



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

Tuteur en entreprise
ALBIEZ Olivier

Suiveur UTBM
HILAIRE Vincent



Test Designer™



Remerciements

Je tiens à remercier tout d'abord mon maître de stage Olivier Albiez pour sa disponibilité, les conseils et toutes les choses qu'il m'a appris.

Un merci particulier à Laurent Py Bruno, Legeard et Stéphane Werba pour m'avoir accueilli au sein de Smartesting.

Un grand merci pour toute l'équipe de recherche et développement de Smartesting avec qui j'ai pu connaître une expérience unique en son genre et des plus instructives.

Table des matières

Remerciements	ii
1 Introduction	1
1.1 Présentation de l'entreprise d'accueil	1
1.2 Test Designer	3
1.3 Environnement de travail	4
2 Méthodes Agiles	5
2.1 Contrôle de version	5
2.2 Intégration continue	5
2.3 Itération	6
2.4 Rétrospective	6
2.5 Fiches	7
2.6 Travail en binômes (pair programming)	9
2.7 Code auto-commenté	10
2.8 Pomorodo	10
2.9 Semaine type	11
3 Activités confiées pendant le stage	13
3.1 Développements sur le code de production	13
3.1.1 Observations	13
3.1.2 Descriptions	14
3.1.3 Test Suites	15
3.2 Correction de bugs	17
3.3 Documentation	17
3.4 Validation	17
3.5 Amélioration du code existant (refactoring)	18
3.6 Prototypes	18
3.7 Administration système	18
3.8 Réunions et Meeting corporate	18
3.8.1 Visite de BNP Paribas	18
3.8.2 Amélioration du process, évolution du fonctionnement de l'équipe .	19
3.8.3 Evaluation	19

4 Conclusion	20
4.1 Connaissances acquises	20
4.1.1 Programmation JAVA	20
4.1.2 Test unitaires	20
4.1.3 Design de code	20
4.1.4 Travail en équipe	20
4.1.5 Méthodes Agiles	20
4.2 Apport à l'entreprise d'accueil	20
4.3 Difficultés rencontrées	20
A Lexique	22
A.1 Technologie JAVA	22
A.2 Langage JAVA	22
A.3 MBT	22
A.4 Agile	22
A.5 eXtreme Programming	23
A.6 Vélocité	23
A.7 Itération	23
A.8 Intégration continue	23
A.9 Programmation en binôme	24
A.10 Tests unitaires	24
A.11 Tests haut niveau	24
A.12 Spike	24
A.13 Tests fonctionnels	25
A.14 Refactoring	25
A.15 Serialisation	25
A.16 IntelliJ IDEA	25
A.17 Client XP	25
A.18 Commit	26
A.19 Rollback	26
A.20 Road-map	26
A.21 Code contest	26
A.22 OCL	26
B Bibliographie / Netographie	27
C Pomodoro Technique	28

Table des figures

1.1	Centre Temis à Besançon	1
1.2	Organigramme	2
1.3	La solution Smartesting	3
2.1	Gestion de conflits dans IntelliJ IDEA	6
2.2	Visualisation des révisions du code source avec TRAC	7
2.3	Visualisation et gestion de l'intégration continue	8
2.4	Iteration	8
2.5	Fiches	9
2.6	Tableau d'avancement des fiches	9
2.7	Pair programming	10
2.8	Exemple de code auto-commenté	11
3.1	Intégration des observations dans Together	14
3.2	Intégration des observations dans Test Designer	14
3.3	Passage des descriptions des modeleurs à leur publication	15
3.4	Exemple de fichier YAML	16
3.5	Exemple de fichier YAML pour les TestSuite	16

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 LIFC¹. 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 ².



FIG. 1.1: Centre Temis à Besançon

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 (cf. figure 1.1 p.1). 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

¹Laboratoire d'informatique de Franche-Comté

²Recherche et développement

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.

