

DEVELOPPEMENT PILOTE PAR LES TESTS D'ACCEPTATION AVEC ROBOTFRAMEWORK

par Craig Larman et Bas Vodde

Version 1.1

Le Développement Piloté par les Tests d'Acceptation est une pratique essentielle mise en œuvre par des équipes Agiles et Scrum avec succès. Le changement réside dans la façon de tester en utilisant des exemples/tests pour clarifier et documenter les exigences. Ce court papier est un échantillon du chapitre Test du livre Practices for Scaling Lean & Agile Development: Large, Multisite, and Offshore Product Development with Large-Scale Scrum.

INTRODUCTION AU DEVELOPPEMENT PILOTE PAR LES TESTS D'ACCEPTATION

Le Développement Piloté par les Tests d'Acceptation (A-TDD)¹ est une pratique de découverte collaborative des exigences où les exemples et les tests automatisés sont utilisés comme spécifications, créant des spécifications exécutables. Elles sont créées par l'équipe et le Product Owner, et toutes autres parties prenantes lors d'ateliers de définition d'exigences.

Les tests comme exigences, les exigences comme tests, d'après Melnik and Martin², « Avec l'augmentation des formalités, les tests et les exigences deviennent non dissociables. A la limite, les tests et les exigences sont équivalents ». Les tests doivent être précis afin d'être automatisés. L'A-TDD exploite cette formalité et formule les exigences en les écrivant sous la forme de tests automatisables.

¹ Le Développement Piloté par les Tests d'Acceptation est aussi connu sous le nom de Test d'Acceptation Agiles ou Développement Piloté par les Histoires de Test

Ndt : dans cette traduction, Acceptance Test Driven Development (A-TDD) a volontairement été traduit par Développement Piloté par les Tests d'Acceptation, pour de plus ample recherche sur le net, garder plutôt la terminologie anglaise. L'[Institut Agile](#) le référence sous le nom de [Développement par les tests clients](#).

² Ndt : Grigori Melnik et de Robert C. Martin, le célèbre Uncle Bob,

http://www.computer.org/portal/cms_docs_software/software/homepage/2008/s1mel.pdf

www.robotframework.org

Copyright (c) Craig Larman & Bas Vodde 2010 – All rights reserved

Traduit par Laurent Carbonnaux en septembre 2011

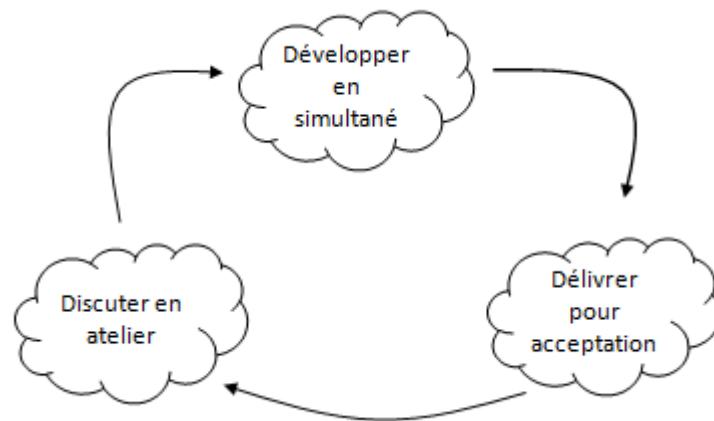
Des ateliers pour clarifier les exigences. La clarification des exigences lors d'ateliers face à face est une pratique utilisée depuis l'invention de La Conception Conjointe d'Application(JAD)³. L'A-TDD exploite, de la même façon, la conversation face à face lors d'ateliers afin d'écrire des tests automatisables.

Ingénierie simultanée. Une itération est le plus couramment de deux semaines. C'est rapide et l'équipe a donc besoin de définir une façon de travailler en simultané. Le développement séquentiel en itérations courtes ne fonctionne pas. Nous avons vu des équipes réinventant encore et encore l'A-TDD simplement parce qu'elle n'avait pas répondu à la question « *Comment pouvons nous faire notre travail en même temps* »

Prévention plutôt que détection. Lorsque vous incluez des personnes spécialisées en tests dans les ateliers de définition d'exigences, elles peuvent poser les questions relatives aux tests ou utiliser des stratégies normales de test telles que les tests aux limites. En ce sens, elles aident à améliorer les exigences et prévenir les défauts.

Comment fonctionne l'A-TDD ? La figure 1.1 en présente un aperçu.

