Approche Agile du Model-Based Testing pour les tests fonctionnels des SI d'entreprise

Elodie Bernard

THÈSE DE DOCTORAT

DE L'ÉTABLISSEMENT UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ

PRÉPARÉE À L'UNIVERSITÉ DE FRANCHE-COMTÉ

École doctorale n°37

Sciences Pour l'Ingénieur et Microtechniques

Doctorat d'Informatique

par

ÉLODIE BERNARD

Approche Agile du Model-Based Testing pour les tests fonctionnels des SI d'entreprise

Thèse présentée et soutenue à Besançon, le 23 mars 2021

Composition du Jury :

BLANC Xavier	Professeur des Universités, Université de Bordeaux	Rapporteur
GROZ Roland	Professeur des Universités, Université de Grenoble	Rapporteur
BOUQUET Fabrice	Professeur des Universités, Université de Franche-Comté	Président du jury
KRAMER Anne	Docteur-Ingénieur, sepp.med GmbH	Examinateur
LEGEARD Bruno	Professeur des Universités, Université de Franche-Comté	Directeur de thèse
AMBERT Fabrice	Maître de conférence, Université de Franche-Comté	Co-encadrant de thèse

N° | X | X | X |

REMERCIEMENTS

Tout d'abord, je voudrais exprimer mes sincères remerciements à mon directeur de thèse, le professeur Bruno Legeard et mon co-directeur, le maitre de conférences Fabrice Ambert pour leur soutien continu durant cette thèse, leur contribution et leurs vastes connaissances et expertises. Leurs enseignements m'ont aidée tout au long de cette thèse.

Je remercie ensuite Xavier Blanc et Roland Groz qui ont accepté d'être les rapporteurs de cette thèse et qui ont étudié avec attention mon travail.

Je remercie aussi Fabrice Bouquet et Anne Kramer qui m'ont fait l'honneur de faire partie de ce jury et accepter de juger les travaux réalisés durant cette thèse

Mes sincères remerciements vont également à tous les collaborateurs de Sogeti qui ont rendu la réalisation de cette thèse possible.

Un grand merci également à tous les membres de Smartesting qui ont toujours prêté attention à mes remarques et réflexions et avec lesquelles j'ai pu travailler en synergie pour proposer des approches adaptées à l'environnement industriel, soutenues par leurs outils.

Je remercie aussi les personnes avec qui j'ai pu travailler durant ces trois années. Sans leur précieux soutien et leur participation aux expérimentations, il n'aurait pas été possible de mener à bien ces recherches. Je pense tout particulièrement à Laura Prouteau qui s'est particulièrement investie dans les expérimentations et qui m'a apporté son soutien continu.

SOMMAIRE

TABLE DES FIGURES

I

CONTEXTE ET MOTIVATIONS

1

PROBLÉMATIQUES ET POSITIONNEMENT DE LA THÈSE

Ce chapitre présente les problématiques et le positionnement de la thèse. La première section présente les enjeux des tests logiciels dans la transformation digitale et Agile des grandes organisations. La seconde section traite du cadre industriel de cette thèse. Les sections qui suivent détaillent les problématiques de recherche, ainsi que les contributions et publications de la thèse. Pour finir, nous détaillons l'organisation de l'ensemble de ce manuscrit.

1.1/ ENJEUX DES TESTS LOGICIELS DANS LA TRANSFORMATION AGILE

La transformation digitale et Agile des grandes organisations induit un besoin d'évolution en profondeur des pratiques des tests logiciels [28]. La mise en production des évolutions des systèmes informatiques à un rythme de plus en plus rapide, sur des systèmes de plus en plus complexes, remet en cause les pratiques traditionnelles en particulier pour les tests fonctionnels qui constituent le focus de cette thèse. Ces pratiques sont fondées sur une forte composante de tests manuels, et une maîtrise de la couverture des tests peu systématique [63, 32]. Un enjeu des tests logiciels dans la transformation Agile se trouve dans la capacité des approches et outils de test à s'adapter aux courtes itérations de l'Agile. Ils doivent aussi parvenir à gérer l'expansion des systèmes développés constitués d'un ensemble de produits, sous-systèmes et composants, dans des contextes d'Agilité à l'échelle [1], cela tout en garantissant la qualité des systèmes en production.

Pour garantir cette qualité, les approches et outils de test doivent permettre d'assurer la couverture des exigences à différents niveaux : les tests doivent permettre de vérifier

1. On appelle "Agilité à l'échelle" les contextes organisationnels ou plusieurs équipes en Agile contribuent en parallèle à un même système complexe.

indépendamment chacun des composants, produits et sous-systèmes. Mais les tests doivent aussi permettre de vérifier le fonctionnement transverse de la solution métier. De plus, pour garantir la qualité des tests dans le temps, il est important de mettre en œuvre des mécanismes de qualité pour la conception, la maintenance et l'exécution des tests.

L'adaptation à l'Agilité signifie que les approches et outils devront être efficaces dans la conception pour différents objectifs de couverture de tests, tels que tester un petit ensemble de User Story [2] ou maintenir opérationnel un ensemble de tests de bout en bout en reprenant les principaux cas d'utilisation de la solution métier. Ceci de manière à gérer les évolutions du système dans sa globalité et étendre le référentiel de tests de non-régression.

Un autre enjeu des tests logiciels dans la transformation Agile est de parvenir à assurer la maintenance des suites de tests manuels [3], grandissantes au fil des itérations. Ces suites de tests manuels sont généralement décrites en langage naturel et, avec le temps, deviennent désorganisées et plus difficiles à utiliser et à maintenir. Il existe là un réel défi à proposer une approche outillée permettant de maintenir ces suites de tests manuels afin de les rendre plus utilisables et plus faciles à maintenir, dans des contextes où il faut être en mesure d'agir rapidement et efficacement. Enfin en terme d'exécution des tests, les approches et outils devront supporter l'exécution de tests manuels et automatisés avec une adaptation transparente des premiers aux seconds. L'exécution systématique de l'ensemble des tests manuels n'est pas possible d'où la forte nécessité d'automatiser l'exécution des tests. La mise en place de processus d'automatisation est un point clé dans la réussite des projets, mais cette activité reste actuellement complexe et coûteuse [88]. Elle permet pourtant de garantir la qualité de l'application dans les différentes itérations, notamment dans les contextes Agile, où les livraisons sont régulières et où le temps nécessaire pour tester manuellement croît au fil des itérations. Automatiser l'exécution des cas de test [3] permet d'alléger la charge de tests manuels et facilite la détection de régressions.

Depuis plus de deux décennies, les approches du test à partir de modèle - appelé Model-Based Testing (MBT) - ont émergé pour assurer un support de bout-en-bout aux activités de test, dans un continuum allant de la capture des spécifications à la génération automatique des cas de test et des scripts de test automatisés [4] [92, 26, 95]. Mais les enquêtes sur les pratiques du test montrent de façon récurrente la très faible adoption du Model-Based Testing par les testeurs fonctionnels en place dans les organisations, sans progression significative ces dernières années [45, 46, 25]. C'est particulièrement le cas pour le test des systèmes d'information et du logiciel métier d'entreprise qui est le

2. Nous avons conservé en anglais le terme "User Story", qui désigne une expression de besoin à un niveau de granularité fin, car c'est un usage courant en milieu industriel.

3. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

4. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

domaine visé par cette thèse.

Etablir une approche du Model-Based Testing adaptée aux tests fonctionnels en Agile des applications d'entreprise a constitué la motivation principale de notre thèse réalisée dans un cadre industriel que nous décrivons dans la section suivante.

1.2/ CADRE INDUSTRIEL DE LA THÈSE

La thèse présentée dans ce manuscrit a été réalisée au sein de l'institut FEMTO-ST - UMR 6174 et de la société Sogeti dans le cadre d'une thèse CIFRE démarrée en novembre 2017.

Sogeti est une société de service en informatique, composante du groupe Capgemini, qui possède une position de leader dans le domaine des tests logiciels. C'est par exemple la première société qui a proposé et documenté une approche globale des tests avec TMap au début des années 2000, et l'a régulièrement adaptée depuis avec TMap Next [43] et TMap HD [13].

Au niveau mondial, Sogeti possède plusieurs pôles de recherche et d'innovation, et met aussi en oeuvre une recherche partenariale avec le tissu académique, dont cette thèse CIFRE constitue un exemple. La thèse a été réalisée au sein du pôle innovation de Lyon, en étant dans un premier temps rattachée au centre de services SNCF de Sogeti Rhône-Alpes puis ensuite aussi en lien avec des projets liés à des clients de Sogeti dans le domaine de la banque et assurance au niveau national.

Les activités réalisées durant cette thèse se sont ainsi imprégnées de l'analyse des difficultés rencontrées au quotidien par les équipes de test dans les différentes activités de conception, d'implémentation et d'automatisation des tests au sein de plusieurs projets conduits pour des clients de Sogeti, en particulier dans le contexte d'une évolution vers l'agilité de ces grandes organisations.

Le résultat de ces activités de recherche industrielle a été la proposition et l'évaluation d'une approche du Model-Based Testing pour les tests fonctionnels des applications des grands systèmes d'information d'entreprise, dans des contextes organisationnels évoluant vers l'agilité. Ce focus méthodologique a été facilité par le partenariat qui s'est mis en place entre l'éditeur Smartesting Solutions Services (éditeur des supports outillés de la thèse) et Sogeti. Cette collaboration nous a permis de nous focaliser sur la méthodologie, la prescription pour l'outillage et la validation des approches élaborées dans le contexte des projets qui ont fourni le contexte expérimental de la thèse.

Nos résultats ne sont cependant pas liés à un outillage particulier, car comme nous l'explicitons dans le chapitre 3, l'approche proposée peut être implémentée avec différents

outils du marché. Mais grâce à ce partenariat entre Sogeti et Smartesting, nous avons pu évaluer les différents aspects de l'approche étudiée et de l'outillage associé de façon concrète et dans des contextes clients variés avec pour objectif d'en éprouver la capacité à être applicable en réel avec les équipes existantes. Il s'agissait aussi pour nous de faire le lien entre la recherche en Model-Based Testing et la pratique industrielle en proposant des approches novatrices et applicables sur les projets traités par Sogeti.

1.3/ OBJECTIFS ET PROBLÉMATIQUES DE RECHERCHE DE LA THÈSE

Les objectifs de la thèse s'inscrivent dans le contexte des grands systèmes d'information d'entreprise s'organisant selon des cycles de vie itératifs et incrémentaux du développement logiciel, typiques de l'Agilité et de l'Agilité à l'échelle. Un des objectifs de la thèse est de définir une approche du Model-Based Testing permettant un processus systématique des tests fonctionnels de ces grands systèmes d'information à partir de modélisations légères des processus métier et des règles de gestion à tester.

Cette approche du Model-Based Testing et les modélisations associées doivent satisfaire plusieurs critères pour répondre à ces objectifs, notamment :

- être utilisables par des praticiens, testeurs fonctionnels de profession ainsi que d'autres parties prenantes du projet, tels les analystes métier et les Product Owners [5]. Ceci dans l'objectif de favoriser la communication autour des besoins métier et de faciliter les interactions entre parties prenantes.
- faciliter la conception et l'implémentation des cas de test lors d'itérations courtes (de 2 à 4 semaines) de développement du logiciel, grâce à des bonnes pratiques de modélisation et de gestion des modèles.
- faciliter l'automatisation des tests, par une transcription simplifiée des cas de test manuels vers des scripts de tests automatisés en facilitant la collaboration entre les testeurs fonctionnels et les automaticiens de test.
- permettre l'intégration de l'automatisation des tests dans les itérations par des approches et bonnes pratiques de construction des mots-clés [6] et de gestion des données de test.

Pour répondre à ces besoins, nos contributions ont visé à étendre l'état de l'art en Model-Based Testing pour le test des systèmes d'information d'entreprise, sur l'automatisation

5. Nous avons choisi de conserver ce terme en anglais car c'est cet usage que nous rencontrons sur les projets.
6. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

des tests dans ce contexte et sur le test dans l'Agilité et l'Agilité à l'échelle. Ainsi nous avons défini et expérimenté dans le contexte de plusieurs projets du centre de services, une approche baptisée ALME - *Agile Lightweight Model-Based Testing for Enterprise IT*[7]. L'approche ALME adapte les concepts et pratiques du MBT avec les objectifs suivants :

1. Simplifier la notation de modélisation MBT pour la rendre facilement utilisable par les testeurs fonctionnels dans le domaine des logiciels des systèmes d'information d'entreprise. Cette notation doit permettre de scénariser les tests en formulant graphiquement les parcours applicatifs [7] à tester, et faciliter le partage et la discussion de ces représentations graphiques avec les parties prenantes orientées métier (Product Owner, analystes métier) du projet.

2. Adapter le MBT aux approches de développement itératives et incrémentales. L'adaptation à l'agilité signifie que le MBT devrait être efficace pour différents objectifs de couverture de tests, tels que le test d'une User Story, la conception de tests de bout en bout reprenant les principaux cas d'utilisation des applications, ou permettre de reproduire un scénario d'utilisation spécifique qui crée des problèmes en production. Par conséquent, l'approche de MBT et l'outillage associé doivent être très flexibles et ne jamais nécessiter une modélisation complète de l'application pour soutenir la production de cas de test.

3. Supporter l'exécution de tests manuels et automatisés avec une adaptation transparente du premier au second. L'exécution manuelle des tests d'acceptation par les testeurs pendant l'itération agile est généralement la première étape, puis l'équipe de test doit étendre le référentiel de tests de non-régression. L'outillage MBT doit à la fois soutenir l'utilisation et faciliter/automatiser l'adaptation des cas de test manuels vers les scripts de tests automatisés. Cette approche devra fournir un support itératif aux activités d'automatisation, en particulier pour faciliter la collaboration entre les testeurs fonctionnels et les automaticiens de test.

Un autre objectif a découlé de notre analyse de l'état des pratiques des tests en Agile : nous avons constaté qu'il n'existait que peu d'étude sur les tests dans l'Agilité à l'échelle notamment sur les tests inter-équipes du point de vue de la pratique de terrain. Cela s'explique par le caractère récent de la mise en place des méthodes de l'Agilité à l'échelle telles que SAFe [8], Spotify [9], Scrum of Scrums [10] ou LESS [11]. Un objectif de notre travail de thèse a donc été d'analyser les pratiques actuelles en test dans l'Agilité à l'échelle pour intégrer cet aspect des tests inter-équipes dans l'approche ALME.

7. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1.
8. Scaled Agile Framework, https ://www.scaledagileframework.com/
9. Spotify, https ://engineering.atspotify.com/
10. Scrum of Scrums, https ://www.scrum.org/
11. Large-Scale Scrum, https ://less.works/

Enfin, le dernier aspect couvert par nos travaux de thèse concerne l'analyse des référentiels de test et leur *refactoring*[12]. Lors de nos recherches sur les pratiques en test dans l'Agilité nous avons observé que le sujet de la gestion des suites de tests manuels dans le temps et leur vieillissement constitue une problématique difficile en pratique, mais sans proposition de solution pour gérer et améliorer itérativement ces référentiels de test grandissants. Notre objectif dans cette thèse pour ce sujet a été de proposer et d'évaluer une approche pour le refactoring des tests manuels visant à en réduire le coût de maintenance en améliorant la qualité structurelle.

Dans la section qui suit, nous présentons la synthèse des contributions réalisées dans le cadre de cette thèse ainsi que la liste des publications qui y sont associées.

1.4/ CONTRIBUTIONS ET PUBLICATIONS DE LA THÈSE

Les contributions de ces travaux de recherche répondent aux problématiques et objectifs de recherche défini précédemment en quatre points :

- Par l'élaboration de l'approche ALME en termes de processus, de bonnes pratiques et d'activités des rôles impliqués, ainsi que son expérimentation et évaluation en contexte projet.
- Par la définition et l'évaluation d'une approche d'automatisation des tests dans le contexte de l'approche ALME.
- Par la définition et l'évaluation d'une approche de refactoring des suites de tests manuels en lien avec l'approche ALME.
- Par l'analyse de l'état des pratiques en test dans l'Agilité à l'échelle au travers d'une étude qualitative et quantitative.

Ces contributions ont été contextualisées dans le domaine industriel quand cela a été possible et ont fait l'objet de plusieurs publications, pour certaines dans des workshops et conférences académiques, et pour d'autres avec un objectif de dissémination de nos travaux vers les praticiens des tests logiciels. Nous structurons cette liste de publications en 3 parties :

- Les publications relatives à la définition et l'évaluation de l'approche ALME dans plusieurs contextes de projet, en incluant la gestion de l'automatisation.
- Les publications relatives au refactoring des tests manuels.
- Les publications relatives à l'analyse des pratiques de test dans l'agilité à l'échelle.

12. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

1.4.1/ PUBLICATIONS RELATIVES À L'APPROCHE ALME

Ces publications concernent à la fois les aspects clés de l'approche, en particulier du point de vue de la modélisation, et les pratiques collaboratives en Agile de l'ATDD visuel [13] liées à l'approche ALME.

- *Elodie Bernard, Fabrice Ambert, Bruno Legeard and Arnaud Bouzy, Lightweight Model-Based Testing for Enterprise IT, 2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), A-MOST - 14th Workshop on Advances in Model-Based Testing, Vasteras, Suède, 2018, pp. 224-230, doi :10.1109/ICSTW.2018.00053.*
  Dans cet article, nous présentons une approche légère de MBT et un outil, appelé Yest, dédié au test des systèmes d'information des entreprises et basé sur les processus métier. Cet outil permet de représenter graphiquement des processus métiers et de les animer au moyen de tables de décisions. La prise en main de l'outil par les testeurs fonctionnels ne nécessite aucune compétence lourde de modélisation, comme UML par exemple.

- *Elodie Bernard, Fabrice Ambert and Bruno Legeard, A Lightweight MBT Approach for Visual Acceptance Test Driven Development, QRS 2019, 19th IEEE International Conference on Software Quality, Reliability and Security, July 2019, Sofia, Bulgaria. (hal-02867886).*
  Dans cet article, nous introduisons et positionnons le concept de l'ATDD (Acceptance Test Driven Development) visuel, expérimenté sur de larges projets informatiques des systèmes d'information. Les enjeux abordés sont de parvenir à mettre en oeuvre une approche collaborative de scénarisation des tests qui s'adapte aux itérations courtes des contextes Agile tout en répondant aux challenges de créer et maintenir de manière efficiente des scripts de tests automatisés.

- *Elodie Bernard, Processus de test fondé sur la conception visuelle des tests : Retour d'expérience sur un projet en Agile, Chapitre de livre « Les tests en Agile », 8 pages – ISBN : 978-2956749004, Editeur Comité Français des Tests Logiciels, Avril 2019.*
  Dans ce chapitre de livre, nous présentons un processus de test fondé sur la conception visuelle des tests à travers un retour d'expérience sur un projet en Agile. Nous détaillons chaque phase de notre approche et la manière dont nous l'avons mise en application dans un contexte industriel de projet Agile.

- *Elodie Bernard, Laura Prouteau, Gestion simple et visuelle des exigences via des représentations graphiques, JFIE 2019, Journée Française d'Ingénierie des Exi-*

13. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

gences[14], Novembre 2019, Paris, France.
Dans cette conférence, nous avons présenté comment réaliser une gestion simple et visuelle des exigences en lien avec la scénarisation des tests, illustrée par des expérimentations réalisées en contexte projet.

La définition et la mise en application de l'automatisation des tests dans ALME a fait l'objet des publications et disséminations suivantes :

- *Elodie Bernard, Fabrice Ambert and Bruno Legeard, Supporting Efficient Test Automation using Lightweight MBT, IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW), 3rd edition of INTUITEST-BEDS (International Workshop on User Interface Test Automation and Testing Techniques for Event Based Software), Porto, Portugal, 2020, pp. 84-94, doi : 10.1109/ICSTW50294.2020.00028.*
  Cet article montre comment nous avons expérimenté une approche légère d'automatisation des tests basée sur des modèles pour répondre à la fois aux défis de la productivité et de la pertinence (c'est-à-dire leur alignement sur les besoins des entreprises) des scripts de tests automatisés. Elle intègre l'automatisation des tests par un processus et une chaîne d'outils simples, expérimentés sur de grands projets informatiques.

- *Elodie Bernard, Experience Report : Visual Test Design for Test Automation in Agile of Large-Scale IT Systems, Novembre 2018, Conference UCAAT (User Conference on Advanced Automated Testing), Paris, France, https ://ucaat.etsi.org/*
  Cette conférence présente un retour d'expérience sur la manière dont nous avons expérimenté une approche légère d'automatisation des tests basée sur des modèles sur de grands projets de systèmes d'information.

1.4.2/ PUBLICATIONS RELATIVES AUX TRAVAUX SUR LE REFACTORING DES TESTS MANUELS

Note approche du refactoring des suites de tests manuels a donné lieu à trois publications dont une lors de la conférence ICST 2020 :

- *Elodie Bernard, Julien Botella, Fabrice Ambert, Bruno Legeard and Marc Utting, Tool Support for Refactoring Manual Tests, IEEE 13th International Conference on Software Testing, Validation and Verification (ICST) - Industry Track, Porto, Portugal, 2020, pp. 332-342, doi : 10.1109/ICST46399.2020.00041.*
  Nous décrivons dans cet article comment nous avons appliqué diverses techniques

14. www.cftl.fr/JFIE/accueil/

de machine-learning et de NLP (ainsi que d'autres algorithmes) pour le remaniement de suites de tests manuels. Nous présentons la manière dont nous intégrons ces techniques dans un outil permettant d'explorer et de visualiser les suites de tests. Nous proposons aussi un processus de remaniement que nous évaluons sur plusieurs suites de tests manuels de l'industrie et rendons compte des gains de temps qui ont été obtenus.

- *Elodie Bernard, Arnaud Bouzy, Reprise et optimisation d'un patrimoine de tests manuels – Une approche visuelle fondée sur l'IA, JFTL 2019 – Journée Française des tests logiciels, Paris, France, http ://www.cftl.fr/JFTL/accueil/*
  Dans cette conférence nous présentons les premiers travaux et expérimentations que nous avons menés avec un outil pour le remaniement et visualisation de suites de tests manuels. Nous présentons les résultats d' expérimentations menées sur des suites de tests manuels de l'industrie dans le but d'éprouver la solution et d'être en capacité de présenter un processus de remaniement des suites de test.

- *Elodie Bernard, Retour d'expérience sur le refactoring des tests avec l'outil Orbiter, La taverne du testeur, https ://latavernedutesteur.fr, mars 2020*
  Dans cet article, nous présentons un retour d'expérience sur le refactoring des tests avec l'outil Orbiter. Nous présentons les diverses fonctionnalités de l'outil et les résultats que nous avons obtenus pour le refactoring de suites de tests manuels de l'industrie.

1.4.3/ PUBLICATIONS RELATIVES À L'ÉTUDE SUR LES TESTS INTER-ÉQUIPES DANS L'AGILITÉ À L'ÉCHELLE

Enfin les résultats de notre étude qualitative et quantitative sur les tests dans l'Agilité à l'échelle ont été présentés dans le cadre de trois disséminations industrielles :

- *Elodie Bernard, Claude Barreau, Fabrice Grimbert Testing in SAFe® : Coordinate and optimize test efforts with visual ATDD - Experience report at Orange, Conference UCAAT (User Conference on Advanced Automated Testing), Octobre 2019, Paris, Bordeaux, https ://ucaat.etsi.org/*
  Dans cette conférence, nous montrons comment nous coordonnons et optimisons l'effort de test avec de l'ATDD visuel basé sur des représentations graphiques. Nous présentons nos expérimentations menées dans l'industrie au sein d'Orange.

- *Elodie Bernard, Tests logiciels et Agilité à l'échelle, La taverne du testeur https ://latavernedutesteur.fr, aout 2020*

Dans cet article, nous présentons nos premiers constats en lien avec notre étude sur les tests dans l'Agilité à l'échelle. L'étude a pour objectif de mieux comprendre les pratiques en test notamment au niveau inter-équipes dans des contextes d'Agilité à l'échelle par une étude quantitative sous forme d'enquête et qualitative sous forme d'interviews.

- *Elodie Bernard, Enquête sur les tests dans l'agilité à l'échelle, JFTL 2020 – Journée Française des tests logiciels. http ://www.cftl.fr/JFTL/accueil/*
  Dans cette conférence, nous présentons une partie de nos résultats de notre enquête sur les tests dans l'Agilité à l'échelle. Nous avons extrait un panel de réponses obtenues lors de l'enquête et fait le parallèle avec les éléments issus des interviews de notre analyse qualitative. Nous avons choisi de présenter les résultats directement en lien avec les activités menées dans le domaine industriel, car les participants à cette conférence sont majoritairement issus du domaine industriel. Ainsi nous présentons un ensemble de constats et pratiques identifiés au niveau inter-équipes dans des contextes d'Agilité à l'échelle.

1.5/ ORGANISATION DU MANUSCRIT

La thèse s'organise en 3 parties et est divisée en 8 chapitres.

La partie I, "Contexte et motivations" comprend 2 chapitres. Le présent chapitre introduit cette thèse en présentant les enjeux des tests logiciels dans la transformation Agile, suivi du cadre industriel de la thèse. Nous présentons ensuite les objectifs et questions de recherche de la thèse et les contributions et publications qui y sont liés. Le chapitre 2 présente l'état de l'art, avec 4 sections dédiées respectivement au Model-Based-Testing, l'Agilité et l'Agilité à l'échelle, le test refactoring et l'automatisation des activités de test.

La partie II traite des contributions techniques et des expérimentations. Elle se divise en 4 chapitres. Le chapitre 3 présente notre approche ALME. Elle est divisée en 6 sections. Après la section d'introduction de notre approche, nous détaillons notre processus d'ATDD visuel par la scénarisation des tests, puis les rôles associés à celui-ci ainsi que les artefacts qui le composent. Nous présentons ensuite nos expérimentations et apportons une conclusion à ce chapitre. Le chapitre 4 présente notre processus d'automatisation qui compose notre approche ALME. Il se divise en 5 sections où nous décrivons en premier lieu ledit processus. Ensuite, nous détaillons les étapes qu'il contient notamment la complétion de la couche d'adaptation suivie de la production des scripts de tests automatisés. Nous illustrons ensuite nos propos par des expérimentations et concluons ce chapitre. Le chapitre 5 présente notre approche de reprise des suites de tests manuels en 3 sections. La 1ere est consacrée à la présentation de notre approche, la 2eme à nos

expérimentations puis nous concluons ce chapitre. Le chapitre 6 présente notre contribution sur les tests dans l'Agilité à l'échelle divisée en 4 sections. Nous commençons par détailler notre analyse qualitative, puis quantitative, ensuite nous procédons à la synthèse et la formulation de réponses et enfin nous concluons.

Enfin, la partie III vise la conclusion et les perspectives de cette thèse.

En annexe, des compléments sont apportés :

- L'annexe A traite de la terminologie, avec d'une part un glossaire des termes utilisés de façon spécifique dans l'approche ALME, et d'autre part un glossaire qui reprend la définition normalisée dans ISO/IEEE/IEC 29119-1 ou dans le glossaire de l'ISTQB [15] de certains termes lorsque cela est utile pour la thèse.
- Les annexes B et C apportent des compléments pour les expérimentations des chapitres 3 et 4.
- L'annexe D présente des données détaillées de l'enquête que nous avons mené sur les tests dans l'agilité à l'échelle.

15. ISTQB - International Software Testing Qualification Board - Il s'agit d'une association professionnelle internationale qui défini un schéma de certification des compétences des testeurs. Le glossaire des termes des tests logiciels de l'ISTQB constitue une référence de la profession du point de vue de la terminologie.

2

ÉTAT DE L'ART

Ce chapitre présente notre état de l'art sur les quatre axes de recherche qui composent la thèse. La première section traite de l'état de l'art en Model-Based Testing. La seconde présente l'Agilité et l'Agilité à l'échelle. La troisième concerne le refactoring des tests et la dernière l'automatisation de l'exécution des tests.

2.1/ MODEL-BASED TESTING

Le Model-Based-Testing (MBT) est une technique de test visant à représenter un système sous forme de modèles pour permettre la génération et l'exécution des tests [26]. C'est un champ de recherche largement abordé dans la littérature scientifique. Apparu dans les années 70, il n'a cessé d'évoluer depuis. Cela est visible par le grand nombre de publications existant dans ce domaine et des nombreuses techniques de MBT proposées [26, 3, 31, 98, 49]. Les techniques de MBT sont souvent utilisables à travers des supports outillés. Ainsi ces outils permettent aux industries qui choisissent le MBT de vérifier leurs systèmes. Comparé aux nombres d'articles proposant des approches de MBT, il existe peu de ressources scientifiques évaluant l'application du MBT dans l'industrie [26]. Ce nombre limité de retours d'expériences peut s'expliquer par le fait que chaque contexte industriel à ses propres caractéristiques et une approche appliquée dans un cadre industriel particulier ne s'appliquera pas forcement dans un autre [3]. Il existe donc aujourd'hui un réel challenge à étudier comment le MBT s'inscrit dans l'industrie. Car malgré le nombre de techniques et outils de MBT existants, ils sont peu diffusés dans l'industrie. Le MBT User Survey 2019 [46] montre un relatif affaiblissement de l'intérêt par rapport à la même enquête réalisée en 2014. Cette faible dissémination s'explique en partie par la faible adaption des techniques et outils de MBT au cycle de développement logiciel qui est devenu majoritairement en Agile, car la complexité des modèles utilisés en MBT devient un frein lors de cycles de développement court, de type itératif et incrémental [31, 3]. Pour répondre à ce besoin lié aux pratiques de l'Agilité, le MBT connait donc depuis les années 2000 une évolution constante en tendant de plus en plus vers sa simplification

[98, 49].

Les sections qui suivent présente l'historique du MBT d'une part du point de vue académique et d'autre part industriel. Nous nous focalisons en particulier sur la notation BPMN (Business Process Modeling Notation) qui est une technique couramment utilisée dans le domaine du test des logiciels d'entreprise et du Système d'information qui est notre cible pour cette thèse. Nous étudions aussi comment le MBT s'inscrit dans les modes de développements en Agile dans la section 2.2.

2.1.1/ MODEL-BASED TESTING DANS LA LITTÉRATURE

HISTORIQUE

Paul C. Jorgensen dans le livre "The craft of model-based-testing" [38] définit le MBT comme "l'utilisation de modèles afin de tester un ensemble de stimuli et de réponses que l'on souhaite tester sur un système sous test (SUT) [1]". Cette définition se prête bien aux premières recherches sur le sujet qui ont débutées dans les années 70. Les travaux, en 1976, de Ramamoorthy [74] décrivent une technique de génération de données de test. Dans cette approche, pour un programme FORTRAN donné, un ensemble de chemins peut être identifié comme satisfaisant à certains critères de test. À partir de là, une exécution symbolique permet la génération d'un ensemble d'entrées pour reproduire les chemins possibles du programme.

En 1978, les laboratoires Bell proposent une approche fondée sur une modélisation par machines à états finis (FSM) [20]. Depuis ces 1^{ers} travaux, les stratégies, les techniques et la définition même du MBT ont évoluées. De la fin des années 70 et la fin des années 90 des techniques de MBT se sont développées. Dalal en 1999 [23] définit le MBT comme "une technique nouvelle et évolutive permettant de générer une série de cas de test à partir des exigences", ces exigences étant les critères de test que l'on inscrit dans les modèles sous différentes formes. C'est en 2007 que Dias Neto présente une vue d'ensemble des évolutions qui ont eu lieu dans le domaine [6]. Il identifie près de 219 approches différentes de MBT. Il présente un ensemble de 7 champs permettant de caractériser les différentes approches de MBT et de les comparer. Ces champs sont les suivants :

- Le *niveau de test* qui définit quel niveau d'abstraction est utilisé par le MBT pour vérifier le comportement du logiciel. Les niveaux utilisés sont : système, intégration, composant et non régression.

1. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

- L'*existence d'un support outillé* qui définit si une approche de MBT peut automatiser ses activités par des algorithmes ou des instructions dans un outil.
- Le *domaine des SUT* qui définit le champ d'application d'une approche de MBT, le type de SUT où l'approche de MBT peut être exécutée (système embarqué, client-serveur, application web, etc)
- Le *modèle comportemental* qui représente les caractéristiques des SUT qui peuvent être testés par une approche de MBT. C'est principalement la capacité ou non de l'approche de MBT à modéliser les comportements du SUT.
- Les *critères de couverture des cas de test* qui définissent les règles utilisées pour générer des cas de test à partir des modèles. Il existe deux types de critères : le flux de données et le flux de contrôle.
- Les *critères de génération de cas de test* qui décrivent les activités à suivre par une approche MBT pour utiliser un modèle comportemental pour la génération de cas test ou leur exécution.
- Le *niveau d'automatisation* qui définit la proportion du nombre d'activités automatisées qui composent une approche de MBT. La principale caractéristique du MBT est la possibilité de génération (et l'exécution) automatisée des tests. En général, les approches MBT sont composées d'activités automatisées et non automatisées.

En 2012, Utting et Legeard [92] complètent cette vision et proposent une taxonomie du MBT qui depuis a servi de base à de nombreux articles. Au sujet des niveaux de test, ils se concentrent sur le niveau système car il est le plus répandu. Ils expriment aussi que la majorité des techniques et approches de MBT ont aujourd'hui un support outillé et n'existent pas seule. Ils traitent en détail :

- des spécifications des modèles comportementaux,
- de leur champ d'application,
- de leurs caractéristiques (déterministe ou non par exemple),
- des paradigmes,
- des techniques telles que basées sur les transitions, les flux de données, etc.

Ces auteurs complètent les champs portant sur les notions des critères de couverture des cas de test, de génération des cas de test et le niveau d'automatisation. Pour eux, les outils de MBT permettent la génération des cas de test, et différentes technologies sont présentées telles que la génération aléatoire, les algorithmes basés sur la recherche,

etc. L'exécution des tests peut être réalisée en ligne ou hors ligne. En ligne, l'outil de MBT va directement s'exécuter sur le SUT, hors ligne les tests vont être générés et devront par la suite être exportés vers un outil dédié à l'exécution des tests. En 2015 une alternative à cette taxonomie est publiée dans laquelle sont reprises : les notions de notations des modèles, la sélection des critères de test, les méthodes de génération de test et l'exécution de test [56]. Cette nouvelle taxonomie ajoute les notions d'artefacts de test représentés par le modèle et le support entre cas de test abstrait [2] ou exécutables. En 2017, la taxonomie initialement émise en 2012 est reprise et développée [53] sur la génération des cas de test et la documentation des tests. Et pour finir en 2019, une étude extrait les différents domaines du MBT en se basant sur les travaux de Utting et Legeard et sur l'analyse des études secondaires publiées pour produire une nouvelle taxonomie [95]. Elle ne présente que les domaines les plus étudiés dans le MBT, en particulier le paradigme et la spécification des modèles [66]. On apprend aussi que les écrits traitent plus des niveaux de test composant et intégration et moins des niveaux système et acceptation. De ce fait, les langages employés par les outils sont en majorité des notations basées de type diagrammes états-transitions, avec l'utilisation d'UML(Unified Modeling Language). Quant à la sélection des critères de test, elle est dominée par la couverture des éléments du modèle, telles que les états et les transitions. Pour les autres aspects (méthodes de génération, technologie de génération, et exécution), il n'y a pas de tendance particulière qui se dessine, chaque outil utilisant des techniques variées de génération. Après cette brève revue historique de l'évolution du MBT, la section suivante est consacrée au détail du processus de MBT.

PROCESSUS DU MBT

Indépendamment de l'outil utilisé, le processus de MBT peut être divisé en 5 phases interconnectées :

- Modéliser le système sous test et/ou son environnement.
  Cette phase a pour objectif de faire une représentation du SUT. Cette représentation doit faire figurer l'ensemble des comportements métier à vérifier.

- Générer des tests abstraits à partir du modèle.
  Une fois qu'un ensemble de modèles a été construit, il est possible de générer des cas de test abstraits. Ces cas de test abstraits ne contiennent que des données abstraites, c'est à dire sans valeur concrète. Cela permet de couvrir les exigences métier sans entrer dans le détail de l'implémentation des valeurs des données de test. La phase d'implémentation vient ensuite pour réaliser la concrétisation des

2. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

tests abstraits permettant de les rendre exécutables.

- Concrétiser les tests abstraits pour les rendre exécutables.
  La concrétisation des tests abstraits consiste à remplacer les valeurs abstraites des données par des valeurs concrètes, mais aussi à détailler et documenter de façon plus précise les étapes de test [3].
- Exécuter les tests sur le SUT et établir les résultats.
  L'exécution des tests dans le cadre du MBT peut se faire directement dans l'outil ou hors de l'outil.
- Analyser les résultats des tests.
  Comme pour l'exécution des tests, l'analyse des résultats peut se faire directement sur les modèles ou dans un outil externe.

L'ensemble de ces étapes permet la mise en place du MBT sur un projet. Afin d'étendre nos connaissances sur cette mise en place du MBT, nous l'étudions dans les contextes industriels dans la section suivante.

2.1.2/ MODEL-BASED TESTING DANS L'INDUSTRIE

Nous nous sommes attachés précédemment à étudier l'historique des travaux en Model-Based Testing dans le contexte académique. Dans cette section, nous nous concentrons sur la mise en œuvre du MBT dans l'industrie. Comme indiqué au début de ce chapitre, l'analyse des applications du MBT dans l'industrie est moins développée que l'étude des techniques MBT [26]. Dias-Neto et al. présentent en 2010 les concepts, techniques et challenges du MBT et ils mettent notamment en avant le manque de connaissances scientifiques sur l'application des techniques de MBT dans l'industrie. Malgré tout, des retours d'expériences existent notamment concernant le support d'outils de MBT. ALi et al. [3] présentent leurs retours d'expériences de l'outil TRUST : Transformation-Based Tool For UML-Based Testing. Ils présentent ainsi un ensemble de challenges du MBT auxquels ils essayent de répondre. Parmi les points soulevés, les auteurs relèvent le manque de compétences au sein des équipes en matière de modélisation pour construire les modèles nécessaires aux tests. Afin de produire des modèles de qualité, il est essentiel de maîtriser le langage de notation utilisé. De plus, ils précisent que l'absence de modélisation métier existante peut complexifier la tâche de modélisation car il est nécessaire de revenir aux exigences. Pour pallier au manque de connaissance métier, des ateliers doivent être organisés pour permettre la conception des modèles. En construisant l'outil TRUST, les auteurs ont essayé de répondre à ces challenges. Les leçons qu'ils ont apprises sont qu'il faut construire des modèles précis,

3. Une définition de ces termes sont définies en Annexe A.1

corrects et complets. Sans cela la génération des tests est impossible où inappropriée, en décalage avec le réel. Ils mettent en avant que ces activités de conception des modèles ne sont pas triviales. L'UML peut être complexe à utiliser pour représenter facilement des systèmes complexes. De l'expertise et du temps sont nécessaires. On retrouve les mêmes constats avec des outils de MBT produit par Siemens, sepp.med, Microsoft et Conformiq par exemple pour lesquels chaque retour d'expériences précise que des experts du domaine ont dû intervenir pour la phase de modélisation ou qu'un temps de formation conséquent a dû être prévu pour permettre la réalisation des modèles. Dans un article présentant les retours d'expérience sur ces 4 outils [61], on peut relever que la formation au MBT a requis 81 jours-hommes sur l'outil de Siemens par exemple. Sur les autres expérimentations, on ne connaît pas la durée de formation ou alors il est précisé que ce sont des experts du domaine qui ont réalisé les travaux. À noter que pour les quatre retours d'expériences, le gain à utiliser le MBT a été démontré.

Lors de nos recherches, il est apparu que les outils de MBT utilisés dans l'industrie intégraient de manière régulière l'UML. Cependant, nous avons remarqué que la BPMN était aussi plus largement présente présente sur ces dernières années. Cela vise en particulier à répondre aux challenges de simplification des notations. Nous présentons le détail de notre étude de l'état de l'art sur cette approche dans la section suivante.

2.1.3/ La notation BPMN

La modélisation des processus métier est un élément déterminant au sein des entreprises d'aujourd'hui. Différentes approches de modélisation existent, mais c'est le BPMN qui est l'un des plus populaire actuellement [57].

