



Test Designer™

UNIVERSITÉ DE TECHNOLOGIE DE BELFORT-MONTBÉLIARD

# Développements en JAVA au sein d'une équipe utilisant l'eXtreme Programming

Rapport de stage ST40 - P2009

**BOUVIER Marc**

Département Génie Informatique

## Entreprise SMARTESTING

TEMIS Innovation - 18 Rue Alain Savary  
25000 Besançon  
[www.smartesting.com](http://www.smartesting.com)

Tuteur en entreprise  
**ALBIEZ Olivier**

Suiveur UTBM  
**HILAIRE Vincent**



Test Designer™





# Remerciements

Bla bla bla bla. Bla bla bla bla.

Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla.

Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla.

Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Bla bla bla bla. Je vous aime tous !

# Table des matières

<b>Remerciements</b>	<b>ii</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Présentation de l'entreprise d'accueil . . . . .	1
1.2 Test Designer . . . . .	2
1.3 Environnement de travail . . . . .	2
<b>2 Méthodes Agiles</b>	<b>4</b>
2.1 Contrôle de version . . . . .	4
2.2 Intégration continue . . . . .	4
2.3 Itération . . . . .	5
2.4 Rétrospective . . . . .	5
2.5 Fiches . . . . .	6
2.6 Travail en binômes (pair programming) . . . . .	8
2.7 Validation . . . . .	9
2.8 Semaine type . . . . .	9
2.8.1 Morning meeting . . . . .	10
2.8.2 Point technique . . . . .	10
2.8.3 Pomorodo . . . . .	11
<b>3 Activités confiées pendant le stage</b>	<b>12</b>
3.1 Découverte exploratoire de l'existant . . . . .	12
3.2 Développements sur le code de production . . . . .	12
3.2.1 Observations . . . . .	12
3.2.2 Descriptions . . . . .	13
3.2.3 Documentation . . . . .	15
3.2.4 Validation . . . . .	15
3.2.5 Amélioration du code existant (refactoring) . . . . .	15
3.2.6 Administration système . . . . .	15
3.2.7 Meeting corporate . . . . .	15
3.2.8 Visite de BNP Paribas . . . . .	15
3.2.9 Amélioration du process, évolution du fonctionnement de l'équipe .	15
<b>4 Conclusion</b>	<b>17</b>
4.0.10 Connaissances acquises . . . . .	17
4.0.11 Apport à l'entreprise d'accueil . . . . .	17
4.1 ce que j'ai apporté . . . . .	17

---

4.2 ce que j'ai appris . . . . .	17
4.3 difficultés . . . . .	17
<b>A Lexique</b>	<b>19</b>
A.1 Technologie JAVA . . . . .	19
A.2 Langage JAVA . . . . .	19
A.3 MBT . . . . .	19
A.4 Agile . . . . .	19
A.5 eXtreme Programming . . . . .	20
A.6 Vélocité . . . . .	20
A.7 Itération . . . . .	20
A.8 Intégration continue . . . . .	20
A.9 Programmation en binôme . . . . .	20
A.10 Tests unitaires . . . . .	21
A.11 Tests haut niveau . . . . .	21
A.12 Spike . . . . .	21
A.13 Tests fonctionnels . . . . .	21
A.14 Refactoring . . . . .	22
A.15 IntelliJ IDEA . . . . .	22
A.16 Client XP . . . . .	22
A.17 Commit . . . . .	22
A.18 Rollback . . . . .	22
A.19 Code contest . . . . .	22
A.20 OCL . . . . .	22
<b>B Bibliographie / Netographie</b>	<b>23</b>
<b>C Pomodoro Technique</b>	<b>24</b>

# Chapitre 1

## Introduction

### 1.1 Présentation de l'entreprise d'accueil

Créée en 2003 par Laurent Py et Bruno LEGEARD, la société LEIROS est spécialisée dans le test logiciel. Elle est issue du projet Smart Testing™ au LIFC<sup>1</sup>. L'objectif premier de LEIROS a été d'industrialiser le projet Smart Testing™. LEIROS a ensuite changé de nom pour devenir SMARTESTING en juin 2008. En septembre 2008, Smartesting ouvre sa filiale à Bangalore, en Inde. SMARTESTING compte environ trente-cinq personnes dont onze dans le service R&D<sup>2</sup>.

Smartesting est implantée dans différents points stratégiques. En France, le siège social ainsi que le centre R&D sont à Besançon dans les locaux de l'hôtel d'entreprises TEMIS Innovation. Ainsi la R&D reste proche géographiquement de l'université et de la recherche qui y a lieu. À Paris et à Amsterdam aux Pays-Bas se trouvent les agences où travaillent les commerciaux et les avant-vente. Smartesting s'est implantée à Bangalore, en Inde où elle compte réaliser la moitié de son chiffre d'affaires en 2010 grâce au boom de l'offshore où le marché du test logiciel est en pleine expansion.

Smartesting développe dans le secteur grandissant du test logiciel. Ce marché devrait atteindre 13 milliards de dollars en 2010 (selon une étude de Gartner). Le test logiciel, en particulier le test fonctionnel devient une phase clé du développement logiciel. Des besoins très stricts pour les milieux bancaires par exemple obligent ces entreprises à faire appel à des ingénieurs et des architectes de test afin de concevoir les tests logiciels qui permettront par exemple de garantir la stabilité, la non regression et le bon fonctionnement de gros projets. SMARTESTING propose Test Designer, une solution de génération automatique de référentiels de test à plusieurs niveaux.

L'entreprise est organisée autour d'un directoire de trois personnes : Laurent PY, Bruno LEGEARD et Stéphane WERBA. Je travaille dans le service de R&D de SMARTESTING. L'équipe est composée de 11 ingénieurs dévelopeurs qui améliorent sans cesse le produit Test Designer ainsi que les connecteurs et les publishers y sont développés. L'équipe

---

<sup>1</sup>Laboratoire d'informatique de Franche-Comté

<sup>2</sup>Recherche et développement

fonctionne autour de méthodes Agiles, en particulier eXtreme Programming. Ces sujets seront développés dans les sections qui vont suivre.

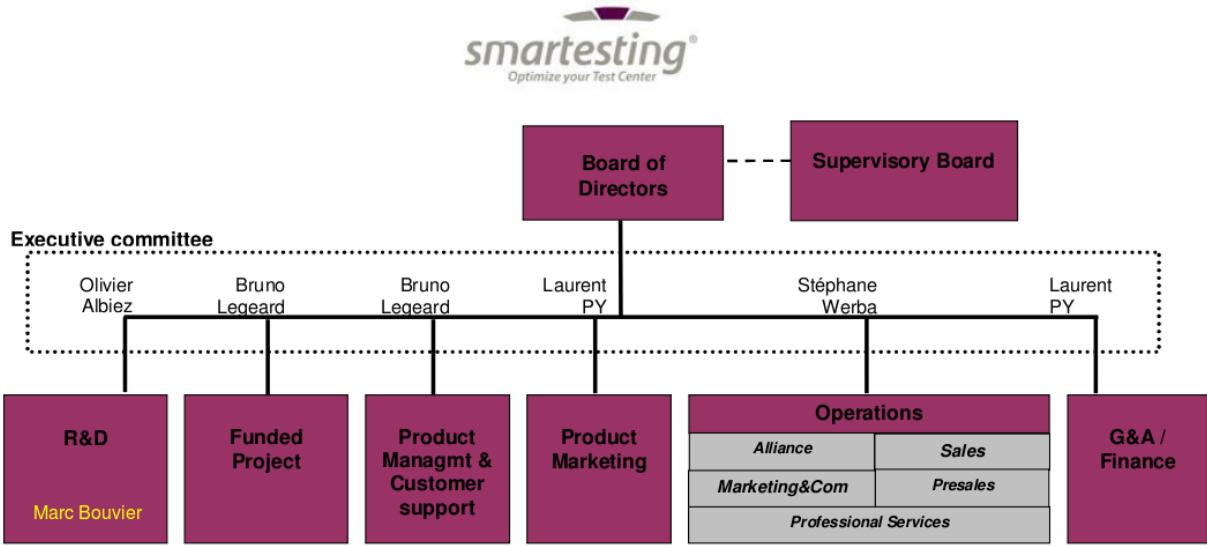


FIG. 1.1: Organigramme

## 1.2 Test Designer

La solution Test Designer permet de générer des référentiels de tests fonctionnels à partir de modèles UML pour le test/footnoteModel Based Testing. Elle fait le lien entre la modélisation de tests et le management de tests.