Figure 1.1 Aperçu A-TDD



L'A-TDD consiste en trois étapes :

1. Discuter les exigences lors d'un atelier.
2. Les Développer en simultané pendant une itération
3. Livrer le résultat aux parties prenantes pour acceptation

Discuter. Les exigences sont découvertes au travers de discussions lors d'ateliers de définition des exigences⁴. Les participants à cet atelier sont l'équipe pluridisciplinaire, le Product Owner ou ses représentants et toutes autres parties prenantes pouvant fournir de l'information sur les exigences. Une question primordiale à se poser durant ces ateliers

³ Ndt : Joint Application Design (JAD)

⁴ Gojko Adzic les appelle Ateliers de spécification

est « *Imaginez le système au final. Comment voudriez-vous l'utiliser et qu'attendez-vous de lui ?* » Cette question entraîne des exemples d'utilisation, et ces exemples peuvent être écrits sous forme de tests : les exigences. L'atelier doit être employé plus comme un moyen de discussion, de découverte des exigences, que comme moyen formel de définition des tests.

Développer. A la fin de l'atelier, les exemples sont *distillés*⁵ en tests. Toutes les activités nécessaires à l'implémentation des exigences sont faites en simultané. Cela inclus :

- ☐ Faire le code qui liera les tests et le système à tester (« Bibliothèques de tests » and « Tables de plus bas niveau » dans Robotframework ou « Fixtures » en Fit)
- ☐ Implémenter l'exigence pour que le test passe
- ☐ Mettre à jour l'architecture et autres documents internes en fonction des décisions de l'équipe
- ☐ Ecrire les documents à destination du client pour cette exigence
- ☐ Des tests exploratoires supplémentaires

La liste exacte dépend du produit, du contexte, des accords de travail, et de la Définition de Fini⁶.

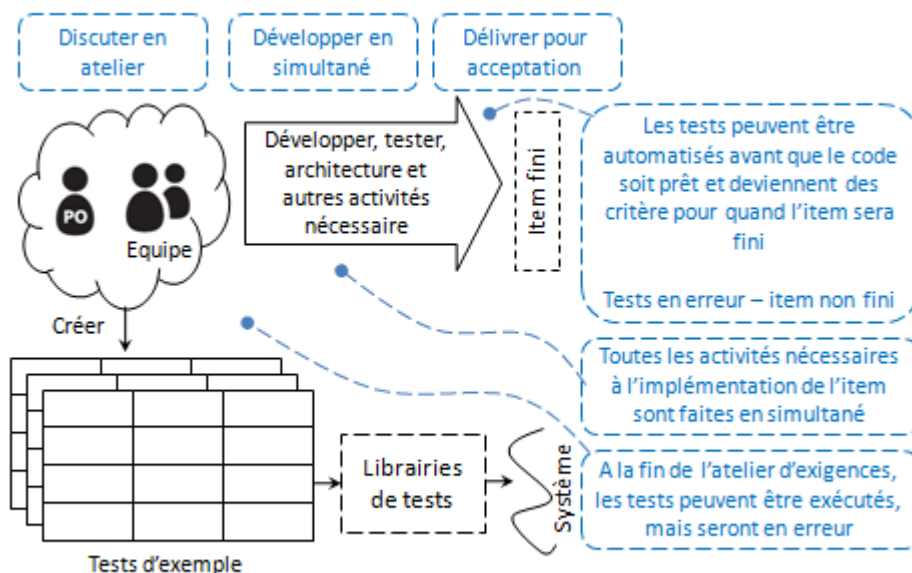
Livrer. Quand les tests passent, l'exigence est revue avec le Product Owner et les parties prenantes. Cela peut entraîner de nouvelles exigences ou un changement dans les tests existants.

⁵ Elisabeth Hendrickson considère la distillation comme une étape de l'A-TDD

⁶ Le « Définition de Fini » est un terme Scrum définissant le contrat entre le Product Owner et l'Equipe sur ce qui doit être inclus dans une itération. Une courte introduction à Scrum (Scrum Primer) peut être trouvée sur <http://www.scrumprimer.org>

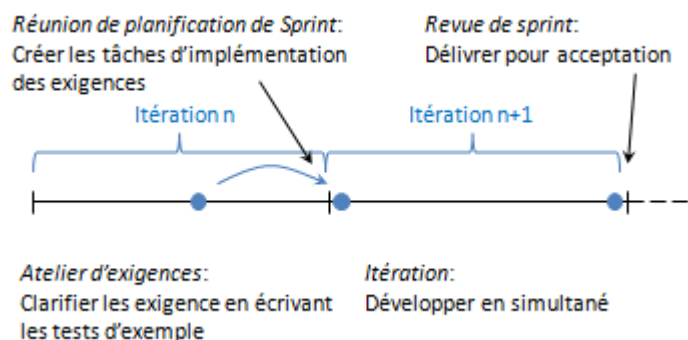
Une vue plus détaillée décrivant l'A-TDD est fournie en Figure 1.2.