Dans "Introduction To BPMN" [96], Stephen A. White présente la Business Process Management Initiative (BPMI) qui a développé en 2004 une norme de notation pour la modélisation de processus. Cette norme se traduit en français par « modèle de procédé d'affaire et notation » plus connue sous le nom de Business Process Modeling Notation (BPMN).

L'objectif premier du BPMN était de fournir une notation facilement compréhensible par tous les utilisateurs professionnels, des analystes métier qui créent les premières maquettes des évolutions, jusqu'aux développeurs implémentant les nouvelles fonctionnalités et aux testeurs les vérifiant. Cette notation vise aussi à améliorer la qualité des processus modélisés [84, 27]. Monique Snoeck, Isel Oca [84] ainsi que Jan Recker [76], et d'autres énoncent que le BPMN est *de facto* devenu la notation standard pour la modélisation des processus. Ce point sera développé plus tard, la section qui suit décrit l'historique et les fondamentaux de cette notation.

MODÉLISATION À TRAVERS LE BPMN

Le BPMN a connu différentes versions, mais à l'origine 4 catégories d'éléments de modélisation de base ont été créées :

- Les objets de flux : ils permettent de représenter les événements et les activités dans le processus, ainsi que des points de choix qui expriment les différents chemins possibles.
- Les connecteurs d'objets : ils permettent de relier entre eux les différents objets dans les processus. Ils sont de différents types, les flux de séquence, de message, et les associations.
- Les piscines et couloirs : les piscines représentent les participants au processus et les couloirs permettent de subdiviser les piscines pour faire intervenir différents acteurs au sein d'un même groupe et ainsi organiser les activités.
- Les artefacts : ce sont des éléments comme des objets de données, des groupes, des annotations. Ils ont pour objectif d'apporter de l'information au diagramme sous une forme libre pour détailler par exemple les entrées et sorties, les données attendues, etc.

Un exemple de modélisation présenté par Stephen A. White est visible dans la figure 2.1, où l'on voit les catégories évoquées. Tous ces éléments permettent ainsi à chaque acteur du projet de matérialiser ses besoins et donc de communiquer une grande variété d'informations à différentes parties prenantes à une solution. Ceci sous une forme commune, ce qui répond au 1^{er} objectif exprimé qui était de fournir une notation facilement compréhensible.

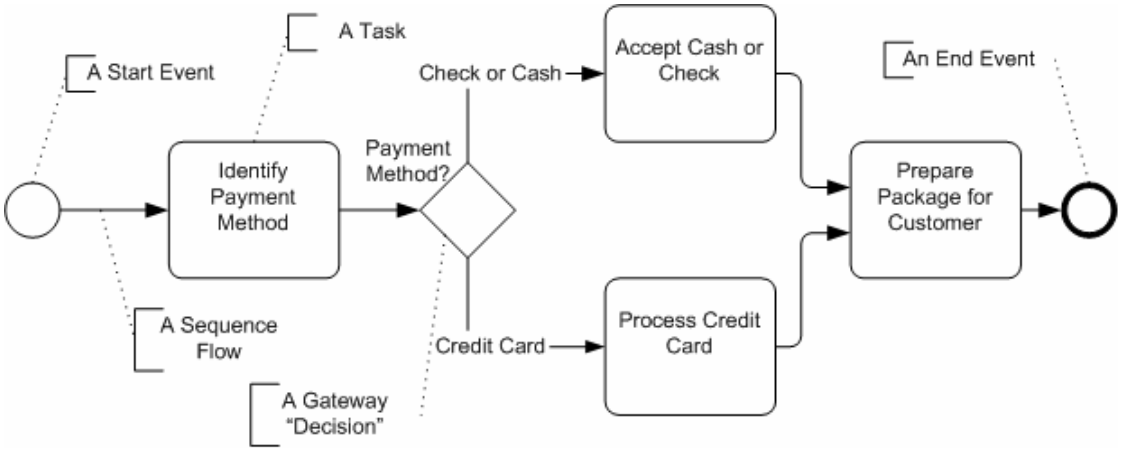


FIGURE 2.1 – Exemple de modèle en BPMN

Le second objectif était de proposer un modèle interne capable de générer des scripts pour l'exécution des processus modélisés. Ceci a été fait à l'aide du BPEL4WS (Business Process Execution Language for Web Services). Le BPEL est un langage basé sur XML qui modélise un processus métier comme une combinaison d'un ensemble de services web élémentaires [59].

L'objectif était de faire correspondre automatiquement le BPMN avec les spécifications BPEL exécutables. Cependant, cette translation n'était pas aisée à mettre en place. Les outils peinaient à fournir un support de modélisation adéquat et il existait des incohérences entre le BPMN et BPEL qui pouvaient impacter la conversion d'un modèle vers un autre [37, 52, 67]. Par ailleurs, l'orientation technique et la rigueur des spécifications de BPEL s'opposent à l'intuitivité des modèles, et entraînent des difficultés de compréhension lorsque des acteurs non techniques tels que les analystes métier sont impliqués.

Après plusieurs évolutions, la version originelle a connu un changement marqué avec la sortie de la version 2.0 [18]. La spécification BPMN 2.0 étend la portée et les capacités du BPMN 1.2 dans plusieurs domaines, notamment la gestion du BPEL. Les améliorations fondamentales du BPMN 2.0 sur le sujet sont :

- l'ajout d'éléments (visuels) pour les gestionnaires (handlers) du BPEL,
- la spécification d'une approche basée sur des modèles pour la mise en correspondance d'un sous-ensemble de BPMN 2.0 avec le BPEL,
- un format d'échange basé sur XML,
- la définition d'une sémantique opérationnelle pour le BPMN 2.0 qui reste proche du BPEL.

Ceci a ainsi permis de répondre en partie aux problématiques de transformation entre BPMN et BPEL. En partie, car seulement un sous ensemble des éléments du BPMN est naturellement supporté par le BPEL et ces ajouts ont fait apparaître de la complexité au BPMN 2.0 [52].

USAGE DU BPMN

L'objectif premier du BPMN est de fournir une notation facilement compréhensible par tous les utilisateurs, nous étudierons ici l'usage qui en est fait.

En 2012, Michele Chinosi et Alberto Trombetta [18] étudient l'usage du BPMN.

Dans leur article, les auteurs présentent les résultats d'interviews et d'études antérieures [65] qui indiquent que le BPMN est d'abord employé à des fins documentaires (52 %),

puis pour l'exécution de processus métier (37 %), et pour finir pour de la modélisation de processus (11 %). Par ailleurs, l'usage des éléments du BPMN est concentré sur une partie d'entre eux. Entre 30 et 40 % des utilisateurs considèrent que des labels textuels sont suffisants, et évitent de spécifier des attributs pour les piscines, couloirs, lignes, flux, etc. 40 % ajoutent des valeurs comme des timers, des messages, des erreurs sur des événements, et parmi eux seulement 33 % définissent des scripts et des règles sur les points de choix et activités. Et finalement, 37 % des interviewés ne se soucient pas du tout des attributs et propriétés.

Ces résultats sur l'usage du BPMN tendent à montrer que malgré la grande diversité d'éléments offerts par cette notation, l'utilisation reste "limitée". D'une part, les utilisations de BPMN se concentrent à 2 domaines précis : la documentation et l'exécution des processus métier. Cette tendance rejoint l'analyse qualitative [76] produite par Recker, Jan C. en 2010, qui énonce que "les applications "classiques" telles que la documentation, la refonte, l'amélioration continue et la gestion des connaissances dominent dans l'application du BPMN". Et que "les domaines d'application plus techniques tels que le développement de logiciels, la gestion des flux de travail ou la simulation de processus ne sont pas (encore) répandus". Cette notion transparaît aussi dans l'expérience menée par Michael zur Muehlen et Danny T. Ho [100] où ils déclarent que "malgré l'introduction de l'ensemble des artefacts du BPMN, le public a tendance à se souvenir des éléments de base et a ignorer les constructions qui ont une signification plus riche". De plus, dans cette expérimentation, ils concluent que l'usage d'un sous-ensemble limité d'éléments de BPMN a permis aux participants de communiquer facilement, dans la mesure où ils ont suggéré de manière proactive des changements et besoins en matière de documentation.

D'autre part, la diversité des éléments de modélisation mis à disposition par le BPMN n'est pas exploité. Les praticiens du BPMN préfèrent des spécifications à l'aide de labels, en évitant, voire en n'utilisant pas les attributs et propriétés des éléments de notation. C'est Michael zur Muehlen et Jan Recker qui ont montré dans leur étude [65] que finalement sur 50 objets de modélisation, en moyenne seuls 9 étaient utilisés lors de la construction d'un modèle.

En conclusion, malgré la popularité du BPMN, et les larges capacités de modélisation que propose la notation, un sous ensemble restreint d'éléments de modélisation est utilisé. Dans la prochaine section, nous étudierons les outils de BPMN et regarderons s'ils suivent les tendances d'usage exprimées ici.

BPMN ET OUTIL

Depuis la création du BPMN, de nombreux outils supportant cette notation on fait leur apparition. En 2010, Recker [76] établit un top 10 des outils les plus utilisés pour modéliser le BPMN.

C'est sans conteste Microsoft Visio qui est le plus employé à 18,2 % d'usage. En second vient ITP Process Modeler avec 7,8 % puis SparxSystems Enterprise Architect 6,9 %, Visual Paradigm Visual Architect 6.2 %, Telelogic System Architect 5,7 % et Intalio BPMS 5 %.

En 2012, on retrouve les mêmes tendances [18], mais avec la disparition de Microsoft Visio (car non considéré comme un "réel" outil de BPMN [41]). En effet, l'auteur explique qu'il ne s'agit que d'un outil de création de diagrammes puisqu'il ne met pas en œuvre les méta-modèles abstraits du BPMN. En tant que tel, il n'assure aucun contrôle syntaxique et ne permet pas la simulation ou l'exécution, contrairement aux outils de BPMN qui permettent cela comme BizAgi, Bonita, Signavio et Trisotech qui sont les 1^{ers} outils utilisés.

Les outils se spécialisent et visent des applications métiers différentes : par exemple, System Architect et Enterprise Architect sont plutôt dédiés à la documentation de l'architecture d'une entreprise, son métier, ses technologies, systèmes, etc. Les outils BizAgi modeler et Bonita sont eux, plutôt orientés vers la construction de processus métier pour la génération de rapports et de documentations. Le choix d'un outil plutôt qu'un autre semble être guidé en premier lieu par la facilité d'usage de l'outil, vient ensuite la conformité avec les notations du BPMN 2.0 et enfin la présence d'un support à la validation. Les aspects plus techniques tel que le support au WSDL, au BPMN 1.0 et au WS-BPEL n'apparaissent qu'après [18]. Ceci explique l'arrivée de nouveaux outils plutôt dédiés à la modélisation de processus purement métier et moins à la cartographie d'architectures ou de systèmes. Les outils confirment bien les tendances d'utilisation avec une prédominance d'outils plus orientés modélisation de processus que leur exécution. Depuis ces études, peu de constats de l'usage d'outils dans l'industrie ont été menés, c'est ce que présente Mateja Kocbek [41] dans son état des lieux sur le BPMN en 2015.

QUALITÉ ET BPMN

La question de la qualité des modèles est un sujet qui a été régulièrement traité sur le BPMN au cours de ces dix dernières années. En 2015, H. Leopold, J. Mendling, et O. Günther [50] reprennent les travaux entrepris par Jan Recker en 2010 [76] pour les compléter notamment, car il aborde peu les questions de qualité des modèles.

Les problématiques autour de la qualité des modèles sont divisées en 3 groupes : la

structure, la disposition, le nommage.

La structure se réfère à l'utilisation correcte et cohérente des éléments de modélisation tels que les activités, les connecteurs, les événements, etc. Pour ce groupe, les problématiques dominantes sont les sous-processus ayant un lien incohérent avec le processus principal, c'est à dire un parcours principal concernant un acteur via un couloir et son sous-processus concernant un autre acteur. Ensuite vient l'utilisation erronée de messages, fait par l'association d'un type de message à un type de nœud non adapté au message. Et ensuite l'existence de fusions multiples au sein d'un même objet (qui entraîne des exécutions répétées non souhaitées.

La disposition concerne le bon positionnement des éléments du modèle pour permettre une bonne compréhension visuelle de celui-ci. L'élément de peine sur la disposition est en grande partie la taille excessive des diagrammes. En effet, quand la taille du diagramme excède un certain format, il devient compliqué pour les acteurs consultant le modèle de le comprendre et d'appréhender les fonctionnalités qu'il présente. Ceci est vrai aussi dans le cas de la superposition d'éléments qui ne devraient pas pouvoir se chevaucher, tels que des connecteurs et les noeuds. Ensuite des comportements incohérents en entrée et en sortie avec de mauvaises directions dans la modélisation créent des incompréhensions.

Pour finir, le nommage fait référence à l'utilisation correcte du langage naturel dans les modèles. L'absence de certains éléments dans le glossaire (les objets de données, les rôles) crée des doublons dans les données, les rôles et complique la maintenance des modèles. Les points de choix, les activités et les événements avec un label non conforme/irrégulier créent des ambiguïtés dans les modèles pour les utilisateurs.

Ces observations relevées par les auteurs [50] montrent que de nombreux concepts structurels, tels que la cohérence entre les modèles, leur tailles ainsi que le nommage des éléments du modèle semblent être lié à des problématiques de qualité. Ils supposent que ces problématiques liées à la qualité émergent d'un manque de clarté des recommandations et bonnes pratiques de modélisation et que les choix de représentation du BPMN peuvent provoquer des erreurs.

Ainsi de nombreux auteurs [50, 60, 84] proposent des recommandations simples pour répondre à ces problèmes spécifiques, tels que d'utiliser des points de choix en sortie d'activités au lieu des divisions implicites dans celles-ci, décomposer en sous-processus quand les modèles deviennent trop grands, etc. En conclusion, ce que démontrent ces travaux c'est que le BPMN peut induire des erreurs par les possibilités de modélisation qu'il offre, et que son usage ne peut être fait sans définir un cadre de bonnes pratiques pour limiter les problèmes de qualité.

2.1.4/ POSITIONNEMENT DES TRAVAUX DE LA THÈSE PAR RAPPORT À L'ÉTAT DE L'ART

Malgré des retours d'expériences positifs, la dissémination du MBT dans l'industrie ne progresse que très lentement [46]. Cela peut s'expliquer par la non-correspondance initiale des principes du MBT avec ceux des préceptes Agiles. [61]. En effet, le MBT nécessite une base solide pour son introduction dans le cycle de développement. Il requiert également une formation à l'outil et langage pour les acteurs du projet, et un effort initial important pour construire les modèles. Cependant, le MBT tend à se simplifier avec l'usage de notations plus allégées. C'est le cas du BMPN qui est une notation plus adoptée dans l'industrie pour modéliser des processus métier. De manière générale, le BPMN est bien accepté. Il présente de nombreux avantages (notation simple, facile à comprendre, etc.) et peu d'inconvénients à proprement parler, car ils sont plus liés au fait que la notation est peu employée dans son intégralité et sur des domaines spécifiques. Cependant malgré son utilisation de prime abord accessible, la mise en place de recommandations, bonnes pratiques et de normes communes semblent être un point crucial pour la qualité des modèles réalisés.

Un sujet ayant de l'intérêt et qui est peu exploré est la simplification des notations de MBT. L'étude d'une notation proche du BPMN, en introduisant un cadre de bonnes pratiques est l'objet d'un des axes de recherches de la thèse. Cette notation devra permettre de modéliser des systèmes complexes pour le Model-Based Testing sans être elle même complexe. Sa prise en main et sa mise en place devront être "allégées" pour permettre de réaliser des modélisations pour les tests sans devoir représenter le système à tester dans son intégralité.

2.2/ AGILITÉ ET AGILITÉ À L'ÉCHELLE

Cette section a pour objectif de présenter notre état de l'art en Agilité et en Agilité à échelle. Nous nous attachons particulièrement à l'étude des tests et du MBT dans ces contextes. La première partie traite de l'Agilité au sens large et la seconde partie détaille l'usage de l'Agilité dans des contextes à l'échelle, c'est à dire avec plusieurs équipes en Agile contribuant ensemble à un grand système.

2.2.1/ AGILITÉ

En 2001, un groupe de praticiens de l'ingénierie logicielle se sont accordés sur un ensemble commun de valeurs et de principes qui sont connus sous le nom de Manifeste Agile [75].

Le manifeste Agile contient quatre déclarations de valeurs :

- Les individus et leurs interactions plus que les processus et les outils.
- Des logiciels opérationnels plus qu'une documentation exhaustive.
- La collaboration avec les clients plus que la négociation contractuelle.
- L'adaptation au changement plus que le suivi d'un plan.

L'élan Agile a donné lieu à la création de différentes approches dans l'objectif de mettre en pratiques ces valeurs et les principes associés au sein des organisations. Parmi ces approches, on peut citer : Extreme Programming, Scrum et Kanban. Elles se distinguent par leurs domaines d'application et leurs manières d'implémenter les valeurs et les principes Agile.

Nous présenterons Scrum, car c'est l'approche la plus utilisée à 54 % dans le 13e rapport annuel sur l'état de l'Agile [94]. Ensuite viennent à 24 % des approches hybrides, suivies de Scrumban à 8 %, et Kanban à 5 %. Scrum est un framework de management Agile qui contient les pratiques suivantes [1] :

- La division du projet en itérations (appelées sprints) de durée fixe (habituellement de 2 à 4 semaines).
- Des Incréments Produit : Chaque résultat de sprint est un produit potentiellement livrable/déployable.
- Le Backlog Produit : Le Product Owner (PO) gère une liste priorisée d'éléments de produits planifiés. Généralement ces éléments sont représentés par des User Stories (US) [4] qui sont des exigences haut niveau décrivant un comportement attendu de l'application sous la forme de : En tant que, je veux, afin de.
- Le Backlog de Sprint : Au début de chaque sprint, l'équipe Scrum sélectionne un ensemble d'éléments à partir du Backlog Produit.
- La définition de Terminé : C'est une liste de critères appropriés pour la complétude du sprint.
- Le timeboxing : Seules les tâches, les exigences, ou les fonctionnalités que l'équipe souhaite finir dans le sprint font partie du Backlog de sprint.
- La transparence : L'équipe de développement rapporte et met à jour le statut du sprint sur une base journalière appelée le Daily Scrum. Cela permet de rendre

4. Nous choisissons le terme User Storie(s) et son abréviation US dans ce manuscrit en suivant la norme industrielle de l'ISTQB

visibles à toutes les parties intéressées, le contenu et la progression du sprint en cours.

Scrum définit trois rôles :

- Le Scrum Master qui assure que les pratiques et que les règles Scrum sont implémentées et suivies.
- Le Product Owner qui représente le client, génère, maintient et priorise le Backlog Produit.
- L'équipe de développement qui développe et teste le produit. L'équipe est auto organisée.

Scrum ne fournit pas de guide sur comment le développement et les tests doivent être réalisé dans un projet Scrum, donc chaque équipe est autonome et assure la qualité du produit qu'elle fournit. Bien qu'aucune approche ne soit définie par Scrum ni dans le manifeste Agile sur la manière de réaliser les activités de test, nous étudions dans la partie qui suit comment le test est réalisé dans des contextes Agile.

AGILITÉ ET TEST

L'Agilité a complètement bouleversé les approches traditionnelles de test [87]. Comparé au cycle en V où les tests sont préparés durant une longue période et exécutés dans leur globalité une fois tous les développements terminés, dans les approches Agile, le test est réalisé tout au long des itérations. Les phases de conception, exécution et mise à jour des tests sont cycliques. Appliquer des approches traditionnelles de test ne se prête pas à l'Agile, et tester dans un contexte Agile peut s'avérer être un véritable challenge [71]. Sean Stolberg [85] présente dans son article une tentative de mise en pratique des approches de test traditionnelles au cours d'un sprint. Le résultat est qu'il n'exécutait pas ses tests au cours du sprint et qu'il accumulait une dette en test. Ceci se traduit par une couverture partielle des exigences. Or l'un des aspects les plus importants des tests consiste à valider que le système répond à ses exigences [39]. Et comme pour la gestion des tests dans sa globalité, la vérification et le traitement des exigences nécessitent des approches particulières propres aux contextes Agiles, afin d'être en capacité de gérer les changements en cours d'itération. Une analyse au plus tôt des exigences fait partie des points clés, comme l'énonce Stolberg en disant qu'il faut "insérer les activités de test au plus tôt dans le développement du sprint, pour obtenir la couverture nécessaire et être en mesure de tester, de trouver et de corriger les défauts avant la fin du sprint". Ce précepte constitue le cœur du test en Agile comme cela est reporté dans de nombreux autres

articles [71, 86, 87]. La pratique du Test Driven Development (TDD), consistant à écrire les tests avant le développement est largement adoptée dans les contextes Agile [79]. Une autre différence majeure entre cycle traditionnel et cycle Agile concerne l'implication des différents acteurs dans les activités de test. Sur des projets par phases séquentielles (cycle en V), comme en Agile, toutes les parties prenantes au projet portent la responsabilité de la qualité, mais en cycle en V, seules les acteurs dédiés à la qualité mènent des activités de test. Alors que dans les contextes en Agile, les acteurs du projet assument la responsabilité de la qualité, mais prennent aussi part aux activités de test de différentes manières et travaillent ensemble [87]. Les méthodes et les outils de test aspirent donc à s'adapter pour mieux correspondre aux profils distincts contribuant au projet : testeurs, développeurs, PO (Product Owner), analystes Métier, etc. Parmi les facteurs clés contribuant à la coordination et la compréhension des activités de test, la communication a une importance capitale [71, 86]. Michael Puleio présente notamment dans "How not to do Agile testing" [71] les difficultés qu'il a rencontrées à mettre en place des tests dans une équipe Agile, et comment une forte communication leur à permis d'avancer. Il précise que "la communication est la clé, peu importe ce que vous faites. Si vous ne parvenez pas à trouver un langage commun et que [...] les autres membres de l'équipe ne parviennent pas à comprendre ce que vous dites ; essayez de trouver un moyen de partager suffisamment d'expériences et de connaissances de votre contexte pour communiquer efficacement". Dans mes recherches, j'étudie de quelles façons la communication peut être facilitée pour la mise en place d'une approche MBT en Agile.

AGILITÉ ET MBT

Dans la littérature quand on recherche des articles mentionnant l'Agilité et MBT, les résultats ne semblent pas de prime abord associer positivement ces deux domaines. Masoumeh Taromirad et Raman Ramsin [89] déclarent que "de nombreuses méthodes de MBT ne sont pas directement applicables aux méthodes Agile/légères, car les pratiques prescrites par le MBT ne sont pas particulièrement compatibles avec celles proposées par les méthodes Agile : en général, les méthodes Agile tendent à simplifier et à accélérer le processus de développement en déconseillant la production, et la mise à jour ultérieure, des modèles". C'est notamment ce qu'énonce David Fagaro [31] en disant que "des modèles abstraits sont indispensables, ce que les outils actuels de MBT ne permettent pas de faire". Or des auteurs traitent de l'usage du MBT en contexte Agile. C'est le cas de Renate Löffler, Baris Güldali, Silke Geisen [55] qui présentent une approche de MBT adaptée à Scrum et où des modèles sont utilisés pour capturer les exigences métier et guider le test d'acceptation. La démarche permet d'une part, d'obtenir une description du SUT sous forme de modèles (UML) et d'autre part de générer des cas de test. On retrouve des études similaires avec Bernhard Rumpe [79] et d'autres [30, 16].

M. Katara et A. Kervinen [39] proposent une solution qui repose sur l'existence d'experts pour concevoir des modèles de test. Ils interfacent les systèmes de génération de tests avec des cas d'utilisation qui sont convertis en séquences de mots clés, correspondant à des événements utilisateur à un niveau d'abstraction élevé. Ils notent cependant que dans leur approche, la relation entre exigences, documents de conception et modèles manquent de clarté, de plus ces propositions de recherche ne sont pas validées de façon conséquente sur des projets industriels. Il apparaît ainsi que le MBT en Agile reste un sujet ouvert pour la recherche en Génie Logiciel.

2.2.2/ AGILITÉ À L'ÉCHELLE

La concurrence grandissante sur les marchés internationaux a conduit les entreprises à employer de nouveaux modes de développement de leurs solutions [15]. L'Agilité, basée sur des cycles courts, des livraisons régulières et à forte valeur ajoutée, est bien adaptée aux besoins des entreprises de raccourcir les délais de mise en production - le "time-to-market" en anglais [64] [94]. Au fil des années, les entreprises et leurs projets n'ont cessé de croître pour employer des centaines d'individus sur des multitudes de produits interconnectés au sein de grand SI. Aujourd'hui, l'Agilité initialement conçue pour être utilisée sur des périmètres réduits et avec une seule équipe (moins de 10 personnes) [15] se voit adaptée et transformée par les entreprises pour coordonner les activités d'équipes de plus en plus volumineuses et distribuées. Cette coordination n'est pas aisée en Agile [64, 28], de ce fait les acteurs de l'Agilité ont étudié comment maintenir les préceptes Agiles tout en permettant une coordination des activités de chaque partie prenante à une solution : c'est là qu'est apparue le concept de l'Agilité à l'échelle. Dès la publication du Manifeste Agile, des articles commencent à mentionner le nombre de personnes impliquées dans les projets Agiles et à étudier le nombre souhaitable de personnes qui devraient y participer pour faciliter l'organisation Agile [97, 42].

Dans la littérature, différents points de vue existent sur ce que représente l'Agilité à l'échelle [28]. Les divergences s'expliquent par le fait que l'Agilité à l'échelle en tant que telle ne dispose pas de précepte aussi précis que ceux de l'Agilité traditionnelle. Hormis que c'est une adaptation de celle-ci. Le mot adaptation revêt ici une importance particulière, car elle est construite différemment selon chaque entreprise et chacune réinvente en fonction de ses besoins l'Agilité pour la transformer en Agilité à l'échelle [7]. D'une part, il existe la vision où l'Agilité à l'échelle est représentée uniquement par le grossissement des équipes. Au lieu de construire des équipes Agiles de moins de 10 personnes, on tend plutôt vers des équipes de plus grande taille, cassant les codes de l'Agilité de base. D'autre part, l'approche est dans la multiplication des équipes travaillant sur une solution commune.

Aujourd'hui, la vision qui semble se démarquer se trouve dans la combinaison d'équipes Agiles de plus grande taille et distribuées sur plusieurs sites et/ou pays, tout en intégrant une notion de spécialisation [15]. La spécialisation représente le fait que pour construire de larges solutions les équipes doivent se consacrer sur des aspects métier particuliers du produit. La vision complète de la solution, comme le préconise l'Agilité traditionnelle, peut difficilement exister lors de la confrontation à de vastes et complexes systèmes tels que sont ceux abordés dans les contextes d'Agilité à l'échelle.

La construction d'un ensemble de nouvelles approches se traduit par des échecs de mise en place ou de véritables difficultés à l'implémentation d'un processus stable et efficace pour gérer la coordination des activités dans un contexte à l'échelle [64].

Dans ce contexte, des approches et frameworks dédiés à l'Agilité à l'échelle ont fait leur apparition. Même si l'Agilité à l'origine ne prévoyait pas nécessairement de gérer cette mise à l'échelle, il est important de préciser que Scrum comporte un événement pour gérer la coordination des équipes Scrum : le Scrum of Scrums meeting. Dans la section suivante, nous présentons les frameworks les plus utilisés dans la mise en oeuvre de l'Agilité à l'échelle.

FRAMEWORKS D'AGILITÉ À L'ÉCHELLE

Dès 2001, le Scrum of Scrums meeting [80] est introduit pour coordonner les activités de plusieurs équipes Agiles. Il s'agit de réunir régulièrement les équipes Scrum et les faire répondre à 3 questions :

1. Qu'a fait votre équipe depuis la réunion précédente qui soit pertinent pour une autre équipe ?
2. Que fera votre équipe d'ici à la prochaine réunion qui intéresse d'autres équipes ?
3. Quels sont les obstacles auxquels votre équipe est confrontée et qui affectent les autres équipes ou requièrent leur aide ?

Ceci permet ainsi la coordination des équipes Scrum et est appliqué par exemple dans le framework d'Agilité à l'échelle Less[48], crée bien plus tard en 2013. Craig Larman et Bas Vodde ont crée Less pour appliquer Scrum au développement de produits à très grande échelle, multisites et offshore. Dans ce cadre, différentes équipes existent : transversales, intercomposants et bout en bout. Elles existent toutes sur un seul niveau, et travaillent en commun avec un seul PO. L'organisation ainsi constituée est appelée framework 1. Quand le produit excède 100 personnes, le framework 2 est mis en pratique pour coordonner les différents groupes d'équipes Agiles. Dans ce cas, plusieurs PO interviennent. Sur le plan des pratiques, Less applique les pratiques de Scrum en les faisant

évoluer et en renforçant la coordination avec plus que le Scrum of Scrums meeting. Par exemple, Less abandonne la notion d'incrément potentiellement livrable en fin d'itération et supprime par la même occasion tout l'aspect organisationnel lié à la livraison de ces incréments et qui impliquerait un grand nombre de partie prenante (dans des phases d'analyse, d'intégration, d'étude, de test, etc). De plus chaque équipe est concentrée sur la réalisation du produit final, plutôt que sur la réalisation de "sa partie".

En 2007, SAFe est créé par Dean Leffingwell avec d'autres collaborateurs [93]. Le framework SAFe définit trois niveaux d'organisation : Équipe, Programme et Portefeuille. Chacun d'eux se caractérise par un ensemble d'activités spécifiques et sont interdépendants.

Dans SAFe, le niveau équipe représente l'ensemble des équipes Scrum, elles mêmes composées de cinq à neuf membres. On y retrouve les personnes travaillant à la spécification, au développement et la vérifications des projets (développeurs, testeurs, Scrum master, PO, etc).

Le niveau programme regroupe un ensemble d'acteurs comme le Product Manager (PM), le Release Train Engineer (RTE), le system architect/engineer. Chacun a des responsabilités précises : le PM porte la responsabilité du Backlog du programme, qui regroupe les grandes fonctionnalités à développer par les équipes. Il est aussi responsable des PO au sein des équipes Agiles. Un architecte système est présent pour réaliser l'architecture initiale et guider l'architecture de tous les projets du programme. Et le RTE a pour rôle de faciliter les processus en place au niveau du programme, de lever les obstacles, de gérer les risques et de contribuer à l'amélioration continue au niveau du programme en manageant entre autres les Scrum master.

Le niveau portefeuille pour finir, correspond au plus haut élément d'autorité sur le produit final. Il est composé de personnes en charge de la supervision globale de la solution. On y retrouve les Epic Owners qui expriment leurs besoins au moyen des épopées de la solution qu'ils partagent avec les membres des autres niveaux. L'architecte de la solution qui a comme rôle de définir l'architecture du train (de manière similaire à l'architecte du programme qui gère l'architecture des équipes).

En 2008, Spotify a proposé un framework qui vise à faciliter la coordination de plusieurs équipes Agiles autour de la même solution [5]. Les équipes Agiles se nomment des *Squads* et n'appliquent pas nécessairement Scrum, elles peuvent appliquer Scrum, Kanban, ou la combinaison des deux. Elles travaillent avec des PO. Jusqu'ici, Spotify peut sembler très proche de Less. Les différences résident dans le fait que Spotify introduit plus de niveaux et des rôles avec des responsabilités. Le niveau 1 correspondant aux Squads, le regroupement de Squads autour d'un domaine est appelé *Tribu* (niveau 2) et dirigé par un leader de tribu. On trouve aussi la notion de *Chapitre* qui regroupe des personnes partageant les mêmes compétences (par exemple le test) au sein d'une tribu. Enfin, les *Guildes* sont des groupes d'individus partageant des connaissances, outils et

pratiques à l'ensemble des équipes.

L'émergence de ces différents frameworks a permis aux grands groupes de commencer leur transformation de l'Agile vers l'Agilité à l'échelle. Nous décrivons cette transformation dans la section suivante.

PASSAGE DE L'AGILE VERS L'AGILITÉ À L'ÉCHELLE

Avec l'arrivée de méthodes structurées et préparées pour répondre aux besoins de mise à l'échelle, la construction des solutions aurait pu être facilitée. Or ce n'est pas le cas et de nombreux articles relatent les difficultés et challenges auxquels sont confrontés les projets dans l'implémentation de ces nouveaux frameworks. Certains auteurs évoquent le fait que l'Agile ne se prête pas aux contextes à grande échelle [75, 19].

Dan Turk, Robert France et Bernhard Rumpe présentent les limites des processus de développement Agile [91]. Sur six limitations exprimées, cinq concernent les difficultés liées à une mise à l'échelle. Le développement dans des équipes distribuées, larges, en outsourcing, la réutilisation d'artefacts communs inter-équipes, et les projets complexes constituent ces limitations. La dernière relève de la faiblesse de l'Agile à apporter du support pour concevoir des logiciels de sécurité critique.

Indépendamment du framework utilisé, l'Agilité à l'échelle semble poser un grand challenge. Pour les groupes ayant relevé ce défi avec succès (comme Ericsson [19]) d'autres problématiques peuvent exister et sont peu abordées telles que les activités du point de vue du test des solutions. Étant donné les difficultés non négligeables pour créer de nouveaux produits à large échelle, comment parvenir à coordonner les efforts de chacun pour garantir la qualité du système implémenté et avoir une approche efficiente pour les tests au niveau du système intégré ? Nous abordons ce sujet dans la section qui suit.

AGILITÉ À L'ÉCHELLE ET TEST

Dans la littérature, beaucoup d'écrits se concentrent sur la mise en place d'une solution globale, avec différents acteurs, rôles, et objectifs, mais très peu concernent les tests. L'Agilité traditionnelle doit parfois faire face à de grands challenges pour introduire des stratégies de test efficaces et adaptées à des cycles courts au sein d'une équipe Agile. L'Agilité à l'échelle, elle, doit se confronter aux mêmes challenges et s'adapter aux nouveaux défis provoqués par la délocalisation des équipes et la gestion de leur coordination. Ceci tout en maîtrisant l'acculturation même à cette approche. Cette acculturation s'avère visible dans la littérature.

En 2016, Bass a mené une étude [7] basée sur 46 interviews auprès de praticiens de l'Agilité à l'échelle avec différents profils. La question principale de cette étude était

”comment les praticiens décrivent-ils l'inventaire des artefacts qu'ils utilisent dans les programmes de développement de logiciels à grande échelle ?”. Suite aux interviews 25 catégories ont été définies, divisées en 5 grands groupes : fonctionnalité, sprint, version, produit et gouvernance du programme. La notion de produit est propre aux contextes à l'échelle. En effet pour adapter l'Agilité à des contextes à grande échelle, les industries divisent leur solution en produits. C'est-à-dire qu'au lieu d'avoir une solution globale avec un ensemble de produits existant comme un tout et maintenu par une équipe de plusieurs dizaines de personnes, chaque produit va être traité individuellement (par une équipe de moins de 10 personnes) pour former la solution complète. Chaque produit traite d'une grande fonctionnalité de la solution globale. Selon l'auteur, ces produits sont ”des éléments concrets du processus de développement des systèmes, destinés à assurer la cohésion des équipes travaillant sur un même programme de développement”. La hiérarchisation de ces groupes est la suivante : tout d'abord, le développement d'une solution est organisé par la gouvernance du programme. Elle définit les fonctionnalités à implémenter qui seront développées dans différents produits lors d'itérations appelées sprints. La succession des sprints conduira à l'achèvement d'une version qui contiendra un ensemble de fonctionnalités qui auront été développées.

Le test figure dans le groupe de gouvernance du programme à travers le plan de test. Le plan de test décrit l'approche globale des tests pour le programme de développement. À ce niveau, le test est bien considéré et doit être reporté dans les autres niveaux. Au niveau produit, bien que les tests d'acceptance existent, ils ne sont pas forcément exécutés à chaque release : à cause de la non-disponibilité des environnements pour la réalisation des tests par exemple. Au niveau release, on évoque les tests de non-régression : ils ne sont pas, eux non plus, exécutés à chaque release, car considéré comme trop long et coûteux. Quant aux tests d'intégration, peu de détails apparaissent hormis la mention d'exécution des cas passants. Au niveau du sprint, aucun élément de test n'est remonté. Dans les features, les critères de test sont abordés, mais dans l'exemple de témoignage apporté par l'étude, un Scrum Master précise que ”parfois des critères de test sont associés aux US”. Ceci met en avant que la pratique ne fait pas l'objet d'une mise en place systématique. Un autre exemple concerne la validation de la conformité des US : c'est la conformité des tests unitaires qui guide le déploiement ou non.

Ainsi, dans cette étude, bien que le test fasse partie intégrante de la solution, souvent les témoignages mettent en avant des pratiques approximatives et non structurées du test, ce qui nous a semblé un état commun de l'état de l'art des tests dans l'Agilité à l'échelle.

POSITIONNEMENT DE NOS TRAVAUX PAR RAPPORT À L'AGILITÉ À L'ÉCHELLE

Depuis sa création l'Agilité n'a cessé d'évoluer et de se diversifier par de nouvelles méthodes et pratiques. Les pratiques de l'Agilité avaient été initialement pensées pour de petits périmètres, mais elles sont aujourd'hui utilisées dans de larges organisations pour de très grands systèmes intégrant de multiples applications, sous-systèmes et composants. L'Agilité dite traditionnelle est aujourd'hui largement diffusée dans les organisations de toutes natures et dimensions, ce qui induit un impact important sur les pratiques des tests logiciels. Mais le contexte de l'Agilité à l'échelle crée de nouveaux challenges pour permettre de garantir la qualité au niveau du système complet, ce qui nous a conduits dans cette thèse à étudier l'état des pratiques actuelles pour les tests inter-équipes. Ce travail a pris la forme d'une étude qualitative et d'une étude quantitative qui constituent l'un des résultats de nos travaux, ouvrant sur des perspectives de mise en oeuvre du MBT et notre approche ALME adaptée à ce contexte.

2.3/ REFACTORING DES TESTS

L'évolution des suites de tests fait partie des projets, que cela soit en cycle en V ou en Agile, les tests changent avec les évolutions de l'application. Les cas de test deviennent plus longs et plus nombreux, des incohérences peuvent apparaître, des doublons, des redondances, etc. Ces évolutions conduisent à l'obsolescence des suites de test et du référentiel [70]. C'est encore plus vrai dans des contextes Agile, où les tests évoluent à chaque itération. Il devient alors coûteux d'étudier l'ensemble des tests du référentiel durant la phase d'analyse : beaucoup de tests à considérer et aucune méthodologie disponible pour accompagner les testeurs durant cette phase. Ces constats coïncident avec les résultats de l'enquête 2019 du Comité Français des Tests Logiciels, dans laquelle 82% des personnes interrogées indiquaient qu'elles avaient été confrontées à un problème d'obsolescence des référentiels de tests [25]. Afin de mieux comprendre cette problématique de maintenance des suites de test, nous avons étudié les techniques et outils pouvant exister pour le refactoring des suites de test.

2.3.1/ TECHNIQUES

La littérature traite largement du refactoring des cas de test unitaires et des tests d'acceptation automatisés. Différents algorithmes et solutions existent pour traiter du refactoring des tests unitaires ou d'acceptation lors de migrations ou évolutions de l'architecture de projets comme le font par exemple Krüger [47], Passier [69] ou Mazedur [73]. Krüger s'intéresse aux techniques de migration des cas de test lors de la refonte d'un système.

Au lieu de se contenter de migrer ces deux éléments, il propose une cartographie pour identifier les tests clonés et ceux qui devraient être mis à jour après l'extraction du code. Sur la même thématique, Passier présente une approche qui consiste à suivre les modifications apportées lors des refactorings, à analyser leur utilité sur les suites de tests existantes et à donner des conseils aux développeurs sur la manière de mettre à jour les suites de tests pour les faire migrer. Enfin Mazedur présente lui une architecture pour mieux maitriser la maintenance des suites de tests au format BDD dans les environnements utilisant des micros-services. Même si ici on se rapproche du langage naturel avec le BDD, on reste éloigné de la maintenance de suites de tests rédigées en langage naturel et non reliées spécifiquement à des éléments de code.