La modélisation de tests se fait à l'aide de modeleurs UML basés sur Eclipse<sup>3</sup>. Les modeleurs supportés actuellement par Test Designer sont Borland Together 2008 et IBM Rational Software Modeler 7.0.5 & 7.5.

## 1.3 Environnement de travail

La première tâche qui m'a été confiée à mon arrivée fut d'installer mon poste de travail. Un ordinateur avec Ubuntu 8.10 m'a été confié et j'y ai installé IntelliJ Idea 8.0 (IDE<sup>4</sup>), adapter quelques options de configuration au développement sur le projet Test Designer.

<sup>3</sup>Eclipse est une plateforme applicative sur laquelle peuvent se greffer des applications clientes sous forme de plugins(extensions)

<sup>4</sup>Environnement de développement intégré

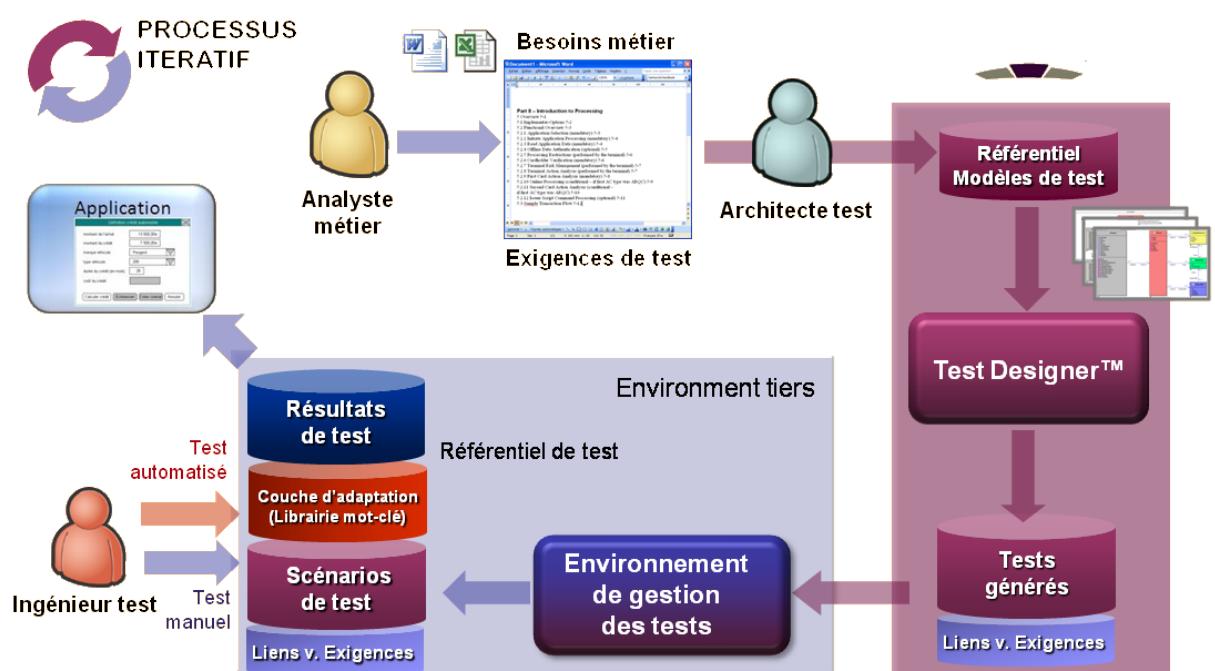


FIG. 1.2: La solution SMARTESTING

# Chapitre 2

## Méthodes Agiles

L'équipe dans laquelle je fais mon stage utilise des méthodes Agiles(cf. lexique A.4 p.19) et eXtreme Programming(cf. lexique A.5 p.20) en particulier. Ces méthodes ne sont pas figées et doivent être adaptées à chaque équipe. Il est donc courant et primordial que certaines pratiques soient remises en cause (cf. tableau 3.2 p.16). Ce chapitre donne un aperçu des pratiques XP utilisées par l'équipe R&D de SMARTESTING dans le cadre du développement de la solution Smartesting.

### 2.1 Contrôle de version

Le projet ainsi que différentes ressources de SMARTESTING sont versionnées sur un serveur Subversion (SVN). L'environnement de développement IntelliJ IDEA est compatible avec SVN et permet une utilisation efficace et agréable de cet outil. Le contrôle de version est primordial dans le processus de développement pour plusieurs raisons. Il sert de “garde-fou” ; lorsqu'un développement n'aboutit pas il est facile de Rollback(cf. lexique A.18 p.22). Il permet également grâce à la gestion des conflits de gérer des modifications effectuées sur les mêmes fichiers par des binômes différents (cf. figure 2.1 p.5). TRAC permet également de faire un suivi rapide du code source qui a été “commit” (cf. figure 2.2 p.6).

### 2.2 Intégration continue

L'équipe utilise l'intégration continue pour construire et tester en permanence le projet. Cela permet d'avoir un retour très rapide sur la qualité des dernières fonctionnalités qui ont été intégrées. Des tâches automatisées très différentes tournent en continu sur différents. Le serveur d'intégration continue tourne sous la plateforme Hudson (cf. figure 2.3 p.7). Ce serveur pilote différents agents d'intégration continue en leur envoyant des tâches à effectuer. Ces tâches peuvent être le build complet du projet, installation automatique des plugins sur les modeleurs, tests unitaires, tests de validation, tests FIT... Un panneau lumineux visible de toute l'équipe permet de réagir très vite en cas d'échec du build ou des tests. Les développeurs (en général Batman) agissent au plus vite pour réparer les erreurs mises en évidence.