Figure 1.2 A-TDD plus en détail



Les étapes de l'A-TDD correspondent bien avec le cycle d'itérations de Scrum (Figure 1.3).

Figure 1.3 Etapes A-TDD mappées sur les itérations Scrum



Discuter en atelier. Avant la réunion de planification détaillée de sprint⁷, l'équipe, le Product Owner et les parties prenantes clarifient les exigences de manière collaborative lors d'un atelier.

Développer en simultané. Les tâches d'implémentation des tests/exigences sont créées pendant la réunion de planification détaillée de sprint et l'implémentation s'effectue pendant le sprint. Toutes les activités sont faites « à peu près en même temps ».

⁷ Lors de la réunion de planification de sprint partie 1 ou lors de la réunion de raffinement du backlog.

Délivrer pour acceptation. L'incrément de produit fini, c'est à dire les tests d'acceptation exécutés sans erreur, sont livrés pour acceptation aux parties prenantes et discutés ensemble lors de la revue de sprint.

EXEMPLE : ROBOT FRAMEWORK

Ce chapitre présente un exemple simple de l'utilisation de Robot Framework sur un système que nous utilisons depuis de nombreuses années. Le développement original n'utilisait pas l'A-TDD ou Robot Framework. L'exemple est neuf, le système est vieux. Le système est de taille moyenne⁸, l'exemple démontre toutefois les points clés utilisables aussi sur des développements plus larges.

Robot Framework est un socle de tests automatisés pilotés par mots-clés⁹. Il a été créé par Pekka Klärck¹⁰ chez Nokia Siemens Network en 2005. Un des premiers objectifs de ce socle était le support de l'A-TDD. Il est passé en Open Source en 2008 et est disponible sur le site www.robotframework.org.

Le produit utilisé dans cette étude de cas est un système d'information de conférence produit pour un centre de congrès. L'accès au système est disponible au travers de « piliers d'informations » distribués par le centre de congrès. Les visiteurs peuvent y accéder pour trouver des informations sur la conférence en cours ou le centre de congrès lui même, telles que :

- ☐ Où puis-je trouver le stand du vendeur X ?
- ☐ Comment puis-je m'y rendre ?
- ☐ Où se trouve le restaurant le plus proche ?
- ☐ Que penses les autres visiteurs de la conférence ?

Ces piliers sont maintenus et contrôlés de manière centralisée. L'information contenue dans le système *doit obligatoirement* provenir du processus existant de préparation de la conférence. La mise à jour du système ne doit entraîner aucun travail supplémentaire de préparation pour une nouvelle conférence. Le nouveau système doit s'adapter au système existant et aux façons de travailler existantes. Et bien sûr, le système doit être « haut en couleur » : sons, graphiques, et autres cloches et sifflets.

Exemple Un : La liste des Vendeurs

Le premier exemple est simple mais démontre quelques points clés. Le client a demandé la possibilité de lister tous les vendeurs et de les afficher de façon « sympa ». Après des

⁸ Des produits plus grands proviennent de domaines plus complexes ou non familiers. Ce qui les rend moins facile d'utilisation pour un exemple tenant en quelques pages.

⁹ Les frameworks de tests pilotés par mots-clés utilisent des mots –clés dans les données pour déterminer les actions à prendre sur ces données. Les mots-clés sont appelés parfois des mots d'actions.

¹⁰ De son vrai nom Pekka Laukkanen.

discussions abstraites sur la notion de sympa, nous lui avons demandé un exemple. Le premier exemple était un descriptif du processus, un exemple (Figure 1.4) avec trois vendeurs dans la base, listés par ordre alphabétique et avec trois colonnes.