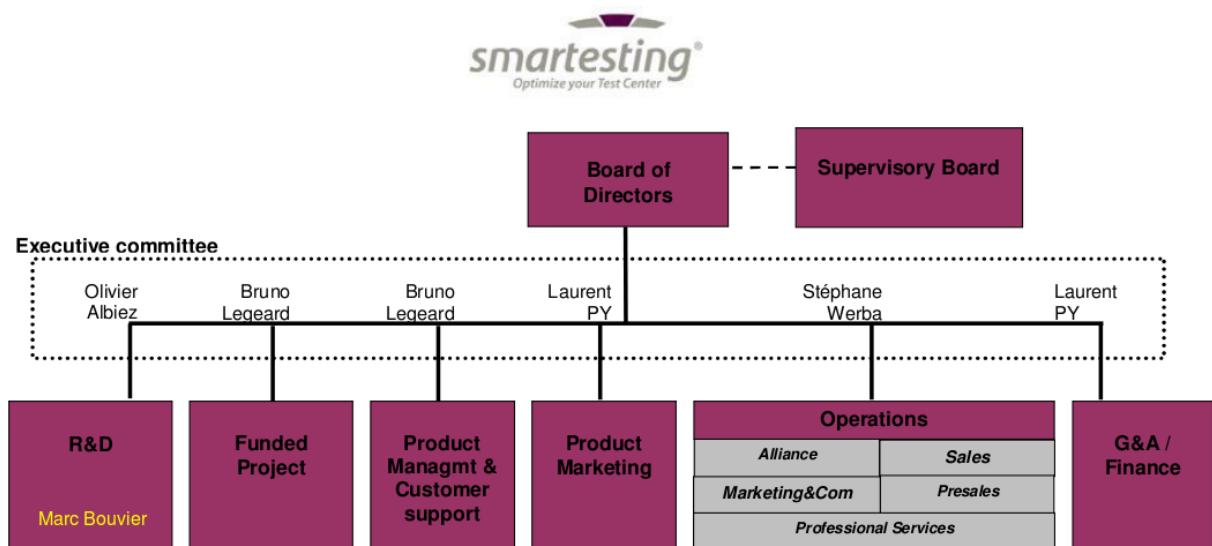


FIG. 1.2: Organigramme

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

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.

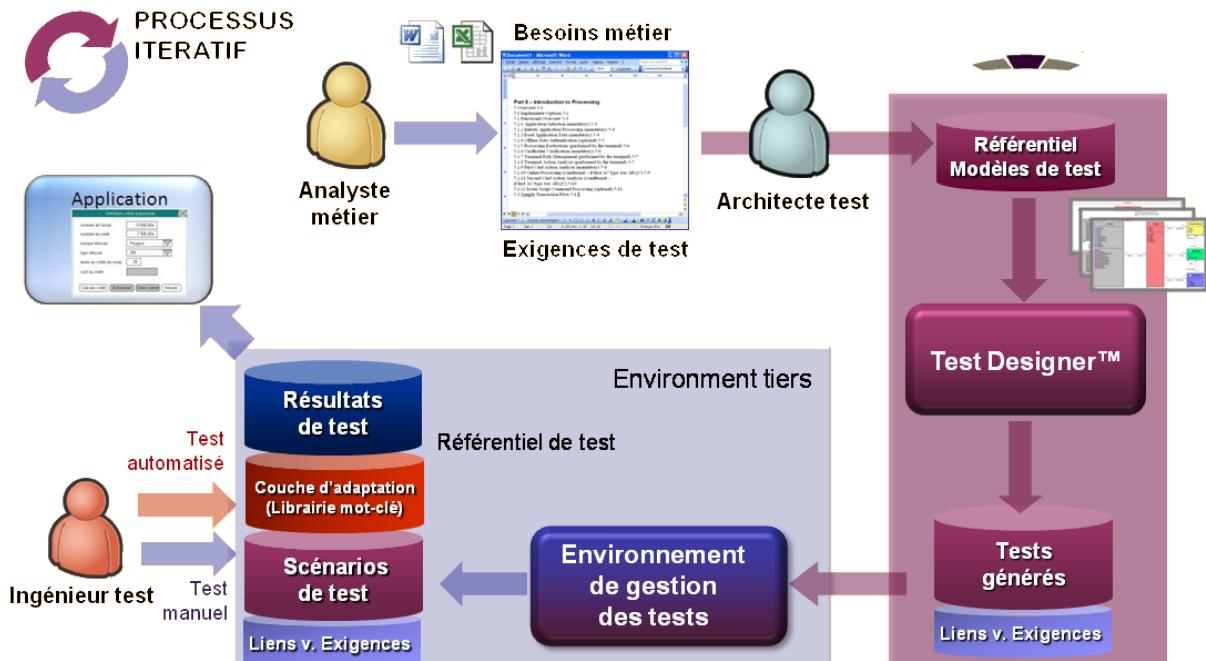


FIG. 1.3: La solution Smartesting

La modélisation de tests se fait à l'aide de modeleurs UML basés sur Eclipse³. Les modeleurs supportés actuellement par Test Designer sont Borland Together 2008 et IBM Rational Software Modeler 7.0.5 & 7.5. Il s'agit de modeleurs basés sur la plateforme Eclipse. Eclipse est une architecture de plugins qui communiquent les uns avec les autres pour former une application. C'est via ce mécanisme que Smartesting a développé deux plugins d'exportation de modèle pour les deux modeleurs précédemment cités.

Une fois le modèle de tests exporté il est utilisable par Test Designer pour générer automatiquement des référentiels de tests. Ces tests sont stockés dans un référentiel de test. Ces tests pourront être publiés vers des environnements tiers (HP Quality Center, tests JUnit, XML, Specifications pdf...).