Ignore whitespace: Leading and trailing

42683(Read-only)

44443

```
private DescriptionRenderer() { }  
public static DocumentBuilderUtils.DocumentBuilderAction processElement(DocumentBuilderUtils.DocumentBuilder processElement(description.getChild("body"), Font.NORMAL) return paragraph(newArray(actionCollector, DocumentBuilder )) }  
private static void processElement(final Parent description, final int style, final Collection<DocumentBuilderUtils.DocumentBuilder final FontAdapter adapter = new FontAdapter(); adapter.appendStyle(style); for (final Content content : (List<Content>) description if (content instanceof Text) { actionCollector.add(text((Text) content).get continue; } final Element tag = (Element) content; if (tag.getName().equals("strong")) { processElement(tag, adapter.getStyle() | Font continue; } if (tag.getName().equals("em")) { processElement(tag, adapter.getStyle() | Font continue; } if (tag.getName().equals("u")) { processElement(tag, adapter.getStyle() | Font continue; } if (tag.getName().equals("strike")) { processElement(tag, adapter.getStyle() | Font continue; } if (tag.getName().equals("u1")) { final Collection<DocumentBuilderUtils.DocumentBuilder processElement(tag, style, items); actionCollector.add(list(newArray(items, DocumentBuilder )) continue; }  
public final class SpecificationDescriptionRenderer implements public DocumentBuilderUtils.DocumentBuilderAction render( final Collection<DocumentBuilderUtils.DocumentBuilder processElement(description, Font.NORMAL, actionCollector return paragraph(toArray(actionCollector, DocumentBuilder )) }  
private static void processElement(final Parent current, final int style, final Collection<DocumentBuilderUtils.DocumentBuilder final FontAdapter adapter = new FontAdapter(); adapter.appendStyle(style); for (final Object content : current.getContent()) { if (content instanceof Text) { actionCollector.add(text((Text) content).get continue; } final Element tag = (Element) content; if (tag.getName().equals("p")) { final Collection<DocumentBuilderUtils.DocumentBuilder processElement(tag, adapter.getStyle(), paragraph actionCollector.add( paragraph( toArray( paragraphContent, DocumentBuilder )) continue; } if (tag.getName().equals("div")) { processElement(tag, adapter.getStyle(), action continue; } if (tag.getName().equals("strong")) { processElement(tag, adapter.getStyle() | Font continue; } if (tag.getName().equals("em")) { processElement(tag, adapter.getStyle() | Font continue; }
```

Deleted      Changed      Inserted

14 differences

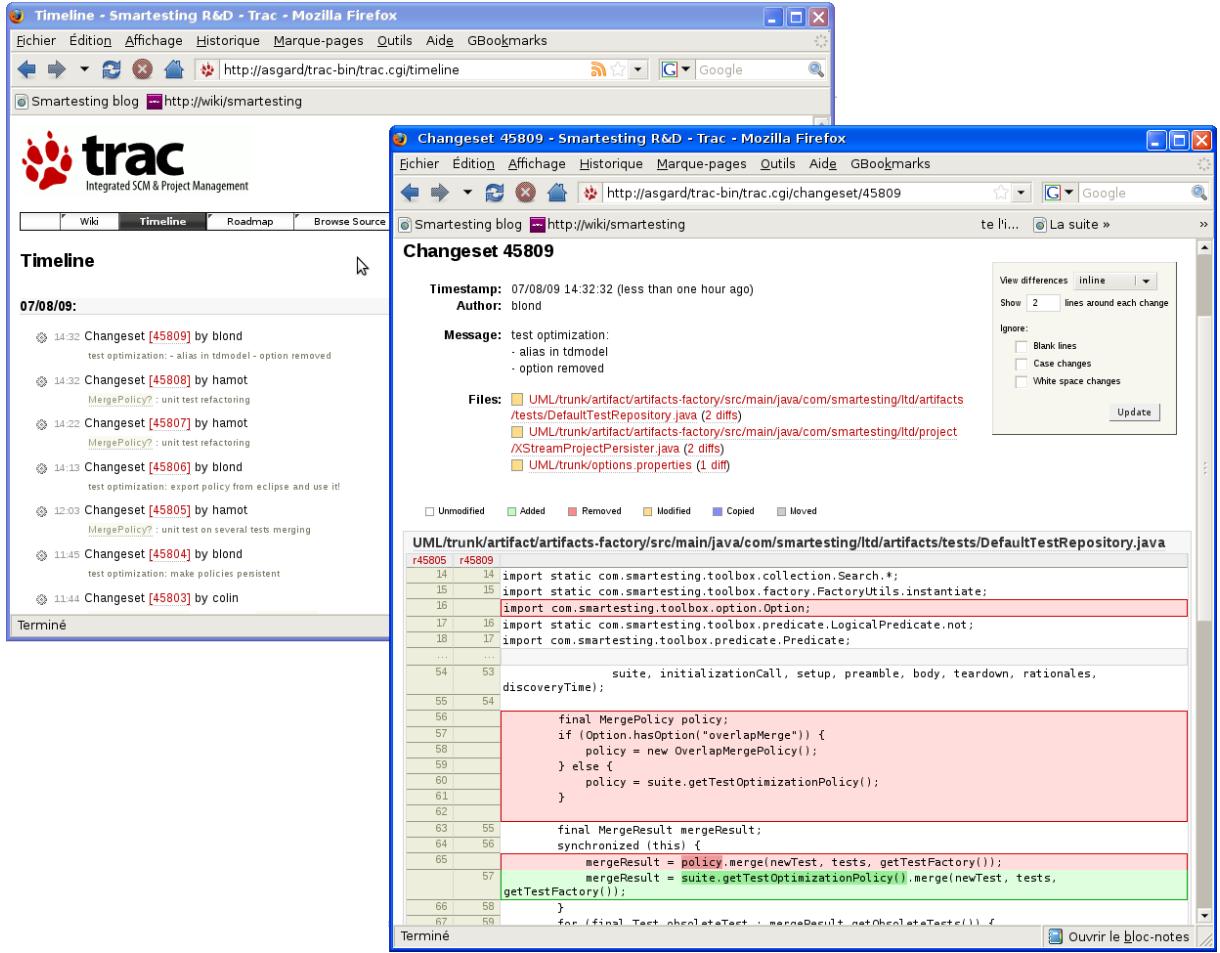
**FIG. 2.1:** Gestion de conflits dans IntelliJ IDEA

## 2.3 Itération

L'équipe organise son travail sous forme d'itérations d'une semaine. Chaque itération traite un nombre de fonctionnalités limité qui est quantifié en points de vitesse (cf. lexique A.6 p.20). À la fin de mon stage la vitesse variait entre 7 et 13 points par semaine. Une itération est planifiée lors de la rétrospective de l'itération précédente. Un itération compte un certain nombre de points de vitesse qui constitue une estimation de la durée requise pour développer les fonctionnalités planifiées. La durée d'une itération à la première semaine de mon stage était de deux semaines. Elle est tout de suite passée à une semaine. À la fin d'une itération le produit est livré. Les versions mineures sont disponibles pour les consultants afin de les tester sur le terrain et d'obtenir des retours en condition réelle. Les versions majeures de fin de jalon (Milestone) bénéficient d'une période de validation approfondie et sont disponibles aux utilisateurs finaux.

## 2.4 Rétrospective

Lors de la rétrospective d'une itération, les membres de l'équipe de R& D se réunissent pour parler de la précédente itération et pour planifier celle à venir. La réunion commence par un “check in” pendant lequel une question est posée (par exemple “Comment voyez



**FIG. 2.2:** Visualisation des révisions du code source avec TRAC

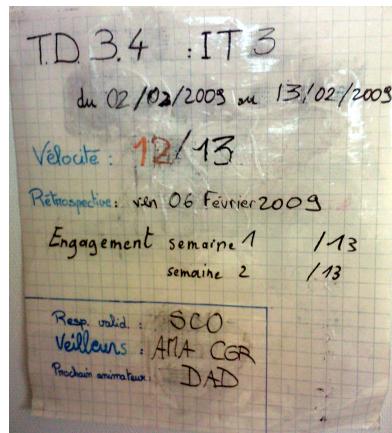
vous Test Designer dans un an ?") en général afin de se remémorer l'itération (se remettre mentalement dedans). Un point est sur des statistiques (métriques) telles que le nombre de lignes de code, la couverture de tests unitaires (cf. lexique A.10 p.21) et haut niveau (cf. lexique A.11 p.21) ou encore la vélocité atteinte par rapport aux objectifs. L'attention est ensuite portée sur les fonctionnalités qui ont été réalisées ou non lors de l'itération. La réunion se continue par les discussions diverses, chacun peut discuter de diverses choses : apparition de problèmes, amélioration du fonctionnement de l'équipe. La réunion se termine enfin par l'annonce de la prochaine vélocité et l'attribution des responsabilités pour l'itération qui vient. En effet chaque itération a des responsables différents (animateur de la rétrospective, validation, veilleur ...).