Figure 1.4 Exemple de processus de sélection des vendeurs

1. Request Show all Vendors
When 3 vendors, Apple, IBM, MS

Logo	Name	Stand#
Apple	Apple	A1-01
IBM	IBM	B2-03
MS	MS	A1-07

Après avoir demandé plus d'exemples (non présentés ici), nous avons découvert que les données présentes en base n'étaient pas cohérentes. Le système devait compenser cette erreur parce que nous n'avions pas la main sur la base de données. Par exemple, il y avait des doublons avec des différences mineures (avec ou sans logo).

Décrire toutes les compensations de données avec des exemples de processus a engendré une multitude de tests similaires, nous sommes donc passés au pilotage des tests par le données pour ces cas métiers particuliers en nous posant la question : « Quelle donnée en base nous amène à une liste des vendeurs sympa ? ». Les résultats sont présentés dans la figure 1.5 et incluent des exemples de tri alphabétique, de suppression de doublons et de sélection avec ou sans logo.

Figure 1.5 Tests pilotés par les données pour la liste des vendeurs

Logo	Name	Stand	Logo	Name	Stand
Apple	Apple	A1-01	Apple	Apple	A1-01
IBM	IBM	B2-03	IBM	IBM	B2-03
MS	MS	A1-07	MS	MS	A1-07
IBM	IBM	B2-07	Apple	Apple	A1-01
Apple	Apple	A1-01	IBM	IBM	B2-03
MS	MS	A1-07	MS	MS	A1-07
Apple	Apple	A1-07	Apple	Apple	A1-01
Apple	Apple	A1-01	Apple	Apple	A1-07
IBM	IBM	B2-03	IBM	IBM	B2-07
No vendors					
Apple	Apple	A1-01	Apple	Apple	A1-01
Apple	Apple	A1-01	IBM	IBM	B2-03
IBM	IBM	B2-03			

Entracte : Aperçu Robot Framework

L'étape suivante consiste à distiller ces exemples dans des tests Robot Framework. Mais comment Robot Framework fonctionne-t-il ?

Les tests Robot Framework sont écrits dans des tables au format HTML, TSV, reST¹¹ ou plein texte. Les formats HTML et plein texte sont les plus couramment utilisés. En HTML, Robot Framework utilise uniquement des tables et ignore tout texte additionnel, qui dans ce cas peut être utilisé comme documentation. Il y a quatre types de tables :

- ❑ *Tables des cas de test*, contenant les cas réels de tests. Le premier champ de la table doit contenir « Test Case » ou « Test Cases. »
- ❑ *Tables des mots-clés*, contient les mots-clés utilisateur de bas niveau utilisés pour construire les cas de test. Le premier champ doit contenir « Keyword » ou « User Keywords. »
- ❑ *Table de paramétrage*, pour importer des fichiers ou définir des métadonnées. Le premier champ doit contenir « Setting » ou « Settings. »
- ❑ *Table de variables*, pour la déclaration des variables globales pouvant être utilisées ailleurs. Le premier champ doit contenir « Variable » ou « Variables. »

Test Cases	Action	Arg
Drink coffee	Drink coffee in liters	10
	Is physical health	OK
Drink coffee over capabity	Drink over the max amount of coffee	
	Is physical health	NOK

La table à gauche montre une table de cas de test avec deux cas de test, dans la première colonne. La seconde colonne contient les mots-clés, et les colonnes suivantes sont utilisées pour passer les paramètres aux mots-clés.

Robot Framework utilise deux type de mots-clés : *mots-clés utilisateur* and *mots-clés de librairie*. Un mot-clé utilisateur est implémenté dans une table de mots-clés, alors qu'un mot-clé de librairie est implémenté par du code, appelé *librairie de test*, faisant l'interface entre le test et le système à tester. L'exécution de la table est en erreur parce qu'il contient trois mots-clés indéfinis : « Drink coffee in liters » (Boire du café au litre), « Is physical health » (Est-ce bon pour la santé), « Drink over the max amount of coffee" (Boire le plus de café possible)¹²

Keywords	Action	Arg
Drink over the max amount of coffee	Drink coffee in liters	\${MAX COFFEE}
	Drink coffee in liters	1

La table à gauche implémente le mot-clé utilisateur « Drink over the max amount of coffee » en buvant un litre de plus que le maximum. \${MAX COFFE} est une variable définie dans une table des variables (non présentée)