Le processus de génération de tests de la solution Smartesting est un processus itératif. C'est à dire que si l'application déjà testée doit évoluer, la prise en compte des nouvelles fonctionnalités, ne générera pas un nouveau coût de génération des tests. Il suffit de modifier le modèle UML de spécifications via le modeleur, puis de générer les tests à

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

nouveau (automatique). Ainsi les architectes de test gagnent un temps considérable à ne pas générer des tests qui sont déjà existants.

Traditionnellement la génération de tests est effectuée manuellement par un ingénieur de tests.

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 (IDE⁴), adapter quelques options de configuration au développement sur le projet Test Designer. Je me suis ensuite familiarisé pendant une semaine avec le code existant. La consigne était de ne demander d'aide de personne pendant une semaine. Au début de la semaine qui suivit, je devais faire une compte rendu de ce que j'avais compris. Ce "test" permet en fait à l'équipe d'avoir un marqueur sur la lisibilité du code source. Après cette phase d'apprentissage de l'existant, j'ai commencé à travailler sur des fonctionnalités de Test Designer.

Le code source de Test Designer (le cœur de métier de Smartesting) est développée en JAVA avec un moteur de génération de tests qui s'appuie sur une approche prover (en c++). L'injection de dépendances est gérée par PICO. Une grande variété de bibliothèques sont utilisées telles que les google collections, JUnit, Mockito, JTidy, ou encore JYaml. Certaines "boîtes à outils" sont néanmoins développés en interne pour les besoins de la production. Les plugins dans les modeleurs utilisent l'architecture en plugin d'Eclipse pour s'y intégrer. En ce qui concerne les publishers, ils sont un fichier XML créé à partir du référentiel de tests. Ce fichier est ensuite exploité par le biais d'une API développée par l'équipe et distribuée aux clients lors de la livraison. Finalement l'API est aussi bien utilisée par les développeurs de l'équipe R&D, les consultants de Smartesting que les clients finaux.

⁴Environnement de développement intégré

Chapitre 2

Méthodes Agiles

L'équipe dans laquelle je fais mon stage utilise des méthodes Agiles(cf. lexique A.4 p.22) et eXtreme Programming(cf. lexique A.5 p.23) 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.19). 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.19 p.26). 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.6). TRAC permet également de faire un suivi rapide du code source qui a été “commit” (cf. figure 2.2 p.7).

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.8). 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 modèleurs, 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.

The screenshot shows a Java code editor with two panes. The left pane displays a class named `DescriptionRenderer` with several methods, including `processElement`. The right pane shows the same class with changes applied, specifically adding a new method `processElement` and modifying the existing one. The code is annotated with line numbers and color-coded markers (red for deleted, blue for changed, green for inserted) to indicate the nature of the differences.

```
private DescriptionRenderer() {  
}  
  
public static DocumentBuilderUtils.DocumentBuilderAction processElement(description.getChild("body"), Font.NORMAL)  
    return paragraph(newArray(actionCollector, DocumentBu  
}  
  
private static void processElement(  
    final Parent description,  
    final int style,  
    final Collection<DocumentBuilderUtils.DocumentBui  
final FontAdapter adapter = new FontAdapter();  
adapter.appendStyle(style);  
for (final Content content : (List<Content>) descript  
    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("ul")) {  
        final Collection<DocumentBuilderUtils.Document  
        processElement(tag, style, items);  
        actionCollector.add(list(newArray(items, Docu  
        continue;  
}  
  
13  
14 public final class SpecificationDescriptionRenderer implements  
15  
16 @  
17     public DocumentBuilderUtils.DocumentBuilderAction render(  
18         final Collection<DocumentBuilderUtils.DocumentBuilder  
19             processElement(description, Font.NORMAL, actionCollecto  
20             return paragraph(toArray(actionCollector, DocumentBu  
21  
22  
23  
24  
25  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41  
42  
43  
44  
45  
46  
47  
48  
49  
50  
51  
52  
53  
54  
55  
56  
57  
14 differences  
Deleted      Changed      Inserted
```

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.23). 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

