

# Le développement agile Java : vive Scrum !

Le développement agile vise à offrir une alternative aux cycles de développement traditionnels dits « en cascade » ou « en V ». Les principes en sont assez simples, mais leur mise en place représente souvent de véritables défis pour les équipes.

L'approche agile, au sens du manifeste agile [1], ne sous-entend pas d'accointances technologiques. Pour autant, les équipes de développement logiciel sont la plupart du temps liées à une technologie, et Java représente une part importante des compétences recherchées aujourd'hui [2]. Les « équipes Java » qui souhaitent tirer profit d'une approche agile sont rapidement amenées à se demander comment mettre en pratique les tests automatiques dans cet environnement.

## Pourquoi un intérêt tout particulier pour les tests automatiques ?

Lorsqu'une équipe découvre l'agilité, elle commence la plupart du temps par mettre en place un cadre de gestion – type Scrum – et elle commence à vivre sur un rythme itératif avec des cycles le plus souvent de 2 ou 3 semaines visant à maximiser l'apprentissage et à favoriser l'amélioration continue. La plupart des équipes héritières d'un processus de développement en cascade vont commencer leur aventure agile en reproduisant la cascade au sein de chaque itération. Cascade dont le goulet d'étranglement est souvent le test manuel en fin d'itération. Cette façon de réaliser

du logiciel atteint très vite une limite. En effet, après quelques itérations, l'équipe se retrouve souvent dans ce qui semble une impasse : comment continuer à ajouter des fonctionnalités alors que les tests de non-régression manuels occupent tout le temps disponible de l'itération ?

Une voix assez commune pour se sortir de cette ornière est alors d'appréhender à automatiser ces tests.

## Mais peut-on automatiser tout type de test ?

Cette question est rarement liée à la possibilité technique d'automatisation. En effet dans le monde numérique tout est techniquement possible. Cette question du « pouvoir » cache une question de l'ordre du « devoir » : devrait-on automatiser tous les tests ? Un premier axe étudié communément repose sur le coût de l'automatisation. Un raisonnement court terme amène les équipes à repousser l'écriture de tests qu'elles ne savent pas écrire. En effet, le temps passé à apprendre à écrire un test est du temps qui leur semble perdu. D'autre part, il arrive qu'une technologie donnée ne dispose pas sur étagère de l'outillage de tests automatisés. Il faut alors envisager de créer cet outillage, ce que les équipes débutantes rechignent à

faire. Ce sont souvent les mêmes qui se retrouvent bloquées après plusieurs itérations. Dans le cas de cette équipe hypothétiquement bloquée, elle serait sans doute dans une situation beaucoup plus confortable si elle disposait d'un harnais de tests qui la rassureraient sur le fait qu'elle n'ait pas détruit les incréments logiciels précédents. Pour cela, une condition nécessaire, c'est-à-dire minimum, est de disposer de tests automatiques qui décrivent le logiciel souhaité par le client : ce sont les tests qui décrivent que l'équipe construit le bon logiciel.

## Et dans l'environnement Java, cet outillage existe-t-il ?

La bonne nouvelle avec Java c'est que l'outillage de test existe et est très mature. Que vous soyez sur des architectures web ou non, toutes les couches logicielles peuvent aujourd'hui être testées/décrites/spécifiées les unes indépendamment des autres. Par mature, nous entendons ici que l'utilisateur de cette technologie a le choix. Un large choix consiste finalement à reconnaître que chacun a une façon de penser personnelle et que la grande diversité d'outils de test lui permettra de choisir ceux qui lui semblent les plus naturels. De façon très anecdotique nous pouvons citer l'exemple de Mockito. L'auteur de Mockito nous explique les raisons qui l'ont poussé à écrire une nouvelle librairie pour écrire des mocks [3]. Si vous partagez son avis sur EasyMock, Mockito vous semblera sans doute plus naturel.

Aujourd'hui, il est possible de couvrir par des tests automatiques toutes les couches logicielles d'un système [4] sans avoir à investir dans la création d'une librairie de tests spécifique ; la librairie qu'il vous faut existe probable-

```
@Test public void
containsLogo() {
    assertThat( homePageContent, hasImageHtmlTag( "logo.png" ) );
}
```

Fig.1

GreenPepperized  - Configure

Execute > for [ refactoring ]

Rights: 2 Wrongs: 0 Errors: 0

Fig.2

Rule for	Delivery service		actually sent ?
recipient	available	send	
joe@example.com	yes	try send	yes
jack@example.com	no	try send	no



ment déjà. Si ces notions sont nouvelles pour vous, une façon simple de faire le grand saut consiste à rechercher sur le web par exemple « TDD Spring » et vous aurez accès à plusieurs articles qui vous aideront à vous lancer dans l'écriture de tests automatiques dans le cadre de l'utilisation de Spring.

## Show me the code ! ;)

Effectivement un test automatique est un programme que l'on écrit comme tous les autres programmes avec un langage ; en l'occurrence ici un code en Java [Fig.1].