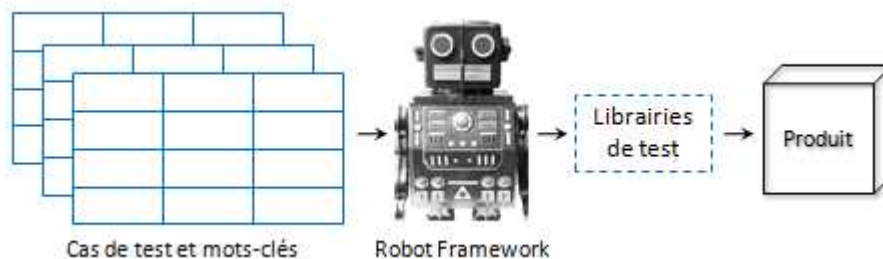
L'exécution de la table est en erreur parce que deux mots-clés sont indéfinis. Ils sont de bas niveau et sont implémentés dans une librairie de mots-clés écrite en Java (possible aussi en Python).

¹¹ TSV est le format à séparation par tabulation. ReStructuredText (reST) est un langage à balise utilisé communément pour la documentation des projets en langage Python.

¹² Ndt : les mots-clés ne sont pas traduits ici, pour garder le lien avec le code plus loin.


```
public class CoffeeTestLibrary
{
    Human humanUnderTest = new Human("Bas");
    public void drinkCoffeeInLiters(Integer liters) {
        humanUnderTest.drinkCoffee(liters);
    }
    public void
    isPhysicalHealth(String expectedHealth) throws Exception
    {
        if (!expectedHealth.equals(humanUnderTest.checkHealth()))
            throw new Exception("Health problem. Expected health: " +
                expectedHealth + " but actual health was " +
                humanUnderTest.checkHealth());
    }
}
```

La figure dessous montre un aperçu de Robot Framework. Les cas de test et les tables de mots-clés sont fournis à RobotFramework. Le socle appelle les librairies, celles-ci appelant le système à tester. Reportez-vous au manuel utilisateur de Robot Framework pour plus de détail.¹³



Contribution : Liste des vendeurs

Nous *distillons* le test issu de l'exemple du tableau blanc. L'exemple 'normal' est supprimé parce qu'il est de *même classe d'équivalence* que celui du test de 'tri'.¹⁴

Ces cas de test sont exécutables, mais sont en erreurs. Robot Framework informe que les mots-clés « Stand input », « Is stand output », et « Has No Extra Stands » sont indéfinis.

¹³ Le manuel utilisateur est disponible sur www.robotframework.org

¹⁴ Les mots-clés de Robot Framework doivent être interprétés des tests orientés données. Dans Fit, ce ne serait pas nécessaire. Cela sera amélioré dans les versions futures.

Figure 1.6 Tables du cas de test de la liste de tous les vendeurs

Test Case	Action	Logo	Name	Place
Sorted				
	Stand input	x	IBM	B2-03
	Stand input	x	Apple	A1-01
	Stand input	x	MS	A1-07
	Is stand output	x	Apple	A1-01
	Is stand output	x	IBM	B2-03
	Is stand output	x	MS	A1-07
	Has no extra stands			
Same vendor, different place				
	Stand input	x	Apple	A1-02
	Stand input	x	Apple	A1-01
	Stand input	x	MS	A1-07
	Is stand output	x	Apple	A1-01
	Is stand output	x	Apple	A1-02
	Is stand output	x	MS	A1-07
	Has no extra stands			
Duplicate entry, logo plus non-logo version				
	Stand input		Apple	A1-01
	Stand input	x	Apple	A1-01
	Stand input	x	MS	A1-07
	Is stand output	x	Apple	A1-01
	Is stand output	x	MS	A1-07
	Has no extra stands			

Les mots-clés peuvent être implémentés sous forme de mots-clés utilisateur ou mots-clés de librairie. Nous choisissons d'implémenter « Is stand output » et « Has No Extra Stands » en mots-clés utilisateur (Figure 1.7).

Figure 1.7 Tables des mots-clés utilisateur de la liste de tous les vendeurs

Keyword	Action	Arg	Arg	Arg
Is stand output	[Arguments]	\${expected_logo}	\${expected_name}	\${expected_place}
	\${actual_logo}=	Get current logo		
	Should be equal	\${expected_logo}	\${actual_logo}	
	\${actual_name}=	Get current name		
	Should be equal	\${expected_name}	\${actual_name}	
	\${actual_place}=	Get current place		
	Should be equal	\${expected_place}	\${actual_place}	
	Increment stand index			
Has no extra stands				
	\${stands_left}=	Stands left		
	Should not be true	\${stands_left}		

La première ligne déclare les trois arguments du mot-clé « Is stand output ». Les lignes suivantes assignent le résultat du mot-clé « get current logo » à la variable `${actual_logo}`, le compare avec la variable `expected_logo`. Les mêmes étapes sont répétées pour les autres valeurs attendues. A la fin, l'index du stand est incrémenté.