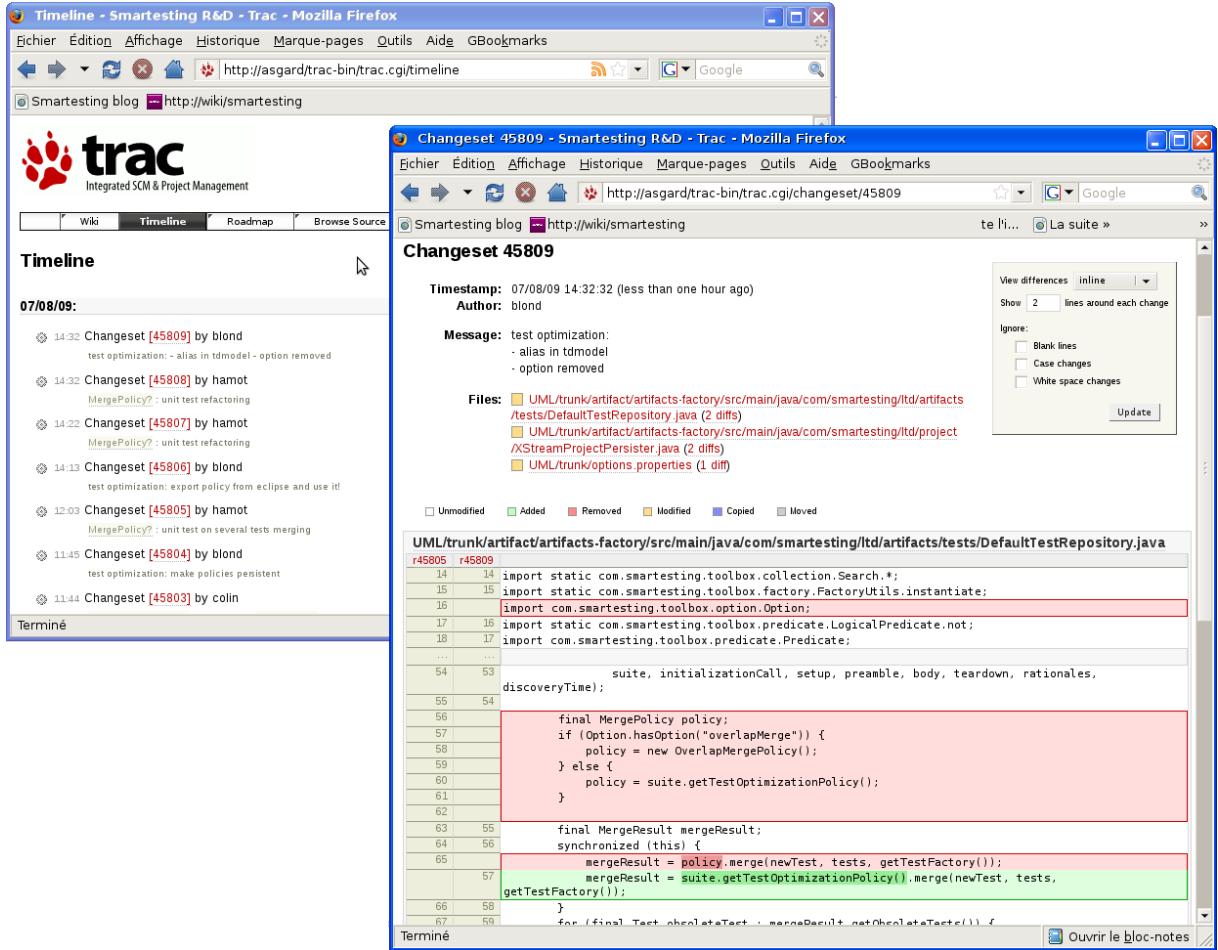


FIG. 2.2: Visualisation des révisions du code source avec TRAC

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.24) et haut niveau (cf. lexique A.11 p.24) ou encore la vitesse 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 vitesse 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'. Below the logo is a search bar and a link to 'ACTIVER LE RAFFRAICHISSEMENT AUTOMATIQUE'. 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. To the right, a large table displays the history of builds for various projects. The columns include 'S' (row number), 'W' (status), 'Job' (name), 'Dernier succès' (last success), 'Dernier échec' (last failure), and 'Dernière durée' (last duration). Each row also includes a small icon representing the project and a 'View' link.