## 2.5 Fiches

Des fiches correspondent aux différentes tâches à effectuer dans le cadre d'une itération. Elles peuvent être de type différent : valeur client (couleur blanche), Tâche technique (couleur verte), Point technique (couleur bleue), Anomalie (couleur rouge), Amélioration de process / Prototype (couleur jaune). Les couleurs des fiches permettent à l'équipe

The screenshot shows the Hudson CI dashboard. At the top, there's a navigation bar with links for 'Nouveau job', 'Administrer Hudson', 'Personnes', 'Historique des builds', and 'Leader board'. The main area features the 'smartesting' logo with the tagline 'Optimize your Test Center'. On the left, there are two sections: 'File d'attente des builds' and 'Etat du lanceur de build'. The 'File d'attente des builds' section lists several jobs: 'ld-smoketest', 'ld-highleveltest', 'ld-unitest-linux', 'ld-t4t', 'ld-build', and 'ld-validationtest'. The 'Etat du lanceur de build' section shows the status of various launchers across different environments: APOLLO, ARTEMIS, CHRONOS, DEMETER, EOLE, EOS, EROS, GAIA, PAN, and PONTOS. Each launcher has a progress bar indicating its current state. On the right, a large table displays the history of builds for each job. The columns include 'S' (row number), 'W' (status icon), 'Job' (job name), 'Dernier succès' (last success), 'Dernier échec' (last failure), and 'Dernière durée' (last duration). Each row also includes a small icon and a link to the build details.

**FIG. 2.3:** Visualisation et gestion de l'intégration continue



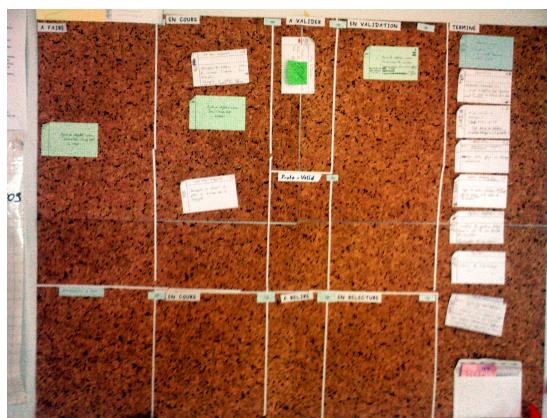
**FIG. 2.4:** Iteration

d'identifier au premier coup d'oeil le travail à réaliser et l'urgence. Si le tableau comporte beaucoup de fiches rouges, elles seront à traiter en priorité car ce sont des anomalies. Les fiches possèdent des points de vélocité qui correspondent à la quantité de travail à

effectuer (en demie-journée par binôme) sur la tâche. Si la fiche est trop importante elle peut être redécoupées en fiches plus petites.



**FIG. 2.5:** Fiches



**FIG. 2.6:** Tableau d'avancement des fiches

## 2.6 Travail en binômes (pair programming)

Une grande partie du travail dans l'équipe est réalisée en binôme. Les binômes changent très souvent. Les avantages du travail en binôme sont multiples. Contrairement à la programmation individuelle les binômes permettent de diffuser plus rapidement le savoir acquis lors du développement et seront plus à même à partager leur connaissances (effet boule de neige). De plus des études montrent que le travail par paires donne généralement un code de meilleure qualité, une meilleure capture des bugs. La communication et la motivation dans l'équipe sont également améliorées par cette méthode. Chaque personne dans le binôme peut prendre le clavier et la souris pour donner ses idées de développement. En général le travail en binôme est un échange d'idées permanent. Chaque binôme, en début de fiche s'engage sur cette fiche et estime le temps qui lui sera nécessaire pour la terminer. Cette estimation est revue régulièrement (lors du morning meeting par exemple) et est communiquée à l'équipe et au client XP.



FIG. 2.7: *Pair programming*

## 2.7 Validation

La validation joue un rôle important lors de l’itération. Une partie de la validation est réalisée en permanence par les serveurs d’intégration continue. Une autre partie (en général, l’utilisation via l’interface homme-machine) ne peut être réalisée que par des personnes qui n’ont pas participé au développement de la fonctionnalité. Cela permet d’avoir une idée plus objective des manipulations à effectuer pour tester la fonctionnalité. Au cours de la semaine la validation est la tâche exclusive de Batman. Il valide une fiche dès qu’elle a été mise “À valider” afin d’avoir le retour le plus rapide sur la fonctionnalité. Lors de la validation, on fait très souvent appel au client XP pour vérifier que la fonctionnalité correspond bien aux besoin énoncés. À la fin de l’itération, lorsque toutes les fiches sont terminées et validées, l’équipe commence une validation de l’ensemble des fiches de l’itération. Lorsque toute l’équipe est satisfaite (les bugs éventuels corrigés, documentation relue ...) la livraison peut avoir lieu.

## 2.8 Semaine type

La semaine Agile à la R&d de SMARTESTING est rythmée par quelques pratiques qui peuvent évoluer.

### Cotation et engagement

Avant de commencer une itération, l’engagement est fait. C’est à dire que les fiches qui vont déterminer les développements de la semaine vont être choisies par le client XP. Les fiches sont posées sur le tableau en vue de tous. La somme des vélocités des fiches correspond à la vélocité déterminée lors de la retrospective de l’itération précédente.

L'équipe peut alors choisir les fiches qui seront commencées dans la semaine, les binômes se créent. Chaque binôme va ensuite voir le client XP afin de confirmer le périmètre fonctionnel de la fiche. Le client XP peut aussi demander la cotation sur des fiches cotée “gros grain” du planning game. L'équipe est réunie et une discussion technique a lieu. Après la discussion l'équipe vote la vitesse estimée de la fiche.

### 2.8.1 Morning meeting

Les membres de l'équipe se réunissent juste avant la pause déjeuner pour parler de ce qu'ils ont fait la matinée. Toute l'équipe est réunie en cercle et se fait passer un objet. Celui qui a l'objet prend la parole. Chaque binôme informe toute l'équipe de l'avancée de son engagement ainsi que des problèmes qu'il rencontre. Le temps de parole est très bref et si un problème mérite une attention particulière il sera traité hors du cercle plus tard. Après le tour du cercle s'il y a des informations complémentaires à mentionner elles le sont. Le meeting se termine ensuite par une pensée du jour.

#### Point perso

Le mardi, juste avant la pause déjeuner, un membre de l'équipe fait une présentation à l'aide d'un projecteur sur un sujet de son choix (pas nécessairement lié à l'informatique d'ailleurs). La présentation doit durer 10 minutes. À la fin, les membres de l'équipe peuvent poser des questions et commenter la manière dont la présentation a été réalisée (intéresser le public, parler clairement, qualité du support...). Cet exercice de communication permet aux membres de l'équipe d'apprendre à partager et à structurer leurs idées. Cela peut être très utile lors de manifestations diverses (xp days, agile tour...). Le point perso a évolué entre le début et la fin de mon stage, et actuellement différentes nouvelles formes sont expérimentées.