Un autre domaine de recherche en lien avec le refactoring des tests est la réduction et la minimisation des suites de tests [40, 4, 99]. S. Yoo présente une étude [99] portant sur la minimisation et la sélection. Ces deux techniques sont très liées et permettent d'identifier les redondances dans les cas de test, et de regrouper les cas de test comprenant des similitudes. Beaucoup de techniques et algorithmes sont présentés, mais ces éléments restent plutôt orientés dans le domaine de la recherche et ne sont pas utilisables tels quels dans l'industrie (sans les adapter par des développements en interne). C'est d'ailleurs sur ce point que Yoo conclut son étude par : "En l'absence d'outils facilement disponibles pour la mise en œuvre des techniques de tests de régression, l'adoption dans la pratique restera limitée."

Khan [40] met en avant qu'aujourd'hui la majorité des outils proposés (82 %) sont orientés uniquement sur la réduction des suites de test sans prendre en compte l'ensemble des autres contraintes qui peuvent toucher les suites de tests (redondance au sein des tests, obsolescence, etc.). Il conclut donc qu'il est important d'apporter plus d'attention aux contraintes des suites de tests et de proposer des solutions polyvalentes pour la réduction de celles-ci.

Enfin plusieurs approches proposent de réduire la taille des suites de tests, mais qu'elles soient basés sur la couverture des besoins [33, 17] ou la similarité des tests [22], toutes comme le refactoring du code des tests, sont majoritairement dédiées aux tests automatisés.

Il existe donc peu de référence au niveau du refactoring des tests en langage naturel dans un référentiel de tests manuels. Borg évoque tout de même cette problématique dans un article [14] où il fait le constat du besoin de restructurer et de réorganiser les tests au niveau des tests d'acceptation du référentiel de test. Mais il traite de la correspondance entre tests d'acceptation automatisés et manuels, et n'évoque pas les problématiques de refactoring du référentiel de tests manuels.

Les références étant rare au niveau du refactoring des cas de test en langage naturel, nous avons orienté nos recherches sur la gestion de la visualisation et du refactoring

des exigences, qui elles sont gérées en langage naturel. Daniel M. Berry [10] et Fabiano Dalpiaz [24] abordent les difficultés et challenges à identifier des redondances, des ambigüités et des incohérences dans les exigences. Pour faire face à ces difficultés, ils présentent des outils basés sur des techniques de NLP (Natural Language Processing) qui permettent d'étudier les exigences et ainsi de les corriger. Un autre champ de recherche intéressant apparenté et parfois combiné avec les techniques de NLP est l'analytique visuelle. L'idée de base de l'analytique visuelle est de représenter visuellement des données afin de permettre à l'utilisateur d'interagir directement avec celles-ci afin d'obtenir rapidement des informations et, en fin de compte, de prendre des décisions optimales sur les données à sa disposition [77]. Ainsi en combinant ces techniques, il est possible dans un premier temps d'obtenir une analyse des données via les techniques de NLP. Puis d'identifier les redondances et autres défauts de manière visuelle à l'aide d'analytique visuelle. Cela permet une manipulation et une analyse plus aisées de celles-ci. Les techniques de NLP et d'analytique visuelle semblant être des éléments se rapprochant le plus du refactoring des suites de test en langage naturel, nous étudions dans la section qui suit comment des outils utilisent ces techniques.

2.3.2/ UTILISATION DES TECHNIQUES ET APPLICATION INDUSTRIELLE

Dans la section précédente, nous avons mis en avant que les techniques étaient plutôt dédiées au refactoring du code des scripts de tests automatisées. Par conséquent, elles ne se prêtent pas à l'étude des cas de test écrits uniquement en langage naturel (qui constitue notre focus sur ce thème). Par contre, les outils basés sur les techniques de NLP ou d'analytique visuelle permettent l'analyse des exigences en langage naturel et aident à leur refactoring. Nous étudions dans cette section différents retours d'expériences sur des techniques et outils de refactoring des exigences.

Fabiano Dalpiaz [24] présente un framework (The Interactive Narrator) qui permet de repérer les ambiguïtés potentielles et les incomplétudes dans les terminologies et désignations utilisées dans les différentes exigences. Son application n'est pas générique, du moins ne pourrait être adaptée au cas de test manuel, car il considère que les exigences ont un format de la forme ”En tant que, Je veux, Afin de”, ce qui n'est pas caractéristique des cas de test. On retrouve le même mécanisme pour d'autres outils (AQUSA software tool, ReCVisu+ [54, 77]). À noter que dans l'évaluation du framework ”The Interactive Narrator” [24], les auteurs suggèrent que leur approche peut conduire à un meilleur résultat qu'une inspection au stylo et au papier, sans qu'aucune différence significative de qualité ne puisse être démontrée.

Sandeep Reddivari et al. présente un outil ReCVisu+ [77] qui permet de faire l'analyse des exigences au moyen de différentes représentations visuelles. L'outil permet notam-

ment le classement des exigences selon différents groupes de fonctionnalités métier. L'utilisateur peut manipuler et raffiner les groupements proposés afin d'étendre l'analyse et ajouter des annotations pour préciser les actions à produire sur les exigences (correction, raffinement, etc.). Par contre, l'utilisateur doit guider l'outil en définissant préalablement des mots clés pour le regroupement. Il ne semble pas possible d'effectuer des corrections directement dans l'outil et ainsi de procéder au refactoring des exigences. Ce constat semble vrai pour plusieurs outils (GATE, AQUSA software tool, ReCVisu+) utilisant l'analytique visuelle, car ils sont plutôt dédiés à l'analyse qu'à la correction des exigences [54, 78, 77]. De plus, il peut être nécessaire de réaliser des développements en interne afin d'utiliser ces outils, notamment pour les adapter aux patterns de la compagnie ou pour permettre l'importation des exigences [78, 54]. Les auteurs indiquent que dans chaque expérimentation qui a été menée, des faux positifs sont apparus, c'est-à-dire des cas pour lesquels l'outil a détecté des défauts alors qu'il n'y en avait pas. Dans ce genre d'outil, l'intervention humaine est nécessaire pour valider l'analyse réalisée [78].

En résumé, les outils d'analyses des exigences regroupent des caractéristiques communes. Quels que soient les techniques et outils utilisés, ils permettent une analyse des exigences, dans la plupart des cas, à travers des tableaux de bords et différents visuels [78, 54, 24]. De manière générale, les résultats présentés, tel que le regroupement des exigences, ne peut pas être sauvegardé pour être appliqué sur les référentiels d'exigences. Des développements en interne doivent être réalisés pour permettre la mise à jour où le réimport des exigences.

2.3.3/ POSITIONNEMENT DES TRAVAUX PAR RAPPORT À L'ÉTAT DE L'ART

L'analyse et le refactoring des suites de test manuels sont des domaines très peu étudiés dans la littérature scientifique. L'analyse du code des tests unitaires et des scripts automatisés est bien présente, mais semble trop éloignée pour être adaptée au langage naturel. Le domaine qui en est le plus proche est celui de l'analyse des exigences. L'identification des défauts dans les exigences, tels que les ambiguïtés et les omissions, est une tâche importante et difficile dans l'ingénierie des exigences. Pour relever ce défi, différents auteurs proposent des outils basés sur des techniques de NLP qui permettent de réaliser une analyse de ces exigences. Un point important relevé dans nos recherches est qu'une intervention humaine pour valider l'analyse produite par les outils de NLP est nécessaire. Car même si des patterns existent, de faux positifs peuvent persister. Cela ne peut être qu'encore plus vrai dans le domaine de l'analyse des cas de test. Contrairement aux exigences, des patterns pré-définis ne peuvent exister pour les suites de test. Les cas de test sont écrits en langage naturel sous des formes extrêmement variées. À notre connaissance, il ne semble pas exister d'outil dédié au refactoring des tests manuels en langage naturel. Il existe donc aujourd'hui un réel défi à proposer des outils dédiés à l'analyse et

au refactoring des tests manuels. Ces outils devront employer des techniques qui permettent d'accompagner au maximum les personnes qui analysent les tests. Il ne faudra pas chercher à mettre en place des techniques complètement autonomes. En effet, l'analyse des exigences a montré que de nombreux faux positifs pouvaient apparaître. Il est vraisemblable que ce soit aussi vrai pour les cas de test manuels. Il faut donc parvenir à mettre en œuvre des mécanismes qui permettent l'accompagnement lors de l'analyse des tests. Cet accompagnement pourrait se faire de manière visuelle, car l'approche fait déjà ses preuves dans l'analyse des exigences. En complément, la modification des cas de test directement dans l'outil et un export facilité vers les outils de gestion de test pourront faciliter l'usage de cet outil.

Dans cette thèse, nous étudions comment répondre aux défis abordés de proposer une approche pour faciliter le refactoring des tests manuels. Nous avons construit l'approche de refactoring en collaborant avec l'équipe de Smartesting en charge de la production de l'outil ORBITER [8], en nous focalisant sur les aspects de méthodologie du refactoring des tests manuels en lien avec notre approche ALME. Ainsi l'approche devra à travers un support outillé permettre l'analyse et le refactoring de suite de tests. En nous basant sur les travaux réalisés en analytique visuelle, nous étudions aussi comment ces suites de tests pourront être visualisées pour faciliter leur analyse et leur refactoring.

2.4/ AUTOMATISATION DE L'EXÉCUTION DES TESTS

L'automatisation de l'exécution des tests reste actuellement complexe et coûteuse en effort des ingénieurs logiciels [88]. Son objectif est de contribuer à la qualité de l'application à travers les différentes itérations, notamment dans les contextes Agile, où les livraisons sont régulières et où le temps pour tester manuellement croît au fil des sprints. Automatiser un ensemble de cas de test permet d'alléger la charge de test manuel [58] et facilite la détection de régressions. On retrouve, entre autres, ces objectifs dans le syllabus Advanced Test Automation Engineer de l'ISTQB [11] où on précise que l'automatisation améliore l'efficacité des tests, réduit le temps d'exécution, etc. D'après ce syllabus, l'automatisation se fait selon différentes couches qui sont les suivantes :

- la génération (manuelle ou automatisée) des tests ;
- la définition des tests (par le choix des données) ;
- l'éxécution des tests permettant d'établir les résultats de l'exécution des tests ;
- l'adaptation des tests qui regroupe toutes les tâches permettant de faire cohabiter et fonctionner ensemble les divers éléments nécessaires à l'automatisation : les scripts, l'environnement d'exécution le SUT, etc.

Dans nos recherches, nous nous orientons sur la génération des tests et la définition des tests, pour les tests fonctionnels à automatiser. Nous étudions en particulier les techniques permettant d'adapter des cas de test destinés à une exécution manuelle en scripts automatisés. Nous nous concentrons particulièrement sur ces phases car elles peuvent s'inscrire dans les approches de MBT [68] et faire partie intégrante de notre approche ALME.

2.4.1/ TECHNIQUES D'AUTOMATISATION

La génération des tests apporte un support outillé pour la conception manuelle des cas de test, le développement, la capture et la dérivation des données de test et la génération automatique des cas de test [11]. La production de scripts de test fait partie des activités pour l'automatisation des tests. Elle consiste de manière manuelle ou automatisée à produire des scripts de test. Les scripts peuvent être produits manuellement sous forme de séquence de mots-clés ou d'appels à des sous-programmes exécutables, avec des paramètres [90]. Une autre alternative consiste à enregistrer les interactions du testeur avec le système sous test, pendant que le testeur effectue un test manuel. Ensuite ce script de tests automatisés pourra être rejoué. Cette approche est connue sous le nom de "capture/rejeu" [11] et est largement abordé dans la littérature [51, 88, 90].

2.4.2/ CAPTURE/REJEU

La forte présence de la technique de "capture/rejeu" dans la littérature peut s'expliquer par le fait qu'elle semble de prime abord présenter de meilleurs résultats que la production manuelle des scripts. Leotta et. Al [51] présentent en effet qu'une approche manuelle consomme de 32 à 112 % de temps supplémentaire pour le développement de suites de tests vis-à-vis d'une approche de "capture/rejeu". Par contre, la maintenance des scripts avec des techniques de "capture/rejeu" s'avère de 16 à 51 % plus coûteuse qu'une approche par écriture manuelle des scripts. Ce faisant, au fil des itérations les coûts cumulés d'une approche manuelle d'écriture des scripts sont inférieurs aux coûts cumulés pour le "capture/rejeu". Toutefois, le "capture/rejeu" présente des avantages comparés à une approche manuelle, notamment sa facilité à obtenir des scripts de test sans avoir de compétences techniques en codage. Les testeurs se contentent de vérifier l'application web testée et d'enregistrer leurs actions [5]. L'inconvénient principal de cette technique est donc plutôt lié à la maintenance des scripts qu'à leur production. Selon la technique utilisée, la production ou la maintenance de scripts de test sont des activités coûteuses dans le processus d'automatisation [88]. C'est pour cette raison que des auteurs comme

5. Une définition de ces termes sont définies en Annexe A.1

Haruto Tanno, Xiaojing Zhang [88] étudient des techniques pour réduire le coût de production et de maintenance des scripts de test. Dans un article, ils présentent notamment comment ils utilisent (entre autre) des techniques basées sur des mots-clés pour diminuer le coût de production et de maintenance des scripts automatisés. Cette technique étant répandue dans la littérature [88, 68, 35], mais aussi dans la pratique industrielle, nous approfondissons nos recherches sur celle-ci dans la section suivante.

2.4.3/ KEYWORD DRIVEN TESTING

Les techniques de test basé sur des mots-clés associent des mots-clés à des comportements du SUT et peuvent enchaîner une ou plusieurs étapes de test. Des exemples de mots-clés sont ”Se connecter”, ”Se rendre sur la page d'inscription”, etc. L'enchaînement de ces mots-clés forme les scripts de test. Selon la granularité des mots-clés, les scripts de test seront plus ou moins abstraits [68].

He et al. [35] énoncent que le Keyword Driven Testing (KTD) peut se résumer par les trois aspects suivants : une couche d'adaptation, la séparation de la description des tests et de leur implémentation et la séparation des données des scripts pour l'exécution. Selon eux ces trois aspects sont indépendants et cela réduit donc les influences mutuelles entre eux. Pour résumer, l'application d'une approche de KTD se réalise selon une série d'étapes indépendantes :

- La construction ou la génération des scénarios de test.
- La complétion de la couche d'adaptation :
  - par la création de bibliothèques de mots-clés ;
  - par la création de la bibliothèque de données de test.
- La production des scripts en assurant une liaison entre les scénarios de test abstraits et les scripts de test.

Les auteurs attribuent de nombreux avantages à l'usage du KDT. Ils parlent notamment des bénéfices de l'abstraction des mots-clés car elle réduit l'effort de maintenance des tests et facilite la construction et la compréhension des scripts [34]. Un autre avantage est que l'utilisation de mots-clés ne nécessite pas de connaissances particulières : si on sait qu'il existe un mot-clé permettant de produire le comportement de connexion à l'application, le testeur n'a pas besoin de comprendre le code qui le compose pour l'utiliser [11].

Enfin on peut produire des scripts de test à partir d'approches comme le MBT. Les tests basés sur des modèles (MBT) peuvent inclure une approche par mots-clés (KDT) pour

la génération automatique des scripts [68, 88]. Nous détaillons l'association de ces deux techniques dans la section qui suit.

2.4.4/ MBT ET KDT AU TRAVERS DES OUTILS

Le MBT permet la génération de cas de test abstraits et le KDT fournit les éléments nécessaires pour la complétion de la couche d'adaptation. Ensuite c'est l'association de ces deux techniques qui permettent la génération des scripts de test. Tanno et Al. [88] proposent un outil qui combine MBT et KDT. Ils proposent une approche pour générer automatiquement des scripts de test à partir de modèles. Pour cela, ils étendent un outil de MBT existant TesMA, dans lequel ils ont ajouté des liens avec des informations physiques pour permettre la génération des scripts. Ces liens se trouvent au niveau des modèles et des cas de test permettant ainsi la génération de script. À notre connaissance, aucun retour d'expérience sur cet outil n'a été établi dans le milieu industriel, du moins utilisé par des acteurs industriels.

Pajunen et Al. [68] présentent eux une approche où ils utilisent un outil de MBT TEMA qui permet l'exécution de scripts basés sur des keywords avec Robot Framework. Robot Framework est un framework générique utilisant des mots-clés et des librairies sur des domaines variés. La suite d'outils TEMA est une suite d'outils de MBT dédiée aux tests d'Interface Homme Machine (IHM). Comme pour Tanno et Al. [88], ils isolent d'une part les aspects concrets et d'autre part les aspects abstraits du SUT. Ici ils font ces abstractions à l'aide des machines à états. À noter qu'ici aussi, nous n'avons pas trouvé d'application industrielle.

Sivanandan et Al. [83] proposent eux aussi une approche associant MBT et KDT. Ils utilisent GraphWalker, un outil permettant de générer des séquences de test à partir de machines à états. Pour la partie automatisation, les auteurs proposent comme Pajunen et Al. [68] l'usage de Robot Framework. Lors de l'exécution du test, les mots-clés exécutés sont sélectionnés sur la base d'un modèle de configuration qui fait le lien entre les étapes de test et les mots-clés. Comme pour les autres outils présentés [88, 68] nous n'avons pas identifié d'application par des acteurs industriels.

En résumé, la combinaison du MBT et du KDT pour l'automatisation est basée sur le fait que le MBT fournit les cas tests et le KDT les éléments nécessaires à l'exécution. Bien que ces deux techniques semblent apporter un support à la production des cas de test et à l'automatisation, peu de retours industriels semblent exister. Nous avons présenté dans la section 2.1 que la faible dissémination du MBT pouvait s'expliquer par la non-correspondance initiale des principes du MBT avec ceux des préceptes Agiles. Nous avons étudié de ce fait si l'automatisation pouvait aussi faire face aux mêmes challenges que le MBT dans les contextes industriels. Collins et Al. [21] sur le sujet on étudié l'au-

tomatisation des tests dans des contextes Agiles. Parmi leurs conclusions figurent des recommandations similaires à celles du MBT. Les techniques d'automatisation doivent s'adapter aux contextes Agiles en proposant une production simplifiée des scripts automatisés. La proposition d'outils accessibles aux membres de l'équipe avec une courbe d'apprentissage adaptée au cycle court devrait aider à leur usage dans ces contextes.

2.4.5/ POSITIONNEMENT DES TRAVAUX PAR RAPPORT À L'ÉTAT DE L'ART

Dans cette section, nous avons étudié l'automatisation de l'exécution des tests. Nous nous intéressons particulièrement à la génération et à la définition des tests automatisés, qui sont des activités pouvant interagir avec les techniques de MBT. Nos recherches nous ont montré que l'automatisation est un point clé dans la réussite des projets et encore plus dans les contextes Agiles. Cependant, comme pour le MBT, l'adaptation de l'automatisation aux contextes Agiles n'est pas nécessairement aisée. De plus en plus, les auteurs tendent vers la simplification de l'automatisation pour répondre aux pratiques légères de l'Agilité. Cependant, des progrès doivent être réalisés pour réduire le temps d'apprentissage des techniques et des outils d'automatisation et les rendre plus accessibles aux personnes qui ne les maîtrisent pas. En plus de cela, la maintenance des scripts doit être facilitée, car elle est encore aujourd'hui un point de difficulté [72]. Dans cette thèse, nous abordons ces défis en proposant une approche légère de MBT qui s'interface avec les frameworks d'automatisation basés sur KDT tels que Robot Framework.

II

Contributions Techniques et Expérimentations

3

CONTRIBUTIONS TECHNIQUES : APPROCHE ALME

3.1/ INTRODUCTION

Comme nous l'avons vu dans le chapitre précèdent, la pratique du Model-Based Testing a commencé à se développer dans l'industrie à partir du début des années 2000, mais son essor reste aujourd'hui faible.

Cela s'explique en particulier par la barrière de la modélisation et par l'adaptation difficile du MBT aux contextes du développement logiciel en Agile.

Historiquement, les approches MBT se sont d'abord fondées sur des formalismes de modélisation permettant de représenter les comportements du système sous test, par exemple, avec des diagrammes états-transitions en UML. La complexité de ces modélisations comportementales détaillées est forte et induit une courbe d'apprentissage longue, ce qui représente un frein important pour l'appropriation des techniques et outils MBT par les testeurs.

Mais la complexité de la modélisation est aussi un frein pour son utilisation en Agile. Les cycles courts des contextes en Agile ne se prêtent pas à une modélisation détaillée du système sous test, car il s'agit de focaliser les tests sur chaque itération et de les faire évoluer de façon incrémentale (ce que ne facilite pas un modèle comportemental complexe).

Ce sont ces challenges que nous adressons dans l'approche du MBT proposée dans cette thèse, en nous focalisant sur le domaine des applications du SI (le logiciel Métier d'entreprise). Nous avons appelé cette approche ALME – *Agile Lightweight Model-Based Testing for Enterprise IT* – pour mettre en avant l'aspect "léger" de la modélisation proposée. Cette volonté d'alléger la modélisation MBT s'appuie sur trois piliers :

- une modélisation orientée workflow (de type processus métier) pour permettre aux

testeurs de penser en termes de parcours applicatifs, c'est à dire de description du flot des actions à réaliser sur le système sous test en vue de le tester ;

- une adaptation au domaine ciblé, car les applications du SI d'entreprise sont testées usuellement avec des scénarios d'usage, c'est à dire de parcours applicatifs à partir des interfaces du système (IH/M ou API) ;
- une notation n'offrant que très peu d'éléments graphiques pour la modélisation de façon à en diminuer la courbe d'apprentissage et à permettre au concepteur des tests de conserver la focalisation sur la scénarisation visuelle des tests lors de leur conception.

L'approche ALME apporte un changement de paradigme par rapport aux concepts historiques du MBT : il s'agit de supporter/faciliter la scénarisation visuelle des tests plutôt que la modélisation du système sous test.

La modèlisation dans l'approche ALME est donc plus proche d'une modélisation des tests que d'une modélisation du système. De plus, il s'agit de promouvoir le travail collaboratif autour de cette scénarisation des tests entre les parties prenantes orientées Métier – PO et analystes métier –, les testeurs et les développeurs - pour permettre l'alignement de l'équipe Agile à l'aide des scénarios de test. Cette approche collaborative correspond à des pratiques du développement logiciel appelées ATDD (Acceptance Test Driven Development) et BDD (Behavior Driven Development) que nous présentons dans la section suivante.

Les travaux réalisés dans cette thèse, dans le cadre d'une collaboration entre la recherche et l'industrie qui a permis des expérimentations de terrain, visent à établir si ce changement de paradigme répond aux besoins des professionnels des tests dans le contexte visé. Pour en permettre la discussion, nous en avons formulé la problématique autour de trois questions de recherche :

- RQ-1 : Dans quelle mesure l'approche ALME facilite la scénarisation des tests fonctionnels dans le contexte des applications du SI ?
- RQ-2 : Dans quelle mesure l'approche ALME peut être adoptée par les praticiens, testeurs fonctionnels de profession ?
- RQ-3 : Dans quelle mesure l'approche ALME est-elle adaptée aux contextes en Agile ?

Les sections suivantes de ce chapitre présentent l'approche ALME au travers du processus d'ATTD visuel que notre approche permet de mettre en oeuvre au sein d'une équipe Agile.

3.2/ PROCESSUS D'ATDD VISUEL PAR LA SCÉNARISATION DES TESTS

Les contextes en Agile ont pour caractéristiques, entre autres, de travailler avec des cycles itératifs et incrémentaux, de tester tôt et de privilégier les individus et leurs interactions. Pour ce faire, des pratiques telles que le BDD (Behavior Driven Development) et l'ATDD (Acceptance Test Driven Development) ont été définies pour faciliter la collaboration entre les parties prenantes de la solution autour des tests d'acceptation.

La différence qui peut être faite entre ces deux pratiques est que le BDD se concentre sur les comportements indépendants à un niveau fin de granularité (typiquement au niveau d'une User Story) et une formulation des scénarios de test dans le langage textuel dédié Gherkin [1] au format Given/When/Then.

Le terme ATDD est issu initialement d'une adaptation de la pratique de tests unitaires TDD - Test Driven Development - au niveau des tests d'acceptation [44]. Cela recouvre des pratiques assez hétérogènes mettant en avant la création collaborative des scénarios de test d'acceptation dès la phase d'expression du besoin lors d'ateliers impliquant PO / Analystes Métier, Testeurs et Développeurs [62]. En ATDD, les scénarios de test d'acceptation peuvent être formulés de façon informelle (c'est à dire en langage naturel). Dans l'approche ALME, nous définissons une pratique de l'ATDD fondée sur la représentation graphique des parcours applicatifs à tester que nous appelons par le vocable ATDD visuel.

ATDD VISUEL

Notre approche est fondée sur la *scénarisation visuelle des tests* [2] fonctionnels par le biais de parcours applicatifs graphiques (de modèles). Ces parcours applicatifs graphiques constituent une abstraction d'un ensemble de scénarios de test, discutés entre les différents acteurs de la solution. Il s'agit de faciliter les interactions entre les membres de l'équipe lors du raffinement progressif de l'expression du besoin par la discussion des scénarios de test d'acceptation.

La construction de notre approche est proche de la façon dont le BDD s'articule. Dans la figure 3.1 (à gauche) est présentée le cycle du BDD tel que proposé par Cucumber [3], qui est un framework très répandu pour l'implémentation du BDD. Pour Cucumber, les entrants à l'approche sont les User Stories qui vont d'abord être découvertes, puis dont les critères d'acceptation sont formulés sous la forme de scénario de test au format "Gi-

1. https://cucumber.io/docs/gherkin/ [Dernière visite : décembre 2020]
2. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1
3. https://cucumber.io/ [Dernière visite : décembre 2020]

ven/When/Then". Cela vise aussi à faciliter leur automatisation par l'utilisation de mots clés. Il est possible de revenir à une étape précédente du cycle si des défauts ou des changements apparaissent.

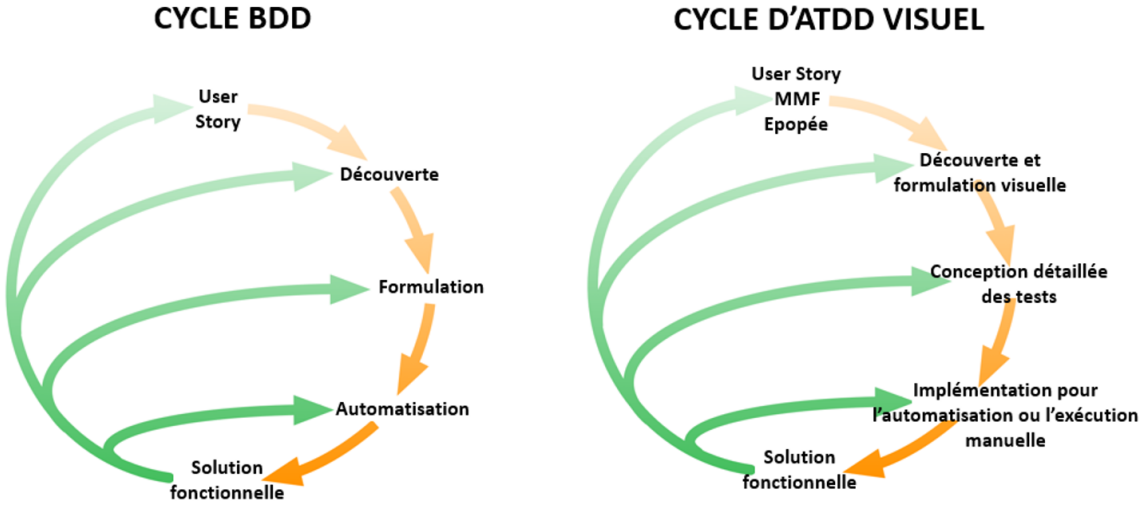


FIGURE 3.1 – Représentation des cycles de BDD (à gauche) et d'ATDD visuel (à droite)

Notre approche, visible dans la figure 3.1 (à droite), se distingue tout d'abord du BDD par le fait qu'elle ne prend pas seulement en compte les User Stories mais aussi des éléments d'expression du besoin de plus haut niveau tels que des épopées représentant des fonctionnalités métier majeures ou des MMF (minimum marketable feature, fonctionnalités minimales commercialisables). Ainsi, nous pouvons couvrir un périmètre ciblé par le biais des User Stories ou un périmètre plus large par le biais des épopées et des MMF. Comme pour le BDD, le cycle de l'ATDD visuel débute par la découverte du besoin et la formulation des scénarios de test d'acceptation.

Une autre différence avec le BDD provient de l'usage de la modélisation qui intervient dès les phases de découverte et de formulation. La représentation des scénarios de test d'acceptation sous la forme de workflows permet de représenter visuellement comment s'articule une fonctionnalité au sein du flot métier et comment chaque fonctionnalité interagit avec les autres.

Enfin, comme pour le BDD, nous pouvons choisir d'automatiser l'exécution des tests ou de rester sur une exécution manuelle des tests. L'automatisation de l'exécution est traitée en détail dans le chapitre 4. Dans ce chapitre, nous considérons que l'exécution des tests peut être réalisée manuellement par un testeur ou automatisée.

À noter que si durant l'exécution des tests, des défauts sont relevés, on peut revenir à n'importe quelle étape de ce cycle d'ATDD visuel. On peut aussi revenir à l'une des premières phases si l'on souhaite implémenter un changement sur les tests produits.

Afin d'étudier et d'expérimenter notre approche en contexte de projet, nous avons tra-

vaillé avec l'outil Yest de Smartesting [4]. Le choix de cet outil s'explique par la longue collaboration existante entre l'éditeur de ce produit et le laboratoire FEMTO-ST. D'une part, l'éditeur nous a permis de mettre en pratique notre approche avec son outil. Et d'autre part, les expérimentations réalisées ont aussi mené à l'évolution de l'outil afin de mieux répondre aux enjeux des tests en Agile qui ont été révélés par cette thèse.

Mais, bien que notre approche soit illustrée dans cette thèse avec un outil particulier, elle peut être appliquée avec d'autres outils MBT, tels que TestOptimal [5] et TestModeller.io [6] par exemple, qui permettent d'appliquer l'approche ALME et les bonnes pratiques que nous présentons plus avant.

La suite de ce chapitre présente les différents aspects de l'approche ALME avec un focus méthodologique et organisationnel (étapes du cycle de l'ATDD visuel, notation de modélisation, rôles intervenants dans l'approche et artefacts produits) pour ensuite en formuler les bonnes pratiques issues de notre expérience.

3.2.1/ Découverte des besoins métier et formulation visuelle des scénarios de tests d'acceptation

Dans notre approche, nous commençons par identifier les entrants qui sont les User Stories, les MMF ou bien les épopées. Cela défini, pour une itération donnée, les évolutions à traiter.

Ensuite, vient la phase de découverte des différents besoins métier identifiés et de formulation graphique des scénarios d'acceptation. L'expression du besoin est généralement réalisée en Agile de façon peu formelle et peu documentée, sous la forme de court texte, de cartes, de tickets, généralement disponibles dans des outils de gestion de projet Agile comme Jira [7], ou autres. La formulation des scénarios d'acceptation en ATDD visuel s'appuie sur une modélisation des parcours applicatifs à tester.

Notation graphique ALME pour la scénarisation des tests

La phase de formulation visuelle des scénarios de test d'acceptation contribue à renforcer la compréhension des besoins métier, tout en représentant ce que l'on cherche à vérifier. Pour cela, il s'agit de modéliser les parcours applicatifs à tester en les décrivant de façon aussi simple que possible en fonction des objectifs du test. Dans la figure 3.2, nous précisons les éléments graphiques de la notation de modélisation utilisée.

4. https://www.smartesting.com/conception-collaborative-tests-yest [Dernière visite : décembre 2020]
5. https://testoptimal.com/ [Dernière visite : décembre 2020]
6. https://testmodeller.io/ [Dernière visite : décembre 2020]
7. https://www.atlassian.com/fr/software/jira [Dernière visite : décembre 2020]

Artefact	Description
	***Point de début:*** *marque le début du parcours applicatif*
	***Tâche:*** *décris les actions à réaliser sur le SUT pour vérifier un comportement et son résultat attendu.*
	***Point de choix:*** *représente les différents chemins possibles*
	***Point de fin:*** *marque la fin du parcours applicatif*
	***Sous-parcours:*** *intègre un parcours dans un parcours existant*
	***Groupement:*** *représente un sous ensemble*
	***Annotation:*** *permette d'écrire du texte libre*

FIGURE 3.2 – Notation allégée pour les représentations graphiques

Du point de vue syntaxique et sémantique, cette notation constitue un sous-ensemble de la notation BPMN 2.0 [8], n'utilisant que septs types d'artefacts là où la version complète de BPMN en propose plus de 50. Comme présenté au début de ce chapitre, il s'agit de conserver le focus sur la scénarisation des tests avec un minimum d'éléments graphiques.

Dans ce sous-ensemble, nous avons considéré un nombre limité de nœuds : le point de départ, les tâches, les points de choix, les points de fin, les sous-chemins, les regroupements et les annotations. Les nœuds sont reliés par un connecteur indiquant le flot du workflow. Les tâches décrivent les actions sur le SUT tandis que les points de choix contrôlent le flot dans le parcours applicatif.

Un sous-parcours est utilisé pour introduire un nouveau parcours en tant que nœud du parcours en cours. Le sous-parcours a son propre diagramme de flot. Il permet d'introduire une décomposition hiérarchique descendante du parcours qui facilite la manipulation du workflow. Enfin, les groupements et les annotations sont présents pour clarifier le modèle en définissant des zones spécifiques dans le parcours applicatif et en ajoutant du texte libre si nécessaire. Ils peuvent aussi bien représenter des acteurs que des fonctionnalités, etc.

Ce nombre limité d'artefacts permet la formulation visuelle des scénarios de test en limi-

8. Notation BPMN 2.0, disponible sur le site de l'OMG https://www.omg.org/spec/BPMN/2.0/

tant la complexité de la modélisation. Cela vise à être accessible aux testeurs fonctionnels pour leur permettre de représenter de manière abstraite les scénarios de test d'acceptation. Un point de début permet de définir le départ du parcours et un point de fin permet d'y mettre fin. On ne propose qu'un type de noeud pour décrire les actions métier, et un pour définir des points de choix. Des sous-parcours peuvent être utilisés pour limiter la complexité visuelle des modèles et permettre à la fois de vérifier des comportements métiers particuliers et des propriétés transverses du SUT.

L'encadré à droite dans la figure 3.3 illustre un exemple d'usage des artefacts de modélisation.

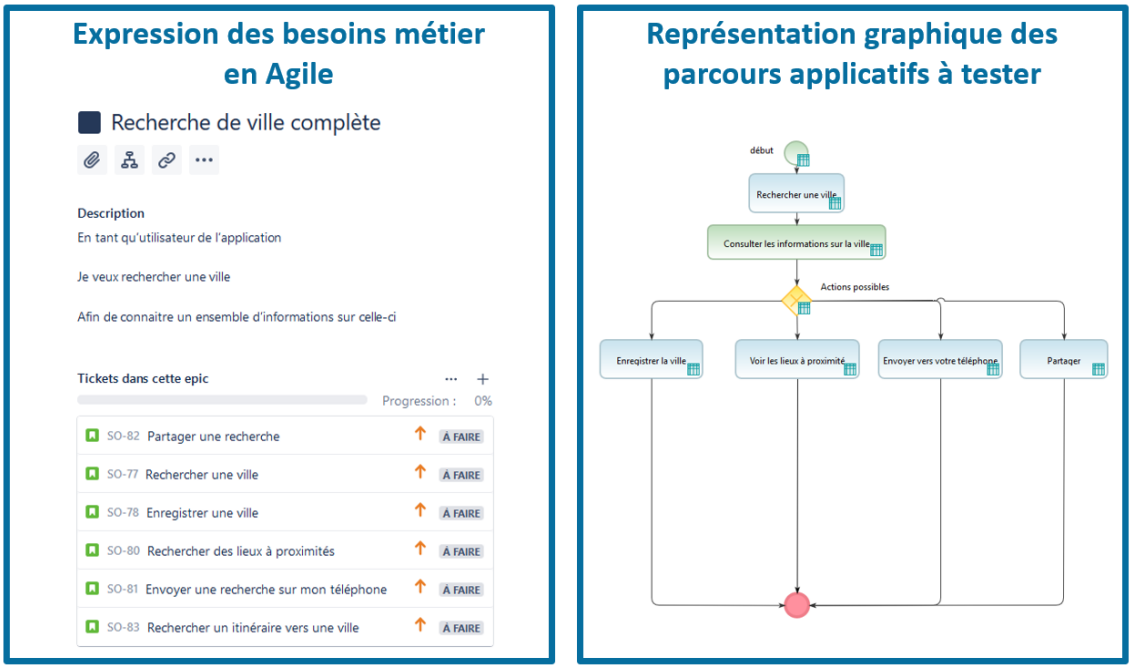


FIGURE 3.3 – Représentation de l'expression des besoins métier en Agile (à gauche) et modélisation des parcours applicatifs correspondant (à droite)

Dans l'encadré de gauche de la figure 3.3 est présenté un formalisme courant en Agile d'expression et de formulation du besoin à l'aide de User Stories (US). Les US sont listées et liées à une épopée par des tickets avec l'outil Jira (déja cité précédemment). À droite est présenté notre approche visuelle de la scénarisation des tests d'acceptation.

L'expression atomique du besoin sous la forme des Users Stories ne permet pas nativement de connaître le lien qui existe entre les différentes fonctionnalités. Mais, les parcours applicatifs modélisés permettent de visualiser quels sont les activités indépendantes les unes des autres et celles qui sont liées. Ainsi dans l'exemple de la figure 3.3, les fonctionnalités "Enregistrer la ville", "Voir les lieux à proximité", "Envoyer vers votre téléphone" et "Partager" ne sont accessibles qu'après la réalisation de l'action "Rechercher une ville". L'approche ALME par ses formalisations visuelles permet de donner une vision globale

des parcours applicatifs que l'on va souhaiter tester (ici illustré sur un exemple de recherche de ville et d'itinéraire).

Pour résumer, l'approche ALME est fondée sur une notation de workflow qui reprend au niveau syntaxique et sémantique un (petit) sous-ensemble de la notation BPMN 2.0. Sa vocation est la scénarisation graphique des tests fonctionnels, sous la forme de workflows décrivant des parcours applicatifs. Ainsi un modèle ALME décrit un ensemble de scénarios de test sous une forme abstraite et graphique. Ces représentations visuelles contribuent à clarifier l'expression du besoin (qui est généralement peu formalisée en Agile) car les scénarios de test décrits de façon abstraite dans les modèles ALME définissent des cas d'usage possibles du système en cours de développement. Les modèles ALME ne remplacent pas la description textuelle des User Stories, mais constituent un complément d'information pour l'équipe. Les modèles ALME sont revus lors d'ateliers impliquant les parties prenantes fonctionnelles Métier (PO et analystes métier), les testeurs et les développeurs pour permettre d'aligner les contributeurs du projet sur une même compréhension du besoin.

Une fois l'ensemble des rôles en accord sur les parcours applicatifs à tester, arrive dans le cycle de l'ATDD visuel, l'étape de conception détaillée des tests.

3.2.2/ CONCEPTION DÉTAILLÉE DES TESTS

Après avoir représenté les besoins métier au moyen de parcours applicatifs graphiques, nous proposons dans notre approche d'effectuer une conception détaillée des tests. Cette conception détaillée consiste à spécifier toutes les informations nécessaires pour les tests. Cela implique la définition des actions, des résultats attendus [9] et si besoin des données abstraites pour définir les conditions permettant de tester les règles métier. Les actions et les résultats attendus sont définis au niveau de chacune des tâches sur les modèles.

Dans la figure 3.3, la première action "Rechercher une ville" peut être définie avec l'action détaillée "Entrer une ville dans le champ de recherche et cliquer sur le bouton rechercher" et comme résultat attendu "Les détails sur la ville sont affichés". La définition des actions de test et des résultats attendus ainsi que le contrôle des règles métier sont des activités habituelles des testeurs fonctionnels.

Selon les outils MBT utilisés, la manière de compléter ces informations varie, généralement ceci est possible par la complétion de tables ou de propriétés associées à chaque tâche du modèle. La figure 3.4 présente trois outils permettant la conception détaillée des tests telle que nous la préconisons dans notre approche, c'est à dire en

9. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

restant simple et accessible pour les testeurs fonctionnels.