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

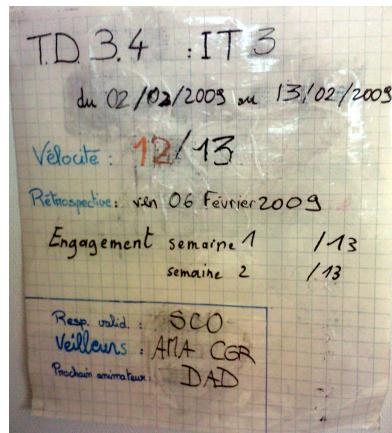


FIG. 2.4: Iteration

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

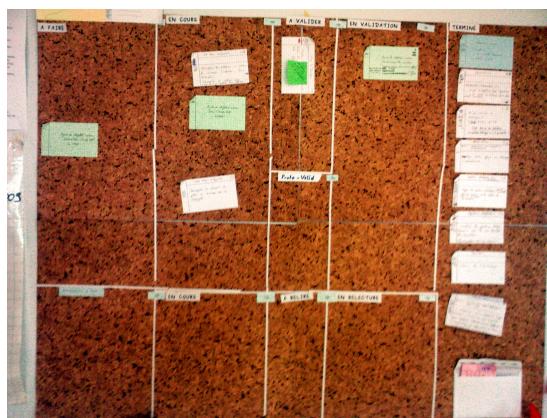


FIG. 2.6: Tableau d'avancement des fiches

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 Code auto-commenté

L'équipe R&D, afin d'avoir un code plus facile à comprendre et à maintenir s'efforce d'auto-commenter son code source. Un code source auto-commenté permet de mettre en avant la sémantique des objets et de leurs méthodes. Ceci permet d'avoir des commentaires explicites pour du code. Dans la mesure du possible on essaye d'avoir des portions de code qui tiennent sur une seule ligne (en particulier dans les tests unitaires). Ces lignes de codes peuvent presque être lues comme des phrases de langage courant dans certains cas. Le code auto-commenté repose sur certains principes de la programmation orienté objet. Le principe de responsabilité simple (un objet à une seule responsabilité) mis à part ses autres avantages rend le code plus lisible. Les imports statiques dans beaucoup de cas rendent le code plus lisible en évitant de mentionner à chaque appel l'origine complète d'un objet. Il est nécessaire d'avoir une code facile à comprendre pour pouvoir le remanier efficacement et s'adapter facilement aux changements.

2.8 Pomorodo

Un mois avant la fin de mon stage l'équipe a commencé à expérimenter la technique du pomodoro(cf. annexe C p.28 qui avait été présentée lors du meeting corporate le 10 avril. Cette technique consiste à découper la journée en sous-unités de temps (pomodoros) afin de garder le “focus” sur la tâche qu'on est en train de réaliser. Un pomodoro dure 25 minutes et pendant ce temps on se consacre uniquement à la tâche et on ne doit pas être interrompu. Une pause de 5 minutes permet de se changer les idées, prendre du recul et de décider si on continue sur la tâche, si on change de direction, ou si on arrête. J'ai participé activement à cette expérience en proposant et en introduisant pendant les pauses des animations liées au théâtre d'improvisation que je pratique régulièrement dans

```

public class EmfModelExtractor implements ModelExtractor {
    private final ExtractorDriver driver;

    public EmfModelExtractor(final ExtractorDriver driver) {
        this.driver = driver;
    }

    public ActionModelProvider extractModel(
        final ModelHandler modelHandler,
        final DescriptionRepository descriptionRepository,
        final EclipseIssueReporter issueReporter) {
        final Model model = modelHandler.getModel(Model.class);
        final ModelBuilder modelBuilder = createModelBuilder(model.getName());
        final Map<Class, ClassBuilder> classesToFactory = new TreeMap<Class, ClassBuilder>(NAMED_ELEMENTS_COMPARATOR);
        try {
            final DataExtractor extractor = new EmfDataExtractor(driver);
            final ModelStructureTranslator modelStructureTranslator = new ModelStructureTranslator(
                extractor, descriptionRepository, driver);
            modelStructureTranslator.translateRootPackage(
                model,
                modelBuilder.createRootPackage(extractor.extractUniqueIdentifier(model), "root"),
                classesToFactory,
                modelBuilder.getModel(),
                issueReporter);
            modelStructureTranslator.translateAssociations(classesToFactory, model, issueReporter);
            new StatechartsTranslator(descriptionRepository, driver).translateStatecharts(
                model, classesToFactory, issueReporter);
        } catch (ModelFactoryException e) {
            issueReporter.report(error(EMPTY_LOCATION, e.getMessageBundle()));
            return null;
        } catch (CancelTranslationException e) {
            issueReporter.report(error(EMPTY_LOCATION, e.getMessageBundle()));
            return null;
        }
        return new DefaultActionModelProvider(modelBuilder.getModel());
    }
}

```

FIG. 2.8: Exemple de code auto-commenté

le cadre d'un club de l'Association des Etudiant de l'UTBM.

2.9 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 vélocité estimée de la fiche.

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 en 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.

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.21 p.26) 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 Days¹. Les points techniques, au cours de mon stage n'ont plus eu de date précise mais étaient organisés selon les besoins de l'équipe.

¹conférence 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 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 7). 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.

3.1.1 Observations

Au début de mon stage, la première fonctionnalité sur laquelle j'ai travaillé fut des observations. Dans le produit, les observations sont des stéréotypes¹ attachés à des opérations de classes du modèle de test. Les observations sont déclenchées par des triggers² selon une pré-condition décrite dans le langage OCL(cf. lexique A.22 p.26). Dans les modeleurs, 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.

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 rendus compte, après avoir passé 4 points de vitesse sur cette fiche

¹stéréotypes au sens UML

²Événement déclencheur

cotée à 2, que c'était plus compliqué que nous l'imaginions. Nous avons alors suspendu la fiche en attente de conseils d'OBEO³. Finalement la après leur réponse, nous avons dû Rollback(cf. lexique A.19 p.26) 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.14) 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.14).