#### Discussion sur un livre

L'équipe lit un chapitre d'un livre pendant la semaine et se réunit le mercredi avant la pause de midi pour le commenter. Lorsque je suis arrivé elle lisait “The Art of Agile Development”. Cette activité permet déjà de lire en anglais, mais également d'améliorer la cohésion de l'équipe par l'éventuelle adoption de pratiques nouvelles. Plusieurs des pratiques évoquées dans ce livre ont été adoptées après mon arrivée. Ce fut le cas de “No bugs”, “Done-done”, “Slack time” par exemple. Cette pratique a disparu avec le temps car plusieurs personnes de l'équipe n'y trouvaient plus d'intérêt. Différentes pratiques sont en cours d'expérimentation pour remplacer la lecture.

### 2.8.2 Point technique

Le Jeudi matin en début de matinée a lieu le point technique il peut varier dans son contenu. Cela peut aller de la discussion d'un point technique rencontré lors de l'itération à un code contest(cf. lexique A.19 p.22) de 30 minutes. L'équipe profite de ce point pour apprendre de nouvelles techniques, faire de la “veille technologique”, faire le point sur des

événements tels que les XP Days<sup>1</sup>. Les points techniques, au cours de mon stage n'ont plus eu de date précise mais étaient organisés selon les besoins de l'équipe.

### **2.8.3 Pomorodo**

Un mois avant la fin de mon stage l'équipé a commencé à expérimenter une technique de gestion du temps qui avait été présentée lors du meeting corporate le 10 avril. Cette technique consiste à découper

---

<sup>1</sup>conférence sur les méthodes agiles <http://www.xpday.fr/>

# Chapitre 3

## Activités confiées pendant le stage

Pendant mon stage j'ai participé à beaucoup de fiches différentes, il serait long et ennuyeux de les détailler toutes sachant que je participai à entre 1 et 4 fiches par semaine. Je présenterai dans les grandes lignes les fonctionnalités majeures auxquelles j'ai participé.

### 3.1 Découverte exploratoire de l'existant

TODO

### 3.2 Développements sur le code de production

Parmi les activités qui m'ont été confiées, j'ai participé à la réalisation de fiches blanches (cf. 2.5 6). Ce code est appelé “code de production” car il couvre les besoins fonctionnels exprimés par le client XP. La première étape dans le travail sur une fiche est de préciser le périmètre fonctionnel avec le client XP. Ceci permet aux développeurs de connaître son besoin précis. Ensuite, dans la mesure du possible, on crée les tests unitaires qui permettront de savoir si la fonctionnalité est opérationnelle. Les membres du binôme peuvent prendre le clavier au moment où ils le souhaitent et proposer leurs idées. Chaque direction prise et ainsi réfléchie et validée par chaque membre du binôme.

#### 3.2.1 Observations

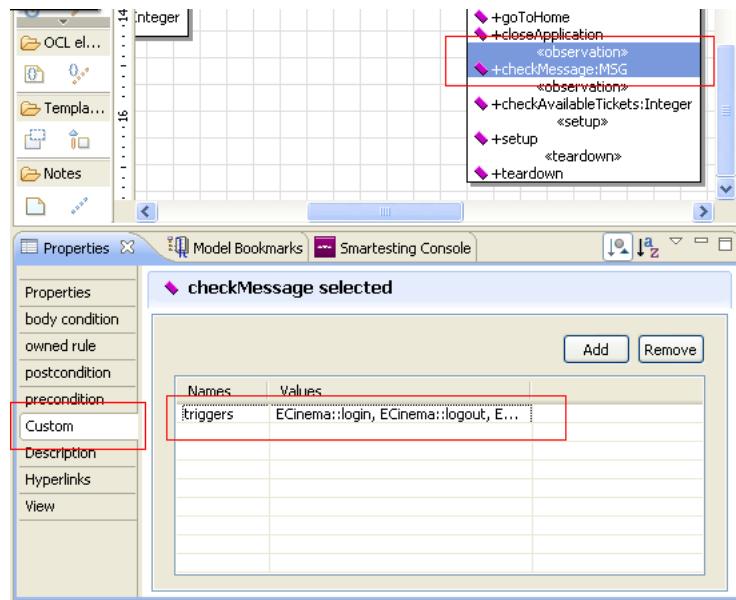
Au début de mon stage, la première fonctionnalité sur laquelle j'ai travaillé fut des observations. Dans le produit, les observations sont des stéréotypes<sup>1</sup> attachés à des opérations de classes du modèle de test. Les observations sont déclenchées par des triggers<sup>2</sup> selon une précondition décrite dans le langage OCL(cf. lexique A.20 p.22). Dans les modéleurs, à l'export, ce stéréotype est reconnu et intégré au modèle propre à Test Designer. Les observations font office de “points de contrôle” permettant de vérifier que le système sous test réagit correctement.

---

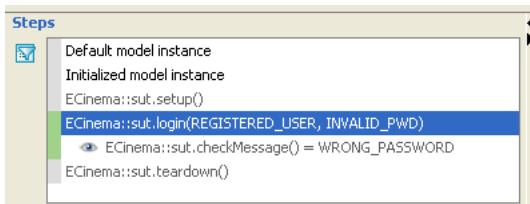
<sup>1</sup>stéréotypes au sens UML

<sup>2</sup>Évènement déclencheur

Les observations sont introduites dans les deux modeleurs principaux de façon différente. Dans RSM, les stéréotypes et la gestion des triggers spécifiques à SMARTESTING proviennent d'un profil personnalisé. L'utilisation des profils dans le modeleur Together ont été l'occasion d'une autre fiche à laquelle j'ai contribué. Cependant mon binôme et moi-même nous sommes rendu compte, après avoir passé 4 points de vélocité sur cette fiche cotée à 2, que c'était plus compliqué que nous l'imaginions. Nous avons alors suspendu la fiche en attente de conseils d'OBEO<sup>3</sup>. Finalement la après leur réponse, nous avons dû Rollback(cf. lexique A.18 p.22) notre travail. La solution adaptée finalement fut d'utiliser le code péexistant. C'est à dire utiliser des propriétés "Custom"(cf. figure 3.1 p.13) pour gérer les triggers dans Together. Ces triggers servent à déterminer les opérations que l'on souhaite observer. Une fois définies dans le modèle UML et après export, les observations sont visibles dans Test Designer (cf. figure3.2 p.13).