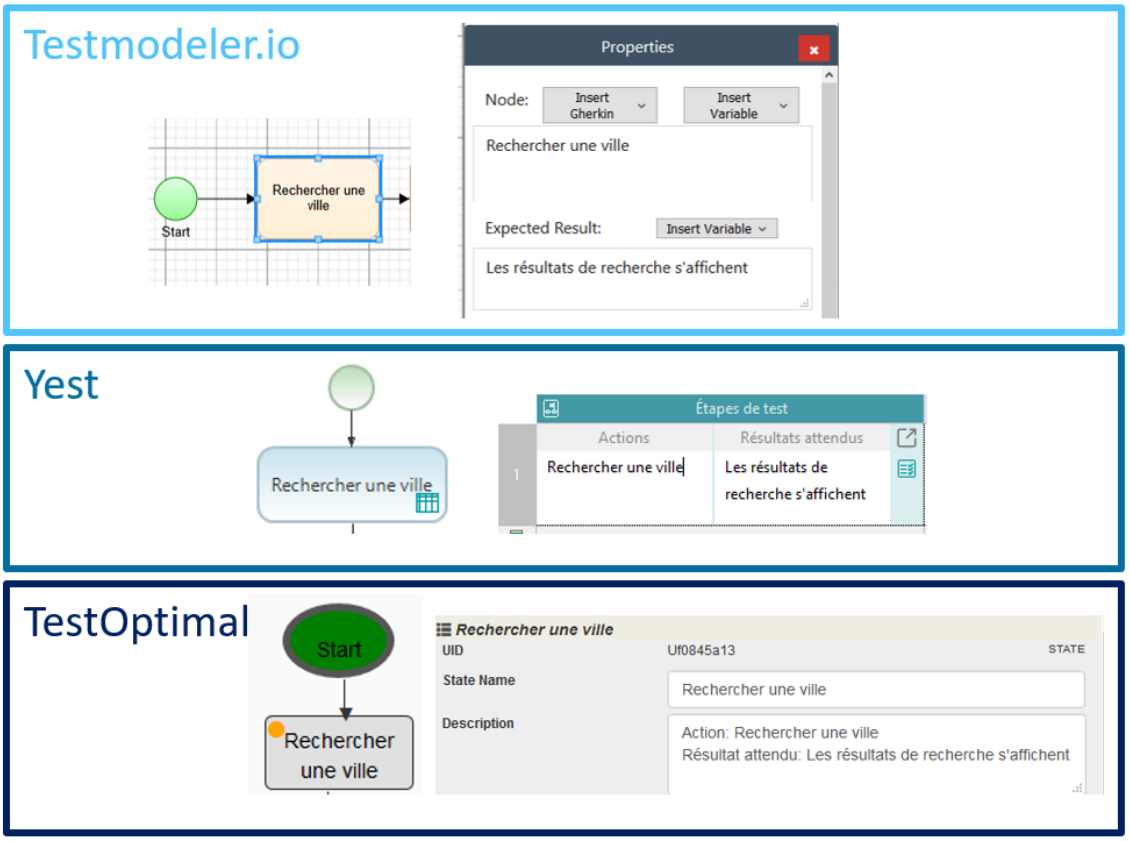


FIGURE 3.4 – Conception détaillée des tests selon différents outils

En phase de conception détaillée des tests, l'objectif est de réaliser la couverture des conditions de test avec des données abstraites. Ces données de test abstraites sont appelées données de conception [10], car elles interviennent durant la phase de conception des tests et sont utilisées dans les modèles. Elles peuvent être créées à l'aide de différents mécanismes tels que des tables, des fichiers Excel ou d'autres moyens permettant leur instanciation ultérieure en données de test concrètes. La concrétisation des données de test fait partie de la phase suivante d'implémentation des tests. Il est mieux de ne définir que les données nécessaires, c'est-à-dire celles qui découlent des exigences ou qui sont utiles pour gérer les flots métier dans les parcours applicatifs. Ceci permet de limiter le nombre de données de conception de test à maintenir et de se focaliser sur les objectifs de test.

10. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

Bonne pratique

Choix des données de conception de test : dans la phase de conception détaillée des tests, seules des données abstraites en lien avec les règles métier doivent être intégrées.

BONNE PRATIQUE 1 – Choix des données de conception de test

Prenons l'exemple d'une application de e-commerce qui propose des bons d'achat en fonction de la valeur du panier client. Les règles métier sont les suivantes :

- Si le montant du panier est de moins de 30€, le client n'obtient pas de bon d'achat.
- Si le montant du panier est entre 30 et 50€, le client obtient un bon d'achat de 5€.
- Si le montant du panier est entre 50 et 100€, le client obtient un bon d'achat représentant 10% de la valeur du panier.
- Si le montant du panier est de plus de 100€, le client obtient un bon d'achat de 10€.

Les données à faire figurer durant la phase de conception détaillée des tests sont celles qui traitent de règles métier. Dans notre exemple, ce sont les plages du montant du panier et les montants des bons d'achat associés. La figure 3.5 illustre une réalisation avec l'outil Yest, à l'aide d'une table spécifiant les conditions de test sur des données abstraites.

	montant_panier	montant_bon	Étapes de test	
1	*moins de 30€*	*pas de bon*	Pour un panier de moins de 30€ Vérifier le montant du bon d'achat	Le bon d'achat est de 0€
2	*entre 30 et 50€*	*un bon de 5€*	Pour un panier entre 30 et 50€ Vérifier le montant du bon d'achat	Le bon d'achat est de 5€
3	*entre 50 et 100€*	*10% du montant du panier*	Pour un panier de entre 50 et 100€ Vérifier le montant du bon d'achat	Le bon d'achat représente 10% du montant du panier
4	*plus de 100€*	*un bon de 10€*	Pour un panier de plus de 100€ Vérifier le montant du bon d'achat	Le bon d'achat est de 10€

FIGURE 3.5 – Exemple de donnees de conception de test

Ici chaque exigence figure sur une ligne. Avec d'autres outils, ceci pourra se faire par la définition de contraintes ou d'autres mécanismes.

Les modèles de parcours applicatifs complétés par la description des étapes de test et la couverture des conditions de tests sur les données de conception permettent de générer les cas de test abstraits. Pour un testeur connaissant bien le système sous test, ces cas

de test abstraits peuvent suffire pour guider l'exécution des tests. Il/elle sera à même de compléter les données de test manquantes ou trop abstraites.

Mais dans le cas général, définir les valeurs concrètes des données de test nécessaires à l'exécution permettra une formulation plus précise des cas de test. Dans notre exemple, l'application n'a pas de champ où la valeur "moins de 30€" apparaît, mais plutôt "5€", "10€", etc. Ces valeurs attendues par l'application sont ce que nous appelons les données d'implémentation des tests[11]. Elles remplacent les valeurs abstraites des données par des valeurs concrètes interprétables par le SUT. Les données d'implémentation facilitent ainsi l'exécution manuelle, et elles sont nécessaires pour l'exécution automatisée.

La section suivante présente les activités d'implémentation des tests dans ALME pour l'exécution manuelle et l'automatisation.

3.2.3/ IMPLÉMENTATION POUR L'ÉXÉCUTION MANUELLE ET L'AUTOMATISATION

Une fois les cas de test conçus, l'objectif de l'implémentation des tests est de permettre l'exécution manuelle ou automatisée des tests. Les cas de test manuels et les scripts automatisés partagent la transcription des données de conception vers les données concrètes d'implémentation, qui consiste à faire correspondre toutes les données abstraites des modèles avec des valeurs concrètes.

Reprenons l'exemple de l'application de e-commerce introduit précédemment (en 3.2.2) et complétons le pour réaliser la transcription des données de conception en données d'implémentation. Les valeurs de données de conception exhibées pour le montant du panier sont "moins de 30€", "entre 30 et 50€" , "entre 50 et 100€" et "plus de 100€". Pour la donnée de conception sur le montant du bon d'achat à obtenir, les valeurs sont : "pas de bon", "un bon de 5€", "10% du montant du panier" et "un bon de 10€".

Pour faire correspondre les données de conception aux données d'implémentation, différents mécanismes sont possibles. Dans notre approche, nous préconisons de faire ce rapprochement à l'aide de tableaux, car cela est actuellement déjà un usage habituel pour les testeurs fonctionnels dans leur travail sur les données de test. Les tableaux de données sont aussi utilisés dans les différents outils de MBT sous une forme similaire. De plus, les tableaux sont souvent utilisés pour extraire des données de test à partir de sources existantes, telles que des bases de données de test ou de production.

11. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

Bonne pratique

Correspondance des données de conception et d'implémentation : utiliser des tableaux pour associer à chaque donnée de conception, sa ou ses données d'implémentation.

BONNE PRATIQUE 2 – Données de conception et d'implémentation

Dans la figure 3.6 nous présentons différentes formes de tableaux utilisées dans deux outils de MBT.

Yest

	Nom du jeu de données	montant_panier	montant_panier_SUT	montant_bon	montant_bon_SUT
1	**Jdd panier moins de 30€ [1]**	*moins de 30€*	*1*	*pas de bon*	*0*
2	**Jdd panier moins de 30€ [29]**	*moins de 30€*	*29*	*pas de bon*	*0*
3	**Jdd panier entre 30 et 50€ [30]**	*entre 30 et 50€*	*30*	*un bon de 5€*	*5*
4	**Jdd panier entre 30 et 50€ [31]**	*entre 30 et 50€*	*31*	*un bon de 5€*	*5*
5	**Jdd panier entre 30 et 50€ [49]**	*entre 30 et 50€*	*49*	*un bon de 5€*	*5*
6	**Jdd panier entre 30 et 50€[50]**	*entre 30 et 50€*	*50*	*un bon de 5€*	*5*
7	**Jdd panier entre entre 50 et 100€[51]**	*entre 50 et 100€*	*51*	*10% du montant du panier*	*5,1*
8	**Jdd panier entre entre 50 et 100€[99]**	*entre 50 et 100€*	*99*	*10% du montant du panier*	*9,9*
9	**Jdd panier plus de 100€[100]**	*plus de 100€*	*100*	*un bon de 10€*	*10*
10	**Jdd panier plus de 100€[101]**	*plus de 100€*	*101*	*un bon de 10€*	*10*
11	**Jdd panier plus de 100€[200]**	*plus de 100€*	*200*	*un bon de 10€*	*10*

TestOptimal

jdd_valeur_panier NEW_DS_11

Variables (width 165 px)

Name	montant_panier	montant_panier_SUT	montant_bon	montant_bon_SUT
Domain	moins de 30€ entre 30 et 50€ entre 50 et 100€ plus de 100€	1 29 30 31 49 50 51 99	pas de bon un bon de 5€ 10% du montant du panier un bon de 10€	0 5 5,1 9,9 10
Data Type	int float **txt** bool	**int** float txt bool	int float **txt** bool	**int** float txt bool
Derived?	☐	☐	☐	☐
Coupling				

Data Table (rows: 11)

	montant_panier	montant_panier_SUT	montant_bon	montant_bon_SUT
1	moins de 30€	1	pas de bon	0
2	moins de 30€	29	pas de bon	0
3	entre 30 et 50€	30	un bon de 5€	5
4	entre 30 et 50€	31	un bon de 5€	5
5	entre 30 et 50€	49	un bon de 5€	5
6	entre 30 et 50€	50	un bon de 5€	5
7	entre 50 et 100€	51	un bon de 5€	5
8	entre 50 et 100€	99	10% du montant du panier	5,1
9	plus de 100€	100	10% du montant du panier	9,9
10	plus de 100€	101	un bon de 10€	10
11	plus de 100€	200	un bon de 10€	0

FIGURE 3.6 – Exemple de correspondance entre données de conception et d'implémentation

Chaque ligne correspond à un cas de test. Dans les colonnes, les données dont le nom fini par "_SUT" correspondent au données d'implémentation. Les autres sont les données de conception. Si l'on regarde les deux premières lignes des jeux de données, on spécifie que si la donnée de conception "montant_panier" à pour valeur "moins de 30€" on utilise en valeur de donnée d'implémentation dans un cas "1" et dans un autre cas "29". Le même mécanisme est présent pour le montant du bon où chaque valeur abstraite correspond à des valeurs concrètes.

Une fois la transcription des données de conception vers des données d'implémentation réalisée, l'exécution manuelle des tests est possible. Les outils que nous préconisons sont en capacité pour chaque étape de test de remplacer les données de conception par des données d'implémentation, et ainsi produire des cas de test exécutables. Cette exécution est possible soit directement dans l'outil de MBT soit dans un outil externe. Dans l'outil Yest, la génération de cas de test permet de visualiser sa couverture du parcours applicatif (visible dans la figure 3.7).

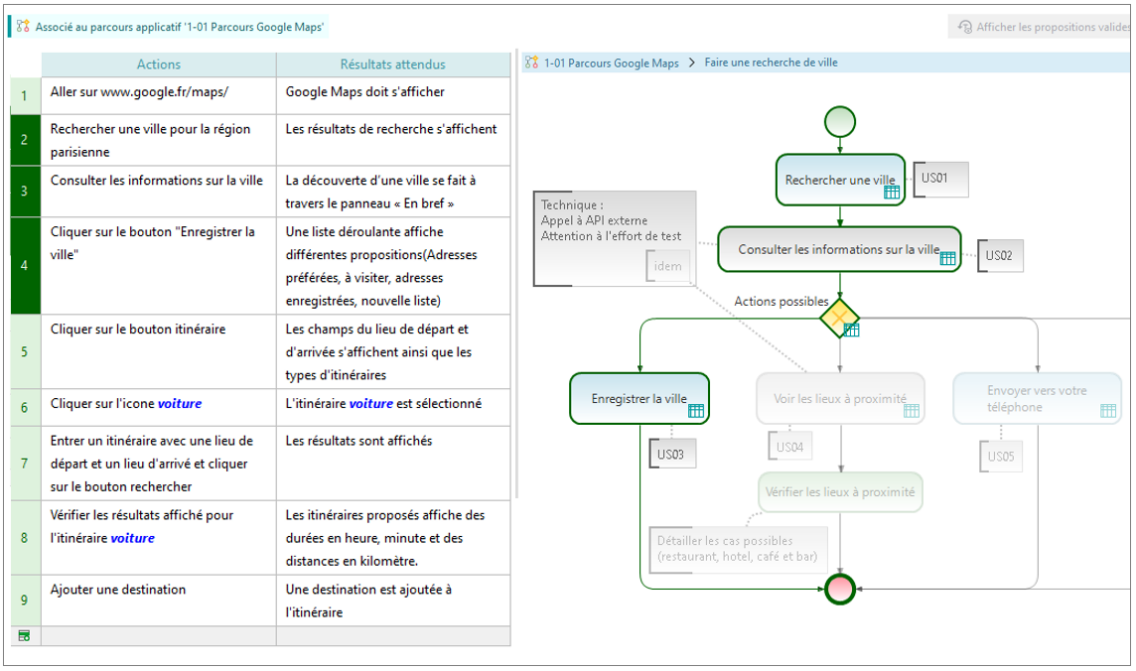


FIGURE 3.7 – Exemple de cas de test et de sa couverture du parcours applicatif

Dans le cas où l'exécution n'est pas possible dans l'outil, la publication pour l'exécution se fait dans un outil externe. La figure 3.8 présente un cas de test publié dans l'outil de gestion des tests Squash TM [12] que nous utilisons au sein du centre de services Sogeti.

Pour produire les scripts de test automatisés, il est nécessaire de compléter une couche d'adaptation [92] pour permettre l'exécution automatique de ces scripts. Cette partie de l'approche ALME est traitée au chapitre 4.

12. cf. https ://www.squashtest.com/

#	Ex.	PJ	Actions	Résultat attendu
1	☑	🔗	Aller sur www.google.fr/maps/	Google Maps doit s'afficher
2	☑	🔗	Rechercher une ville pour la région parisienne : Paris	Les résultats de recherche s'affichent
3	☑	🔗	Consulter les informations sur la ville	La découverte d'une ville se fait à travers le panneau « En bref »
4	☑	🔗	Cliquer sur le bouton itinéraire	Les champs du lieu de départ et d'arrivée s'affichent ainsi que les types d'itinéraires
5	☑	🔗	Cliquer sur l'icone voiture	L'itinéraire voiture est sélectionné
6	☑	🔗	Entrer un itinéraire avec une lieu de départ : Paris et un lieu d'arrivé : Lyon et cliquer sur le bouton rechercher	Les résultats sont affichés
7	☑	🔗	Vérifier les résultats affiché pour l'itinéraire voiture	Les itinéraires proposés affiche des durées en heure, minute et des distances en kilomètre.
8	☑	🔗	Ajouter une destination : Bordeaux	Une destination est ajoutée à l'itinéraire

FIGURE 3.8 – Exemple de cas de test publié dans l'outil de gestion de test Squash

SYNTHÈSE

Dans cette section, nous avons présenté le cycle de l'ATDD visuel qui constitue une pratique centrale de notre approche ALME. Nous avons notamment présenté la formulation visuelle des scénarios de test d'acceptation par des modèles de workflow. Puis, nous avons décrit la conception détaillée des tests et enfin l'implémentation des données abstraites de test en données concrètes pour l'exécution manuelle et l'automatisation. Nous avons mis en lumière la simplification de la notation de modélisation en utilisant un nombre très restreint d'éléments graphiques pour en faciliter l'accès aux testeurs fonctionnels. Dans la section suivante, nous décrivons et discutons des rôles contribuant à l'ATDD visuel.

3.3/ RÔLES DANS LE PROCESSUS D'ATDD VISUEL

En Agile, les individus et leurs interactions font partie des principes majeurs : notre approche doit donc permettre de favoriser ces interactions. Ces échanges interviennent à différents niveaux : en interne au sein des équipes, entre équipes, avec les représentants métier et le client. Il est donc capital de parvenir à fournir à chaque acteur une vision claire et simple des besoins en y intégrant les propriétés qui lui sont utiles. Selon le cadre Agile appliqué (Scrum, XP, etc.) on retrouve différentes terminologies pour désigner les parties prenantes d'une solution. Ici nous faisons le choix de représenter les acteurs de la solution par des rôles types intervenants dans la plupart des projets :

- Les développeurs / les responsables techniques : les personnes concernées par les développements et techniques de la solution. Leurs rôles sont de construire la solution et son architecture.
- Les testeurs : les personnes en charge de la conception et de l'exécution des tests.

- Les analystes métier : les personnes en charge d'analyser les besoins métier.
- Les automaticiens de test : les personnes en charge de l'automatisation des tests.
- Les responsables de test, coachs de test : les personnes responsables des tests au sein des équipes, ou extra équipe.
- Les représentants du métier : les personnes faisant partie de l'entité métier utilisant la solution et ayant une forte connaissance métier. Leurs rôles sont de donner des lignes directrices pour la formulation des besoins.
- Les clients / les utilisateurs : les personnes à qui est destinée la solution.

Dans les sections précédentes, nous avons présenté comment notre approche s'articulait en différentes étapes : la découverte et la formulation visuelle des besoins, la conception détaillée des tests, l'implémentation pour l'exécution manuelle ou l'automatisation. Pour chacune de ces étapes, différents rôles sont impliqués. Dans cette section, nous présentons comment les rôles mentionnés ci-dessus interviennent dans les différentes étapes du cycle de l'ATDD visuel.

3.3.1/ DÉCOUVERTE ET FORMULATION VISUELLE

L'étape de découverte et de formulation visuelle des scénarios de test d'acceptation est celle qui implique le plus de rôles. Chaque rôle a besoin durant cette étape de comprendre les besoins métier et les représentations visuelles y contribuent. Elles visent d'une part à représenter les parcours applicatifs à tester et d'autre part à renforcer les interactions entre les rôles du projet, permettant leur alignement sur une même compréhension du besoin. Pour faire le parallèle avec les cérémonies Agiles, la formulation peut se dérouler durant la construction du Backlog du produit et durant les ateliers de raffinement des US. Mais aussi à tout autre moment qui requiert une mise à jour des besoins métier.

Dans notre approche, si nous nous plaçons dans le cadre d'une itération, nous proposons que les analystes métier, avec le soutien des représentants métier, voir même du client, construisent une première représentation graphique abstraite des parcours applicatifs à traiter pour l'itération en cours. Cette représentation peut se situer au niveau de fonctionnalités définies dans des fiches MMF ou épopées.

Dans la figure 3.9, nous présentons un exemple de représentation graphique abstraite de parcours applicatifs. Cet exemple représente deux épopées. L'une a déjà été développée, la recherche d'un itinéraire et l'autre est à réaliser, la recherche d'une ville. Le lien entre ces deux épopées nous indique qu'il existe une relation entre ces fonctionnalités. Ici, cela

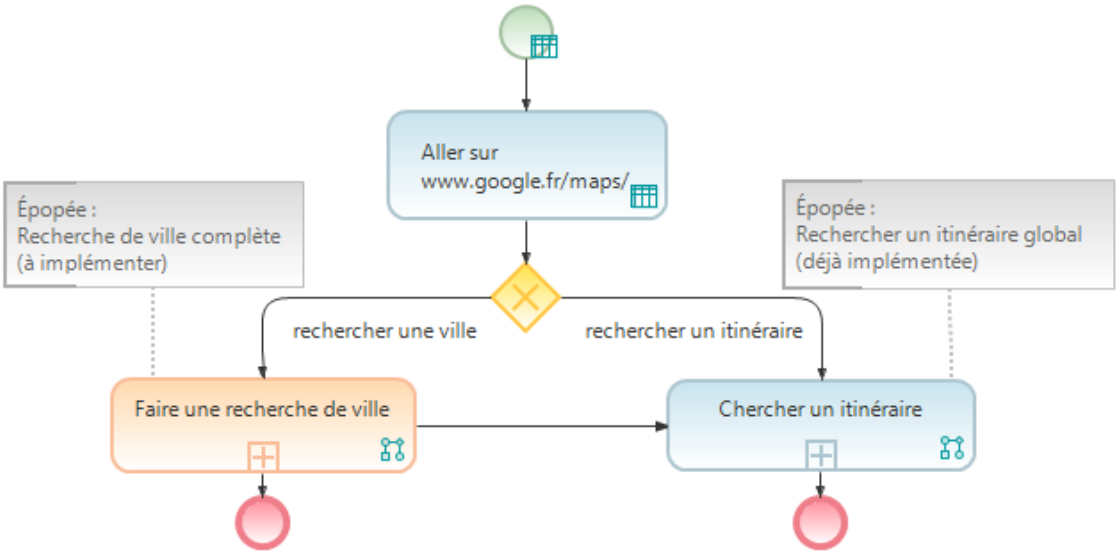


FIGURE 3.9 – 1ère représentation abstraite des parcours applicatifs pour une itération

signifie que la recherche d'une ville devra permettre la recherche de l'itinéraire pour y accéder.

La formulation des parcours applicatifs peut montrer des éléments déjà existants et des éléments liés aux nouvelles fonctionnalités à implémenter. Cela permet notamment de préparer les tests de régression (en plus des tests d'acceptation) et d'échanger sur les différents points d'attention. Avec cette représentation graphique, chaque rôle peut avoir une vision des nouveaux développements à réaliser et connaître les impacts avec l'existant. Les représentants métier pourront s'exprimer s'ils constatent des omissions. Notre exemple est volontairement simple, mais cette approche fonctionne à plus grande échelle, ce que nous présentons dans les expérimentations décrites dans ce chapitre.

De plus, nous nous sommes placés dans le contexte de l'itération, mais le même travail peut être effectué en amont au moment de la définition du besoin (Backlog du produit). Chaque épopée ou MMF peut être représentée ainsi que leurs interactions. Ensuite cette représentation pourra être complétée au fil des itérations comme nous l'avons présenté (raffinement des US), et détaillée au niveau des US. Suivant l'organisation de l'équipe, c'est l'analyste métier ou bien le testeur qui sera chargé de construire les représentations plus détaillées associées aux US, mais dans tous les cas, les testeurs et les développeurs seront plus impliqués durant cette phase de représentation des US que durant la phase de représentation des épopées ou des MMF. Notez que cette représentation est directement liée à la représentation précédente grâce à la notion de sous-parcours présentée précédemment.

La figure 3.10 présente les interventions de ces deux rôles. Les développeurs ou les responsables techniques pourront mettre en avant les US avec des spécificités tech-

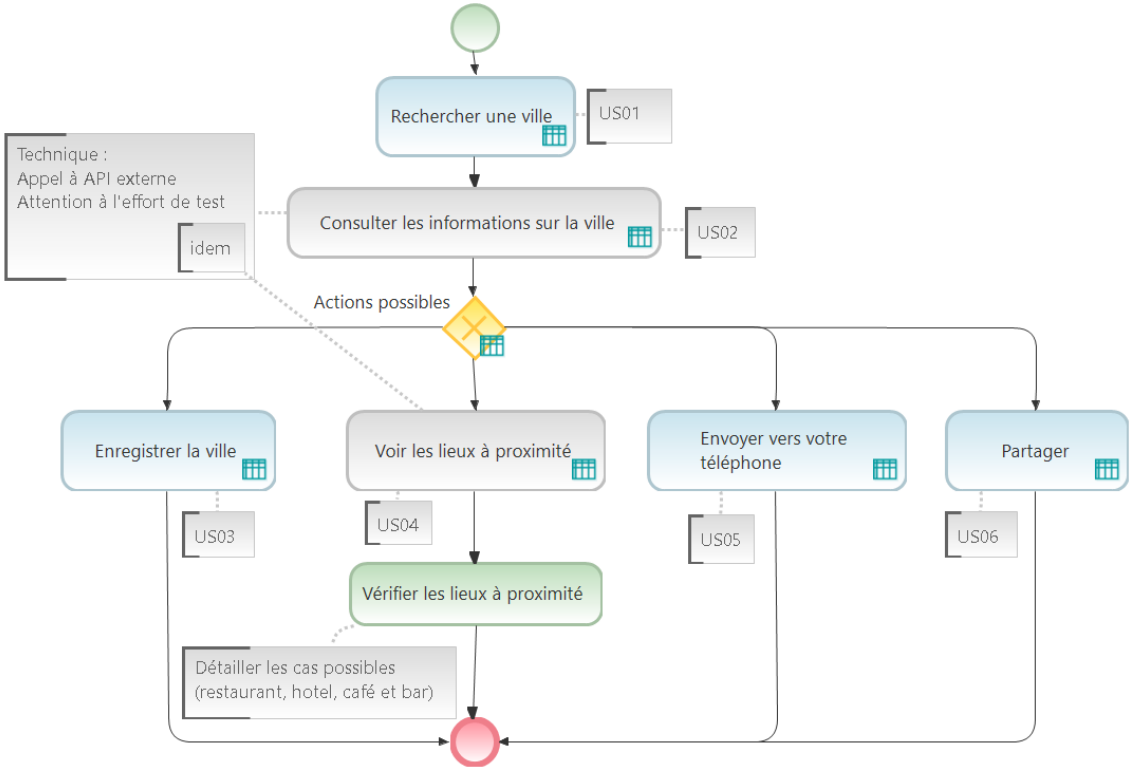


FIGURE 3.10 – Raffinement des parcours applicatifs pour une itération

niques. Dans l'exemple, les US avec des particularités techniques apparaissent en gris, et un commentaire présente brièvement la technicité. Dans notre cas, il s'agit de l'appel à une API externe. Cette information est intéressante non seulement pour le personnel technique, mais aussi pour les testeurs qui devront éventuellement consacrer un effort de test supplémentaire sur ces US. Pour leur part, ils pourront ajouter des informations spécifiques aux tests. Pour l'US04, qui permet de voir les lieux à proximité d'une ville, aucune information n'est spécifiée sur les lieux à proximité. Le testeur peut donc déjà anticiper la conception détaillée du test en ajoutant une tâche de vérification des lieux à proximité et un commentaire.

En ajoutant des commentaires dans la représentation, toutes les parties prenantes disposent d'informations concernant les points d'attention à gérer au cours d'une itération. Ils peuvent notamment en discuter et ajouter, si cela est pertinent, des éléments à la représentation. Nous soulignons ici l'utilisation d'une API externe, ainsi que l'effort de test à fournir sur les résultats renvoyés par l'API ainsi que sur les lieux à proximité.

Durant la phase de découverte et de formulation visuelle des parcours applicatifs, nous proposons donc une approche qui permettent à chaque partie prenante de participer à leur expression et leur raffinement, en particulier par les annotations sur le modèle. Ensuite ces représentations doivent être développées pour la conception détaillée des tests. Ce qui est l'objet de la sous-section suivante.

3.3.2/ LA CONCEPTION DÉTAILLÉE DES TESTS

La conception détaillée des tests est assurée par le testeur et selon le contexte par le responsable de test, du moins sous son pilotage. La conception détaillée des tests a pour objectif de spécifier les informations nécessaires pour les tests. Les testeurs réalisent la conception des tests au niveau de leur équipe durant les itérations. Le responsable de test peut être amené à intervenir dans le cadre des tests transverses. En effet, les tests transverses étant de plus haut niveau, il peut être possible que les testeurs au sein des équipes n'aient pas la vision nécessaire pour construire les tests de bout en bout. Le responsable de test a donc pour rôle de coordonner la conception des tests pour assurer les tests transverses sur la solution. Un autre rôle qui peut intervenir durant cette phase est l'automaticien de test. Nous préconisons de rapprocher les testeurs et les automaticiens pour la construction de l'automatisation. Bien que la conception des tests ne traite pas directement de l'implémentation de la couche d'adaptation des tests, ces deux rôles pourront ensemble étudier les représentations des parcours applicatifs pour anticiper l'automatisation et s'assurer que tous les éléments métier pour l'automatisation sont présents.

Bonne pratique

Construction conjointe de l'automatisation : associer les testeurs fonctionnels aux automaticiens de test pour initier l'automatisation.

BONNE PRATIQUE 3 – Construction de l'automatisation

Ainsi l'organisation des modèles et des informations associées dans ALME vise lors de la phase de conception des tests à permettre aux rôles impliqués de collaborer plus aisément. Le testeur fonctionnel peut à l'aide de représentations visuelles et l'enchaînement des étapes de test présenter les comportements métier à automatiser à l'automaticien de test pour apporter plus de vision métier à celui-ci. À l'inverse, quand le testeur fonctionnel manque de vision métier sur l'enchainement global des fonctionnalités, il peut se rapprocher des responsables de test pour compléter sa vision métier pour la construction des tests de bout en bout. Dans la sous-section suivante, nous présentons comment le testeur intervient dans la phase d'implémentation pour l'exécution manuelle ou l'automatisation.

3.3.3/ IMPLÉMENTATION POUR L'EXÉCUTION MANUELLE OU L'AUTOMATISATION

Comme pour la conception détaillée des tests, l'implémentation pour l'exécution manuelle ou en vue de l'automatisation va principalement être réalisée par le testeur. C'est ce rôle qui réalise l'exécution des tests et donc qui aura besoin des données d'implémentation pour l'exécution. Généralement, les testeurs peuvent être autonomes pour réaliser la transcription des données de conception de test vers les données d'implémentation. Mais parfois, ils peuvent ne pas avoir les informations nécessaires pour cela.

Afin d'obtenir les données nécessaires à l'exécution, les testeurs peuvent s'adresser dans un premier temps aux développeurs qui peuvent leur présenter les données qu'ils ont déjà manipulées dans l'environnement de test du SUT. Il leur est aussi possible de s'adresser aux automaticiens de test qui ont l'habitude de manipuler les données d'implémentation pour l'automatisation. Cependant, dans certains cas, ces données ne sont pas disponibles et il faut s'adresser aux représentants du métier pour les obtenir. À l'aide du support des représentations graphiques, les testeurs (et toute autre partie prenante au projet qui aurait besoin de données de test) peuvent présenter les fonctionnalités pour lesquelles des données de test sont nécessaires.

Dans la figure 3.11, les éléments en rouge sont ceux pour lesquels il manque des données. On voit rapidement que l'ensemble des fonctionnalités peut être vérifié hormis le partage. Ainsi les représentations graphiques permettent aux testeurs de mettre en avant les données manquantes et de présenter l'impact sur les tests. Ici l'absence de données sur les supports de partage empêche les testeurs de contrôler ces supports. À plus large échelle cela permet d'avoir une vision rapide de l'impact du manque de données sur un ensemble de fonctionnalités à traiter.

Une fois que le testeur a effectué la transcription des données de conception vers des données d'implémentation, il peut exécuter ces tests manuels. Nous détaillons le rôle du testeur pour l'automatisation dans le chapitre 4.

Dans cette section, nous avons présenté comment les différents rôles du projet intervenait dans notre approche. Nous avons notamment présenté comment chaque rôle pouvait collaborer pour faciliter la découverte des besoins métier et la formulation des parcours applicatifs à implémenter et à tester, ainsi que la conception et l'implémentation des tests. Dans la section qui suit nous détaillons quelles sont les pratiques à mettre en place pour faciliter la maintenance des artefacts dans notre processus d'ATDD visuel.

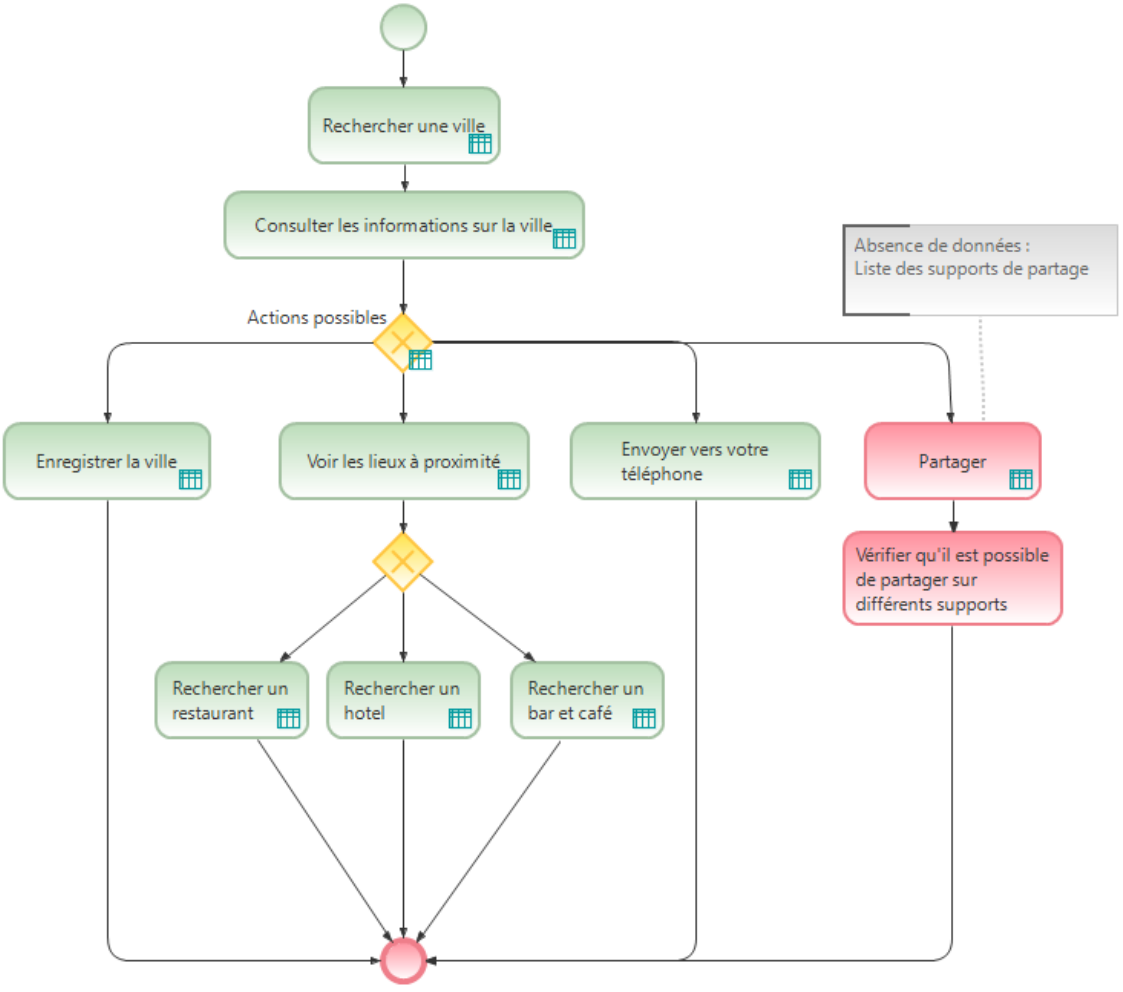


FIGURE 3.11 – Exemple de support visuel pour la gestion des données de test

3.4/ TRAÇABILITÉ ET CYCLE DE VIE DES ARTEFACTS DANS LE PROCESSUS D'ATDD VISUEL

Dans notre approche nous gérons un ensemble d'artefacts. Ce sont les modèles, les données de conception de test et les données d'implémentation de test. On traite aussi pour la partie automatisation des données manipulées par les mots-clés. Nous détaillons la gestion et la maintenance de ces données dans le chapitre 4 lié à l'automatisation. Dans cette section nous nous focalisons donc sur la traçabilité et le cycle de vie des modèles et des données de conception/d'implémentation de test.

3.4.1/ GESTION DES MODELES

Dans la pratique des projets, il est courant de construire plusieurs modèles qui utilisent des comportements communs. Par comportement commun, nous entendons un groupe d'étapes de test : des actions et des résultats attendus qui sont similaires. Un exemple est la phase de connexion au SUT qui peut être commune à plusieurs parcours applicatifs couvrant différentes fonctionnalités. Une bonne pratique consiste à intégrer dans les modèles un sous-parcours de "connexion" commun qui décrit le comportement de connexion. Ce sous-parcours "commun" est un modèle intégrant toutes les étapes nécessaires pour établir la connexion à l'application et qui peut être intégré dans tout autre modèle.

Ainsi, lorsque nous créons un nouveau modèle pour tester un nouveau comportement, il est possible de réutiliser les parcours de connexion existants pour construire le processus à tester. Dans ce cas, il est pertinent que le cas de connexion aboutisse, car il ne sert à rien de vérifier un comportement métier global en arrêtant le test sur un échec de connexion. Il est préférable d'orienter la construction des parcours communs autour des cas de passants plutôt que vers les cas d'erreur. Cela s'explique par le fait que l'objectif des parcours communs est de faciliter la construction de comportements métier de bout en bout dans lesquels l'objectif est de vérifier la séquence d'actions se déroulant nominalement sur le système et non la vérification ciblée de toutes les propriétés de l'interface. Les cas de test visant à vérifier les propriétés ciblées du système se feront par le biais de modèles dédiés qui traiteront des cas d'erreur et des exigences et règles de gestion particulières à vérifier. L'utilisation de ces parcours ciblés pour établir un comportement métier de bout en bout n'est pas recommandée, car la création et la maintenance d'un parcours avec trop de propriétés sont complexes. Chaque parcours traite des données spécifiques par rapport aux exigences à vérifier et l'assemblage avec d'autres parcours peut entraîner des incohérences, voire des incompatibilités entre les données manipulées entre chaque modèle. C'est pour ces raisons que le choix de parcours communs ne représentant que

des cas nominaux est un bon choix. Cette approche réduit le temps nécessaire pour produire de nouveaux scénarios en utilisant les artefacts existants. De plus, le fait de disposer de parcours communs pour les comportements spécifiques passants réduit le temps de maintenance. En effet, si un changement intervient sur la phase de connexion, il est facile de modifier le parcours lié à la connexion et ainsi tous les parcours qui utilisent le parcours de connexion seront à jour, sans modifier chaque modèle principal un par un.

Bonne pratique

Intégration de sous-parcours commun : identifier les différents comportements de l'application et à repérer ceux qui peuvent être utilisés dans plusieurs cas de test et créer un ensemble de sous-parcours communs pouvant être utilisés plusieurs fois plutôt que des modèles distincts et uniques.

BONNE PRATIQUE 4 – Intégration de sous-parcours

Ces sous-parcours communs peuvent s'apparenter au mécanisme de refactoring visant à éviter la duplication de code. Chaque parcours représente des fonctions (pour établir un parallèle avec le développement) et comme des fonctions il est possible de les paramétrer, dans le contexte des parcours à l'aide des tables de décision qu'ils contiennent.