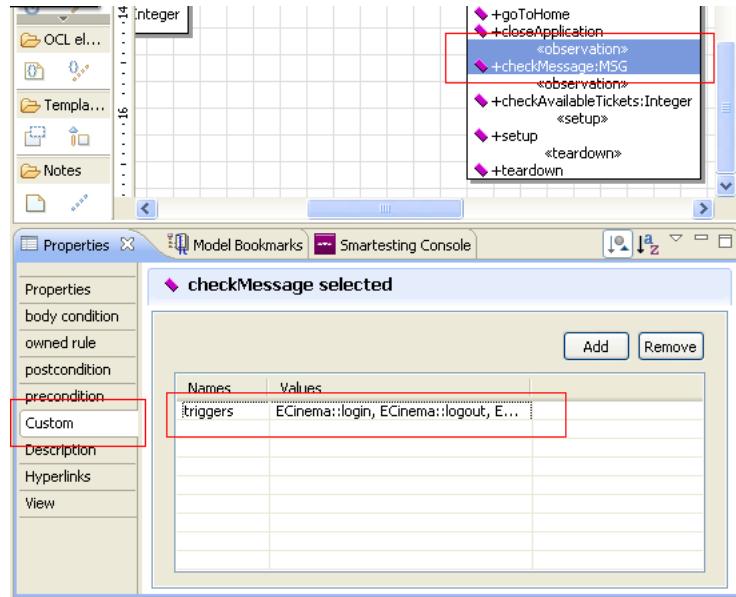


FIG. 3.1: Intégration des observations dans Together

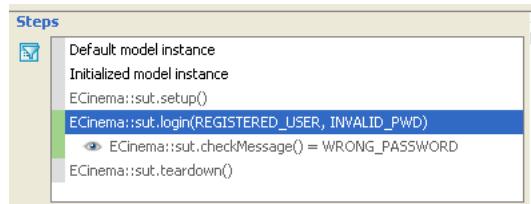


FIG. 3.2: Intégration des observations dans Test Designer

3.1.2 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 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.24) nous nous sommes aperçu que plusieurs publishers supportaient des balises HTML. Il était également possible de mettre du texte en forme en amont dans les modeleurs (cf. tableau

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

3.1 p. 15). Néanmoins certaines version 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. 15). 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 <i><u>
	RSM 7.0.0.x	texte brut
	RSM 7.0.5.x	HTML <i><u> + centrer + paragraphes
	RSM 7.5	HTML <i><u>+ centrer + paragraphes + autres
Publishers	HTML	
	Quality Center	texte brut et HTML <i><u>
	Specification (pdf)	<ins>
	Prototype RQM	HTML <i><u>

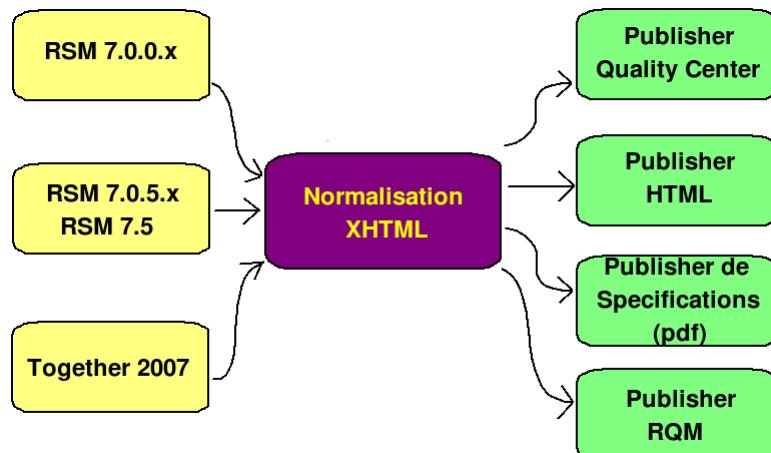


FIG. 3.3: Passage des descriptions des modeleurs à leur publication

3.1.3 Test Suites

Les suites de test permettent d'avoir une unité structurante pour tout un ensemble de test à partir des modeleurs. Elles définissent le périmètre des tests. Auparavant, une suite de test était considérée implicitement dans le modèle. Les différents packages⁴ contenant des informations relatives aux suites étaient analysés puis elles étaient personnalisées (filtre) dans le Target Manager.

⁴Équivalent d'un dossier pour stocker des ressources dans Eclipse dans le cadre de la modélisation

En fin du dernier jalon, guidé par des besoins de pouvoir remonter des informations des suites jusque là difficilement accessibles. Le client XP après plusieurs discussions avec l'équipe a décidé qu'il fallait externaliser les suites de test dans des fichiers de type texte. Cette modification de la structure des suite doit permettre d'être une étape de base pour rendre la solution Smartesting "verticale" afin de pouvoir par exemple s'intégrer à SAP. La solution choisie a été d'utiliser un fichier texte écrit dans un langage facile à écrire et à comprendre : Yaml⁵. Yaml est dit "user friendly" facile à lire et à comprendre pour un utilisateur.

```
# un commentaire et un tableau
users:
  - Toto
  - Titi

# utilisation de booléens
vrai: true
faux: false

# tableaux associatifs multidimensionnels
foo:
  toto: gentil
  titi: mechant
```

Fig. 3.4: *Exemple de fichier YAML*

```
--- !TestSuite
identifier: 4d517188-80af-4255-9405-abc367dcb5a7
initialModelInstance: initial_model_instance
version: "1.0"
```