Lorsque le test est exécuté, Robot Framework informe que cinq mots-clés sont indéfinis : stand input, get current logo, get current name, get current place, et stands left. Ces mots-clés seront implémentés dans une librairie de test.

Notre système à tester est écrit en C. Nous pouvons l'appeler au travers d'une interface utilisateur (non recommandé), ou nous pouvons appeler le code C directement. Nous choisissons ce dernier cas et implémentons la librairie de test en langage Python qui permet facilement l'appel de code C en passant par la librairie externe `ctypes`¹⁵. Le code de la librairie de test :

```
from ctypes import *

class conferencekeywords:

    def __init__(self):
        self.conf = cdll.LoadLibrary("stands.so")
        self.conf.init()
        self.stand_index = 0
        self.logo_mark = ["", "x"]
        self.conf.get_place_at_index.restype = c_char_p
        self.conf.get_name_at_index.restype = c_char_p

    def increment_stand_index(self):
        self.stand_index = self.stand_index + 1
```

¹⁵ Fonctionne extrêmement bien pour le code C. Pour le C++, préférez l'utilisation de SWIG ou Boost Python.

```
def stands_left(self):
    return self.conf.stand_output_at(self.stand_index)

def stand_input(self, logo, name, stand):
    has_logo = logo == "x"
    self.conf.add(has_logo, c_char_p(name), c_char_p(stand))

def get_current_logo(self):
    logo = int(self.conf.get_logo_at(self.stand_index))
    return self.logo_mark[logo]

def get_current_name(self):
    return self.conf.get_name_at_index(self.stand_index)

def get_current_place(self):
    return self.conf.get_place_at_index(self.stand_index)
```

Le code est simple, la seule chose qu'il fait est d'appeler l'interface C.

Le système à tester est lié dans une librairie partagée et chargée par l'appel `cdll ctypes`. Le résultat est assigné à la variable `conf` et est utilisé pour appeler les fonctions C dans la librairie partagée. Le code additionnel spécifie les paramètres et le code retour ; doit être fait absolument s'il ne s'agit pas d'entiers, le type par défaut.

Le test est exécuté, avec succès, si les fonctionnalités sont implémentées.

Exemple Deux : Importer des fichiers AutoCAD

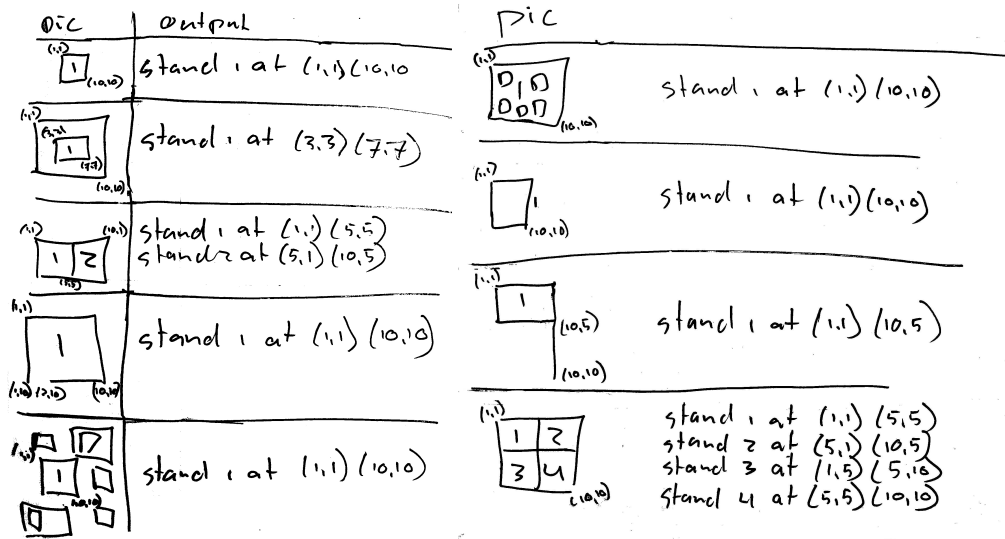
Le département Implantation utilise AutoCAD pour créer la mise en place des conférences. Elle contient la position des stands et tout ce qui est relatif à la conférence, incluant des symboles étranges et les prises électriques. Ces dessins CAD sont *censés être lisibles par des humains*.