Dans l'exemple de la figure 3.12, dans les zones "Contexte 1 : Avec bonnes pratiques" et "Contexte 2 : Sans bonnes pratiques", nous considérons deux contextes identiques, mais le 1er applique des bonnes pratiques et l'autre pas. Les comportements sont similaires par exemple au début du parcours, mais aussi sur certaines fonctionnalités par la suite. Au lieu de créer différents sous-parcours comme dans le contexte 2, il est préférable d'utiliser le même sous-parcours comme indiqué dans le contexte 1. Le nombre d'artefact de test à maintenir est réduit à 3 dans le contexte 1, contre 5 dans le contexte 2 alors que tous deux vérifient les mêmes propriétés.

Au fil des itérations et des évolutions, le nombre de modèles va croître. Nous préconisons donc d'appliquer les pratiques présentées pour limiter le nombre de modèles. Notamment en créant des modèles communs qui pourront évoluer et non pas être dupliqués à chaque itération.

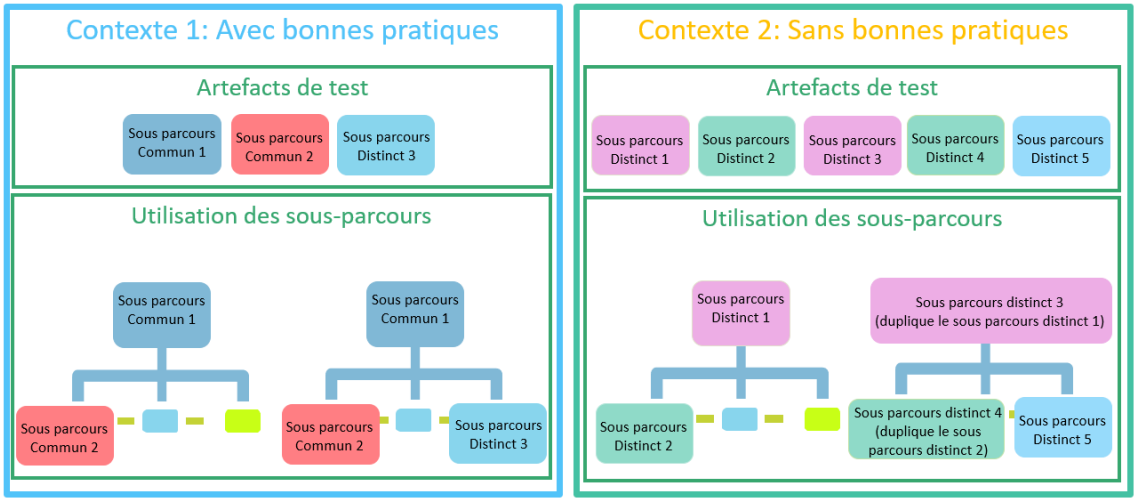


FIGURE 3.12 – Bonnes pratiques de gestion des sous-parcours

Bonne pratique

Organisation des modèles par fonctionnalité : afin de ne pas perdre de visibilité sur les modèles existants, nous conseillons de les organiser par fonctionnalité et sous fonctionnalité métier.

BONNE PRATIQUE 5 – Organisation des modèles

Prenons l'exemple d'une application de e-commerce. Elle se compose d'une fonctionnalité permettant la connexion, l'accès à des produits, à un panier d'articles, etc. Chacune de ces fonctionnalités persiste dans le temps et est amenée à évoluer. Dans la figure 3.13 nous présentons un exemple d'organisation des modèles vis-à-vis d'une application de e-commerce.

On peut voir dans cette figure 3.13 des modèles liés à la connexion de notre exemple. L'un traite de la connexion client l'autre de la connexion vendeur. Ici le fait d'avoir deux modèles s'explique par le fait que l'on est sur deux acteurs différents qui n'ont pas accès au même portail de connexion ni aux mêmes fonctionnalités. D'où le choix de faire deux modèles. Sur la fonctionnalité "Produits" on peut retrouver deux modèles l'un gérant l'affichage des produits de manière générale et l'autre dans le détail des fiches produits. L'objectif est de construire au fil des itérations des modèles pour chaque fonctionnalité et réutiliser quand cela est possible des modèles existants. Dans la section suivante, nous décrivons comment gérer les données de conception de test que manipulent les modèles.

FIGURE 3.13 – Exemple d'organisation des modèles par fonctionnalité

3.4.2/ GESTION DES DONNÉES DE CONCEPTION DE TEST

Comme évoqué, les données de conception de test sont utilisées dans notre approche lors de la conception détaillée des tests. Elles sont spécifiées dans les modèles et sont présentes pour gérer les besoins métier. Nous avons déjà préconisé de ne définir que les données nécessaires, c'est-à-dire celles qui découlent des exigences ou qui sont utiles pour gérer les flots métier. Pour rappel, ceci permet de limiter le nombre de données à maintenir et de se focaliser sur les objectifs de test. Cependant même si on ne définit que les données nécessaires, le nombre de données va croître au fil des itérations. Chaque évolution peu entraîner de nouvelles conditions de test et donc de nouvelles données. L'important est de garder une visibilité sur les données existantes au fil des itérations pour éviter des redondances dans celles-ci. Les doublons, qu'ils soient au niveau des données, ou de leur valeur, ne peuvent engendrer que des confusions et un temps de maintenance supplémentaire. D'où l'intérêt d'organiser intelligemment ces données. Les outils de MBT proposent chacun des mécanismes différents de gestion des données. Cependant, ils permettent généralement d'organiser hiérarchiquement les données.

Bonne pratique

Organisation des données par fonctionnalité : nous préconisons d'organiser les données par fonctionnalité et sous fonctionnalités métier (comme pour les modèles).

BONNE PRATIQUE 6 – Organisation des données

Ainsi en cas d'évolution à traiter sur une fonctionnalité, il est possible directement de consulter les données et leurs valeurs en lien avec la fonctionnalité et ainsi éviter la création de nouvelles données similaires.

Pour faire suite aux données de conception de test, nous présentons dans la section suivante la gestion des données d'implémentation.

3.4.3/ GESTION DES DONNÉES D'IMPLÉMENTATION DE TEST

La gestion des données d'implémentation permet d'assurer la traçabilité entre les conditions de test exprimées sur les données de conception et les valeurs concrètes à mettre en œuvre pour assurer leur vérification. Dans la figure 3.6, nous avons présenté un exemple de correspondance entre données de conception et d'implémentation. Cela permet notamment de connaître les montants réels à compléter dans l'application pour vérifier les règles de gestion.

Bonne pratique

Nommage des données d'implémentation : nous préconisons pour faciliter la gestion des données d'implémentation de veiller à nommer de manière rigoureuse ces données ainsi que les jeux de données qui les contiennent.

BONNE PRATIQUE 7 – Nommage des données

Dans la figure 3.6, dans Yest, nous avons par exemple nommé chaque ligne selon la propriété qu'elle vérifiait (panier à mois de 30€, entre 30 et 50€, etc). Dans Yest et TestOptimal, nous avons terminé les noms de nos données d'implémentation par "_SUT" pour signifier que ces données étaient celles à utiliser dans le système sous test. De la même manière, il est important de nommer les jeux de données de manière explicite afin d'éviter leur duplication. Pour cela, les noms de jeux de données peuvent contenir les données principales qu'ils utilisent, dans notre exemple "panier", "montant".

Nous avons présenté dans cette section des bonnes pratiques pour la gestion des artefacts (modèles et données) de notre approche ALME. Afin d'éprouver notre approche, nous avons mené des expérimentations que nous présentons dans la section suivante.

3.5/ EXPÉRIMENTATIONS

Ces expérimentations ont été réalisées dans le contexte de nos activités de thèse au fil des 3 années. Nous organisons dans la suite leur restitution en deux parties.

L'expérimentation 1 détaille la mise en pratique de l'approche ALME sur un exemple d'illustration.

Dans l'expérimentation 2, nous avons étudié deux contextes de projet auxquels nous avons participé. Dans un cas, nous avons appliqué notre l'approche dans sa globalité (contexte 2) et dans l'autre nous avons travaillé dans un contexte Agile, sans appliquer notre approche (contexte 1). Il s'agit ainsi de discuter dans quelle mesure le Model-Based Testing avec ALME est adapté au contexte Agile en comparaison avec une approche de conception manuelle des tests.

3.5.1/ EXPÉRIMENTATION 1 : MISE EN PRATIQUE DES NOTATIONS DE L'APPROCHE ALME

Pour illustrer notre approche, nous présentons un court exemple permettant d'en discuter toutes les étapes. Il s'agit de la vérification d'un système de billetterie de train sur le site web Oui.sncf. Oui.sncf est une agence de voyages française en ligne (https://www.oui.sncf).

Nous avons mené nos expérimentations sur cette application, car elle est assez représentative d'un site web marchant. Nous avons notamment travaillé sur un périmètre mettant en oeuvre des fonctionnalités communes des sites en ligne qui sont l' inscription sur le site et la recherche et la commande d'un produit, dans ce cas, des billets de train. Nous avons choisis ces deux fonctionnalités car ce sont des modélisations qui pourraient s'adapter pour tous sites de ce type. Ainsi nous pouvons éprouver notre approche sur un contexte particulier mais relativement généraliste.

Pour réserver un billet de train, les utilisateurs peuvent choisir les villes de départ et d'arrivée et la date. Pour créer un compte, l'utilisateur doit entrer un prénom, un nom, une date de naissance, un email et un email de confirmation.

Nous allons présenter comment nous avons utilisé la modélisation ALME pour traiter de la vérification de la réservation de billet et l'inscription. Pour cela, nous avons défini un ensemble d'exigences.

BESOIN MÉTIER À VÉRIFIER

Nous listons ci-dessous l'ensemble des exigences que nous avons définies lors de cette expérimentation :

- Il est possible de rechercher un train entre deux villes à une date précise (exigence_réservation_010).
- Dans le formulaire de réservation de billets de train, il est impossible de choisir une date de départ antérieure à la date du jour (exigence_réservation_011).
- Dans le formulaire de réservation des billets de train, il est impossible de choisir la même ville d'arrivée et de départ (exigence_réservation_012).
- Dans le formulaire de création de compte, si les informations sont valides, la création de compte est possible (exigence_compte_010).
- Dans le formulaire de création de compte, le format du prénom et du nom doit être valide (pas de caractères spéciaux)(exigence_compte_011)
- Dans le formulaire de création de compte, l'email et l'email de confirmation doivent être identiques pour créer le compte (exigence_compte_012).
- Dans le formulaire de création de compte, l'email ne doit pas déjà exister dans la base de données pour permettre la création d'un compte (exigence_compte_013).

Une fois ces exigences définies nous avons modélisé les besoins métier avec une représentation graphique visible dans la figure 3.14.

DÉCOUVERTE ET FORMULATION VISUELLE

La modélisation réalisée en figure 3.14 s'inscrit durant la phase de découverte et de formulation visuelle des parcours applicatifs. Nous avons volontairement appliqué beaucoup de détails durant cette phase afin de présenter les capacités de modélisation dans notre approche.

Nous avons ”divisé” notre représentation en deux parties. La partie gauche du modèle représente la création du compte et la partie droite la réservation du train. Considérons la partie réservation à droite du modèle. Le processus consiste en une succession de quatre tâches. Tout d'abord, il faut accéder à la page de réservation des billets de train. Ensuite, nous saisissons les informations nécessaires à la recherche d'un billet. Cela fait, nous validons la recherche. Si les informations sont valides (choix de gares existantes), il est possible sélectionner un train, sinon un message d'erreur sera affiché.

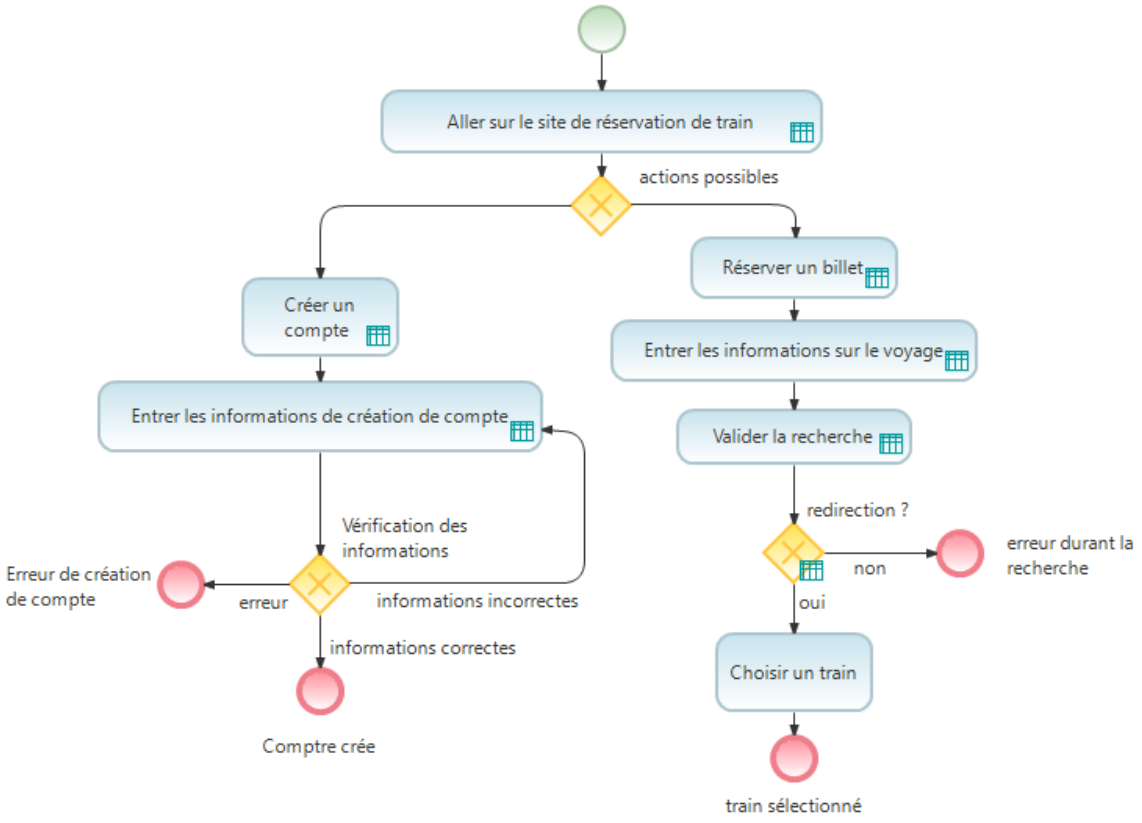


FIGURE 3.14 – Modélisation des parcours applicatifs du site oui.sncf sur le périmètre choisi

La seconde partie de notre modèle à gauche traite de la création du compte. Ici nous modélisation un parcours itératif. Dans le détail, la première étape du processus consiste à accéder à l'espace de création de compte. Ensuite, les informations nécessaires à la création de compte sont complétées et leur validité est vérifiée pour déterminer si elles permettent la création du compte ou non. Le caractère itératif est induit par le fait qu'il est possible de ressaisir les informations de créations de comptes si celles si sont invalides.

Nous avons été en mesure durant cette phase de découverte et de formulation visuelle de représenter simplement le parcours d'inscription et de recherche d'un produit sur une même interface. Nous avons volontairement représenté ces deux comportements distincts sur un même modèle pour montrer qu'il était possible de modéliser des comportements métier riches de manière simple et visuelle.

CONCEPTION DÉTAILLÉE DES TESTS

Lors de la conception détaillée des tests nous avons complété les informations nécessaires aux tests. Nous avons notamment complété les étapes de test, et entré des données de conception dans des tables de décision liées aux tâches et aux points de choix dans notre modèle. La figure 3.15 présente la table associée à la tâche "Entrer les informations de création de compte".

	nom	prénom	email	confirmation_email	Étapes de test		Exigence
1	*valide*	*valide*	*valide*	*identique*	Entrer les informations de création de compte correctement	Les informations sont entrées	exigence_compte_010
2	*valide*	*invalide*	*valide*	*identique*	Entrer les informations de création de compte avec un prénom invalide	Les informations sont entrées	exigence_compte_011
3	*invalide*	*valide*	*valide*	*identique*	Entrer les informations de création de compte avec un nom invalide	Les informations sont entrées	exigence_compte_011
4	*valide*	*valide*	*valide*	*différent*	Entrer les informations de création de compte avec un email et un email de confirmation différents	Les informations sont entrées	exigence_compte_012
5	*valide*	*valide*	*déjà en base*	*identique*	Entrer les informations de création de compte avec un email déjà en base de données	Les informations sont entrées	exigence_compte_013

FIGURE 3.15 – Table de décision "Vérification des informations"

La table comporte 5 lignes, chacune correspond à un comportement qui doit être testé pour couvrir les différents besoins métier pour la création de comptes. Les données de conception nécessaire pour couvrir les objectifs de test sont le prénom, le nom, un email et une validation d'email. Dans la table 3.15 nous avons donc fait apparaître uniquement ces quatre données (comme préconisé dans la BONNE PRATIQUE 1) et nous leur avons associé des valeurs abstraites pour vérifier nos différentes propriétés. Ainsi pour le nom, prénom et le mail, nous avons comme valeurs possibles "valide" ou "invalide". Et pour la confirmation d'email "identique", signifiant qu'il est identique à l'email renseigné et "différent" pour signifier que l'email de confirmation est différent de l'email renseigné.

Ces combinaisons de valeurs permettent de vérifier les différents objectifs de test.

Chaque ligne de la table va ainsi produire un cas de test abstrait que nous cherchons à vérifier. La génération des tests produit 5 cas de test abstraits (un par ligne à couvrir).

IMPLÉMENTATION POUR L'EXÉCUTION MANUELLE OU L'AUTOMATISATION

Une fois les tests générés nous les avons implémentés afin de permettre leur exécution. Dans cette expérimentation nous nous focalisons uniquement sur l'implémentation pour l'exécution manuelle. L'implémentation pour les tests automatisés est traitée dans le chapitre 4.

Lors de la phase de conception détaillée des tests, nous avons spécifié 5 cas à traiter se formalisant par 5 cas de test. Ces cas permettent notamment de vérifier la validé du nom, du prénom et du mail dans l'inscription. Nous avons donc défini dans la figure 3.15 des valeurs abstraites pour vérifier ces propriétés qui étaient "valide" ou "non valide". Pour permettre l'exécution des tests, et comme nous l'avons préconisé dans la BONNE PRATIQUE 2 , nous avons associé chacune de nos données abstraites à des données concrètes avec un tableau, visible dans la figure 3.16.

	Nom du jeu de données	prénom	prénom_SUT	nom	nom_SUT	email	email_SUT	confirmation_email	confEmail_SUT
1	**Jeu inscription valide**	*valide*	*Paul*	*valide*	*Dupont*	*valide*	*paul.dupont@mail.com*	*identique*	*paul.dupont@mail.com*
2	**Jeu inscription prénom invalide [1]**	*invalide*	*exP0-???*	*valide*	*Dupont*	*valide*	*paul.dupont@mail.com*	*identique*	*paul.dupont@mail.com*
3	**Jeu inscription prénom invalide [2]**	*invalide*	*1234*	*valide*	*Dupont*	*valide*	*paul.dupont@mail.com*	*identique*	*paul.dupont@mail.com*
4	**Jeu inscription nom invalide [1]**	*valide*	*Paul*	*invalide*	*exP0-???*	*valide*	*paul.dupont@mail.com*	*identique*	*paul.dupont@mail.com*
5	**Jeu inscription nom invalide [2]**	*valide*	*Paul*	*invalide*	*1234*	*valide*	*paul.dupont@mail.com*	*identique*	*paul.dupont@mail.com*
6	**Jeu inscription email invalide [1]**	*valide*	*Paul*	*valide*	*Dupont*	*valide*	*paul.dupont@mail.com*	*différent*	*dupont@mail.com*
7	**Jeu inscription email invalide [2]**	*valide*	*Paul*	*valide*	*Dupont*	*valide*	*enBase@mail.com*	*identique*	*enBase@mail.com*

FIGURE 3.16 – Tableau d'implémentation des données

Nous avons par exemple choisi des noms, prénoms et emails que le SUT peut accepter. Mais aussi des valeurs qui devront être refusées. La ligne 3 par exemple permet de vérifier un cas ou l'inscription échoue, car le prénom à un format invalide.

Dans le détail, nous avons fait le choix de tester plusieurs nom, prénom et mail invalides. Ceci montre qu'un cas abstrait peut donner lieu à plusieurs cas de test concrets [13]. Ceci explique que notre table de données concrètes de la figure 3.16 présente plus de lignes que le tableau des valeurs abstraites en 3.15.

Dans cet exemple, nous avons pu facilement associer chacune de nos données de conception abstraite à des données d'implémentation concrète et ainsi permettre une

13. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

exécution future. Nous avons différencié nos données ici via leur nommage comme nous l'avons préconisé dans notre approche (BONNE PRATIQUE 7).

Dans cette expérimentation nous avons appliqué l'approche ALME de bout en bout pour permettre l'exécution manuelle des tests. Nous avons présenté comment des notations simples permettaient de vérifier des besoins métier liés à une souscription et un achat de produit. Nous avons aussi montré comment nous faisions le lien entre les données abstraites liées aux besoins métier et les valeurs concrètes pour l'exécution. Dans la prochaine expérimentation, nous présentons comment nous avons mis concrètement en place notre approche en contexte projet.

3.5.2/ EXPÉRIMENTATION 2 : APPROCHE ALME DANS UN CONTEXTE INDUSTRIEL

Dans cette section, nous restituons la mise en oeuvre de l'approche ALME dans un projet sur lequel nous avons travaillé et nous comparons cette expérience avec celle résultant d'une conception manuelle des tests, dans les deux cas en Agile. Il s'agit d'évaluer si notre approche permet un gain au niveau de la phase de conception des cas de test. Les paramètres recueillis pour évaluer l'efficacité de l'approche ALME sont le temps consacré à la conception et à implémentation des tests [12], le nombre de cas de test produits et le nombre d'étapes par test, ainsi que le nombre d' US traitées.

CONTEXTE 1 : CONCEPTION DES TESTS SANS L'APPROCHE ALME

Ce projet porte sur le test d'une application web de gestion de la qualité dans une société ferroviaire française. Le développement est réalisé en Agile avec des sprints de 4 semaines et une stratégie de test pour vérifier les fonctionnalités mises en œuvre à la fin de chaque sprint. Pendant un an et demi, des cas de test manuels ont été développés par deux testeurs fonctionnels, sans aucune disposition concernant un futur plan d'automatisation des tests. Notre expérimentation commence un an et demi après le début du projet. Pendant 3 mois, nous avons réalisés les phases de découverte du besoin et de conception/implémentation des tests.

CONTEXTE 2 : CONCEPTION DES TESTS AVEC L'APPROCHE ALME

Le second contexte porte sur le test d'une application web pour gérer les compétences des employés dans la même entreprise que le contexte 1. Le développement est réalisé en Agile avec des sprints de 4 semaines et une mise en pratique de l'approche ALME. Comme pour le 1er contexte, nous avons choisi une période de 3 mois (après 6 mois de

développement) pour étudier les phases de découverte et de formulation des parcours applicatifs à tester, et la conception et l'implémentation des tests avec ALME. Le choix d'attendre 6 mois pour évaluer l'approche a été guidé par le fait que nous voulions évaluer notre approche lorsque le projet était suffisamment démarré. C'est-à-dire qu'une 1ère livraison client a eu lieu et qu'un effort régulier existe à chaque sprint (ce qui n'est pas le cas durant les premières itérations). La conception des tests a été assurée par deux testeuses fonctionnelles n'ayant pas d'expérience préalable en MBT. L'une est testeuse depuis plus de 10 ans, connaissait le contexte métier et avait pour habitude de concevoir les tests manuellement. L'autre est une nouvelle testeuse, débutante dans le domaine du test et sur le contexte métier, et qui n'avait pas conçu de test avant cette expérimentation.

Ces deux contextes nous permettent ainsi de confronter une approche de conception de tests manuels avec la conception de tests avec l'approche ALME dans des conditions identiques de compétences initiales des testeurs.

DÉCOUVERTE ET FORMULATION

Dans les deux contextes étudiés, l'expression du besoin est réalisé par des User Stories. Ces US avaient exactement la même structure pour les deux projets : une présentation du contexte suivie d'une description d'un ensemble de règles de gestion. Ces règles devant être traitées dans les cas de test. La base de représentation des besoins metier était donc la même pour nos deux contextes. La différence est que sur le contexte 2, la représentation visuelle des parcours applicatif a été introduite. Un exemple de représentation visuelle est visible en annexe B.1.1

Du point de vue de la construction de ces représentations graphiques, elles n'ont pas posé de difficultés particulières dans leur réalisation. Pour la testeuse qui découvrait l'expression des besoins, c'était un moyen simple d'accroître ses connaissances métier grâce à la construction des représentations qui décrivaient la séquence des fonctionnalités. De plus, les modèles étant classés par fonctionnalité comme le préconise la BONNE PRATIQUE 5, la testeuse pouvait cibler rapidement les fonctionnalités qu'elle souhaitait étudier.

Constatation : meilleure vision sur les besoins métier.
L'application de bonnes pratiques a permis de simplifier l'activité de découverte et de formulation visuelle des parcours applicatifs. Le choix de séparer les parcours a limité la complexité des représentations visuelles. En complément, le classement de ces parcours par fonctionnalité a apporté de la visibilité sur les besoins métier.

CONSTAT 1 – Meilleure vision sur les besoins métier

Pour la testeuse plus expérimentée, les représentations visuelles lui ont permis de signaler plus facilement les points d'attention aux développeurs et de mieux justifier l'effort de test nécessaire sur certaines fonctionnalités.

Constatation : Les représentations visuelles facilitent la communication au sein des équipes.
Les représentations peuvent être un support à la communication en permettant à chaque partie prenante à la solution d'exprimer son point de vue et mettre en avant des points d'attention.

CONSTAT 2 – La communication au sein des équipes

Dans le contexte 1, avec uniquement la description textuelles des User Stories et des règles de gestion, l'équipe rencontrait différentes difficultés. Tout d'abord, les dépendances et enchaînements des US n'étaient pas complètement bien perçus par l'équipe. L'ordre des développements n'était donc pas priorisé pour permettre le test au plus tôt. De plus, l'absence d'atelier pour discuter des US autour de représentations graphiques menait parfois à des mises à jour tardives durant l'itération. Même si des ateliers étaient menés autour des US, chaque partie prenante n'avait pas forcément assez de visibilité pour mettre en avant des points métiers qui auraient dû être revus, car incomplets ou erronés. Le test ne commençait donc pas au plus tôt, mais intervenait plutôt à la fin des itérations. Ceci rendait donc le rythme projet moins soutenable en fin d'itération avec de nombreuses corrections à effectuer tardivement pour assurer la qualité des livrables.

Sur le second contexte, ce problème est moins fréquent, car les différentes parties prenantes ont pu identifier plus tôt des défauts de conception ou de développement pour permettre aux développeurs de corriger durant l'itération.

CONCEPTION DÉTAILLÉE DES TESTS

Dans le contexte 1, la conception détaillée des tests a été réalisée dans l'outil de gestion de test Squash TM en se basant sur les US. Il est possible de voir un exemple de cas de test anonymisé dans la figure 3.17 en haut de la capture.

Les cas tests ont été initialement écrits de manière abstraite sans inclure de données concrètes. Ils étaient écrits un par un, parfois en utilisant des étapes de test provenant de scénarios de test créés lors d'itérations précédentes.

Pour le contexte 2, illustré avec un cas de test anonymisé dans la figure 3.18, la conception abstraite des tests a également été réalisée en premier lieu. Cependant, au lieu d'écrire les cas de test un par un, un ensemble de représentations graphiques ont été conçues. Elles étaient composées d'étapes de test visant à couvrir les besoins du métier. Ensuite, les cas de test ont été générés avec l'outil MBT (Yest dans notre cas) de génération de tests.

Pour la testeuse la plus expérimentée, la conception détaillée était au départ déroutante, car elle a entraîné un changement de ses pratiques. Au lieu de choisir une règle métier et de concevoir un test associé, elle devait faire des représentations visuelles qui contenaient des règles métier et ensuite générer les tests. Malgré le changement, elle a retrouvé des points de repère, notamment en précisant les actions et les résultats attendus. De plus, elle a trouvé que la conception des tests était à la fois plus rapide et plus rigoureuse. Plus rapide, car la génération des tests permettait une couverture optimisée des règles métier, contrairement à la conception manuelle des tests, qui nécessitait plus de réflexion et donc plus de temps pour orchestrer la séquence de contrôle des règles métier. Et plus rigoureuse, parce que l'utilisation de tableaux pour spécifier les différents cas possibles a permis une plus grande exhaustivité.

Pour la testeuse novice, la conception des tests n'a pas présenté d'obstacle particulier. Comme elle n'avait aucune expérience préalable de la conception de tests, elle n'a pas été affectée par la manière dont les tests sont conçus dans l'approche ALME. Elle a pu rapidement utiliser les modèles établis pendant la phase de découverte et de formulation des parcours applicatifs à tester, et les compléter par des actions et des résultats attendus pour vérifier les règles métier.

Constatation : la conception détaillée est plus rapide et rigoureuse.
La génération des tests donnent une couverture plus rapide des règles métier.
Les tables de décision donne un cadre pour définir rigoureusement les conditions de test des règles métier.

CONSTAT 3 – La conception détaillée est plus rapide et rigoureuse

Une fois la conception détaillée des tests réalisée vient la phase d'implémentation des tests qui est l'objet de la sous-section suivante.

IMPLÉMENTATION POUR L'EXÉCUTION MANUELLE OU L'AUTOMATISATION

Dans le contexte 1, les données d'implémentation ont été directement intégrées dans les cas tests en utilisant des paramètres, comme le montre la partie en bas de la figure 3.17. Dans cet exemple, la liste des points clefs, qui est une donnée métier nécessaire pour les tests, est spécifiée à l'aide de paramètres et d'ensembles de données.

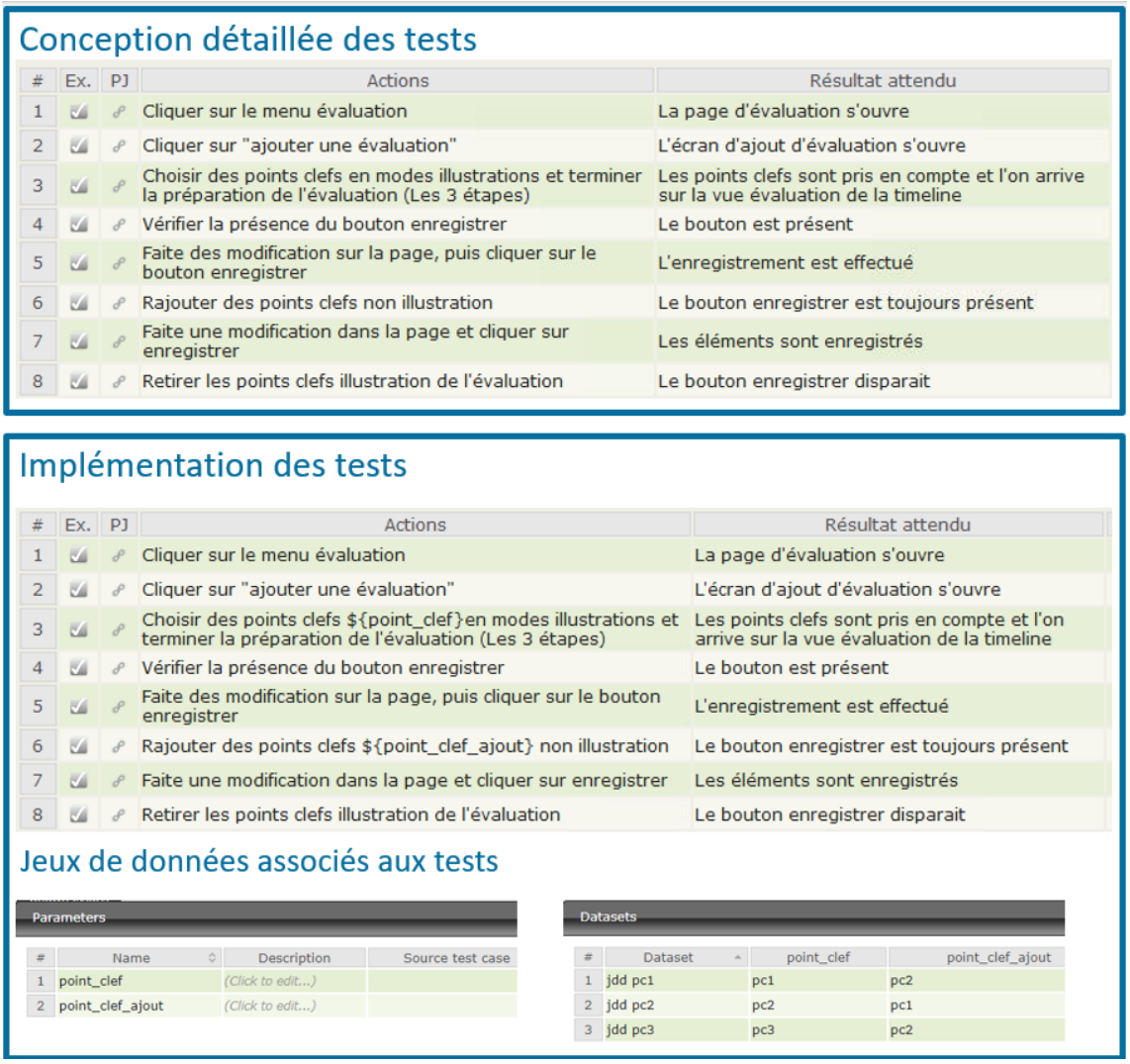
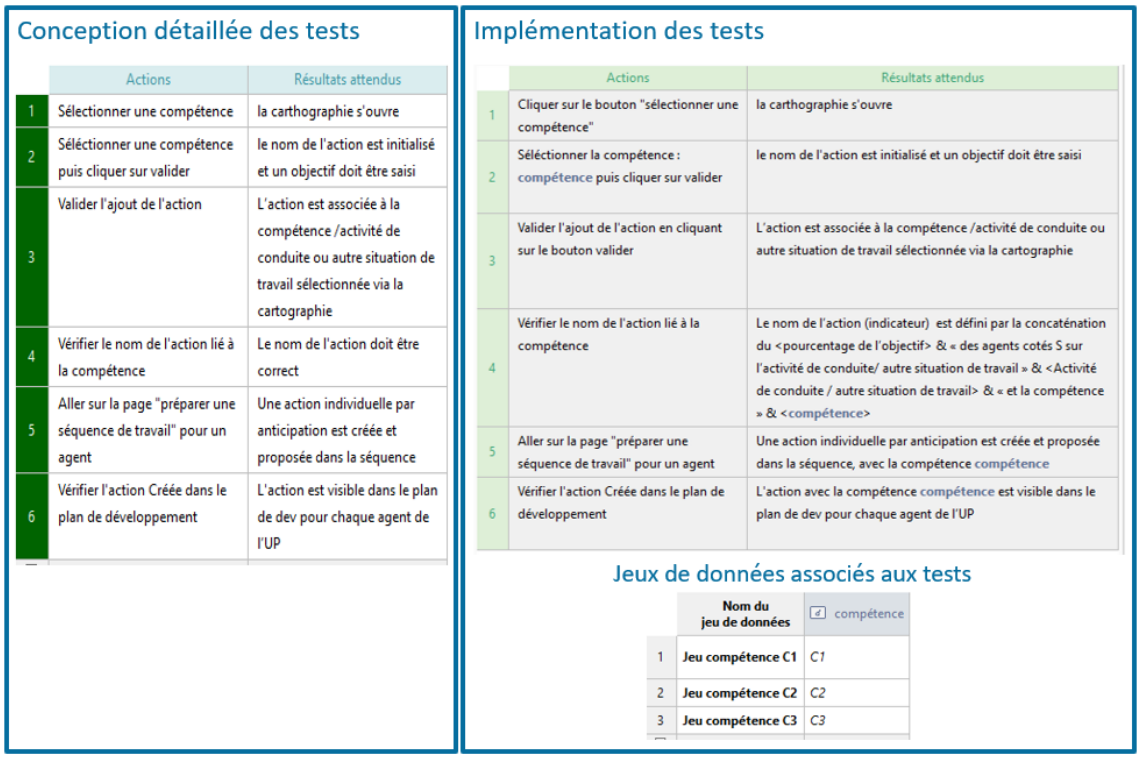


FIGURE 3.17 – Exemple de cas de test durant la phase de conception des tests et la phase d'implémentation des tests, contexte 1

Dans le second contexte, un mécanisme similaire à été utilisé pour relier des données abstraites à des données concrètes pour l'exécution, comme le montre la partie droite de la figure 3.18. Cependant, les données étaient classées par fonctionnalité comme préconisé dans la BONNE PRATIQUE 6 et nommées de manière explicite comme préconisé dans la BONNE PRATIQUE 7.

Nous pouvons voir que les données qui étaient exprimées de manière abstraite lors de



Conception détaillée des tests

	Actions	Résultats attendus
1	Sélectionner une compétence	la carthographie s'ouvre
2	Séléctionner une compétence puis cliquer sur valider	le nom de l'action est initialisé et un objectif doit être saisi
3	Valider l'ajout de l'action	L'action est associée à la compétence /activité de conduite ou autre situation de travail sélectionnée via la cartographie
4	Vérifier le nom de l'action lié à la compétence	Le nom de l'action doit être correct
5	Aller sur la page "préparer une séquence de travail" pour un agent	Une action individuelle par anticipation est créée et proposée dans la séquence
6	Vérifier l'action Créée dans le plan de développement	L'action est visible dans le plan de dev pour chaque agent de l'UP

Implémentation des tests

	Actions	Résultats attendus
1	Cliquer sur le bouton "sélectionner une compétence"	la carthographie s'ouvre
2	Séléctionner la compétence : compétence puis cliquer sur valider	le nom de l'action est initialisé et un objectif doit être saisi
3	Valider l'ajout de l'action en cliquant sur le bouton valider	L'action est associée à la compétence /activité de conduite ou autre situation de travail sélectionnée via la cartographie
4	Vérifier le nom de l'action lié à la compétence	Le nom de l'action (indicateur) est défini par la concaténation du <pourcentage de l'objectif> & « des agents cotés S sur l'activité de conduite/ autre situation de travail » & <Activité de conduite / autre situation de travail> & « et la compétence » & <compétence>
5	Aller sur la page "préparer une séquence de travail" pour un agent	Une action individuelle par anticipation est créée et proposée dans la séquence, avec la compétence compétence
6	Vérifier l'action Créée dans le plan de développement	L'action avec la compétence compétence est visible dans le plan de dev pour chaque agent de l'UP

Jeux de données associés aux tests

	Nom du jeu de données	compétence
1	Jeu compétence C1	C1
2	Jeu compétence C2	C2
3	Jeu compétence C3	C3

FIGURE 3.18 – Exemple de cas de test durant la phase de conception des tests et la phase d'implémentation des tests, contexte 2

la phase de conception détaillée des tests sont remplacées par des données concrètes mises en œuvre par des ensembles de données pour tester plusieurs valeurs.

Dans la section suivante, nous présentons les résultats obtenus en comparant les phases de conception et d'implémentation entre l'approche ALME utilisée dans le contexte 2 et l'approche manuelle utilisée dans le contexte 1.

RÉSULTATS ET DISCUSSION DES QUESTIONS DE RECHERCHE

Lors de nos expérimentations de thèse, nous avons confronté la conception et l'implémentation des tests entre les contextes 1 et 2 sur une période de 3 mois avec les métriques suivantes :

- le nombre de cas de test produit.
- le nombre d'étapes par cas de test.
- le temps passé à la conception et l'implémentation des tests. Nous utilisons les heures et les minutes comme unité de mesure pour mesurer la moyenne des temps de conception par cas de test.

Nous présentons dans le tableau 3.19 les résultats obtenus à la suite de ces phases de conception et d'implémentation des tests sur chacun des projets.

	Moyenne pour une itération			
	Nombre de cas de test	Nombre d'étape de test par cas de test	Nombre d'étape de test pour tous les cas de test	Temps de conception par étape de test
Contexte 1	102	10	1020	1min42s
Contexte 2	131	15	1965	54s

FIGURE 3.19 – Résultats pour la conception et l'implémentation des tests pour les contexte 1 et 2

Durant cette période pour le contexte 1, il a été produit en moyenne par itération : 102 cas de test, 10 étapes par cas de test (soit 1020 étapes de test). Le temps de conception pour une étape de test en moyenne est de 1 min 42s.

Pour le contexte 2, en moyenne 131 cas de test ont été produits, 15 étapes par cas de test (soit 1965 étapes de test). Le temps de conception pour une étape de test en moyenne est de 54s.