Fig. 3.5: *Exemple de fichier YAML pour les TestSuite*

Malgré sa simplicité apparente, l'extraction des suite de test a été difficile. Tout d'abord l'équipe ayant peu de connaissances dans Eclipse sur certains domaines relatifs à cette fonctionnalité, il a fallu un peu de temps d'adaptation. Les différences entre les modeleurs ont rendu la tâche encore plus difficile. Plusieurs étapes ont été nécessaires lors de l'élaboration de cette fonctionnalité. Un fois le standard de format de fichier établi pour le suites de test, il a été possible de créer des outils permettant de prendre en compte ce fichier lors de l'extraction du modèle. Une interface graphique de type "wizard" a été créée pour facilier la saisie des données par l'utilisateur.

D'un autre côté un autre binôme travaillait sur gestion de l'écoute des fichiers de suites de test. Ceci a pour but de s'assurer que les données inscrites dans l'éditeur du fichier de

⁵"YAML Ain't Markup Language"; il s'agit d'un langage dédié à la serialisation (cf. lexique A.15 p.25)

suites de test sont bien mises à jours lors de la modification des noms de package, leur déplacement ou autre. Cette fonctionnalité étant difficile à mettre en place sans générer des familles de bugs génantes, elle a été abandonnée.

Les suites de test ont fait l'objet d'une release 3.4.1 peu de temps après la release majeure du jalon de la version 3.4. Cette livraison a eu lieu en début de mois de juillet.

3.2 Correction de bugs

La correction de bugs fait partie du travail des développeurs même si ce n'est pas forcément agréable. Toutefois les tests, la validation, l'intégration continue et les itérations courtes rendent la détection de bugs très rapide. La plupart des bugs sont en général détectés et corrigés avant la livraison en fin de semaine. Dans l'équipe on considère qu'il existe deux types de bugs. Les bugs client sont des bugs qui ont été remontés par les utilisateurs finaux après une release officielle. Les autres bugs qui sont détectés pendant l'itération ou après des releases mineures ne sont pas considérés comme des bugs client. Cette distinction est importante car, en adoptant la pratique "No Bugs" l'équipe s'engage à livrer un produit de qualité avec un minimum d'anomalies. L'équipe R&D s'est fixée comme objectif pour l'année d'avoir moins de 40 bugs client. Cet engagement influe sur la prime de l'équipe.

3.3 Documentation

Certaines fonctionnalités nécessitent d'être documentées dans le guide de l'utilisateur. La documentation de Test Designer est entièrement en anglais au format HTML. Les parties relatives à l'outil lui même et aux plugins dans les modeleurs peuvent être accompagnées de captures d'écran lorsque cela est nécessaire. Dans ce cas il arrive très souvent que la documentation soit dédoublée dans le cas des différents modeleurs. Même à part le guide de l'utilisateur, il est nécessaire de mettre à jour les release notes lors de la livraison d'un version destinée à des utilisateurs finaux. La documentation, une fois réalisée est soumise à une relecture par une autre personne.

3.4 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 fiche 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.

3.5 Amélioration du code existant (refactoring)

Le refactoring est une pratique courante dans les développements de l'équipe R&D de Smartesting. Il consiste à rendre le code plus facile à maintenir ou à lire sans pour autant modifier les fonctionnalités. Cela peut se faire de différentes manières. Le code peut être modifié en vue d'être auto-commenté (cf. 2.7 p.10). Il peut être découpé en d'autres méthodes et classes pour être plus facile à tester unitairement. Il peut aussi être déplacé dans d'autres packages.

Le refactoring a lieu dans le cadre de travail sur du code de production lors de fiches blanches. Il peut également être mis en pratique lors de réduction de dette de code⁶ pendant le "slack-time".

3.6 Prototypes

3.7 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 backup⁷.

3.8 Réunions et Meeting corporate

TODO : définition, utilité, ma participation ...

3.8.1 Visite de BNP Paribas

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

⁶La dette de code est un phénomène qui a lieu lorsqu'un code source produit devient de plus en plus dur à maintenir du fait de modifications successives ou d'un mauvais design. Ce phénomène est à éviter à tout prix car plus la dette de code est grande plus il est difficile de la réduire.

⁷sauvegarde automatique

3.8.2 Amélioration du process, évolution du fonctionnement de l'équipe

J'ai été ammené à l'occasion de retrospectives 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.

3.8.3 Evaluation

Tableau des évolution des pratiques XP

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

Chapitre 4

Conclusion

4.1 Connaissances acquises

4.1.1 Programmation JAVA

4.1.2 Test unitaires

(cf. ?? p.24)

4.1.3 Design de code

approche différente de spec en amont

4.1.4 Travail en équipe

Travail en Open-Space

4.1.5 Méthodes Agiles

Initiation à la méthode Agile en particulier à l'eXtreme Programming

4.2 Apport à l'entreprise d'accueil

4.3 Difficultés rencontrées

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.22) 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.17 p.25) 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. Il est très rare et peu souhaitable que la livraison d'une itération ne dégage pas de valeur pour les utilisateurs (en général testeurs).