**FIG. 3.1:** Intégration des observations dans Together



**FIG. 3.2:** Intégration des observations dans Test Designer

### 3.2.2 Descriptions

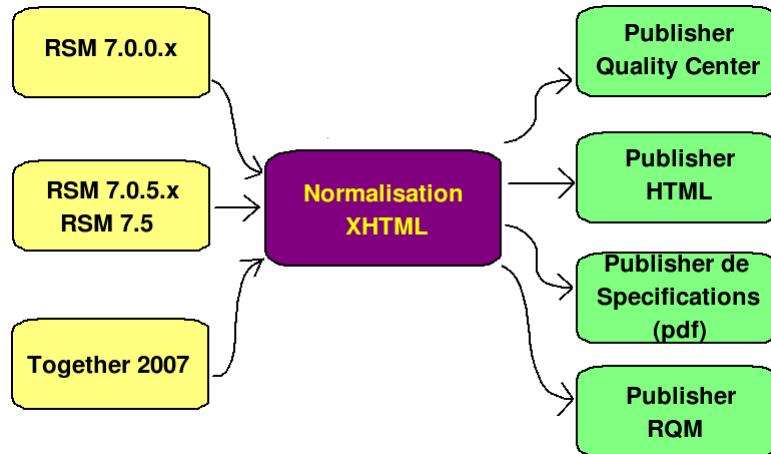
Dans test Designer, plusieurs éléments ont besoin d'être décrits afin d'être exploitable à la publication par exemple dans la publication HTML, HP Mercury Quality Center

<sup>3</sup>OBEO est une société de conseil experte dans le domaine de la modélisation EMF/GMF sous Eclipse

ou encore dans Rational Quality Manager. A l'origine les descriptions qui étaient déjà implémentées étaient publiées en texte brut. Après un court spike (cf. lexique A.12 p.21) nous nous sommes aperçus que plusieurs publishers supportaient des balises HTML. Il était également possible de mettre du texte en forme en amont dans les modeleurs (cf. tableau 3.1 p. 14). Néanmoins certaines versions de modeleurs n'ont pas les mêmes standards de mise en forme des descriptions. Ainsi il a fallu passer par une étape intermédiaire pour unifier les descriptions dans un format spécifique interne. Il est par exemple nécessaire d'enlever certaines balises en trop ou encore de convertir des balises en d'autres plus globalement supportées. Nous avons choisi d'utiliser le format XHTML avec l'aide de la bibliothèque JTidy (cf 3.3 p. 14). Les descriptions obtenues sont ensuite "rendues" pour être affichées ainsi qu'on le souhaite dans les différents publishers.

**TAB. 3.1: Compatibilité HTML des modeleurs et publishers**

Type	Nom	Balises supportées (simplifié)
Modeleurs	Together 2007	HTML <b><i><u>
	RSM 7.0.0.x	texte brut
	RSM 7.0.5.x	HTML <b><i><u> + centre + paragraphes
	RSM 7.5	HTML <b><i><u> + centre + paragraphes + autres
Publishers	HTML Quality Center Specification (pdf) Prototype RQM	texte brut et HTML <b><i><u> <strong><em><ins> HTML <b><i><u>



**FIG. 3.3: Passage des descriptions des modeleurs à leur publication**

fiche à laquelle j'ai participé traitait d'un bug d'IHM<sup>4</sup>. TODO : Correction bug affichage no changes, travail sur Setup/teardown,

<sup>4</sup>Interface homme-machine

### 3.2.3 Documentation

### 3.2.4 Validation

La validation des fiches est obligatoire pour considérer que la fonctionnalité lui correspondant est terminée. Elle consiste au test de la fonctionnalité sur l'application après commit. La validation se fait généralement en binôme et il faut chercher toutes les combinaisons en rapport avec la nouvelle fonctionnalité qui sont susceptible d'échouer, de vérifier si toutes les objectifs de la fiche sont remplis. Après avoir testé un maximum de cas de figures il faut faire appel au Client XP qui va s'assurer que l'engagement a été rempli. Une fois cela fait, la fiche est terminée et son nombre de points de vélocité peut être ajouté à ce qui a déjà été fait

### 3.2.5 Amélioration du code existant (refactoring)

TODO : utilité ,exemple,

### 3.2.6 Administration système

Sur une courte période j'ai effectué des tâches d'administration système. En particulier au moment de l'intégration des Google Apps dans le fonctionnement de Smartesting. La nécessité de partager des calendriers et de pouvoir y accéder via des plateformes mobiles a amené Smartesting à envisager d'utiliser Google Apps. Ainsi, en binôme avec Olivier, nous avons appréhendé le panneau d'administration ainsi que les différents services utilisables. Chaque calendrier donne la possibilité d'être exporté, ainsi nous avons pu réaliser une routine de backup<sup>5</sup>.

### 3.2.7 Meeting corporate

TODO : definition, utilité, ma participation ...

### 3.2.8 Visite de BNP Paribas

Mes impressions, les decisions, Smart et BNP ...

### 3.2.9 Amélioration du process, évolution du fonctionnement de l'équipe

J'ai été ammené à l'occasion de rétrospectives ou de réunions à réfléchir sur le processus de développement au sein de l'équipe. L'équipe de R&D est très concernée par l'amélioration du processus et toute pratique peut être remise en cause si elle ne convient pas à l'équipe. Chaque décision quelle qu'elle soit doit être approuvée par toute l'équipe avant d'être prise. Je parlerai plus en détail à travers d'exemple du type de décisions qui sont prises sur ce sujet.

Tableau des évolution des pratiques XP

---

<sup>5</sup>sauvegarde automatique

**TAB. 3.2:** Evolution des pratiques XP au cours du stage

Pratique	Début du stage	Fin du stage
Pair programming	✓	✓
Itération	2 semaines	1 semaine
Intégration continue	✓	✓
Niko Niko	✓	✗
Lecture	✓	✗
Point perso	hebdomadaire	nouvelles expérimentations
Pomodoro	✗	✓
Test driven development	✓	✓
“Done done”	théorique	adopté et normalisé
“No Bugs”	imprécis	engagement
Slack Time	✗	adopté et normalisé
Rétrospective	1 à 2h le lundi matin	Timeboxée et juste après la livraison
Point technique	assez réguliers	moins nombreux
Veilleur	✓	Robin cumule son rôle
Batman/Robin	✗	✓

TODO : Pomodoro, Decisions lors de retrospectives, Iteration 1 semaine, Slack, reduction de vitesse, Done-done, rédaction(et simplification) des standards, Objectifs R&D, ...

# Chapitre 4

## Conclusion

### 4.0.10 Connaissances acquises

Programmation JAVA Utilisation de tests unitaires???. Design patterns Travail en Open-Space Initiation à la méthode Agile en particulier à l'eXtreme Programming

### 4.0.11 Apport à l'entreprise d'accueil

#### 4.1 ce que j'ai apporté

#### 4.2 ce que j'ai appris

#### 4.3 difficultés

blabla

## **Annexes**

# Annexe A

## Lexique

### A.1 Technologie JAVA

JAVA est le nom d'une technologie mise au point par Sun Microsystems qui permet de produire des logiciels indépendants de toute architecture matérielle. Cette technologie s'appuie sur différents éléments qui, par abus de langage, sont souvent tous appelés Java.

### A.2 Langage JAVA

Le Langage Java est un langage de programmation informatique orienté objet créé par James Gosling et Patrick Naughton employés de Sun Microsystems avec le soutien de Bill Joy (cofondateur de Sun Microsystems en 1982), présenté officiellement le 23 mai 1995 au SunWorld. Le langage Java a la particularité principale que les logiciels écrits avec ce dernier sont très facilement portables sur plusieurs systèmes d'exploitation tels que Unix, Microsoft Windows, Mac OS ou Linux avec peu ou pas de modification... C'est la plate-forme qui garantit la portabilité des applications développées en Java. Le langage reprend en grande partie la syntaxe du langage C++, très utilisé par les informaticiens. Néanmoins, Java a été épuré des concepts les plus subtils du C++ et à la fois les plus déroutants, tels que l'héritage multiple l'embauche par Jim Blandy de Karl Fogel, qui travaillait déjà sur un nouveau gestionnaire de version.