Entre ces deux contextes, la volumétrie en nombre de cas de test produits parait assez proche avec 102 cas de test pour le 1er contexte et 131 pour le second. Cependant si l'on compare le nombre d'étapes de test et le temps pour les concevoir, le temps de conception est presque deux fois rapide (47%) sur le second contexte où nous avons appliquer l'approche ALME. En complément, dans ce second contexte le temps passé à mettre à jour les cas de test a été distingué du temps passé à créer de nouveaux cas de test. Sur l'étude de conception de 266 cas de test avec 110 nouveaux cas de test et 156 en cours de mise à jour, nous avons observé un gain moyen de 60% sur le temps de mise à jour des cas de test, soit un temps deux fois plus rapide que la phase de conception.

Constatation : l'approche ALME est plus efficace qu'une approche manuelle.

Pour ces expérimentations, pour des contextes assez similaires, avec une source de conception et d'implémentation de cas de test similaire, les résultats de nos expériences indiquent que la conception de tests avec l'approche ALME est 47% plus efficaces qu'une approche de conception manuelle (voir la figure 3.20).

CONSTAT 4 – L'approche ALME est plus efficace qu'une approche manuelle

L'efficacité de l'approche ALME vis à vis d'une approche manuelle peut s'expliquer par le fait que la conception des tests est complètement refondée. La conception manuelle des tests se fait généralement dans un outil de gestion et d'exécution des tests tel que Hp ALM, Squash TM ou Excel (Ici, Squash TM est utilisé).

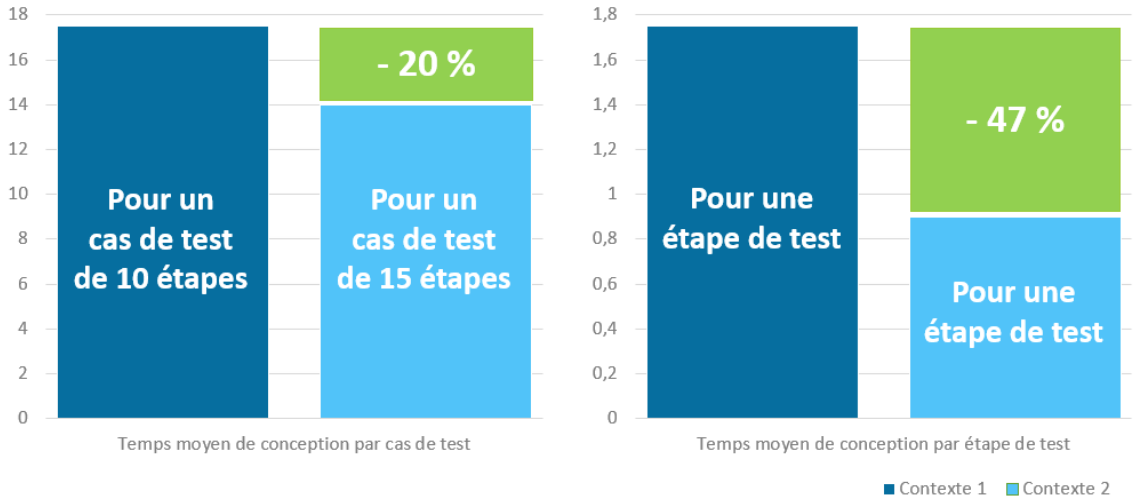


FIGURE 3.20 – Résumé des résultats de l'étude sur les temps de conception entre la conception et l'implémentation manuelles par rapport à ALME

Dans ces outils, en cas d'évolution affectant un ensemble de cas de test, il est nécessaire d'aller rechercher plusieurs cas de test existants et de les combiner pour former de nouveaux cas de test. Ou alors de les concevoir en ne partant de rien. Dans le cas de mise à jour affectant un ensemble de cas de test, il est nécessaire d'appliquer individuellement des mises à jour à chaque cas de test pour assurer leur non-obsolescence.

Avec l'approche ALME que nous préconisons (et l'outillage MBT adapté), il est facile d'identifier les modèles affectés par des changements, de les mettre à jour selon les besoins et de générer ensuite tous les cas de test mis à jour, sans avoir à les modifier individuellement. Cette génération de cas de test fait gagner beaucoup de temps, ce qui est possible grâce à la facilité d'utilisation de l'outil, d'où l'aspect "léger" qui permet des mises à jour faciles et rapides dans les modèles, puis la régénération des cas de test. Cette tendance est confirmée par l'étude des temps de mise à jour. Par rapport à une approche manuelle, la conception avec le MBT léger donne de bons résultats et est plus efficace pour la conception et l'implémentation des cas de test et très efficace pour la mise à jour des cas de test.

Les résultats de ces expérimentations nous permettent de discuter les trois questions de recherche posées en début de ce chapitre (section 3.1).

Pour les questions RQ-1 et RQ-2 qui concernent la pertinence de l'approche dans les contextes existants : la réalisation des modèles ALME pour scénariser les tests fonctionnels a été réalisé par des testeuses fonctionnelles, sans connaissance initiale en modélisation (ni en programmation). La montée en compétence a été rapide (quelques heures pour se saisir des éléments graphiques) et la production des tests a été conforme à ce qui était attendu par l'équipe. Mais cette appropriation a nécessité de mettre en place

de bonnes pratiques sur la gestion des artefacts et l'organisation du travail collaboratif. C'est cette expérience que nous avons capitalisé dans les éléments de l'approche ALME sur la gestion des modèles (section 3.4.1).

Pour la question RQ-3, qui concerne les cycles courts en Agile, les expérimentations ont montré que la scénarisation visuelle des tests était adaptée à une démarche itérative et incrémentale. Mais, cela nécessite, là aussi, de bonnes pratiques de modélisation : des modèles focalisés sur l'objectif de test [14] et avec une factorisation des fonctions communes (tels que présentés en section 3.4.2). Ces résultats expérimentaux devront être consolidé dans le futur à partir des remontées d'usage de l'approche ALME en projet réel.

DISCUSSION SUR LES BIAIS DE VALIDITÉ

Le contexte des projets cités, ainsi que l'expérience / la formation des deux testeuses ayant réalisé l'expérience, nous paraissent représentatifs de la réalité de terrain des tests fonctionnels dans le domaine des applications du SI d'entrreprise.

Le principal biais de nos résultats nous semble liés au besoin de reproduire l'expérience plusieurs fois pour en tirer des moyennes statistiques ; Nous n'avons pas pu réitérer dans la durée de la thèse du fait du temps consacré aux contributions et expérimentations sur les 3 autres sujets liés à notre approche ALME : l'automatisation de l'exécution (cf. chapitre 4), le refactoring des tests manuels (cf. chapitre 5) et l'étude des pratiques des tests inter-équipes dans l'agilité à l'échelle (cf. chapitre 6). Refaire d'autres expériences comparatives de l'approche ALME pour en vérifier les résultats obtenus reste à réaliser à l'issue de cette thèse.

3.6/ SYNTHÈSE

Nous avons présenté dans ce chapitre notre approche ALME au travers du cycle de l'ATDD visuel qui en constitue le fondement.

Notre approche est basée sur un ensemble d'étapes : la découverte du besoin métier et la formulation visuelle des parcours applicatifs à tester, la conception détaillée des tests et l'implémentation pour l'automatisation ou l'exécution manuelle, en s'appuyant sur des représentations visuelles simples, afin de rendre le MBT plus accessible pour les testeurs fonctionnels.

En complément, nous nous sommes attachés à adapter l'approche MBT aux contextes en

14. Ce terme est défini en Annexe A.2 à partir de la norme ISO/IEEE/IEC 29119-1 et du glossaire de l'ISTQB.

Agile à l'aide de bonnes pratiques qui facilitent la gestion des modèles dans un contexte itératif et incrémental. Il s'agit aussi de faciliter la collaboration à partir de l'expression des besoins métier entre les différents rôles de l'équipe en Agile avec la formulation visuelle des parcours applicatifs.

Nous avons également expérimenter l'approche ALME en contexte projet en impliquant les testeurs en place.

La première expérimentation nous a permis de montrer l'adéquation de la notation de modélisation pour représenter simplement les parcours applicatifs à tester. Dans la deuxième expérimentation (en contexte 2), avec un nombre limité d'artefacts de modélisation, les testeuses fonctionnelles ont été en mesure de représenter les règles métier à vérifier et de générer les cas de test requis. Cela montre que notre approche est capable de vérifier de grands systèmes tout en étant simple à utiliser, accessible aux testeurs fonctionnels.

Ensuite, nous avons cherché à évaluer si, en plus de faciliter la scénarisation des tests fonctionnels des applications du SI, il était également possible de le faire avec un rythme qui se prête à des itérations en contexte Agile. Pour ce faire, nous avons appliqué avec succès notre approche dans un contexte de projet en Agile avec des itérations de 4 semaines, et montrant un effort de conception des tests divisé par 2.

Le chapitre suivant traite de l'automatisation de l'exécution des tests avec ALME au cours d'une itération.

4

CONTRIBUTIONS TECHNIQUES : AUTOMATISATION DES TESTS FONCTIONNELS DANS L'APPROCHE ALME

L'accélération du rythme de mise en production des évolutions des logiciels est un élément clé de la compétitivité. Cette accélération est désormais soutenue par une approche continue du développement et du déploiement (DevOps), et nécessite une transformation profonde des processus de développement, des technologies, des pratiques et de la culture organisationnelle. Mettre en production des incréments plus souvent augmente la fréquence des tests et pose donc le problème de l'automatisation de leur exécution.

De plus, la complexité sans cesse croissante des systèmes et le renforcement des exigences de qualité ont entraîné une forte augmentation des coûts de vérification et de validation (d'après le World Quality Report 2019 [1]). Faciliter l'automatisation de l'exécution des tests fonctionnels des grands systèmes constitue donc un enjeu important, que nous abordons dans l'approche ALME avec les objectifs suivants :

- Assurer la pertinence des tests automatisés vis à vis des besoins métier par une approche liant la conception des tests et l'implémentation des scripts.
- Équilibrer les efforts d'automatisation et faciliter la collaboration entre automaticien de test et testeur fonctionnel.
- Proposer des approches et bonnes pratiques pour faciliter la maintenance des artefacts d'automatisation (répertoires d'objets, mots-clés et scripts).

1. https ://www.capgemini.com/research/world-quality-report-2019/

Pour les couvrir, nos contributions dans ce chapitre sont un processus et un ensemble de bonnes pratiques couvrant les différentes activités de l'automatisation des tests.

Elles s'inscrivent dans l'approche ALME et visent à faciliter la transition entre la scénarisation des tests, leur conception détaillée et leur implémentation sous la forme de scripts de test exécutables automatiquement. Il s'agit d'établir une continuité entre la modélisation des parcours applicatifs, intuitive pour les testeurs fonctionnels, et des mots-clés pour automatiser les étapes de test de bas niveau (au niveau de l'interface graphique ou de l'API).

4.1/ PRINCIPE D'AUTOMATISATION DANS ALME

Notre approche d'automatisation des tests articule les principes du test à base de modèles et à base de mots-clés.

Les étapes de test décrites dans le modèle et formant les cas de test sont transcrites en mots-clés d'automatisation de test avec des paramètres (les données de test). Cette conversion est possible grâce à la complétion d'une couche d'adaptation, afin de produire automatiquement des scripts de test. Les sections suivantes décrivent le processus menant à la production de scripts. À ce stade, nous considérons que les modèles et les cas de test manuels ont été produits comme présenté dans le chapitre précédent. Nous décrivons donc ici la complétion de la couche d'adaptation et la production des scripts. Nous détaillons également un ensemble de bonnes pratiques visant à faciliter la maintenance des artefacts d'automatisation qui composent ces différentes activités.

4.2/ COMPLÉTION DE LA COUCHE D'ADAPTATION

La complétion de la couche d'adaptation se compose d'un ensemble d'étapes dans notre approche :

- Construire les répertoires d'objets.
- Implémenter les mots clés.
- Relier les mots clés aux étapes de test des parcours applicatifs.

Pour chacune de ces étapes, nous avons émis des recommandations que nous allons présenter. Nous décrivons comment les rôles qui sont amenés à intervenir durant ces étapes (le testeur fonctionnel et l'automaticien de test) réalisent ces activités.

4.2.1/ CONSTRUIRE LES RÉPERTOIRES D'OBJETS

Les répertoires d'objets sont des bibliothèques contenant tous les objets du SUT nécessaires à l'automatisation, construites par l'automaticien de test, et qui sont intégrés dans la solution d'automatisation [2]. Ainsi la solution d'automatisation se compose des répertoires d'objets qui incluent des boutons, des champs, des formulaires ou tout autre objet sur lequel il est possible d'effectuer des actions pour tester l'application.

Pour tester une interface de connexion, le référentiel d'objets contiendra les champs "identifiant", "mot de passe" et un bouton "soumettre". Selon la technologie d'automatisation utilisée, le répertoire d'objets est construit différemment.

Bonne pratique

Identification précise des objets : les objets doivent être nommés et identifiés avec précision pour faciliter leur gestion et maintenance.

BONNE PRATIQUE 8 – Identification des objets

Fréquemment, l'identification des objets se fait par un "chemin" menant à l'objet. Ce chemin est plus ou moins complexe selon le type de SUT. Dans le cas des applications web, les objets sont identifiés par un xpath, ou un id (entre autres). Avoir un id pour identifier un objet est la meilleure façon de faciliter le processus d'automatisation et sa maintenance. En effet, même si au cours des évolutions de l'application, l'objet se déplace dans la structure, le script sera toujours capable de trouver un objet, contrairement à un xpath. Une bonne pratique pour faciliter la création et la maintenance du répertoire d'objets consiste donc à associer un identifiant (id) à chaque objet de l'application. S'il n'est pas disponible, il faut identifier la propriété la plus forte (c'est-à-dire celle qui est "la plus unique" et la moins sujette au changement) de l'objet pour l'identifier. Par exemple, le nom de l'objet, son texte, ou toute autre propriété susceptible de ne pas changer et permettant de sélectionner l'objet de manière unique.

Quant au nommage, il doit idéalement combiner vision technique et métier. Pour couvrir l'aspect technique, le nommage de l'objet peut contenir le type du champ qu'il référence, par exemple un champ texte, un bouton, etc. Ceci permet de donner un indicateur autant à l'automaticien qu'au testeur fonctionnel sur les actions réalisables sur l'objet (un clic pour un bouton, une complétion pour un champ texte). Pour couvrir la vision métier, l'objet peut mentionner dans son libellé le nom du champ tel qu'il est affiché dans l'application. Ainsi en combinant ces pratiques, des exemples d'objets seraient "champTexte_nomClient', "champTexte_prénomClient", "bouton_validerDevis", etc. Chacun d'eux est as-

2. Une définition de ce terme, issue du glossaire ISTQB est fournie en Annexe A.2

sez explicite pour qu'un testeur fonctionnel lors de la lecture d'un script soit en mesure d'identifier sur quels champs de l'application une action est réalisée. Ainsi, cette pratique contribue à une meilleure compréhension des tests automatisés en facilitant l'identification des objets de test et des actions exercées sur eux dans les scripts.

Bonne pratique

Organisation des objets : les objets doivent être organisés efficacement dans la solution d'automatisation pour permettre une mise à jour facile pour l'automaticien.

BONNE PRATIQUE 9 – Organisation des objets

La construction des répertoires d'objets doit permettre de facilement localiser les objets à mettre à jour lorsque nécessaire. Les objets peuvent être classés par interface ou par fonctionnalité selon la façon dont l'application est mise en place.

4.2.2/ IMPLÉMENTER LES MOTS-CLÉS

Les mots-clés sont des fonctions d'automatisation permettant d'activer les comportements du SUT. Elles sont généralement produites par un automaticien de test. Quelle que soit la technologie utilisée, les mots-clés comportent des propriétés, au minimum un nom et des paramètres (données, objets des bibliothèques d'objets), qui doivent être gérées pour faciliter leur compréhension et leur usage par un testeur fonctionnel ainsi que leur maintenance par un automaticien de test. Voici les propriétés que nous avons traité afin de faciliter la gestion des mots-clés et à plus large échelle la compréhension des scripts.

LES NOMS DES MOTS-CLÉS

Une attention particulière doit être portée au nommage des mots-clés. C'est d'abord par le nom des mots-clés que le testeur fonctionnel et l'automaticien de test en percevront le sens. Il est donc nécessaire de mettre en place un système de dénomination qui permette au testeur fonctionnel de comprendre les scripts et à l'automaticien de test de gérer sa bibliothèque de mots-clés. Les noms des mots-clés doivent être significatifs et correspondre aux opérations qu'ils représentent. Par exemple "Connexion", "RéserverUnVol (Départ, Arrivée)" sont des noms descriptifs et utilisent des données qu'un testeur fonctionnel peut comprendre et utiliser pour relier les mots-clés aux étapes de test.

Bonne pratique

Nommage des mots-clés : les noms des mots-clés doivent être significatifs et correspondre aux opérations qu'ils représentent.

BONNE PRATIQUE 10 – Nommage des mots-clés

LES PARAMÈTRES

Dans l'exemple ci-dessus pour le mot-clé "RéserverUnVol (Départ, Arrivée)", deux paramètres sont utilisés : la ville de départ et la ville d'arrivée. La difficulté dans la gestion de ces paramètres est qu'ils doivent être définis dans un format approprié pour être le plus robuste possible à la modification. Ici, les paramètres sont en attente d'une chaîne de caractères : il y a différentes propriétés à prendre en considération. Si le testeur fonctionnel choisit "paris" comme ville de départ et que le SUT attend "Paris", il est possible que le test échoue à cause de la gestion des majuscules et des minuscules. Dans cette situation, l'automaticien de test pourrait effectuer un traitement sur les données passées en paramètre pour s'assurer qu'une majuscule est présente pour le premier caractère du paramètre et, si ce n'est pas le cas, pour l'intégrer. Mais dans d'autres circonstances, le paramètre ne doit pas être retouché, car le test aura pour but de vérifier la propriété que le SUT doit recevoir le nom exact de la ville, en respectant les majuscules au début du mot. La difficulté dans la gestion des paramètres est de savoir quelles valeurs sont attendues et acceptées.

Bonne pratique

Spécification des paramètres des mots clés : le testeur fonctionnel et l'automaticien de test doivent travailler ensemble et définir correctement le comportement attendu des fonctions ainsi que le format et les valeurs des données acceptées dans les paramètres.

BONNE PRATIQUE 11 – Spécification des paramètres de mots-clés

L'automaticien fournit un mot-clé comme "CompleterChampTexte(champ, valeur)" permettant de compléter un champ texte d'un formulaire web identifié par son nom (paramètre champ) et implémenté avec une valeur (paramètre valeur). Le testeur fonctionnel peut renseigner le paramètre "champ" avec le nom du champ du formulaire web qu'il souhaite alimenter et la valeur qu'il veut entrer dans le paramètre "valeur". Dans cette

configuration, l'automaticien de test doit fournir les valeurs des bibliothèques d'objets (ici le nom des champs du formulaire) au testeur fonctionnel pour permettre à cette technique de fonctionner. D'où l'importance d'une forte synergie entre l'automaticien de test et le testeur fonctionnel et d'une solide gestion du répertoire d'objets. Cette synergie équilibre l'effort d'automatisation entre l'analyste de test et l'automaticien.

LA GRANULARITÉ DES MOTS-CLÉS

La gestion de la granularité des mots-clés est essentielle pour faciliter leur mise en place et leur maintenance. La difficulté sera de choisir quel niveau de granularité mettre en œuvre et quand : faut-il écrire un mot-clé de connexion, ou faire apparaître les champs à compléter suivi d'une validation pour effectuer la connexion ?

Afin de gérer au mieux cette granularité, nous avons établi 5 facteurs qui varient en fonction de la granularité : la réutilisabilité du mot-clé, son effort de maintenance, la compréhension du script produit (pour le testeur fonctionnel et l'automaticien de test) et l'investissement à réaliser d'une part par le testeur fonctionnel et d'autre part par l'automaticien de test. La table de la figure 4.1 classe la granularité en 3 niveaux : faible, moyenne et élevée et selon ces niveaux de granularité les mêmes échelles sont appliquées aux facteurs. De plus, 4 couleurs sont utilisées : vert, jaune, orange et rouge selon que le facteur est bon, moyen, moyen-moins ou mauvais.

Granularité	Réutilisabilité	Effort de maintenance	Compréhension du script	Investissement du testeur fonctionnel	Investissement de l'automaticien de test
Elevée	Elevée	Faible	Faible	Elevé	Faible
Moyenne	Moyenne	Moyen-Elevé	Moyenne	Moyen	Moyen
Faible	Faible	Elevé	Elevée	Faible	Elevé

FIGURE 4.1 – Granularité des mots-clés et facteurs

Un niveau de granularité élevé peut être associé à un mot-clé du type "completerChamp(Champ, Valeur)" : il peut être utilisé pour vérifier diverses fonctionnalités et champs. On peut donc voir que sur un niveau de granularité élevé, la réutilisabilité des mots-clés est élevée, ce qui est un très bon point, mais en contrepartie le travail du testeur fonctionnel se trouve alourdi. Chaque script de test nécessite un grand nombre de mots-clés pour être exprimé. Cela conduit à des scripts peu lisibles, car ils ne comprennent que des étapes pour remplir des champs et cliquer sur des boutons qui ne sont pas représentatifs du comportement métier qui se cache derrière. L'automaticien de test doit créer la bibliothèque de mots-clés, mais elle est limitée et sa maintenance nécessite moins d'efforts. Le principal avantage d'une granularité élevée est donc la réutilisabilité.

Un niveau moyen de granularité peut être représenté par un mot clé du type "choisirUnSiteDeReservation(URLSiteDeRéservation)" : il regroupe l'ensemble des opérations

permettant d'accéder à un site de réservation tout en étant prévu pour être utilisé sur plusieurs sites grâce au paramètre. Par exemple, ce mot-clé pourrait permettre de choisir un site de réservation pour ensuite vérifier le processus de réservation. Dans ce cas le code du mot-clé doit être adapté pour identifier le site où le test est effectué et le mot-clé doit être adapté pour permettre la réservation sur chacun d'eux. Pour un niveau de granularité moyen, tous les facteurs ont un degré moyen sauf l'effort de maintenance qui peut être élevé. L'investissement du testeur fonctionnel comme celui de l'automaticien de test sont moyens, car l'effort d'automatisation est réparti sur les deux rôles. Cela s'explique par le fait que les mots clés sont suffisamment compréhensibles pour apporter une partie de la vision métier, donc les scripts sont plutôt compréhensibles. Par contre, l'effort de maintenance peut être complexe, car les mots clés ne sont pas dédiés à un contexte spécifique, il faut donc être prudent lors de la mise à jour d'un contexte particulier pour éviter de casser les autres contextes.

Un faible niveau de granularité peut être représenté par un mot-clé du type "RéserverUnHotelSiteA(Arrivée, Départ)" : il ne peut être utilisé que dans un cas d'utilisation particulier. Pour ce niveau de granularité, la réutilisabilité est faible, car un mot-clé très spécifique ne pourra pas être réutilisé de manière régulière dans un script totalement différent de celui pour lequel il a été créé. Habituellement, les mots clés de granularité faible couvrent un périmètre précis et ne peuvent pas être utilisés sur d'autres périmètres, mais peuvent être utiles pour les tests de non-régression. Comme les mots-clés ont un faible niveau de granularité, la compréhension des scripts est facilitée, car ils contiennent des mots clés avec des noms significatifs permettant de comprendre les actions effectuées dans le script du point de vue métier. Ainsi, l'investissement du testeur fonctionnel est limité, car il lui est facile, pas à pas, de reproduire le comportement du SUT et de compléter la couche d'adaptation, mais pas pour l'automaticien de test. Il aura un grand nombre de mots-clés à mettre en place et qui seront dédiés à des comportements spécifiques. L'effort de maintenance est donc plus élevé, car les mises à jour des mots-clés doivent être faites pour chacun des périmètres de test, pour un grand nombre de mots-clés. La table de la figure 4.2 donne un exemple de fonctions qui peuvent être utilisées pour différents niveaux de granularité.

Bonne pratique

Granularité des mots-clés : nous préconisons d'adapter la granularité des mots-clés à leur utilisation, à répartir entre des mots-clés de granularité faible, moyenne et élevée et à dupliquer certains comportements sur des niveaux de granularité différents.

BONNE PRATIQUE 12 – Granularité des mots-clés

Granularité élevée	Granularité moyenne	Granularité faible
CompleterChamp (champ, valeur)	choisirSiteRéservation (urlSiteRéservation)	réserverunHotelSiteA (arrivé,départ)
cliquerBouton (bouton)	réserverUnHotel (arrivé,départ,hotel)	réserverunHotelSiteB (arrivé,départ)
completerChamp (arrivé,dateArrivé)		
completerChamp (départ,dateDépart)		
completerChamp (hotel, nomHotel)		

FIGURE 4.2 – Exemple de mots-clés par granularité

Considérons un tout autre exemple, pour la complétion d'un formulaire web comportant 4 champs, certains étant obligatoires et d'autres optionnels. Considérons que notre objectif de test est de vérifier que les règles de gestion liées à chaque champ s'appliquent bien et qu'il est possible de valider le formulaire (selon que les champs sont complétés ou non et que les valeurs utilisées pour les compléter sont valides ou pas). Deux champs sont obligatoires, le nom et le prénom et les deux autres optionnels, le nom marital et la date de naissance. On peut dans ce cas créer 4 fonctions de granularité élevée pour remplir chacun des champs et deux fonctions de granularité moyenne, l'une pour compléter l'ensemble des champs obligatoires et une autre fonction pour compléter les champs optionnels. Enfin une fonction de granularité faible serait une fonction qui permettrait la complétion des 4 champs. Ainsi, en fonction de ses besoins, le testeur fonctionnel peut choisir de remplir le formulaire dans son intégralité (en particulier s'il effectue des tests transverses) ou de ne remplir qu'un seul champ s'il effectue des tests plus ciblés sur un aspect particulier. Ou encore de choisir de ne compléter que les champs obligatoires ou optionnels selon ses objectifs de test.

Il est intéressant de créer et d'utiliser des mots-clés à granularité élevée pour des champs très indépendants. C'est-à-dire des champs pour lesquels les interactions avec d'autres mots clés sont faibles et dont les propriétés peuvent être vérifiées indépendamment des autres éléments de l'application. Cela peut être le cas pour les champs de formulaire. De plus, ces mots clés peuvent être repris et combinés par l'automaticien de test pour construire de nouveaux mots clés de granularité plus faible. Leur utilisation est recommandée dans le cadre de vérifications ciblées (propriétés des champs par exemple) et à proscrire dans la création de scénarios de bout en bout, qui seraient illisibles et difficiles à maintenir. Les mots-clés de granularité moyenne se prêtent à la vérification de

comportements transverses. Il est intéressant de produire et d'utiliser des mots-clés de granularité moyenne lorsqu'un comportement est vérifiable dans différents contextes, interfaces, etc. Les mots-clés à granularité faible sont plutôt dédiés à la construction de scénarios de bout en bout. La séquence de mots-clés à granularité faible doit permettre de vérifier un ensemble de propriétés d'un système. Contrairement aux mots-clés à granularité moyenne, les mots-clés à granularité faible ne se prêtent pas à la vérification de comportement partagé. Ils sont dédiés à un contexte particulier pour la vérification de propriétés ciblées (comportement d'une interface, d'un ensemble).

En combinant ces 3 niveaux de granularité, il est possible d'atteindre un équilibre entre chacun des 5 facteurs et de faciliter ainsi la mise en place et la maintenance des mots-clés. Et aussi d'équilibrer l'effort d'automatisation entre le testeur fonctionnel et l'automaticien de test.

Par l'application des pratiques présentées pour la création des répertoires d'objets et l'implémentation des mots clés nous cherchons à couvrir le 1er objectif énoncé dans ce chapitre qui traite de l'amélioration de la compréhension des tests fonctionnels automatisés. L'amélioration de la compréhension des scripts passe par une définition rigoureuse des objets de l'application dans la solution d'automatisation. C'est le nommage de ces objets qui va entre autres permettre une meilleure compréhension des scripts. C'est également un nommage explicite des mots clés qui va participer à une meilleure lecture des scripts. Ainsi le testeur fonctionnel est en mesure de comprendre le script à travers l'enchaînement des mots clés et de leurs paramètres. Ces derniers étant eux aussi explicites et qui décrivent clairement les objets manipulés par l'application. En complément, ces pratiques participent à un meilleur équilibrage de l'effort d'automatisation entre le testeur fonctionnel et l'automaticien en rendant accessible la production des scripts par le testeur fonctionnel. Nous décrivons cela dans la prochaine section qui traite de la liaison entre les mots clés et les étapes de test.

4.2.3/ RELIER LES MOTS CLÉS AUX ÉTAPES DE TEST

Dans notre approche la production des scripts est basée sur la liaison préalable des étapes de test à des mots clés. Par exemple, pour l'action "se connecter à l'application" on pourra lier un mot-clé "connexion(id,mdp)" qui est une fonction permettant la connexion sur le SUT à l'aide d'un identifiant et d'un mot de passe. Dans les outils, la liaison d'une étape de test à un mot clé peut se faire de différentes manières.

Dans l'outil Yest, pour un groupe de cas de test sélectionné il est possible de lier chaque étape de test à un ou plusieurs mots clés. L'interface permettant cette association est visible en figure 4.3, à gauche. Le code des mots clés n'est pas accessible dans l'outil, mais dans la solution d'automatisation (dans un projet Java-Selenium par exemple).

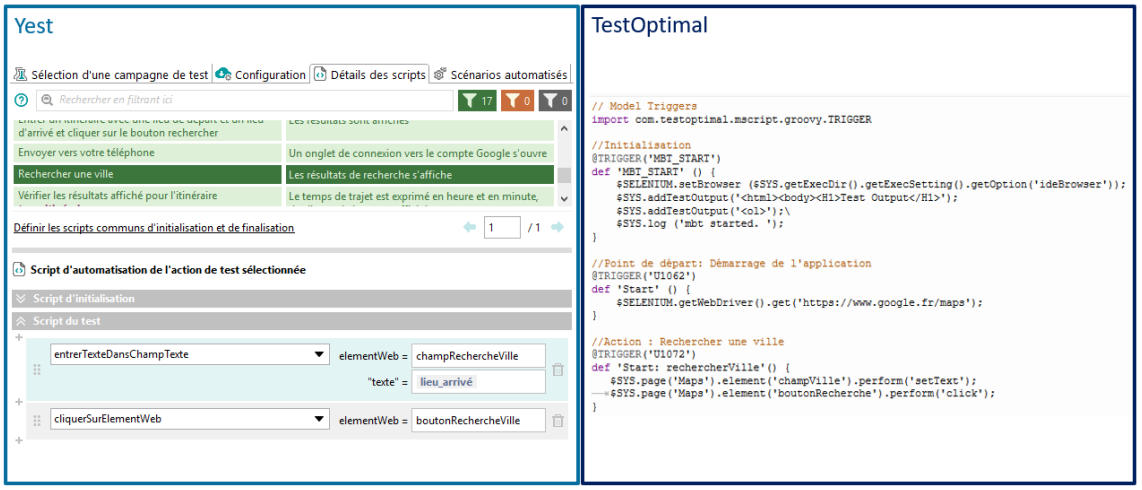


FIGURE 4.3 – Liaison des étapes de test aux mots-clés

Dans TestOptimal, il est possible de faire de même, mais le code des mots clés est directement accessible dans l'outil. Pour chaque tâche contenant des étapes de test, il est possible d'écrire le code d'automatisation associé. Ceci est visible dans la figure 4.3, à droite.

Une fois les mots-clés reliés aux étapes de test, la production de scripts peut être réalisée, comme présenté dans la section qui suit.

4.3/ PRODUCTION DES SCRIPTS

Dans notre approche la production des scripts est réalisée grâce aux représentations visuelles. C'est l'ensemble des mots clés associés aux étapes de test qui la compose qui va permettre la génération des scripts.

Les scripts sont à notre sens l'artefact d'automatisation le plus fragile, leur fonctionnement dépendant à la fois des données et des mots-clés. La plupart des modifications opérées au niveau du SUT, tant au niveau des comportements qu'au niveau des données, nécessitent de reconsidérer les scripts de test.

L'orchestration des mots-clés et la gestion des données qui les composent étant un réel défi, nous conseillons plutôt l'usage de solutions outillées qui permettent la génération de scripts à partir des modèles. L'idée étant de permettre une conversion facile des cas de test manuel vers les scripts automatisés.

Bonne pratique

Écriture des scripts de test : nous préconisons, dans le cadre de l'utilisation d'un outil permettant d'associer les représentations visuelles aux mots-clés d'automatisation, de confier l'écriture des scripts de test au testeur fonctionnel plutôt qu'à l'automaticien de test.

BONNE PRATIQUE 13 – Écriture des scripts de test

En utilisant des outils qui permettent une liaison facilitée des mots clés aux étapes de test, l'effort d'automatisation entre l'automaticien de test et le testeur fonctionnel est mieux équilibré. Habituellement, l'activité d'écriture des scripts est plutôt prise en charge par l'automaticien de test, dans notre approche nous préconisons à la place qu'elle soit prise en charge par le testeur fonctionnel. Différentes raisons guident ce choix. Tout d'abord, le testeur fonctionnel a généralement une meilleure vision des besoins métier que l'automaticien de test. Ainsi en favorisant la production des scripts par le testeur fonctionnel notre approche participe à renforcer le respect des besoins métier. Ensuite, la charge d'automatisation est réduite pour l'automaticien qui n'a plus à produire les scripts. Ainsi l'automaticien peut davantage se concentrer sur la maintenance des mots clés et des objets d'automatisation qui sont des activités plus difficilement réalisables par un testeur fonctionnel et qui sont consommatrice en temps. De cette manière l'effort d'automatisation global est plus équilibré entre ces deux acteurs.

Bonne pratique

Maintenance des modèles : le testeur doit veiller à ce que les modèles soient tenus à jour. Cela signifie qu'il doit intégrer les nouveaux comportements dans les modèles dès que possible et re-générer les cas de test mis à jour.

BONNE PRATIQUE 14 – Tenir à jour les modèles

De cette façon, la couche d'adaptation peut être complétée avant de générer les scripts. L'intérêt de faire les mises à jour régulièrement est que les activités peuvent être réalisées de manière itérative et incrémentale. Ceci afin de s'intégrer efficacement dans les cycles Agiles. Il n'est pas nécessaire que les comportements soient complets pour les vérifier, la vérification est effectuée de manière progressive, dès que possible, comme le recommandent les préceptes Agiles.

L'application de ces différentes bonnes pratiques contribue à maintenir les scripts de test et à faciliter la gestion du processus d'automatisation.

Nous ne présentons pas d'avantage d'éléments sur la gestion des scripts, car elle fait peu partie de notre contribution. Bien que notre approche contribue à faciliter la maintenance des scripts par une génération de ceux-ci, la gestion des scripts dépend de chaque contexte et des solutions employées pour leur exécution.

4.4/ EXPÉRIMENTATIONS

Dans la section ci-dessus, nous avons introduit un ensemble de bonnes pratiques visant à faciliter le processus d'automatisation et sa maintenance dans le cadre de l'approche ALME.

Nous décrivons ici comment nous les appliquons pour tester une application web dans le cadre de deux grands projets informatiques.

Nous avons tout d'abord expérimenté notre approche au sein d'un projet Agile en milieu de développement (après un an et demi de projet). La deuxième expérimentation porte sur un cycle de 10 jours pour évaluer dans quelle mesure l'approche est adaptée à des itérations courtes.

Chacun des projets utilisait un cadre d'automatisation différent : Selenium[3] pour la première expérimentation et UFT (Unified Functional Testing)[4] pour la seconde. Ceci nous a permis de mettre en œuvre l'approche sur deux technologies différentes et de nous assurer que les bonnes pratiques peuvent être appliquées dans différents contextes.

4.4.1/ EXPÉRIMENTATION 1 : INTRODUCTION TARDIVE DE L'AUTOMATISATION DES TESTS FONCTIONNELS DANS L'APPROCHE ALME

La première expérimentation porte sur le même projet que celui abordé dans le chapitre 3 (Expérimentation 2 / Contexte 1). C'est à dire le test d'une application web dédiée à la gestion de l'évaluation des agents dans une société ferroviaire française. Elle a été développée dans un contexte Agile avec des sprints de 4 semaines et une stratégie de test pour vérifier les fonctionnalités mises en œuvre à la fin de chaque sprint.

L'automatisation des tests a commencé 1,5 ans après le début du développement de l'application et a utilisé l'approche ALME avec l'outil Yest et le framework d'automatisation Selenium. Le démarrage tardif de l'automatisation s'explique par diverses contraintes projets (manque de budget, de temps, de personnels) qui ont mené à reporter l'automatisation. Les cas de test à automatiser ont été construits par un spécialiste de l'automatisa-

3. https ://www.selenium.dev/
4. https ://www.microfocus.com/fr-fr/products/uft-one/overview

tion des tests à l'aide de Yest, puis il a complété la couche d'adaptation en collaboration avec un testeur fonctionnel et a construit le code d'automatisation.

Cette expérimentation est focalisée sur la production des tests de régression automatisés. Pour cela, de nouveaux scénarios de test ont été construits à partir de notre approche ALME tel que décrit dans la section suivante.

DESCRIPTION DES ASPECTS FONCTIONNELS DU SUT

Notre expérimentation porte sur un aspect de l'application : le processus d'évaluation. Le principe du processus d'évaluation est de réaliser le contrôle des compétences des employés dans une interface web. À cette fin, l'évaluateur peut choisir un métier, un groupe de veille et le type d'évaluation (parmi ceux auxquels il a accès en fonction de ses droits) et il suit une procédure pour réaliser une évaluation. Ainsi, pour la phase de préparation de l'évaluation, un évaluateur qui a accès à deux métiers, trois groupes de veille et deux types d'évaluation peut réaliser douze évaluations différentes. Selon les métiers et les groupes de veille, l'évaluation est réalisée différemment et de nouvelles actions doivent être produites. En outre, d'autres contrôles sont réalisés au cours du processus d'évaluation, ce qui augmente le nombre de comportements possibles. Ainsi, la vérification du processus d'évaluation est complexe par le nombre de comportements possibles à contrôler en raison d'une grande combinaison de données. Afin d'expérimenter notre approche ALME dans un contexte d'automatisation, nous avons d'abord modélisé les parcours applicatifs à tester. Bien que cette phase sorte du périmètre de l'automatisation même, nous la présentons pour faciliter la compréhension globale de l'expérimentation.

DÉCOUVERTE ET FORMULATION VISUELLE DES PARCOURS APPLICATIFS

Afin de tester le processus d'évaluation, un ensemble de représentations visuelles a été créé en veillant à la mise en œuvre des bonnes pratiques. En particulier, les comportements communs identifiés dans les processus d'évaluations ont été isolés dans des parcours indépendants traitant des cas passants, comme préconisé dans la BONNE PRATIQUE 4.

Nous avons donc traité les comportements spécifiques dans des parcours séparés plutôt que de traiter l'évaluation dans un parcours unique qui aurait dû composer avec les propriétés de chaque cas et donc apporter de la complexité. En complément, les modèles ont été organisés selon les fonctionnalités traitées par l'évaluation, telle que préconisée dans la BONNE PRATIQUE 5.

L'application de bonnes pratiques a permis de simplifier l'activité de découverte et de

formulation visuelle des parcours applicatifs. Ce qui nous amène à une constatation similaire au chapitre 3, c'est à dire une meilleure vision sur les besoins métier (voir CONSTAT 1).

Plus de détails sont disponibles en annexe C.1.1 sur l'organisation des différents modèles.

Une fois la structure globale réalisée nous avons détaillé l'ensemble des étapes de test lors de la phase de conception détaillée des tests.

CONCEPTION DÉTAILLÉE DES TESTS

Pour toutes les étapes de test, nous avons défini un ensemble d'actions et de résultats attendus. En terme de données, nous n'avons inclus dans les parcours que les données nécessaires pour couvrir les exigences, comme nous l'avons recommandé dans la BONNE PRATIQUE 1. Par exemple, nous avons inclus des données sur le type d'évaluation, car selon ce type, les possibilités de notation différent de même que les contrôles à effectuer. Dans la figure 4.4 sont présentées des étapes d'un scénario de test. En bleu apparaît la donnée de conception sur le type d'évaluation. Par contre toutes les données concernant le métier, l'établissement, le type de plan de veille etc, n'apparaissent pas ici car leur concrétisation n'est utile que pour l'exécution. Le fait de ne compléter que les données nécessaires à la couverture des exigences nous a permis d'établir la constatation suivante.

