant la décomposition d'un item en tâches
- Pas d'attribution préalable des tâches à un participant sauf si compétence unique (vérifier alors les possibilités matérielles de réalisation)
- Une fois qu'un Sprint est initialisé, il doit se dérouler comme prévu jusqu'au bout.

Scrum
Sprint Backlog

- Tableau avec un ensemble de « post-it » (un par tâche assignée au sprint) répartis en 3 colonnes
  - À faire
  - En cours
  - Fait
- Passage de « à faire » à « en cours »

  Fait individuellement par chaque intervenant le matin
- Passage de « en cours » à « fait »

  A la fin de chaque journée, si tout ce qui a été spécifié comme indiquant qu'une tâche peut être considérée comme faite est réalisé ; si une tâche n'est pas finie, mise à jour de l'effort restant

Scrum
Sprint au quotidien

- Daily scrum

  Réunion quotidienne (type stand-up) de durée très réduite limitée (≈ ¼ d'heure) où chacun

  - expose
    - Ce qu'il a fait la veille
    - Ce qu'il compte faire ce jour
    - Les embûches éventuelles
  - N'engage pas de discussion

    Si cela semble nécessaire, après le daily scrum, lors d'un « follow-up meeting »
- Développement
- Mise à jour du *Sprint Backlog* et du *Sprint Burndown Chart*

Scrum
Sprint Burdown Chart

Graphique représentant

- En abscisse, l'écoulement du temps en jours du début à la fin du Sprint
- En ordonnée, le montant de travail restant à faire estimé
- La droite représentant un avancement linéaire idéal
- La courbe représentant l'avancement réel

Scrum
Fin d'un sprint

- Finir un Sprint à la date prévue
  - Cela doit être vrai que la totalité du Sprint soit réalisée ou non, sans multiplier les heures supplémentaires
  - En général
    - On commence par sur-estimer ses capacités
    - Puis on sous-estime
    - Enfin, on se règle (au bout de 4 sprints environ)
- Mise à jour du Release/Product Backlog
  - Prévoir une réunion de 5% du Sprint pour ce faire (1 jour sur un sprint d' 1 mois)
- Revue de Sprint
- Rétrospective de Sprint
- Mise à jour du Release/Product Backlog et du Burndown Chart

Scrum
Revue de Sprint

- **Inspection et adaptation concernant le produit**
- Présentation par l'équipe du travail réalisé au Product Owner
- Les « poulets » participent en général à cette réunion
- « Poulets » et « Cochons » sont libres de poser des questions et répondre
- Orientée discussion entre le PO et l'équipe
- Vérification pour savoir si les items sont bien « faits »
- Passe par une démonstration, mais ne doit pas se tenir à cela
- Réadaptation du Product/Release Backlog en fonction des tâches items non réalisés
- Ne doit pas demander plus de 30 minutes de préparation

Scrum
Rétrospective de Sprint

- **Inspection et adaptation concernant le processus**
- Participation de l'équipe et du Scrum Master
  - Scrum Master doit limiter ses intervention pour garder au maximum une position neutre
  - Product Owner pas indispensable, mais le bienvenu
- Principe possible
  - Un tableau avec deux colonne : marche bien/marche mal
  - Chaque intervenant dispose un ou plusieurs items dans chaque colonne
  - Les items répétés sont indiqués par des barres supplémentaires pour chaque occurrence supplémentaire
  - L'équipe discute de modifications légères à essayer dans le prochain sprint pour corriger les défauts

eXtreme Programming

Extreme Programming

- Introduction
  - Une des méthodes mettant en oeuvre les concepts Agiles
  - ...
- Principes généraux
  - Des itérations courtes
  - Pas de spécification et conception globales initiales
  - Les jeux de tests sont préalables au développement et perdurent au fur et à mesure des versions pour garantir la non-régression

Pilotage du projet

- Phase initiale d'exploration
  - Phase très courte (1 mois)
  - 3 objectifs :
    - Définition du contenu fonctionnel de l'application
      - => Liste de "user stories"
    - Établir un premier plan de développement
    - Fournir une première version du logiciel

- Phase de planification
  - 1 user story => 1 estimation de coût en points abstraits
  - Estimation de la vélocité : points productibles par itération ; correction après la première itération
  - Le client affecte les "user stories" aux itérations

User story

- Besoin du client
  - exprimé en quelques phrases
  - informel
- Doit tenir sur une fiche 3" x 5" (ou 8cm x 13cm)
  - Sinon, séparer en plusieurs "user stories"
- Utilisation
  - Estimation d'une charge de travail
  - Définition de tests fonctionnels pour la validation
  - Planification du travail

Déroulement d'une itération

- Réunion préalable avec le client
  - Bilan de l'itération passée
  - Présentation des objectifs (user stories) de la nouvelle itération, et détermination des tâches à effectuer
  - Les développeurs choisissent eux-mêmes leurs tâches, au fur et à mesure de l'avancée de l'itération
- Réunion quotidienne (stand-up) ≤ 15min
  - Point sur l'avancement
  - Présentation des objectifs du jours
  - Présentation des difficultés éventuelles
- Intégration en continu
- Conserver une rythme durable
  - Pas d'heures supplémentaires plus de 2 semaines consécutives