Une exigence pour le système d'information de conférence était de montrer le plan de la conférence. Les visiteurs peuvent sélectionner un endroit du plan et le système leur dit quel est le vendeur. Le centre de congrès demande que le système puisse importer des fichiers AutoCAD DWG créés par le département Implantation. La lecture des fichiers AutoCAD étant lente, ils sont convertis dans un format interne ne gardant que les informations nécessaires, ce qui supprime les *bruits* tels que les prises électriques.

Le DWG contient des lignes décrivant les formes. Une forme avec un numéro est probablement un stand. Le système doit lire le fichier, dessiner la forme, reconnaître les stands et supprimer toute information inutile. Ce n'est pas trop compliqué... sauf que les données sont incohérentes, non prévues pour un ordinateur. Les formes ne sont pas toujours fermées, le nombre n'est pas toujours *exactement* sur un stand, et ainsi de suite.

Lors d'un atelier de définition d'exigences, nous avons discuté des différentes incohérences que le système doit supporter. Nous avons démarré avec des *descriptions abstraites* et avons finis par les exemples, comme sur le Figure 1.8.

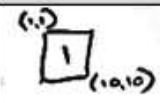
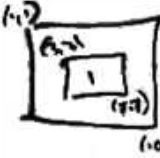



Figure 1.8 Exemples de stands rectangulaires



Ces exemples sont des dessins de formes. La discussion portait sur la distinction entre les stands et le bruit et sur les incohérences dans les dessins.

Les tests distillés de ces exemples sont à la figure 1.9. Les tests contiennent un fichier d'entrée AutoCAD et un fichier de sortie attendu. Nous avons ajouté les images aux tests pour la documentation. Robot Framework ignore toute information hormis les textes dans les tables. Cela permet d'ajouter de la documentation à nos tests de spécifications.

Figure 1.9 Test de reconnaissance de stand

Test Case	Action	Original	Result
Recognize rectangular stands	Detect and Check stands	 simple_rectangle.dwg	simple_rectangle.fpm
	Detect and Check stands	 rect_in_rect.dwg	rect_in_rect.fpm
	Detect and Check stands	 two_rects.dwg	two_rect.fpm
	Detect and Check stands	 non_closed_rect.dwg	non_closed_rect.fpm
	Detect and Check stands	 rect_with_noise.dwg	rect_with_noise.fpm

Le mot-clé « Detect and check stand » présenté en Figure 1.10 est implémenté comme un mot-clé utilisateur.

Figure 1.10 Mot-clé de reconnaissance de stand

Keyword	Action	Arg	Arg
Detect and Check stands	[Arguments]	\${dwg_drawing}	\${fpm_result}
	Convert to FPM	Convert to FPM	Convert to FPM
	Compare FPM files	Compare FPM files	Compare FPM files

La librairie de test appelle l'interface C du système à tester (Non présenté).

Exemple Trois : Laisser des messages

Ce dernier exemple est un exemple de test de processus. Les visiteurs ont besoin de pouvoir laisser des messages aux autres dans le système d'information de conférence. Lors d'une *discussion* en atelier de définition d'exigences, nous sommes passés d'une description abstraite à des *exemples de processus*. Les résultats sont présentés en Figure 1.11.

Figure 1.11 Exemple de processus de messagerie

```

1. select leave message
2. insert name "Bas"
3. insert topic "Great conference"
4. insert message "Party afterwards?"
5. save
6. show saved message

```

```

1. select leave message
2. save
3. error message "you should insert name"

```

```

1. select leave message
2. insert name "Bas"
3. save
4. error message "you have no topic"

```

Les tests *distillés* ont finis à l'identique de ceux de l'exemple au mur. Ils sont présentés en Figure 1.12.

Figure 1.12 Exemple de processus

Test Case	Action	Arg
Leave a message succeeds	Select leave a message	
	Insert name	Craig
	Insert topic	Great conference
	Insert message	Party afterwards?
	Save	
	Is message	\${NORMAL EXPECTED MESSAGE}

Toutes ces actions ont été implémentées en mots-clés utilisateur, afin d'être réutilisables pour de futurs tests de processus. Cela minimise la duplication entre les tests de processus et réduit l'effort de maintenance. Nous passons sur les tests utilisateurs et les bibliothèques de tests, ils sont identiques aux exemples précédents.