Constatation : définition facilitée des données de conception de test.
En ne définissant que les données nécessaires aux tests, la conception des tests a été allégée. Au lieu, de mélanger les données liées aux exigences et à l'implémentation, cette phase n'a concentré l'effort que sur la définition de données propres au contrôle des exigences.

CONSTAT 5 – Définition facilitée des données de test

Des éléments complémentaires sur la conception détaillée des tests sont présentés en annexe C.1.2. Une fois cette conception terminée, l'implémentation des tests est réalisée.

IMPLÉMENTATION DES TESTS

Durant la phase d'implémentation nous avons inclus les données nécessaires à l'exécution. Ces données étaient par exemple le lieu de l'évaluation, le groupe de veille, le plan de veille, etc (Ce sont les elements visibles dans la première étape du cas de test de la figure 4.4). Nous n'avons pas spécifié ces données dans les représentations

	Actions	Résultats attendus
1	Se connecter et réaliser une évaluation du type ***S.A.M.I.*** Aller à l'écran de création d'évaluation Ajouter une évaluation (1/3) Selectionner : -le métier : -l'etablissement/unité/secteur : -le type de plan de veille : -le groupe de veille : -le type : -le sous-type : Cliquer sur suivant	-L'étape 1/3 s'affiche -Le type est sélectionné -Le groupe de veille est sélectionné -Le type est sélectionné -La technologie est sélectionnée
2	Dans l'étape 2/3, ajouter les points clés	Les points clés sont sélectionnés
3	Selectionner les agents	Les agents sont selectionnés

FIGURE 4.4 – Illustration de l'usage des données de conception

visuelles, mais dans un jeu de données pour permettre l'exécution.

Nous utilisons également des données spécifiques aux mots clés : elles ne modifient pas le comportement métier et ne sont pas des données d'implémentation, car elles ne sont pas extraites de la base de données ou de l'application. C'est le cas par exemple du paramètre de type "Webdriver" qui est utilisé pour simuler une interface web dans les scripts. C'est un paramètre des mots-clés, car chacun d'eux le manipule pour effectuer des stimulations sur l'interface de l'application.

Le temps passé sur la phase de représentation des besoins métier, la conception et l'implémentation des tests, représente 30% du processus d'automatisation.

Les 9 scénarios produits au cours de ces phases sont tous convertis en scripts par la complétion de la couche d'adaptation qui est décrite dans la section suivante.

COMPLÉTION DE LA COUCHE D'ADAPTATION

Tout au long de la complétion de la couche d'adaptation, nous avons progressivement construit un répertoire d'objets qui, au final, comptait environ 200 objets. Nous avons appliqué les bonnes pratiques (BONNES PRATIQUES 8 et 9) énoncées sur la gestion des objets. Nous les avons organisés par blocs fonctionnels, par exemple synthèse, compte-rendu, écran de cotation, etc. Ensuite quand cela était possible nous avons identifié les

objets par leur id et sinon par leurs propriétés les plus fortes (texte, lien, etc).

Constatation : organisation des données.
Par l'application des bonnes pratiques, les données ont été correctement organisées. D'une part, le nommage des données permettait une identification rapide de leur type et usage fonctionnel. Et d'autre part, le fait qu'elles soient organisées par fonctionnalité permettait de les retrouver rapidement pour les mettre à jour en cas d'évolution des interfaces.

CONSTAT 6 – Organisation des données

Nous avons également créé un dictionnaire de mots-clés contenant 30 mots-clés. Nous avons préféré des mots-clés à granularité faible parce que nous voulions couvrir un seul domaine spécifique. Cependant des mots clés à granularité plus élevée ont été créés pour faciliter la création de mots clés de plus faible granularité, comme préconisé dans la BONNE PRATIQUE 12. Plus de détails sur le dictionnaire sont présentés en annexe C.1.3

Bien qu'il existe plusieurs niveaux de granularité des mots clés, le testeur fonctionnel utilise principalement des mots clés de granularité faible pour lier chaque étape de test à des mots clés. La figure 4.5 présente un extrait du dictionnaire de mots-clés sous forme de table auquel le testeur avait accès. Son travail est facilité par le fait qu'il manipule des éléments qu'il connaît, c'est à dire des mots clés dont les noms sont en lien avec les étapes de test, comme préconisé dans la BONNE PRATIQUE 10.

	Keyword	param1	param2	param3	param4	param5	param6	param7
2	construireDriver	lien_application	driver					
3	fermerDriver	driver						
4	seConnecter	identifiant	driver					
5	ajouterPointsClesPourEvaluation	listePointsCles	driver					
6	ajouterEvaluationEtape1sur3	metier	etablissement	typeDePlanDeVeille	groupeDeVeille	typeE	sousType	driver
7	validerCotation	avertissement	driver					
8	terminerEvaluation	driver						
9	accederEvaluationTimeLine	driver						
10	seRendreDansEvaluation	driver						
11	effectuerNotationPointsObserves	pointsObserves	notation	precision	driver			
12	effectuerCotation	ChoixCotation	driver					
13	choisirLieuDeSuiviEvaluation	lieu	driver					
14	choisirAgentAEvaluer	agent	driver					

FIGURE 4.5 – Dictionnaire de mots-clés

Toutefois l'automaticien peut mettre à la disposition du testeur fonctionnel des mots clés de granularité plus élevée pour qu'il gagne en autonomie sur la complétion de la couche d'adaptation. C'est par exemple le cas avec un mot-clé "fermerDriver" qui permettrait au testeur fonctionnel de choisir si après un test le navigateur doit être fermé ou non (Laisser ouvert le navigateur permet d'enchaîner plusieurs cas de test). De cette façon, le testeur est complètement autonome pour l'écriture des scripts et peut réaliser pleinement cette activité comme cela est préconisé dans la BONNE PRATIQUE 13.

En complément lorsqu'une étape de test est partagée entre plusieurs cas de test, par

exemple ”effectuer une cotation”, le testeur fonctionnel ne lie un mot clé qu'une seule fois à l'étape de test. Dans l'exemple des cas de test couvrant les types de cotation (issue du parcours C, visible en annexe dans la figure C.1) il y a un total de 94 étapes de test confondues pour ces cas de test. Or le testeur fonctionnel n'a qu'à compléter 11 étapes de test au total pour couvrir l'ensemble des étapes de ces cas de test. Ce mécanisme améliore grandement l'implémentation des étapes de test en réduisant l'effort à fournir pour produire des scripts automatisés.

Constatation : participation active du testeur fonctionnel à l'automatisation des tests.
Les bonnes pratiques, notamment celles proposant l'usage de mots clés de granularité faible et compréhensible du point de vue métier par le testeur fonctionnel ont permis d'impliquer celui-ci pleinement dans le processus d'automatisation.

CONSTAT 7 – Participation active du testeur fonctionnel à l'automatisation des tests

Quant aux paramètres utilisés, leur complétion est réalisée grâce à des fonctions dans la solution d'automatisation qui vont parcourir et récupérer des données sous forme de texte dans des fichiers contenant des données de la base de données. La gestion est entièrement contrôlée par l'automaticien et le testeur fonctionnel n'a pas à s'en soucier. Pour les données de conception saisies par le testeur fonctionnel, par exemple les cotations, les données de conception ont été reprises sans aucune transformation dans les mots clés. Pour plus de détail se reporter à l'annexe C.1.3

PRODUCTION AUTOMATIQUE DES SCRIPTS DE TEST

Une fois la couche d'adaptation complétée, nous obtenons 9 scripts, utilisant les 30 mots-clés et générant 400 lignes de code. Nous avons utilisé un outil de MBT léger, Yest, pour publier nos tests afin de faciliter la production et la mise à jour de nos scripts comme conseillés dans les bonnes pratiques sur la gestion des scripts (BONNES PRATIQUES 13 et 14).

Constatation : production de scripts respectant la vision métier.
Les scripts produits sont de granularité faible permettant une compréhension de ceux-ci par le testeur fonctionnel et sont pertinents vis à vis des besoins métier.

CONSTAT 8 – Production de scripts respectant la vision métier

Un exemple d'extrait de scripts est visible dans la figure 4.6.

Ces scripts sont exécutés dans un environnement d'exécution en utilisant pour certains scripts des données issues des cas de test (dans la figure 4.6 c'est le cas pour le mot-clé

```
@Test
public void testEvaluation_SAMI(){
        seConnecter(getValue( dataName: "id",data), driver);
        seRendreDansEvaluation(driver);
        ajouterEvaluationEtape1sur3(getValue( dataName: "metier",data),getValue( dataName: "structure",data),get
        ajouterPointsClesPourEvaluation(getValue( dataName: "pcl",data),getValue( dataName: "espace",data),getVa
        choisirAgentsAEvaluer(getValue( dataName: "element_veille",data), driver);
        accederEvaluationTimeLine(driver);
        choisirUnAgent(getValue( dataName: "element_veille",data), driver);
        selectionner1erPcEval(driver);
        effectuerNotationPointsObserves( elementsEvalues: "tous", ChoixCotation: "conforme", precision: "non", driver);
        effectuerCotation( ChoixCotation: "S", driver);
        effectuerCotation( ChoixCotation: "A", driver);
        effectuerCotation( ChoixCotation: "M", driver);
        effectuerCotation( ChoixCotation: "I", driver);
        validerCotation( avertissement: "non", driver);
        terminerEvaluation(driver);
        Fonctions.fermerDriver(driver);
    }
```

FIGURE 4.6 – Exemple de code d'un script

"effectuerCotation") et pour d'autres un ensemble de données extraites de la base de données (dans la figure 4.6 c'est le cas pour les fonctions qui manipulent le paramètre "data").

4.4.2/ EXPÉRIMENTATION 2 : AUTOMATISATION DES TESTS FONCTIONNELS DANS L'APPROCHE ALME DURANT UNE ITÉRATION AGILE

La deuxième expérimentation porte sur le test d'une application web inclue dans un grand éco-système de gestion de produits d'assurance. L'application étudiée permet la souscription à un nouveau produit d'assurance. Elle comporte déjà un ensemble de produits qui augmente au fil du temps. L'application est développée dans un contexte Agile avec des sprints de 10 jours. Notre expérimentation a lieu après 6 mois de développement. Durant ces 6 mois, la majorité des développements ont été consacrés à la construction d'une nouvelle base de l'application, sans interface. C'est à partir de 4 mois de développement que les premières interfaces de l'application ont été développées et donc à partir de 6 mois que des tests automatisés ont pu être mis en place. L'automatisation des tests a donc commencé 6 mois après le début du développement avec notre expérimentation utilisant l'approche ALME (avec l'outil Yest) et UFT. Les cas de test construits durant les 6 premiers mois n'ont pas été conçus avec l'optique d'être automatisés par la suite. D'où le besoin de créer de nouveaux scénarios de test. Les cas de test à automatiser ont été construits par une testeuse fonctionnelle à l'aide de Yest, ensuite, un expert en automatisation des tests a complété la couche d'adaptation en collaboration avec la testeuse fonctionnelle, puis a construit le code d'automatisation.

Dans cette expérimentation nous cherchons à vérifier que l'automatisation des tests est

réalisable dans un cycle court en Agile. Pour cela, l'expérimentation a duré 10 jours, le temps d'un sprint. Sur cette période, la testeuse fonctionnelle a été impliquée à plein temps, alors que l'automaticien lui n'est intervenu qu'à partir du 6ième jours, donc durant 5 jours.

DESCRIPTION DU SUT

Cette seconde expérimentation nous amène à nous concentrer sur la commercialisation d'une nouvelle offre de produits d'assurance sur l'application présentée. Cette nouvelle commercialisation donne lieu à un nouveau processus de souscription et donc propose un parcours métier précis via une série d'étapes. Les données manipulées sont principalement des montants numériques liées à des règles de gestion de chiffre d'affaires, de calcul de franchises, etc. Comme pour l'expérimentation précédente, bien que la phase de modélisation des besoins métier sorte du périmètre de l'automatisation même, nous la présenterons pour faciliter la compréhension globale de l'expérimentation.

DÉCOUVERTE ET FORMULATION VISUELLE DES BESOINS MÉTIER

Lors de l'expérimentation, un sous-ensemble de fonctionnalités a été traité (28 exigences), c'est à dire les fonctionnalités du nouveau produit d'assurance à mettre en commercialisation. Ces 28 exigences ont été choisies parce qu'elles appartiennent au sous-groupe des règles métier qui vérifient les propriétés du parcours de souscription selon différentes interfaces.

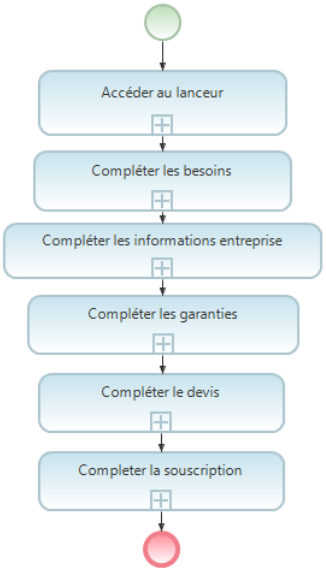


FIGURE 4.7 – Parcours global de l'expérimentation 2

La figure 4.7 présente le parcours global lié à la vérification de la souscription à un contrat dans le cas passant. Ce parcours appelle 6 sous-parcours réalisant chacune des étapes nécessaires à la souscription. Dans le détail on accède à un lanceur, puis on complète les besoins, les informations de l'entreprise, les garanties, le devis et la souscription.

Comme préconisé dans la BONNE PRATIQUE 4, les parcours applicatifs ont été divisés en 2 groupes. Ceux dédiés à la production de cas de test ciblés (permettant de vérifier des propriétés spécifiques du système et les cas d'erreur), et ceux dédiés à production de cas de test nominaux pour la vérification de l'enchaînement de plusieurs fonctionnalités sans erreur. En complément, ces parcours étaient aussi classés par fonctionnalité comme préconisée dans la BONNE PRATIQUE 5.

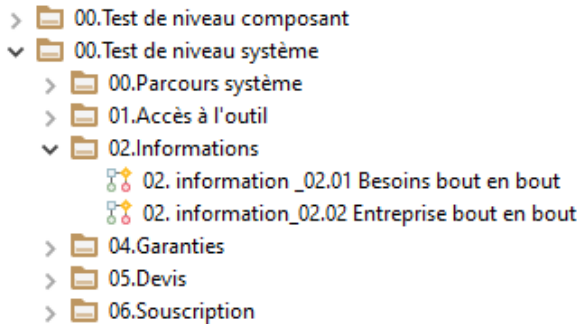


FIGURE 4.8 – Organisation des parcours en appliquant les bonnes pratiques

La figure 4.8 illustre cette organisation. Les parcours de test ciblés se trouvent dans le premier dossier. Le second dossier traite des tests pour les cas nominaux. Chaque sous dossier qui le compose permet de traiter un aspect métier de l'application. Ainsi le dossier "02.Informations" regroupe les parcours permettant de compléter les informations sur le besoin et sur l'entreprise. Plus de détails sur les parcours sont disponible en annexe C.2.1.

La représentation visuelle des parcours applicatifs a permis de mieux comprendre les exigences métier. Nous retrouvons la même constatation que dans l'expérimentation précédente (voir CONSTAT 1). En effet, chaque parcours étant représenté par une séquence visuelle de tâches simples, telles que "Saisir un chiffre d'affaires", puis "Contrôler le montant de la franchise", il était simple de représenter toutes les étapes lors de la signature d'un contrat. Au cours de cette expérimentation, la modélisation n'a pas d'emblée intégrée tout les aspects métier, certains ont tout simplement été oubliés en début de projet. Ceci nous a amené à établir la constatation suivante.

Constatation : intégration des changements facilités.
En suivant la BONNE PRATIQUE 14, la ré-intégration des éléments oubliés quelques jours après le début de la modélisation n'a posé aucune difficulté, il était facile de retrouver les éléments concernés dans le modèle et donc de rapidement les mettre à jour.

CONSTAT 9 – Intégration des changements facilités

Une fois les parcours créés, la testeuse fonctionnelle a présenté les parcours à l'automaticien de test, ce qui nous amené à établir la constatation suivante.

Constatation : collaboration entre testeur et automaticien.
La testeuse fonctionnelle a présenté à l'automaticien de test le parcours global de souscription au moyen de différentes représentations visuelles (en travaillant de manière conjointe avec l'automaticien, elle applique la BONNE PRATIQUE 3). Ceci a permis à l'automaticien d'avoir une vision globale sur le processus à automatiser.

CONSTAT 10 – Collaboration entre testeur et automaticien

La testeuse a ensuite réalisé la conception détaillée des tests.

CONCEPTION DÉTAILLÉE DES TESTS

Pour chaque tâche dans les parcours créé, un ensemble d'actions et de résultats attendus ont été définis. En termes de données, seules les données nécessaires pour couvrir les exigences ont été incluses, comme nous l'avons recommandé dans la BONNE PRATIQUE 1. Par exemple, les données sur le chiffre d'affaires ont été traitées par des classes d'équivalences. Comme dans la première expérience, nous avons pu observer une définition efficace des exigences, comme décrit dans le CONSTAT 5.

Afin de couvrir toutes les exigences, 36 cas de test ont été générés par l'outil. Plus de détails sur la conception de ces tests est disponible en annexe C.2.2

Parmi les cas nominaux, un cas a été identifié pour être converti en script. Le choix d'un seul cas de test s'explique par la durée de l'expérimentation qui ne permettait pas de couvrir l'ensemble des cas produits (bien que l'expérimentation ait duré 10 jours, l'automaticien n'est intervenu que durant 5 jours). Parmi les cas produits nous avons choisi le cas le plus important du point de vue métier, c'est-à-dire celui qui traite de la souscription pour un agent avec un chiffre d'affaires inférieur à 1 000 000 d'euros. Durant cette phase, la testeuse fonctionnelle a pu partager avec l'automaticien de test les étapes de test qui constituaient le cas de test à automatiser (en suivant la BONNE PRATIQUE 3). Ceci a permis à l'automaticien d'initier le développement des mots clés.

Dans la section qui suit nous présentons comment les données de conception ont été liées avec les données d'implémentation pour permettre la future exécution des scripts.

IMPLÉMENTATION DES TESTS

Pour les données de conception existant en tant que classe d'équivalence, un ensemble de données ont été créées pour les remplacer par leurs valeurs réelles dans le SUT, c'est-à-dire la valeur des données d'implémentation. Ces données ont été nommées de manière détaillée et classées par fonctionnalité comme cela a été préconisé dans les BONNES PRATIQUES 6 et 7. Comme dans la précédente expérimentation, nous avons constaté une organisation pus rigoureuse des données (voir CONSTAT 6).

	Nom du jeu de données	profil	Chiffre_daffaires	ChiffreAffaireHT	montant_max_garanti
1	**Jeu garanti 100 000 €**	*Agent*	*CA < 1 000 000 €*	*150 000 €*	*100 000 €*
2	**Jeu garanti 250 000 €**	*Agent*	*CA < 1 000 000 €*	*300 000 €*	*250 000 €*
3	**Jeu garanti 450 000 €**	*Agent*	*CA < 1 000 000 €*	*500 000 €*	*450 000 €*
4	**Jeu garanti 600 000 €**	*Agent*	*CA < 1 000 000 €*	*750 000 €*	*600 000 €*
5	**Jeu garanti 1 000 000 €**	*Agent*	*CA < 1 000 000 €*	*999 999 €*	*1 000 000 €*

FIGURE 4.9 – Extrait du jeu de données

L'implémentation de ces données est réalisée par des jeux de données dans le tableau de la figure 4.9, comme préconisé dans la BONNE PRATIQUE 2.

Plus de détail sur l'implémentation sont présentés en annexe C.2.3. C'est la complétion de ce jeu de données, qui rend les scripts produits exécutables. De plus, l'utilisation des données de conception dans les jeux de données permet de respecter la vision métier, ce qui assure la production de scripts pertinent vis à vis de ces besoins métiers. En complément, nous avons observé un autre fait, décrit dans la constation qui suit.

Constatation : la vision partagée sur les données.
Durant cette phase la testeuse fonctionnelle et l'automaticien de test ont pu échanger sur la gestion des données des scripts. Ensemble, ils ont validé que les données d'implémentation avaient bien le format attendu par l'application, et l'automaticien a pu vérifier que les mots clés traiteraient bien ces données.

CONSTAT 11 – Partage de vision sur les données

COMPLÉTION DE LA COUCHE D'ADAPTATION

La complétion de la couche d'adaptation est une activité partagée entre la testeuse fonctionnelle et l'automaticien de test, comme préconisé dans la BONNE PRATIQUE 3. Pour l'automaticien, cette phase a commencé par la construction du répertoire d'objets, où il a appliqué les bonnes pratiques (BONNES PRATIQUES 8, 9) énoncées sur la gestion des objets.

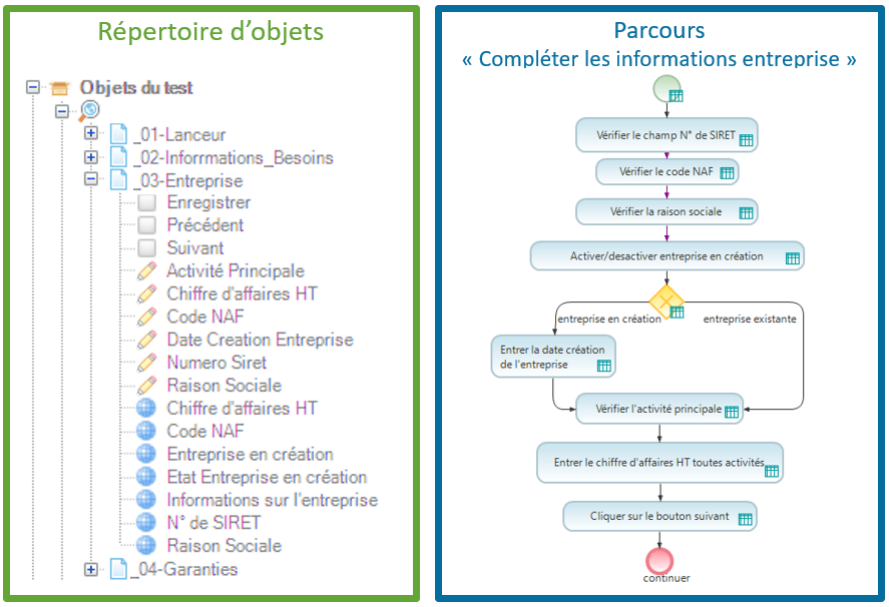


FIGURE 4.10 – Dictionnaire de données et parcours entreprise

Il a organisé les données par interfaces de l'application (visible dans la figure 4.10). C'est-à-dire qu'il défini les données liées au lanceur dans un fichier, celles liées aux besoins dans un autre, etc. Ces interfaces sont visibles dans le parcours applicatif présenté en annexe C.2.4. Cette organisation à donné lieu au même constat que dans la première expérimentation, c'est à dire une organisation rigoureuse des données (voir CONSTAT 6).

Après la création des bibliothèques d'objets, l'automaticien de test a créé un dictionnaire de mots-clés. Ces mots-clés ont une granularité variable, mais privilégient la granularité faible afin de garder les scripts lisibles. Il a donc créé des fonctions pour compléter les champs ainsi que les interfaces. Pour maintenir la vision métier, et suivre les BONNES PRATIQUES 10 et 12, des fonctions de granularité faible avec un nommage adapté aux actions métier ont été utilisé telles que "completerBesoin", "completerInfosEntreprise" et "CompleterGaranties". Elles ont pour objectifs de suivre les actions métier comme dans le parcours de la figure 4.7. Il existe aussi des mots clés de granularité plus élevée tel que "renseignerChamp" pour compléter des champs dans des fonctions de granularité plus faible et donner de l'autonomie à la testeuse. Au final 6 mots-clés ont été produits pour vérifier le périmètre à couvrir.

Le développement des mots-clés a eu lieu parallèlement à la conception et à l'implémentation des tests par la testeuse fonctionnelle. Ainsi, la testeuse fonctionnelle et l'automaticien de test ont pu échanger ensemble sur ces mots-clés. En particulier, l'automaticien a pu présenter les mots-clés créés en les rapprochant des représentations visuelles pour faciliter leur utilisation future par la testeuse lors de la liaison des étapes de test avec les mots-clés. Ainsi il lui a été possible de relier chacun de ces mots clés à l'étape métier à laquelle il correspondait. Ainsi les scripts produits sont pertinents vis à vis de l'aspect métier. La figure 4.11 présente un extrait du script généré suite à la complétion de la couche d'adaptation. La définition de mots-clés de granularité faible, leur usage facilité pour le testeur, et sa collaboration avec l'automaticien nous amènes a faire les mêmes constatations que précédemment (voir les CONSTATS 7 et 10)

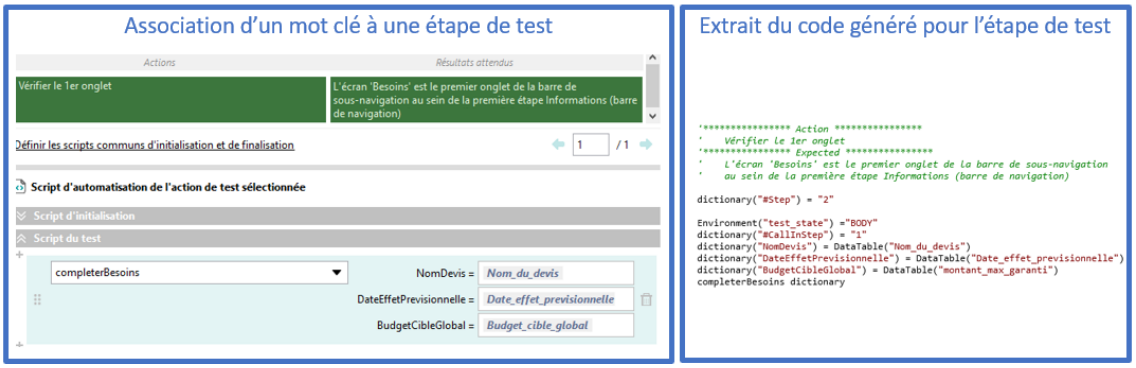


FIGURE 4.11 – Association d'un mot clé à une étape de test

PRODUCTION AUTOMATIQUE DES SCRIPTS DE TEST

Comme pour la première expérimentation, la génération des scripts se fait une fois la couche d'adaptation complétée. Au final, 1 script de test a été produit. Ce script est paramétré à l'aide de 5 jeux de données, ce qui nous permet de vérifier l'ensemble des propriétés que nous souhaitions contrôler.

Cette expérimentation avait pour objectif de vérifier la capacité de notre approche à s'inscrire dans un cycle court en Agile. Nous sommes parvenus durant la durée du sprint (10 jours) à reproduire l'approche de bout en bout. Nous avons créé les modèles représentant les besoins métiers, conçu et implémenté les tests, puis complété la couche d'adaptation et généré un script de test. De plus, durant toutes ces phases, la testeuse fonctionnelle et l'automaticien de test ont pu travailler en synergie pour faciliter chacune de leurs activités (en accord avec la BONNE PRATIQUE 3). En particulier, la testeuse fonctionnelle a pu aider l'automaticien à comprendre les besoins métier grâce aux représentations visuelles et aux cas de test. Elle a également lié les mots clés aux étapes de test et généré le script automatisé, réduisant ainsi l'effort d'automatisation pour l'automaticien. Nous avons donc

pu, en un temps limité, construire l'approche proposée à partir de zéro, montrant qu'elle peut être adaptée à des cycles et des itérations courts en contexte Agile.

4.5/ SYNTHÈSE

Dans ce chapitre nous avons présenté la phase d'automatisation de notre approche ALME. Nous avons notamment décrit les différentes étapes qui la composent : la complétion de la couche d'adaptation et la production des scripts.

Pour chacune d'elles, nous avons défini un ensemble de bonnes pratiques et recommandations visant à faciliter l'automatisation dans des cycles de développement Agile tout en maintenant des scripts pertinents vis à vis des besoins métier. Afin d'éprouver notre approche, nous avons réalisé deux expérimentations dans le contexte de projets existants dans notre environnement de thèse CIFRE, dont nous pouvons extraire les enseignements suivants.

Tout d'abord, l'utilisation de l'approche ALME permet la conception des cas de test à automatiser en conservant une bonne vision du processus métier à tester. En effet, avec des modèles avec un niveau d'abstraction élevé, il est possible de discuter autour de ces modèles et d'étendre directement les objectifs des tests en modifiant le modèle.

De plus, la création d'un ensemble de sous-parcours regroupant les comportements communs de l'application permet de réaliser rapidement de nouveaux parcours applicatifs à tester et de réduire la maintenance de chaque parcours.

Concernant l'implémentation des mots-clés, nous avons constaté que l'utilisation de cette approche permet d'accélérer et réduire le temps de maintenance, y compris en cycle court, mais nous n'avons pas encore assez de cas d'usage pour établir des moyennes fiables de gain de productivité. Nous avons aussi pu observer dans notre seconde expérimentation une forte synergie entre l'automaticien de test et la testeuse fonctionnelle. Ce qui a permis d'équilibrer l'effort de test entre chacun de ces acteurs par une communication facilitée par les supports visuels et l'usage de mots-clés de granularité faible et compréhensibles pour la testeuse fonctionnelle.

Ainsi, les approches et les bonnes pratiques proposées tendent à améliorer la pertinence métier des tests fonctionnels automatisés tout en facilitant leur maintenance. Ceci grâce à une utilisation adaptée des données de conception et d'implémentation de test pour assurer la cohérence entre les règles métier mises en place dans les parcours et les données d'implémentation conditionnées par des exigences. De plus, l'utilisation de mots-clés avec différents niveaux de granularité permet de produire rapidement des scripts lisibles par un testeur fonctionnel.

Concernant la production des scripts : pour une étape partagée N fois, on ne définit qu'une fois l'appel à un ou plusieurs mots-clés pour couvrir cette étape de test. Cela facilite la maintenance. De plus, la possibilité d'inclure des données concrètes dans cette phase permet de valoriser les scripts dès leurs publications. On évite ainsi d'avoir un delta entre l'enchaînement des étapes et les valeurs nécessaires à l'exécution de ces étapes. Cela est géré au niveau de l'outillage MBT permettant ainsi de réduire la maintenance, notamment pour la publication des scripts une fois la couche d'implémentation complétée.

Pour obtenir ces bénéfices de l'approche, nous avons constaté l'importance du suivi des bonnes pratiques telles que celles exposées dans ce chapitre, ce qui implique une formation des testeurs et automaticiens de test intervenants dans l'approche.

5

CONTRIBUTIONS TECHNIQUES : REFACTORING

Comme nous l'avons présenté et discuté dans le chapitre sur l'état de l'art, en section 2.3, les testeurs fonctionnels formulent généralement les cas de test manuels au moyen d'étapes de test et de résultats attendus, rédigés de façon informelle en langage naturel.

Avec le temps une forme d'obsolescence apparaît : des cas de test ne sont plus à jour, d'autres sont totalement ou partiellement redondants et les suites de cas de test sont de plus en plus désorganisées, ce qui les rend difficile à utiliser et à maintenir.

Cette section présente une approche de refactoring [1] des suites de tests manuels afin de les rendre plus utilisables et plus faciles à maintenir. Cela vise la reprise de cas tests existants (et donc n'ayant pas été conçu à partir de modèles) dans le cadre plus général de notre approche ALME. Notre contribution porte à la fois sur la dimension méthodologique et sur son évaluation dans le contexte de projets réels avec l'outil Orbiter de Smartesting, dans le cadre du partenariat avec Sogeti et l'institut FEMTO-ST. Nos expérimentations ont porté sur les deux questions de recherche suivantes :

- RQ-4 : Dans quelle mesure l'approche proposée permet-elle de rendre plus utilisables et maintenables les suites de cas de test manuels ?
- RQ-5 : Dans quelle mesure l'approche outillée de refactoring proposée permet elle un gain d'effort vis-à-vis d'un refactoring sans cet outillage ?

En complément, nous avons établi un ensemble d'objectifs à couvrir pour proposer une approche de reprise des suites de tests qui réponde aux défis rencontrés dans le domaine industriel :

- Proposer une approche qui permette de réaliser la reprise de suites de tests en langage naturel.

1. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1.

- Proposer une approche qui permette d'identifier et corriger efficacement les obsolescences (redondances dans les étapes de test, cas de test en doublon, etc) dans les suites de tests.
- Proposer une approche qui permette de rendre plus réutilisables et faciles à maintenir les suites de tests manuels.
- Proposer une approche qui permette de rendre plus faciles à automatiser les suites de tests manuels.

Nous évaluons notre approche sur plusieurs suites de tests dans le contexte de projets réels, et nous rendons compte des gains de temps qui ont été obtenus.

5.1/ LES ACTIVITÉS DE LA REPRISE DES SUITES DE TESTS MANUELS

Notre démarche de refactoring est structurée en cinq activités itératives :

1. Identifier un périmètre pour le refactoring.
2. Choisir les étapes ou fonctions de test[2] à corriger.
3. Homogénéiser et corriger les étapes ou fonctions de test.
4. Corriger les étapes et fonctions de test dans les cas de test.
5. Paramétriser les étapes et fonctions de test.

Nous présentons en détail chacune de ces activités, en introduisant tout d'abord le vocabulaire utilisé. Nous parlons de suites de tests, de cas de test, de fonctions de test, d'étapes de test, d'actions, de résultats attendus.

L'approche est présentée avec un exemple de refactoring de suites de test. Dans cet exemple 19 cas de test sont répartis dans les dossiers Sprint 1 et Sprint 2. Des cas de test identiques ou présentant des redondances existent entre ces deux dossiers.

Nous voulons éliminer les redondances et les erreurs existantes au sein des cas de test de ces deux dossiers. La figure 5.1 présente la représentation visuelle des cas de test avant de commencer le refactoring. Bien que peu lisible, cette représentation pourra servir de point de comparaison pour les activités successives de refactoring.

2. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

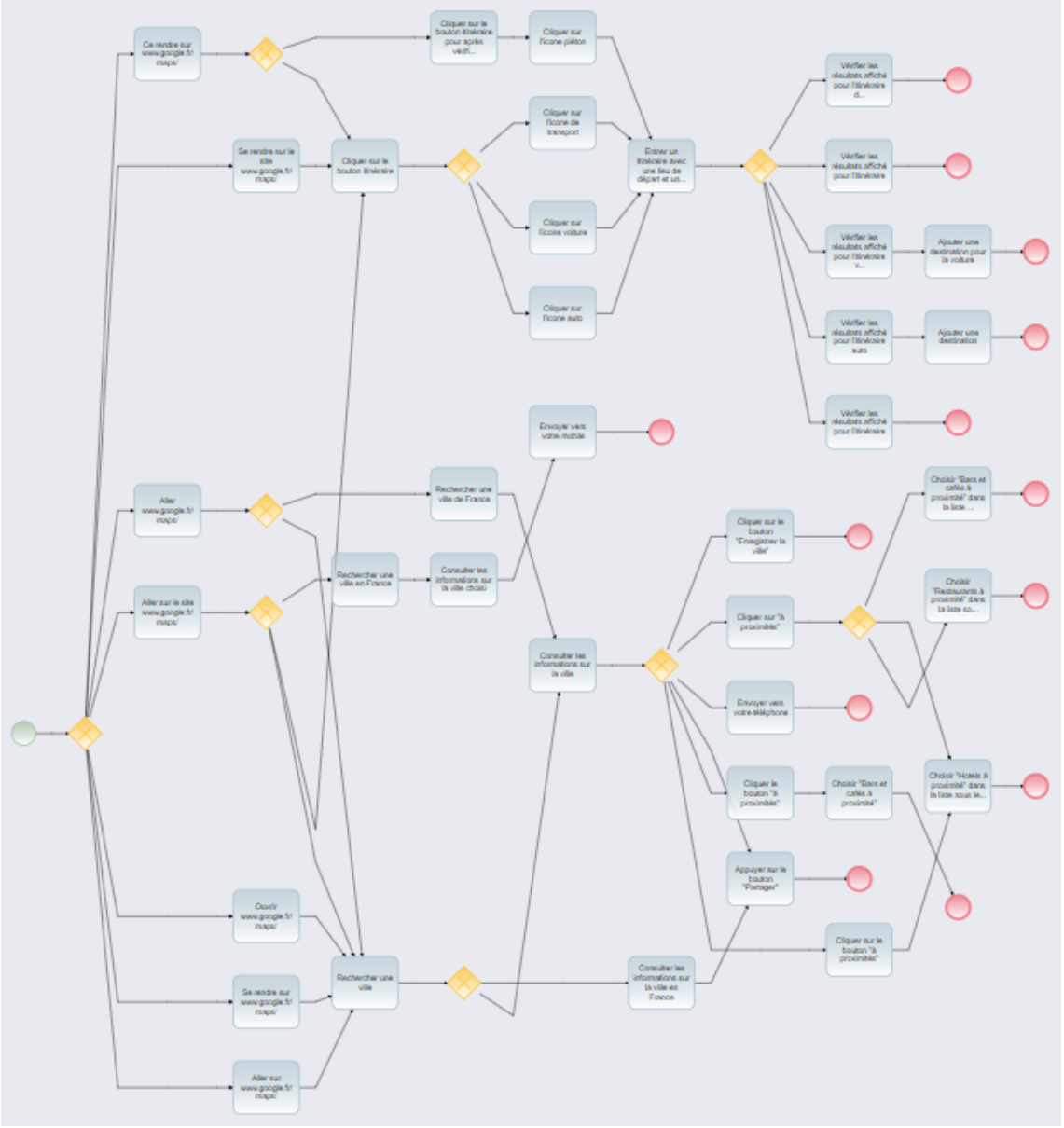


FIGURE 5.1 – Représentation visuelle des cas de test

5.1.1/ IDENTIFIER UN PÉRIMÈTRE POUR LE REFACTORING

L'identification du périmètre des cas de test sur lequel procéder au refactoring constitue la 1ère activité de notre approche.

DESCRIPTION DE L'ACTIVITÉ

Les suites de tests étant des éléments textuels, il est possible d'identifier un premier sous-ensemble de cas de test en se fondant sur l'expérience et les compétences métier. Dans l'approche que nous proposons, un support outillé(Orbiter) nous permet de visualiser les cas de test sous la forme de dendrogramme. Les cas de test sont au préalable importés dans Orbiter à partir d'un outil de gestion des tests (par exemple ALM, Squash, Excel), et ensuite regroupés à l'aide d'algorithmes de regroupement hiérarchique, et affichés dans une vue arborescente à l'aide de dendrogrammes. Il co-existe alors deux regroupements :

- l'un au travers des dossiers du référentiel existant (une structure arborescente)
- l'autre basé sur les similitudes de la description textuelle de chaque cas de test.