Lorsque l'on choisit d'écrire un test, on a pratiquement toujours plusieurs choix d'outils ou de conteneur pour ce test. Par exemple [Fig.1], l'équipe a choisi d'écrire ce test avec Junit [5] dont l'intention est de spécifier qu'un logo doit être présent dans la page d'accueil. Le choix de l'outil dépend la plupart du temps de la personne avec qui il faut avoir une discussion pour pouvoir l'écrire. Le code ci-dessus pourrait ne pas mettre à l'aise une personne assez éloignée de la technique. D'autres outils à vocation plus collaborative ont donc vu le jour. Certains comme GreenPepper [6] se basent sur un wiki et donnent une vision moins technique des tests [Fig.2]. Du choix de l'outil peut dépendre notre capacité à collaborer avec les personnes nécessaires à l'écriture du test. Ce besoin de collaboration devient crucial lorsque les tests sont écrits en premier, c'est-à-dire avant que le programme lui-même n'existe.

## Ecrire les tests avant le programme à tester ?

Oui, c'est là que les tests automatiques décuplent leur valeur.

En effet, considérant comme un minimum les tests qui décrivent que nous avons construit le bon logiciel, cela semble une bonne idée de les connaître avant de commencer. L'avantage de cette approche est de rendre non équivoque la spécification, car les exemples contenus dans le test pourront explorer non seulement les cas nominaux d'utilisation mais également les scénarios exceptionnels.

## Comment faire dans le cas de la reprise d'un code existant sans test ?

Le développement piloté par les tests (TDD) a un impact important sur l'architecture d'une application. Il favorise l'émergence de zones de responsabilité distinctes et découplées, propres à favoriser l'évolutivité d'un système. Les codes, souvent anciens, développés sans tests sont également souvent les moins aptes à évoluer et à livrer de nouvelles fonctionnalités. Lorsque la meilleure idée de l'équipe est de conserver ces programmes, leur reprise de contrôle passe par l'écriture de tests systèmes englobant totalement la zone sur laquelle on souhaite intervenir avant d'entreprendre une ré-écriture de la zone.

## Cela ne semble pas trivial, existe-t-il des formations spécifiques ?

Ne nous trompons pas d'objectif : ce qui est non trivial c'est le développement logiciel. Les pratiques d'ingénierie agiles ne sont pas faciles à maîtriser, mais avec de l'aide et de la discipline, les gains sont vite visibles. Le programme Professional Scrum Developer (PSD) de scrum.org est un stage de cinq jours pour les développeurs d'une équipe Scrum. Le stage apprendra aux équipes à répondre à des demandes de fonctionnalité en construisant des incréments logiciels prêts pour la production en utilisant les pratiques d'ingénierie modernes. Pyxis est partenaire de ce programme et propose des formations dans les technologies Java ou .Net [7].

### Objectifs

Scrum sera mis en place et étudié via des discussions, démonstration et exercices pratiques. Les participants apprendront à utiliser Scrum correctement en étant aidés par l'instructeur. En particulier, les sujets suivants seront étudiés :

- La construction d'équipe performante
- Le travail sur un code pré-existant
- Définition de critères de qualité, de critère d'acceptation
- Définition de « terminé »
- Build automatisé

- Correction d'anomalies
- Vérification de l'élimination des anomalies via des tests automatiques
- Planification de releases et d'itérations
- Estimations
- Création et gestion d'un sprint backlog
- Tenir efficacement une revue de sprint
- Améliorer son processus de développement grâce aux rétrospectives
- Éviter la dette technique grâce à l'architecture émergente
- Le développement piloté par les tests comme outil de conception
- Intégration continue
- Tests de toutes les couches logicielles
- Agir sur les dysfonctions d'équipe

### Audience

Ce cours s'adresse à tout membre d'une équipe de développement – architecte, programmeur, testeur, etc. Nous encourageons les équipes à venir vivre ce stage ensemble. Les personnes venant seules intégreront une équipe. Les participants devront s'auto-organiser pour former des équipes multidisciplinaires. Ces équipes ont en effet besoin d'un ensemble spécifique de compétences pour adresser le cas d'étude proposé aux participants. Les Product Owner, ScrumMaster et autres contreparties sont également les bienvenus. Néanmoins, gardez en tête que tous les participants seront censés se comporter comme des membres d'équipes à part entière et participer aux engagements collectifs de leur équipe.

### ■ Eric Mignot

Pyxis Technologies

- [1] <http://www.agilemanifesto.org>
- [2] <http://www.indeed.com/jobtrends?q=java&l=>
- [3] <http://monkeyisland.pl/2008/01/14/mockito/>
- [4] <http://www.natpryce.com/articles/000772.html>
- [5] <http://www.junit.org>
- [6] <http://www.greenpeppersoftware.com>
- [7] <http://www.pyxis-tech.com/en/services/formation/inscription/>