Conclusion Robot Framework

Ces exemples ont montré quelques fonctions majeures de Robot Framework. D'autres fonctions moins abordées sont :

- ❑ La possibilité de classer les tests à l'aide tags. Peut être utile pour des rapports, des statistiques ou la sélection de test à l'exécution.
- ❑ Disponibilité de bibliothèques de tests génériques, telles que Swing Library, Selenium Library ou *Eclipse Library*¹⁶.
- ❑ Des traces et rapports clairs qui permettent de facilement comprendre ce qui se passe.
- ❑ Facilité d'intégration avec d'autres systèmes tels que les outils de GCL ou d'IC¹⁷
- ❑ Un EDI pour éditer les tests : RIDE.¹⁸

Conclusion

L'A-TDD est une technique collaborative de clarification des exigences qui utilise des exemples exécutables pour explorer les exigences. C'est une technique d'équipe qui crée un environnement collaboratif et parallélise toutes les activités dans une itération. Cela contribue à flouter les rôles traditionnels et crée un environnement coopératif.

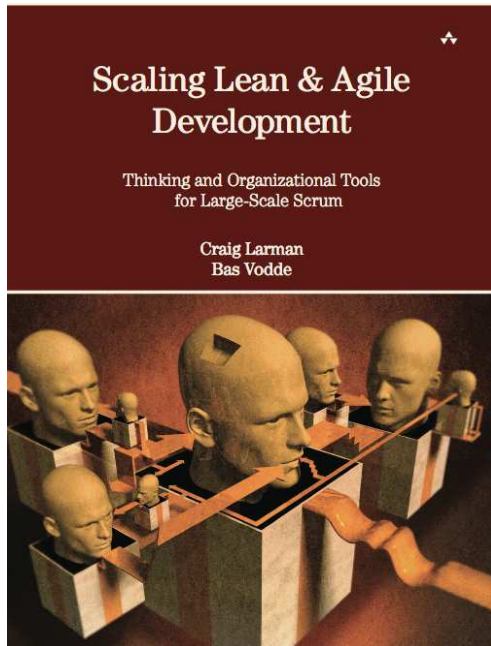
Robot Framework est un socle de test construit pour l'A-TDD. Il utilise un format tabulaire pour décrire les exemples qui deviendront des exécutables par implémentation d'un bout de code d'interface appelé bibliothèque de test. La structure en table multi couche favorise l'écriture en langage naturel des tables de hauts niveaux, rendant les exigences et les tests équivalents.

¹⁶ Ndt : Eclipse Library ajouté à la traduction.

¹⁷ Ndt : GCL Gestion de Configuration Logiciel. IC, Intégration Continue

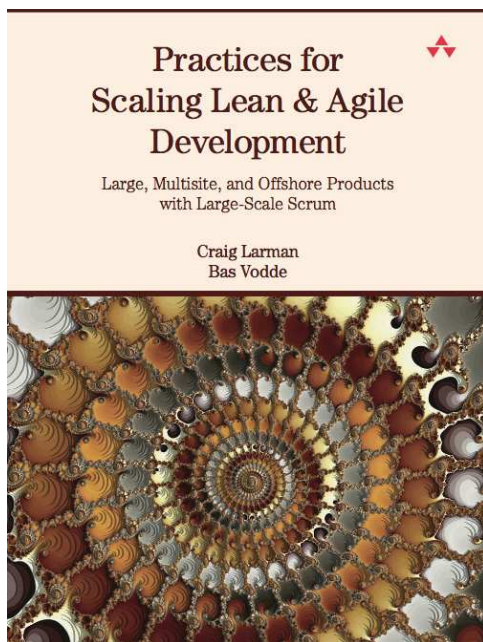
¹⁸ Ndt : EDI, Environnement de développement intégré

Références



Chapitres :

- ☐ Introduction
- ☐ Systems Thinking
- ☐ Lean
- ☐ Queuing Theory
- ☐ False Dichotomies
- ☐ Be Agile
- ☐ Feature Teams
- ☐ Teams
- ☐ Requirement Areas
- ☐ Organization
- ☐ Large-Scale Scrum



Chapitres :

- ☐ Large-Scale Scrum
- ☐ Test
- ☐ Product Management
- ☐ Planning
- ☐ Coordination
- ☐ Requirements
- ☐ Design
- ☐ Legacy Code
- ☐ Continuous Integration
- ☐ Inspect & Adapt
- ☐ Multisite
- ☐ Offshore
- ☐ Contracts