### A.3 MBT

Parmi les différentes approches du test logiciel, MBT (Model Based Testing) a pour objectif de créer des cas de test à partir de modèles abstraits.

### A.4 Agile

Les méthodes Agiles sont des procédures de conception de logiciel. Le client est impliqué au maximum dans le processus de conception. Ainsi, ces méthodes permettent une grande réactivité à ses demandes. Elles visent la réelle satisfaction de son besoin. Le besoin réel du client est prioritaire aux termes du contrat de développement. La notion de

méthode agile est née à travers un Manifeste Agile signé par 17 personnalités en 2001. La notion de méthode agile se limite actuellement aux méthodes ciblant le développement d'une application informatique. Cependant des pratiques de l'agilité sont utilisées dans la gestion de certains projets non-informatiques.

## A.5 eXtreme Programming

L'eXtreme Programming (XP) est une méthode Agile(cf. A.4 p.19) à la philosophie d'accepter le changement plutôt que de le supporter. XP prône les cycles de développement courts (1 à 2 semaines). Un Client XP(cf. A.16 p.22) permet d'avoir des retours rapides et de s'assurer que les exigences fonctionnelles soient bien comprises et implémentées. L'eXtreme Programming repose sur cinq valeurs :

- La communication
- La simplicité
- Le feedback
- Le courage
- Le respect

## A.6 Vélocité

La vélocité est une métrique qui permet de faire une estimation du temps que prendra une tâche. La vélocité est utilisée lors de la cotation des fiches. Une fois les fiches réalisées on peut mesurer la différence entre l'estimation et la réalité. Cela dit ce n'est qu'une métrique, même si elle donne un bon ordre d'idée sur l'écoulement du temps elle ne doit pas être considérée comme un objectif en elle-même.

## A.7 Itération

Une itération a en général une durée de 1 à 2 semaines. Elle se termine par la livraison (mineure ou majeure) du programme. Une rétrospective permet de faire le bilan de l'itération passée.

## A.8 Intégration continue

Lorsqu'une tâche est terminée, les modifications sont immédiatement intégrées dans le produit complet. On évite ainsi la surcharge de travail liée à l'intégration de tous les éléments avant la livraison. Les tests facilitent grandement cette intégration : quand tous les tests passent, l'intégration est terminée.

## A.9 Programmation en binôme

La programmation se fait par deux. Le premier appelé driver (ou pilote) tient le clavier. C'est lui qui va travailler sur la portion de code à écrire. Le second appelé partner (ou

co-pilote) est là pour l'aider en suggérant de nouvelles possibilités ou en décelant d'éventuels problèmes. Les développeurs changent fréquemment de partenaire ce qui permet d'améliorer la connaissance collective de l'application et d'améliorer la communication au sein de l'équipe.

## A.10 Tests unitaires

En programmation informatique, le test unitaire est un procédé permettant de s'assurer du fonctionnement correct d'une partie déterminée d'un logiciel ou d'une portion d'un programme (appelée « unité »). Il s'agit pour le programmeur de tester un module, indépendamment du reste du programme, ceci afin de s'assurer qu'il répond aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances. Cette vérification est considérée comme essentielle, en particulier dans les applications critiques. Elle s'accompagne couramment d'une vérification de la couverture de code, qui consiste à s'assurer que le test conduit à exécuter l'ensemble (ou une fraction déterminée) des instructions présentes dans le code à tester. L'ensemble des tests unitaires doit être rejoué après une modification du code afin de vérifier qu'il n'y a pas de régressions (l'apparition de nouveaux dysfonctionnements). Dans les applications non critiques, l'écriture des tests unitaires a longtemps été considérée comme une tâche secondaire. Cependant, la méthode Extreme programming (XP) a remis les tests unitaires, appelés « tests du programmeur », au centre de l'activité de programmation. La méthode XP préconise d'écrire les tests en même temps, ou même avant la fonction à tester (Test Driven Development). Ceci permet de définir précisément l'interface du module à développer. En cas de découverte d'un bogue, on écrit la procédure de test qui reproduit le bogue. Après correction on relance le test, qui ne doit indiquer aucune erreur.

## A.11 Tests haut niveau

Contrairement aux tests unitaires, les tests haut niveau s'assurent qu'une fonctionnalité est opérationnelle dans sa globalité. Le résultat d'un opération d'un plus haut niveau d'abstraction est donc testé.

## A.12 Spike

Un spike est une recherche en général technologique portant sur un sujet tel que l'adoption d'une nouvelle bibliothèque, sur des directions possibles à prendre pour une grosse fonctionnalité... Elle se fait sur une durée déterminée. À la fin du spike le binôme est amené découper la fiche de spike en différentes plus petites, cela amène souvent à une nouvelle cotation plus fine et un choix technologique.

## A.13 Tests fonctionnels

À partir des scénarios définis par le client, l'équipe crée des procédures de test qui permettent de vérifier l'avancement du développement. Lorsque tous les tests fonctionnels

passent, l’itération est terminée. Ces tests sont souvent automatisés mais ce n’est pas toujours possible.

## **A.14 Refactoring**

La refactorisation, aussi appelé remaniement de code, (anglicisme venant de refactoring) est une opération de maintenance du code informatique. Elle consiste à retravailler le code source non pas pour ajouter une fonctionnalité supplémentaire au logiciel mais pour améliorer sa lisibilité, simplifier sa maintenance, ou changer sa générnicité (on parle aussi de remaniement). Une traduction plus appropriée serait réusinage. C'est donc une technique qui s'approche de l'optimisation du code, même si les objectifs sont radicalement différents.

## **A.15 IntelliJ IDEA**

IntelliJ IDEA est un environnement de développement JAVA qui permet entre autres la refactorisation de code. TODO.

## **A.16 Client XP**

Appelé également client sur site ; TODO

## **A.17 Commit**

TODO

## **A.18 Rollback**

TODO

## **A.19 Code contest**

## **A.20 OCL**

TODO : blabla OCL.

## Annexe B

### Bibliographie / Netographie

Wikipedia

[http://fr.wikipedia.org/wiki/Méthode\\_agile](http://fr.wikipedia.org/wiki/Méthode_agile)

[http://fr.wikipedia.org/wiki/Extreme\\_programming](http://fr.wikipedia.org/wiki/Extreme_programming)

Smartesting

<http://www.smartesting.com>

Google apps

<http://www.google.com/apps/intl/fr/business/index.html>

Pomodoro Technique

<http://www.pomodorotechnique.com>

[http://www.pomodorotechnique.com/downloads/pomodoro\\_cheat\\_sheet.pdf](http://www.pomodorotechnique.com/downloads/pomodoro_cheat_sheet.pdf)

[http://pomodorotechnique.com/resources/cirillo/ThePomodoroTechnique\\_v1-3.pdf](http://pomodorotechnique.com/resources/cirillo/ThePomodoroTechnique_v1-3.pdf)

The Art of Agile Development

By James Shore, Shane Warden, O'REILLY, October 2007

Pragmatic guide to agile software development

<http://www.oreilly.com>

(isbn :0-596-52767-5)

Joel on Software

By Joel Spolsky, Apress, 2004

<http://www.joelonsoftware.com>

(isbn :1-59059-389-8)

Smartesting for DUMMIES (limited edition)

By Remco Kwinkelenberg, Jean-Pierre Schoch, WILEY, 2008

(isbn :978-0-470-74165-8)

## Annexe C

# Pomodoro Technique

**Qu'est ce que c'est ?** La technique du pomodoro a été inventée par Francesco Cirillo. C'est une technique de gestion du temps qui peut être utilisée pour n'importe quelle type de tâche. L'objectif de la technique du pomodoro est de considérer le temps comme un allié dans ce que l'on veut faire et d'améliorer en permanence notre façon de travailler ou d'étudier.