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 généréricité (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 Serialisation

En informatique, la sérialisation (de l'anglais américain serialization, le terme marshalling est souvent employé de façon synonyme) est un processus visant à encoder l'état d'une information qui est en mémoire sous la forme d'une suite d'informations plus petites (dites atomiques, voir l'étymologie de atome) le plus souvent des octets voire des bits. Cette suite pourra par exemple être utilisée pour la sauvegarde (persistance) ou le transport sur le réseau (proxy, RPC...). L'activité symétrique, visant à décoder cette suite pour créer une copie conforme de l'information d'origine, s'appelle la déserialisation (ou unmarshalling).

A.16 IntelliJ IDEA

IntelliJ IDEA est un environnement de développement JAVA qui permet entre autres la refactorisation de code. Il est adapté aux pratiques agiles grâce aux support des serveurs de contrôle de version (en particulier fusion des fichiers avancée). Différents algorithmes rendent son utilisation pratique et agréable (copier/coller avec copie des imports, divers avertissements, formatage automatique de code, gestion des historiques subversion, ...)

A.17 Client XP

Appelé également client sur site, son rôle est de jouer l'interlocuteur au sein de l'équipe de développement. Il décide à chaque itération le fonctionnalité sur lesquelle

l'équipe va travailler. Par le biais de discussions avec les développeurs, il s'assure que ce qui est développé correspond exactement aux besoins. Le client XP vient en général de l'entreprise TODO ???

A.18 Commit

TODO

A.19 Rollback

TODO

A.20 Road-map

La road-map est une planification des tâches à effectuer pour les différents jalons à venir. Elle donne les directions que l'entreprise prend pour l'avenir. Elle est créée par la mission produit qui est constituée du directeur de production, du client XP, d'un représentant des consultants, d'un représentant de la R&D et d'un représentant des ventes. La road-map est élaborée lors d'un jeu de rôles (planning game). Et chaque fonctionnalité cotée gros grain (vision sur le moyen et long terme) est placée sur les différents jalons à venir. Ainsi une planification sur le moyen et long terme (vision sur 3-4 jalons) est faite.

A.21 Code contest

A.22 OCL

TODO : blabla OCL.

Annexe B

Bibliographie / Netographie

Wikipedia

http://fr.wikipedia.org/wiki/Méthode_agile

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://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 pomodoro¹ 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)

¹minuterie 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:

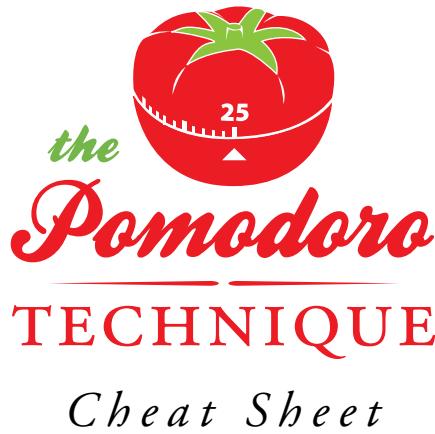
- ❖ Choose the topmost task from the list
- ❖ Set the Pomodoro to 25 minutes
- ❖ Work until the Pomodoro rings
- ❖ Mark the task with an x on the To Do SHEET
- ❖ Take a short break (3–5 minutes)

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

Mots clefs

Agile - eXtreme Programming - JAVA - Test logiciel - Eclipse - Programmation orientée objet - Génie logiciel - Test Driven Developpment

BOUVIER Marc

Rapport de stage ST40 - P2009

Résumé

Mon stage a eu lieu chez Smartesting qui industrialise le test logiciel par une approche basée sur la modélisation. J'ai évolué dans l'équipe de recherche et développement pour participer au développement en JAVA de l'outil de génération de test Test Designer et de ses modules annexes. Cette équipe pratique les méthodes de développement Agiles : eXtreme Programming (XP). L'équipe est très soudée et la communication entre ses membres est très forte. J'ai pu découvrir des techniques de programmation telles que Test Driven Développement, refactoring. Des bonnes pratiques de programmation objet comme le principe de responsabilité simple. Les processus de développement ont varié tout au long de mon stage prouvant qu'il est souvent nécessaire le remettre en question. J'ai découvert des pratiques de l'agilité telles que le "done-done", "no-bugs", "slack-time". J'ai participé à la mise en place d'un système de gestion du temps (pomodoro technique). J'ai pu assister à des meetings regroupant tous les collaborateurs de l'entreprise, découvrir ainsi le fonctionnement d'une entreprise au delà des considérations techniques.

Entreprise SMARTESTING

TEMIS Innovation - 18 Rue Alain Savary
25000 Besançon
www.smartesting.com