Principes généraux de codage

- Principe de base : responsabilité collective du code
- Uniformité du code
  - Norme de codage
  - Référence à une métaphore partagée
- Travail en binôme
  - Le pilote code, le copilote participe
  - Les binômes changent tous les jours
- Importance des tests unitaires
  - Précèdent le codage
  - Sont sans cesse ré-exécutés
- Simplicité du code
  - Ne pas anticiper les généralisations
  - Ne pas optimiser si ce n'est pas nécessaire
- Refactoring en continu

Refactoring
http://www.refactoring.com

- But

Maintenir un code propre et facile à faire évoluer, c'est-à-dire un code tel qu'une évolution donnée n'entraîne qu'une seule modification

- Moyens
  - Supprimer le code mort
  - Supprimer les redondances
  - ...
- Corollaire
  - Adapter éventuellement les jeux de test
- Guide ?
  - AntiPatterns de développement

AntiPattern de développement

- Définition

  Cas typique de mauvais développement auquel est associée une proposition de refactoring pour le corriger.

AP <<The blob>>
la tache (ref. cinématographique)

- Problème
  - Un objet assure l'essentiel des responsabilités, tandis que les autres ne font que contenir des données ou assurer des processus simples
- Echelle
  - Application
- Causes
  - Paresse
  - Précipitation
- Solution
  - Revoir la conception pour mieux distribuer les responsabilités et isoler les effets d'un changement
- Exemple de mise en évidence
  - "Cette classe est le coeur de notre application"

AP <<Continuous obsolescence>> Obsolescence perpétuelle

- Problème

  Une application repose sur plusieurs outils, dont les fréquentes mises à jour rendent difficile l'aboutissement à un développement à jour et fonctionnant avec des versions compatibles de ces outils

- Echelle

  Application

- Causes

  Utilisation d'outils ne faisant pas référence à des standards ouverts

- Solution

  Utiliser des systèmes standards stables. Les nouveautés des autres systèmes ne tarderont pas à y être intégrées

AP <<Lava flow>>
Flot de lave

- Problème

  Une application se trouve progressivement envahie de "code mort".

- Echelle

  Application

- Causes

  code R&D ou prototype passé en production

- Solution

  Pour prévenir : ne pas développer avant d'architecturer

  Pour guérir : procéder à des activités d'exploration du système

AP <<Ambiguous viewpoint>>
Point de vue ambigü

- Problème

  Les analyse et conception orientées objets peuvent concerner différents points de vue, les rendant ni claires ni utiles.

- Solution

  Trois points de vue : métier, spécification (interfaces) et implantation (détail des objets).

  En général, vue "implantation", mais pas forcément la plus utile.

AP <<Functional decomposition>> Décomposition fonctionnelle

- Problème
  - L'application a une structure complètement fonctionnelle, sans vraie structure objet.
- Echelle
  - Application
- Causes
  - Mauvaise maîtrise des concepts objets
  - Reprise d'un projet initial dans un langage non-objet
- Solution
  - Revoir la conception
- Exemple de mise en évidence
  - Méthodes avec des noms "fonctionnels"
  - Pas d'utilisation de la redéfinition

AP <<PolterGeist>>
Esprit frappeur

- Problème

  Une classe "poltergeist" est une classe dont les instances ne sont que transitoires, agissant de manière limitée et par des sortes d'effets de bord sur les données de l'application

- Echelle

  Application

- Causes

  Paresse, Ignorance

- Solution

  Supprimer les classes en question

  Introduire les fonctionnalités qu'elles implantaient dans les classes sur lesquelles elles les appliquaient.

AP <<Boat Anchor>>
Ancre de bâteau

- Problème

  Une ancre de bâteau est une partie d'une application qui ne sert pas à grand chose, et qui correspond souvent à une acquisition payante.

- Echelle

  Application

AP <<Golden Hammer>>
Marteau doré

- Problème

  Un "marteau doré" est un outil utilisé à tort et à travers, et choisi en général par la maîtrise que l'équipe en a, alors qu'il n'est pas forcément le plus adapté.

- Echelle

  Application

- Causes

  Fierté, étroitesse d'esprit

  Formation à rentabiliser

  Une même solution a déjà fonctionné plusieurs fois

- Solution

  Veille techno

  Changement complet du processus

AP <<Dead end>>
Impasse

- Problème

  On intègre dans le développement d'une application une modification d'un composant réutilisable. On aboutit à une impasse si le composant en question n'est plus maintenu ou s'il est modifié sans compatibilité ascendante

- Echelle

  Application

- Solution

  Limiter la modification de COTS (Components off the shelf)

  Passer par des composants intermédiaires pour relier une application aux composants utilisés