### Que faut-il pour commencer ?

**Une minuterie de cuisine** Vous pouvez aussi bien utiliser un pomodoro<sup>1</sup> qu'un timer logiciel. Le régler sur 25 minutes.

**Une feuille de papier** Le papier blanc est idéal, c'est encore mieux avec des lignes et le papier à pomodoro pré-imprimé est parfait !

**Un crayon** Avoir un gomme est un plus.

**Comment commencer ?** L'unité de travail de base peut être découpée en cinq étapes.

- Choisir une tâche à accomplir
- Régler le pomodoro sur 25 minutes
- Travailler sur la tâche jusqu'à ce que le pomodoro sonne et faire une coche en face de la tâche sur la feuille de papier
- Prendre une courte pause (5 minutes)
- Tous les quatre pomodoros prendre une pause plus longue (15-25 minutes)

---

<sup>1</sup>minuterie de cuisine

## TOOLS

To start using the Technique you only need some simple tools:

- ❖ A KITCHEN TIMER (Pomodoro looks fine)
- ❖ A PENCIL
- ❖ A To Do SHEET
- ❖ AN ACTIVITY INVENTORY SHEET
- ❖ A RECORDS SHEET

You can download and print sheets' templates from Pomodoro Techniques web site.

## BASICS

Put all the activities you have to accomplish on the ACTIVITY INVENTORY SHEET. At the beginning of each day select the tasks you need to complete and copy them on the To Do SHEET.

Start working:

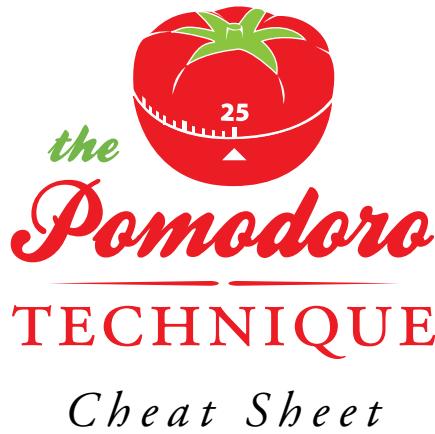
- ❖ Choose the topmost task from the list
- ❖ Set the Pomodoro to 25 minutes
- ❖ Work until the Pomodoro rings
- ❖ Mark the task with an x on the To Do SHEET
- ❖ Take a short break (3–5 minutes)

Keep on working, Pomodoro after Pomodoro, until the task at hand is finished, then cross it out on the To Do SHEET.

Every 4 Pomodoros take a longer break, (15–30 minutes).

## RULES & TIPS

- ❖ A Pomodoro is indivisible
- ❖ If a task takes more than 5–7 Pomodoros, break it down
- ❖ If it takes less than one pomodoro, add it up, and combine it with another task
- ❖ Once a Pomodoro begins, it has to ring
- ❖ The next pomodoro will go better
- ❖ The Pomodoro Technique shouldn't be used for activities you do in your free time. Enjoy free time!



## WHAT IS IT?

The Pomodoro Technique is a time management method that can be used for any kind of task. For many people, time is an enemy. The anxiety triggered by "the ticking clock", especially when a deadline is involved, leads to ineffective work and study habits which in turn lead to procrastination.

The aim of the Pomodoro Technique is to use time as a valuable ally in accomplishing what we want to do in the way we want to do it, and to enable us to continually improve the way we work or study.

## THE GOALS

The Pomodoro Technique will provide a simple tool/process for improving productivity (your own and that of your team members) which is able to do the following:

- ❖ Alleviate anxiety linked to becoming
- ❖ Enhance focus and concentration by cutting down on interruptions
- ❖ Increase awareness of your decisions
- ❖ Boost motivation and keep it constant
- ❖ Bolster the determination to achieve your goals
- ❖ Refine the estimation process, both in qualitative and quantitative terms
- ❖ Improve your work or study process
- ❖ Strengthen your determination to keep on applying yourself in the face of complex situations

## INTERRUPTIONS

Once you've started using the Pomodoro Technique, interruptions can become a real problem.

*Internal interruptions* are distractions that come from you: stand up and get something to eat or drink or to look up something on the Internet this minute.

Make these interruptions clearly visible. Every time you feel a potential interruption coming on, put an apostrophe (') on the sheet where you record your Pomodoros.

Then do one of the following:

- ❖ Write down the new activity on the To Do SHEET under Unplanned & Urgent if you think it's imminent and can't be put off.
- ❖ Write it down in the ACTIVITY INVENTORY SHEET, marking it with a "U" (unplanned); add a deadline if need be.
- ❖ Intensify your determination to finish the current Pomodoro. Once you've marked down the apostrophe, continue working on the given task till the Pomodoro rings.

People who work in social environments have to deal with *external interruptions*: a colleague asks you how to compile a report; an email program constantly beeps every time a new message comes in. A 25-minute or 2-hour delay (four Pomodoros) is almost always possible for activities that are commonly considered urgent.

Make these interruptions clearly visible. Every time someone or something tries to interrupt a Pomodoro, put a dash (-) on the sheet where you record your Pomodoros, apply the Inform, Negotiate, and Call Strategy.

Then apply one of the rules exposed above for internal interruptions.

*Downloadable book and much more on [www.pomodorotechnique.com](http://www.pomodorotechnique.com)*

# Table des figures

1.1	Organigramme	2
1.2	La solution SMARTESTING	3
2.1	Gestion de conflits dans IntelliJ IDEA	5
2.2	Visualisation des révisions du code source avec TRAC	6
2.3	Visualisation et gestion de l'intégration continue	7
2.4	Iteration	7
2.5	Fiches	8
2.6	Tableau d'avancement des fiches	8
2.7	Pair programming	9
3.1	Intégration des observations dans Together	13
3.2	Intégration des observations dans Test Designer	13
3.3	Passage des descriptions des modeleurs à leur publication	14

## Mots clefs

Mot clef 1 - Mot clef 2 - Mot clef 3 - Mot clef 4 - Mot clef 5 - Mot clef 6  
Mot clef 7 - Mot clef 8

**BOUVIER Marc**

**Rapport de stage ST50 - P2009**

## Résumé

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit. Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Sed non risus. Suspendisse lectus tortor, dignissim sit amet, adipiscing nec, ultricies sed, dolor. Cras elementum ultrices diam. Maecenas ligula massa, varius a, semper congue, euismod non, mi. Proin porttitor, orci nec nonummy molestie, enim est eleifend mi, non fermentum diam nisl sit amet erat. Duis semper. Duis arcu massa, scelerisque vitae, consequat in, pretium a, enim. Pellentesque congue. Ut in risus volutpat libero pharetra tempor. Cras vestibulum bibendum augue. Praesent egestas leo in pede. Praesent blandit odio eu enim. Pellentesque sed dui ut augue blandit sodales. Vestibulum ante ipsum primis in faucibus orci luctus et ultrices posuere cubilia Curae; Aliquam nibh. Mauris ac mauris sed pede pellentesque fermentum. Maecenas adipiscing ante non diam sodales hendrerit.

**Entreprise SMARTESTING**

TEMIS Innovation - 18 Rue Alain Savary  
25000 Besançon  
[www.smartesting.com](http://www.smartesting.com)