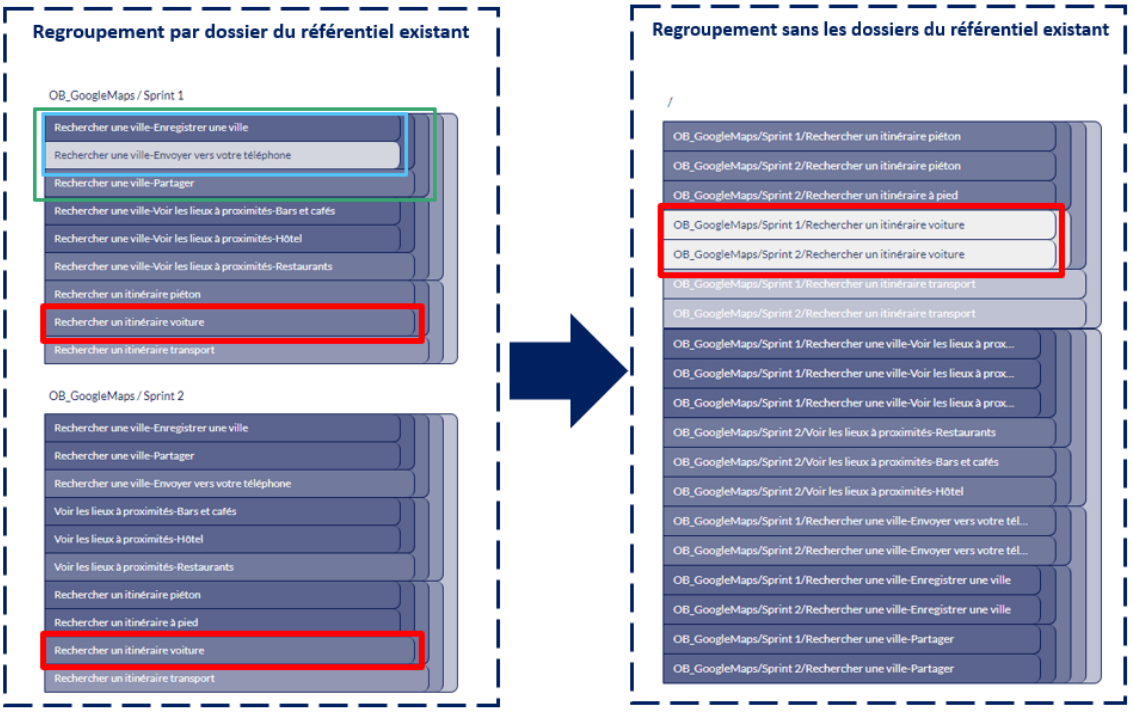


FIGURE 5.2 – Regroupement hiérarchique des cas de test

La figure 5.2 présente ces deux regroupements, à gauche, les cas de test sont affichés à travers les dossiers du référentiel existant (2 dossiers sont présents : Sprint 1 et Sprint 2).

En partie droite de la figure 5.2, l'arborescence existante n'est plus visible et les tests sont uniquement regroupés par similitudes. Cette vue permet de s'affranchir de la séparation en dossiers afin de détecter les similitudes entre les tests des différents dossiers (Sprint 1 et Sprint 2).

En complément, l'outil propose une représentation graphique des parcours applicatifs correspondant aux cas de test du groupe sélectionné (Figure 5.1). Cette représentation, combinée aux libellés des cas de test constitueront les indicateurs de choix des cas de test à remanier.

BONNES PRATIQUES DE L'ACTIVITÉ

Même si le support outillé que nous utilisons permet d'étudier un référentiel dans son ensemble, la complexité que cela engendre conduit souvent à restreindre, dans un premier temps, le périmètre de refactoring. Nous avons établi quelques bonnes pratiques conduisant à identifier efficacement un périmètre.

Bonne pratique

Utiliser la structure du référentiel dans l'objectif de refactoring : L'arborescence de répertoires du référentiel constitue de fait des périmètres de refactoring possibles. La définition de l'objectif de refactoring, en terme de périmètre, peut correspondre à un niveau dans l'arborescence selon que l'on souhaite traiter l'ensemble du référentiel ou seulement une partie.

BONNE PRATIQUE 15 – Utiliser la structure du référentiel dans l'objectif de refactoring

Lors de l'étude d'un sous ensemble, il faut garder en tête que le refactoring bien qu'appliqué à une échelle locale peut avoir des impacts sur l'ensemble du référentiel.

Ensuite nous préconisons de cibler un certain groupe de cas de test pour commencer le refactoring. Ces cas de test sont ceux qui comportent des similitudes. Ainsi nous avons établi la bonne pratique suivante.

Bonne pratique

Baser le refactoring sur des cas de test comportant des similitudes : une bonne pratique est de baser le refactoring sur un groupe de cas de test comportant des similitudes. Le dendrogramme est le support qui permet d'identifier un groupe de cas de test comportant des similitudes.

BONNE PRATIQUE 16 – Baser le refactoring sur des cas de test comportant des similitudes

Une autre bonne pratique que nous avons établi concerne les profondeurs existantes dans le dendrogramme. Celui-ci est composé d'un ensemble de profondeurs allant du niveau le plus bas au niveau le plus élevé. Ce que nous appelons le niveau le plus bas sont les groupes à gauche du dendrogramme dans la figure 5.3 (indiquées en vert et étiquetées comme "niveau le plus bas"). Ainsi, plus les groupes se déplacent vers la droite, moins il y a de similitudes (le niveau le plus élevé est représenté en noir sur la figure 5.3 et étiqueté "niveau le plus élevé").

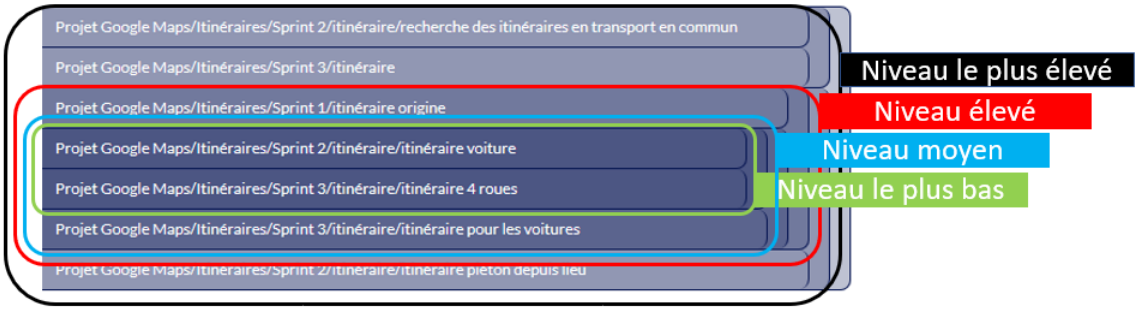


FIGURE 5.3 – Un dendrogramme, montrant des groupes de tests par niveau

Bonne pratique

Choisir un niveau bas de profondeur dans le dendrogramme : ce sont ces niveaux qui comportent le plus de similitudes et sur lesquels nous conseillons de placer les premiers efforts de refactoring.

BONNE PRATIQUE 17 – Choisir un niveau bas de profondeur dans le dendrogramme

Le refactoring des cas de test à partir des niveaux les plus bas conduit parfois à la réévaluation des cas de test aux niveaux supérieurs. C'est pour cette raison que nous préconisons plutôt de choisir des regroupements les plus à gauche dans les dendrogrammes. Un cas de test de niveau légèrement supérieur peut, à la faveur des refactoring,

intégrer le groupe de niveau inférieur. L'inverse n'est pas très fréquent. Ceci s'explique par le fait que de nouvelles évolutions implémentées, peuvent dépendre de fonctionnalités existantes. Les cas de test auront plus de chance de contenir des étapes de test similaires sur des comportements existants que sur les nouvelles fonctionnalités mises en production.

Jusqu'ici nous n'avons considéré que l'étude des différents groupements proposés par les dendrogrammes. L'étude des représentations visuelles est un support pour identifier si les groupements choisis sont adaptés au refactoring. Nous présentons deux bonnes pratiques permettant de vérifier si un groupement est adapté ou non au refactoring.

Bonne pratique

Identifier les périmètres non adaptés au refactoring : Lorsque la représentation visuelle du périmètre sélectionné forme un peigne avec des chemins disjoints, cela peut correspondre à un périmètre de refactoring qui est non adapté, car trop large.

BONNE PRATIQUE 18 – Identifier les périmètres non adaptés au refactoring

Dans la figure 5.4, nous présentons un extrait de suite de tests présentant des chemins disjoints, non adaptés au refactoring. Les cas de test présentés couvrent différentes fonctionnalités : la recherche d'itinéraire, la réservation d'hôtel et le e-commerce. Dans ce cas, la comparaison des cas de test entre eux n'est pas pertinente, car ils ne traitent pas des mêmes domaines. Il faut donc changer de périmètre de refactoring, pour ne traiter que d'un seul domaine et avoir une représentation visuelle plus cohérente.

Bonne pratique

Identifier les périmètres adaptés au refactoring : Lorsque la représentation visuelle du périmètre sélectionné présente un enchevêtrement de chemins avec des étapes de test partagées entre les chemins, il est intéressant de continuer le refactoring.

BONNE PRATIQUE 19 – Identifier les périmètres adaptés au refactoring

La figure 5.5 présente un schéma adapté au refactoring. Les premières étapes de test traitent toutes de la connexion au site Google Maps. La suite de tests choisie traite d'un domaine précis. De plus, la représentation graphique présente différents noeuds regroupant plusieurs chemins. Il existe donc des redondances et il faudra donc déterminer si

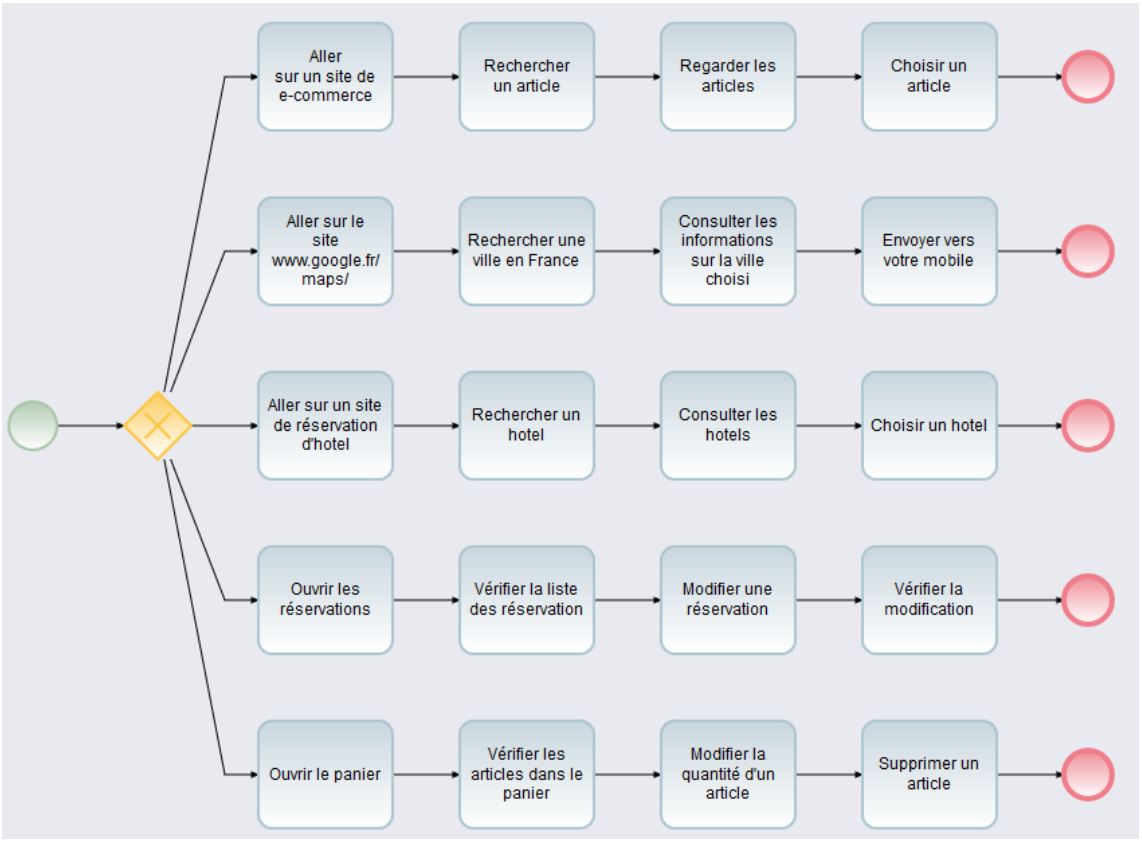


FIGURE 5.4 – Exemple de chemins disjoints dans la représentation visuelle non adaptés au refactoring

cela est correct ou non. Ces éléments en font donc un bon candidat pour le refactoring.

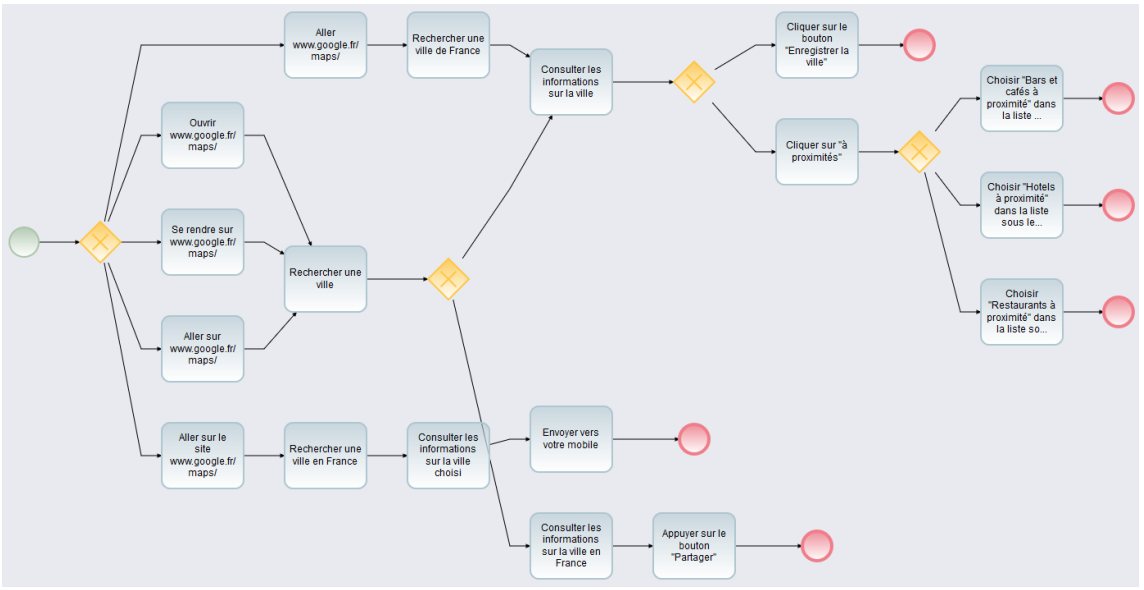


FIGURE 5.5 – Exemple de représentation visuelle adapté au refactoring

ILLUSTRATION SUR L'EXEMPLE FIL-ROUGE

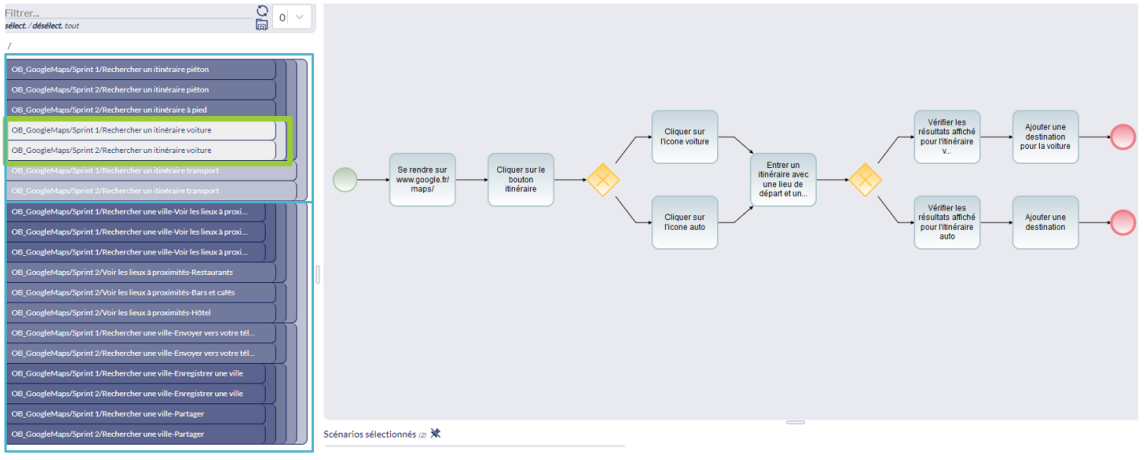


FIGURE 5.6 – Un dendrogramme, montrant des groupes de tests par niveau

L'exemple du référentiel en figure 5.6 se place au niveau 0, sans tenir compte de l'arborescence existante, car nous souhaitons analyser les redondances pouvant exister entre les dossiers. L'analyse du dendrogramme nous donne différentes informations. Tout d'abord, il existe deux grands groupements, entourés en bleu. Nous choisissons d'étudier d'abord le sous-ensemble le plus petit, c'est-à-dire celui composé de 7 cas de test (rectangle du haut). Et pour aller plus loin, le regroupement de cas de test le plus à gauche et de plus petite taille est choisi (entouré en vert). De plus, les libellés des cas de test sont

les mêmes ce qui indique sans doute des redondances. La représentation visuelle est simple et semble présenter des redondances ce qui en fait un bon candidat pour débuter le refactoring.

SYNTHÈSE

L'identification d'un périmètre pour le refactoring est la première activité de notre approche. Elle permet notamment d'identifier rapidement les cas de test qui semblent présenter des redondances et ainsi cibler où commencer les corrections. Le support de l'arborescence des tests sous forme de dendrogramme et de la représentation visuelle des cas de test, associés à la mise en œuvre des bonnes pratiques participent à une identification efficace d'un premier périmètre pour le refactoring.

Après cette activité, le choix des étapes et fonctions de test à corriger peut être réalisée.

5.1.2/ CHOISIR LES ÉTAPES OU FONCTIONS DE TEST À CORRIGER

Une fois qu'un ensemble de cas de test a été sélectionné, leurs étapes et fonctions de test peuvent être inspectées pour voir si un refactoring est nécessaire.

DESCRIPTION DE L'ACTIVITÉ

Nous nous concentrons ici sur les étapes et fonctions de test à corriger et non sur les cas de test, car nous travaillons d'abord sur les étapes et fonctions de test avant d'examiner les cas de test dans leur ensemble. Cela permet d'éviter de contrôler des cas de test dont les étapes n'auraient pas encore été revues et qui pourraient contenir des incohérences. La correction des étapes et fonctions de test apporte de la clarté aux cas de test et facilite donc leur analyse ultérieure. Les représentations graphiques sont un support privilégié pour les opérations de refactoring dans le choix des étapes et fonctions de test à corriger par l'identification de "schémas" conduisant au refactoring rapide des points de redondance. L'identification d'un schéma particulier va conduire à la sélection d'une étape ou fonction de test pour ensuite procéder à sa correction. Nous détaillons l'ensemble des schémas établis à travers des bonnes pratiques dans la section qui suit.

BONNES PRATIQUES DE L'ACTIVITÉ

Afin de faciliter le choix d'une étape de test pour le refactoring, nous avons établi des bonnes pratiques visant à faciliter l'identification de redondances dans les cas de test.

Le ”schéma” le plus utile et rapide à identifier est celui des tâches sur la même ligne verticale.

Bonne pratique

Identifier les étapes de test redondantes grâce aux alignements verticaux : des tâches sur la même ligne verticale indiquent souvent des redondances. Commencer par étudier ces étapes de test est donc une bonne pratique pour identifier rapidement des redondances.

BONNE PRATIQUE 20 – Identifier les redondances grâce aux alignements verticaux

La figure 5.7 montre à plusieurs reprises ce schéma où nous avons mis en évidence des étapes de test présentant des redondances au moyen de rectangles verts. Ce type d'erreur est facilement identifiable visuellement et peut donc être rapidement corrigé en appliquant des terminologies similaires lors de la correction et de l'homogénéisation des étapes et fonctions de test. Appliquer les mêmes terminologies à de nombreux avantages et dans notre cas cela nous permet de savoir si les deux cas de test étudiés sont identiques ou non.

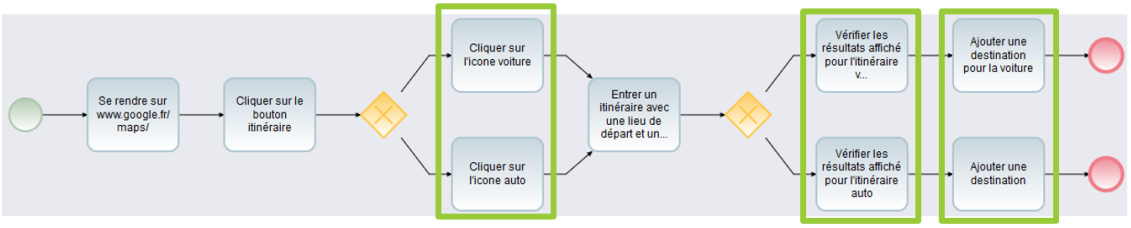


FIGURE 5.7 – Visualisation graphique des suites de tests sous forme de processus métier

Un autre schéma qu'il est possible d'identifier est des boucles, présenté dans la bonne pratique qui suit.

Bonne pratique

Identifier les étapes de test redondantes grâce aux boucles : des boucles dans la représentation visuelle peuvent indiquer des comportements en doublon.

BONNE PRATIQUE 21 – Identifier les étapes de test redondantes grâce aux boucles

La création de doublons peut avoir lieu lorsqu'un testeur utilise des extraits d'anciens cas de test pour en construire de nouveau. Par mégarde, il peut faire l'usage de fonctions de

test identiques au sein du même cas de test par l'usage inadapté de copier-coller. Un exemple de boucle est présenté dans la figure 5.8.

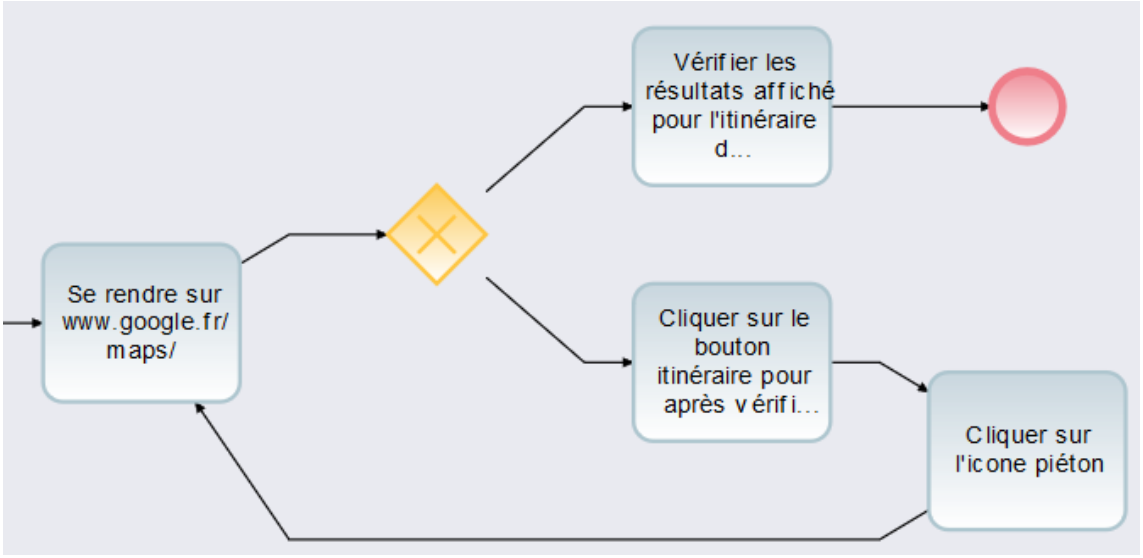


FIGURE 5.8 – Exemple de représentation visuelle avec une boucle

Cependant la présence de boucle dans la représentation visuelle n'induit pas nécessairement l'existence d'un doublon. Cela peut révéler l'usage d'une fonction de test générique telle que la vérification de l'affichage d'un message d'erreur. Pour déterminer s'il existe un doublon ou non, il faut consulter le cas de test dans sa globalité.

ILLUSTRATION SUR L'EXEMPLE FIL-ROUGE

La figure 5.7 présente le diagramme des deux cas de test sélectionnés précédemment dans le dendrogramme traitant de la recherche d'un itinéraire voiture. Chaque rectangle représente une étape de test ou une fonction de test, libellée par le début du texte de son action. La différence entre fonction de test et étape de test se fait après la sélection de l'un d'entre eux par la visualisation du nombre d'occurrences. Comme évoqué, un nombre d'occurrences supérieur à 1 indique que c'est une fonction de test employée plusieurs fois contrairement à une étape de test unique. Ainsi chaque rectangle modélise soit une étape de test soit une fonction de test et aucune n'est identique à une autre, elle diffère soit par l'action, soit par le résultat, soit par les deux.

D'après la représentation graphique, les 1ère et 2ème étapes de test pour la recherche d'un itinéraire sont communes aux deux cas de test. D'après leurs actions, ces deux fonctions de test consistent respectivement à "se rendre sur Google Maps" puis à "cliquer sur le bouton itinéraire". Ensuite les comportements divergent entre les deux cas de test. Dans l'un, l'action est "cliquer sur l'icône voiture" que dans l'autre elle est "cliquer sur l'icône

auto". Ici, les deux actions sont similaires et nous devrons les corriger afin d'avoir une même terminologie pour avoir une même fonction de test.

SYNTHÈSE

La représentation visuelle des parcours métier constitue un support important pour les opérations de refactoring dans le choix des étapes et fonctions de test à corriger ainsi que pour l'acquisition de compétences sur le référentiel de tests étudié. Ici c'est le support des représentations visuelles et l'identification de différents schémas qui vont guider le choix d'une étape ou fonction de test à corriger. Une fois ce choix réalisé, il est possible de passer à l'homogénéisation et la correction des étapes ou fonctions de test.

5.1.3/ HOMOGÉNÉISER ET CORRIGER LES ÉTAPES OU FONCTIONS DE TEST

À ce stade du processus de refactorisation, nous considérons qu'un périmètre pour le refactoring a été identifié, ainsi qu'une étape ou fonction de test pour être potentiellement reformulée. Potentiellement, car ce n'est pas forcement l'étape ou fonction de test choisie que l'on va chercher dans un premier temps à retravailler, mais les étapes ou fonctions de test proches de celui-ci.

DESCRIPTION DE L'ACTIVITÉ

Vis-à-vis de l'approche que nous proposons, nous préconisons une fois une étape ou fonction de test choisie, de consulter les contenus proches de l'étape ou fonction de test sélectionnée pour déterminer s'ils doivent être corrigés. Ce choix se fait par la visualisation des contenus proches dans l'outil que nous utilisons, présent en bas à droite de la figure 5.9.

Dans la figure 5.9, pour une action ou résultat attendu de l'étape ou fonction de test sélectionnée, ici l'action *"Cliquer sur l'icône voiture"* (entourée en vert) la liste de tous les contenus proches est affichée. Pour chacun, l'outil met en évidence les rapprochements envisagés en indiquant (en vert) les termes à ajouter et (en bleu) les termes à supprimer.

Cette mise en valeur des contenus proches est un bon support pour faciliter l'homogénéisation des étapes et fonctions de test. Dans l'exemple présenté, l'action redondante identifiée dans la représentation visuelle : *"Cliquer sur l'icône auto"* est retrouvable facilement dans les contenus proches. L'outil permet, très simplement, de fusionner les actions ou résultats attendus proches que l'on souhaite homogénéiser. Pour que deux étapes de test soient fusionnées en une fonction de test, leurs actions et résultats attendus doivent être identiques.

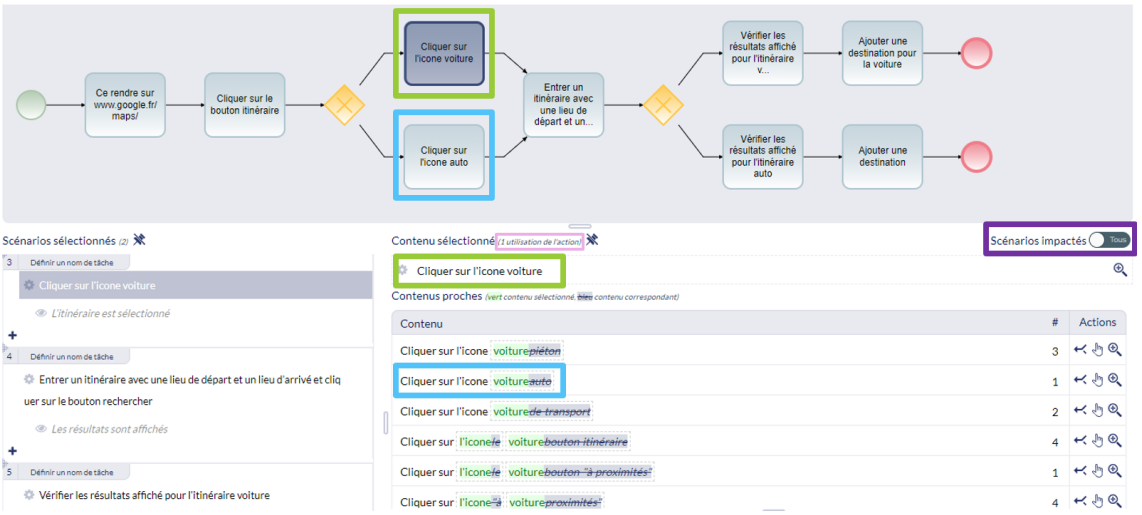


FIGURE 5.9 – Visualisation des contenues proches pour l'homogénisation des étapes de test

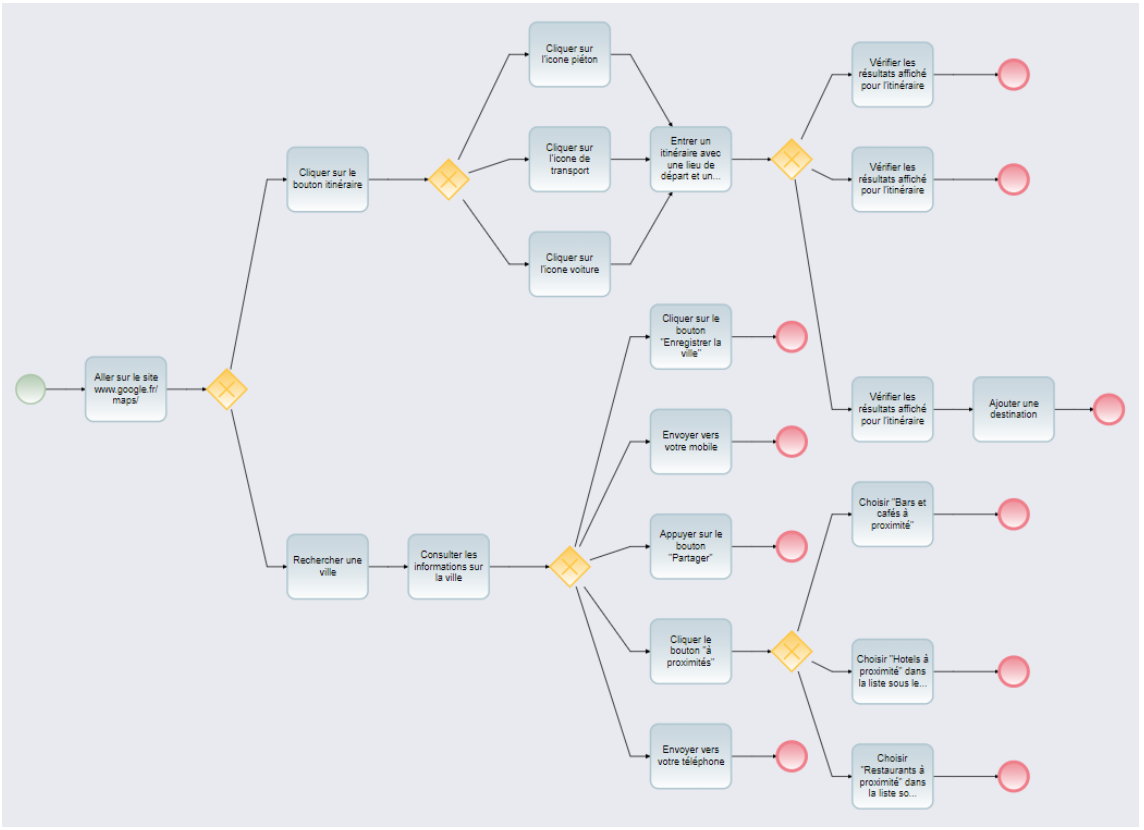


FIGURE 5.10 – Représentation visuelle des suites de tests après l'activité d'homogénéisation

Le processus de refactoring est ainsi basé sur l'alternance entre l'uniformisation des étapes et fonctions de test et la consultation et le choix de nouvelles étapes et fonctions de test par la représentation visuelle des suites de test.

Cette activité d'homogénéisation peut être considérée comme terminée lorsque la représentation visuelle est simplifiée sur le périmètre choisi et que les propositions les plus proches ne permettent pas d'effectuer de nouvelles opérations de mise à jour.

BONNES PRATIQUES DE L'ACTIVITÉ

Afin de faciliter l'homogénéisation et la correction des étapes de test, nous avons établi un ensemble de bonnes pratiques. La première concerne le périmètre d'homogénéisation.

Bonne pratique

Homogénéiser et corriger l'ensemble du référentiel : nous recommandons de travailler sur l'ensemble du référentiel, afin d'éviter tout travail redondant.

BONNE PRATIQUE 22 – Homogénéiser et corriger l'ensemble du référentiel

Au cours de nos expériences d'évaluation, nous avons remarqué qu'en général, lorsqu'une correction devait être effectuée sur un périmètre local réduit, elle devait également être effectuée sur un périmètre plus large.

L'activité initiale de refactoring est plus longue si l'on utilise un périmètre plus large, car elle nécessite l'analyse d'un plus grand nombre de correspondances les plus proches. Néanmoins, l'utilisation d'un périmètre plus large peut faire gagner un temps considérable pour le reste du travail. Il permet d'harmoniser d'autres périmètres avant les activités ultérieures. En revanche, si l'objectif du refactoring n'est pas de travailler sur d'autres parties du référentiel de test, alors le maintien du périmètre de travail dans le groupe de cas de test sélectionné est un bon choix.

Indépendamment du périmètre choisi, il est important de considérer l'impact des modifications et c'est ce que présente la bonne pratique suivante.

Bonne pratique

Considérer l'impact des modifications : avant de procéder à une opération d'homogénéisation, nous recommandons de rechercher l'origine des étapes ou fonctions de test à homogénéiser afin de connaître leur contexte d'utilisation et ainsi garantir la cohérence du référentiel.

BONNE PRATIQUE 23 – Considérer l'impact des modifications

Lors de la mise à jour d'une action ou d'un résultat attendu, il faut garder à l'esprit que la modification sera effectuée pour le nombre d'occurrences indiqué (ce nombre est affiché pour l'action ou le résultat attendu sélectionné et en face de chaque contenu proche).

Concernant les contenus proches, nous avons établi la bonne pratique suivante.

Bonne pratique

Homogénéiser l'ensemble des actions ou résultats attendus proches : nous conseillons d'homogénéiser l'ensemble des actions ou résultats attendus proches du contenu sélectionné pour lesquels les mêmes comportements sont réalisés, mais qui ont des terminologies différentes.

BONNE PRATIQUE 24 – Homogénéiser les actions ou résultats attendus proches

Enfin, la dernière bonne pratique concernant l'activité d'homogénéisation et de correction des étapes et fonctions de test concernent la non application de paramètre.

Bonne pratique

Ne pas intégrer de paramètres prématurément : une opération qui peut renforcer l'uniformité des cas de test est d'y intégrer des paramètres afin de variabiliser des données. Notez que dans l'approche que nous proposons il est important de ne pas appliquer de paramètres au cours de cette activité.

BONNE PRATIQUE 25 – Ne pas intégrer de paramètres prématurément

Si les paramètres sont appliqués prématurément, cela ne provoque pas de blocage pour les activités à venir du processus. Cela constitue simplement une dépense de temps supplémentaire. L'outil propose des résultats moins pertinents pour l'utilisation des données et pour cette raison, on peut être amené à créer les mêmes données plusieurs

fois pour des étapes de test similaires alors qu'en réalité, la paramétrisation n'aurait pu être faite qu'une seule fois. Nous détaillerons l'ensemble les pratiques liées à l'usage des paramètres plus loin dans le processus.

ILLUSTRATION SUR L'EXEMPLE FIL-ROUGE

Dans notre exemple, nous avons appliqué l'homogénéisation à l'ensemble des cas de test et la représentation graphique est simplifiée, comme visible dans la figure 5.10, par rapport à celle initialement présentée en figure 5.1 et les contenus proches proposés ne sont plus pertinents.

SYNTHÈSE

Pour résumer, l'activité d'homogénéisation consiste à consulter les actions et résultats attendus proposés dans les contenus proches de l'étape ou fonction de test initialement sélectionnée, puis de déterminer si ces contenus proches doivent être uniformisés. L'homogénéisation doit être effectuée avec précaution en suivant les bonnes pratiques. Elles mettent notamment en valeur l'impact des modifications apportées ainsi que la manière de le faire en fonction des objectifs de refactoring. Ensuite nous proposons de corriger les étapes et fonctions de test dans les cas de test.

5.1.4/ CORRIGER LES ÉTAPES ET FONCTIONS DE TEST DANS LES CAS DE TEST

Dans notre approche, une fois l'activité d'homogénéisation des étapes et fonctions de test terminée, nous préconisons de travailler sur les cas de test entiers, afin de les corriger, de les homogénéiser, de les optimiser et de les supprimer si nécessaire.

DESCRIPTION DE L'ACTIVITÉ

La consultation des cas de test a différents objectifs et permet d'effectuer différentes opérations, notamment de modifier les étapes de test dans leur contexte d'utilisation en allant plus dans le détail que lors de l'activité d'homogénéisation.

Il convient de garder à l'esprit que les fonctions de test sont utilisées plusieurs fois et qu'une mise à jour doit toujours être considérée en fonction du contexte d'appel. Lors de la consultation des cas de test, il est donc possible d'effectuer un travail de refactoring comme mentionné dans la section précédente, mais aussi de réorganiser les étapes et les fonctions de test. Leur réorganisation peut impliquer :

- leur déplacement : pour une étape de test ou une fonction de test qui n'apparaît pas dans l'ordre attendu d'exécution, il est pertinent de déplacer cet élément pour rétablir un ordre d'exécution adéquat.

- leur suppression : pour une fonction de test sélectionnée, qui apparaît en plusieurs occurrences et dont l'utilisation n'est pas pertinente, il est intéressant de la supprimer afin de réduire le nombre de fonctions de test dans le référentiel.

- leur fractionnement : pour une étape ou fonction de test qui est trop dense et qui contient de nombreuses stimulations à réaliser sur le système. Il est possible de fractionner cette étape ou fonction de test, souvent longue à exécuter et pour laquelle, en cas d'échec, il est complexe d'identifier la source de l'erreur.

En complément, lors de cette activité, le fait de consulter la représentation graphique permet aussi d'identifier les cas de test en doublon et de les éliminer. En effet, l'outil rapproche les cas de test identiques en les mettant côte à côte dans le dendrogramme, mais ne les supprime pas. Il faut donc vérifier, grâce à l'outil, que les 2 cas de test activent bien le même chemin sur la représentation graphique des parcours applicatifs. Si le chemin emprunté est le même, les cas de test sont identiques et la suppression des doublons est possible.

BONNES PRATIQUES DE L'ACTIVITÉ

Afin de faciliter la correction des étapes et fonctions de test, nous avons établi un ensemble de bonnes pratiques. La première concerne la consultation des cas de test.

Bonne pratique

Sélectionner les actions et les résultats attendus : pour chaque cas de test, il est recommandé de lire les actions et les résultats attendus des étapes et fonctions de test et de les sélectionner si une correction est nécessaire.

BONNE PRATIQUE 26 – Sélectionner les actions et les résultats attendus

Cette bonne pratique participe à vérifier qu'aucun contenu proche proposé ne devrait être homogénéisé. Ensuite une autre pratique à appliquer concerne les étapes de test.

Bonne pratique

Scinder les étapes de test : il est suggéré pour certains cas, de scinder ces étapes ou fonctions de test pour éviter un trop grand nombre de stimulations sur le système dans une même étape ou fonction de test.

BONNE PRATIQUE 27 – Scinder les étapes de test

Cependant, toutes les étapes ou fonctions de test denses ne doivent pas systématiquement donner lieu à des segmentations. Il faut distinguer les stimulations sur le système décrivant des comportements métiers hétérogènes (qui peuvent nécessiter une segmentation) et les successions formant un tout (qui n'en nécessite pas forcement).

L'exemple de la figure 5.11 pourrait selon les cas ne pas nécessiter de segmentation, en fonction des objectifs de test. Elle présente à gauche un cas de test avant d'avoir été fractionné et à droite après le fractionnement. Un exemple ne nécessitant pas de segmentation est celui de la saisie d'une adresse. Il serait normal de trouver la liste des champs à compléter tels que le numéro de rue, le code postal et la ville dans une même étape ou fonction de test et ne pas séparer dans des actions distinctes chacune de ces complétions de champs.

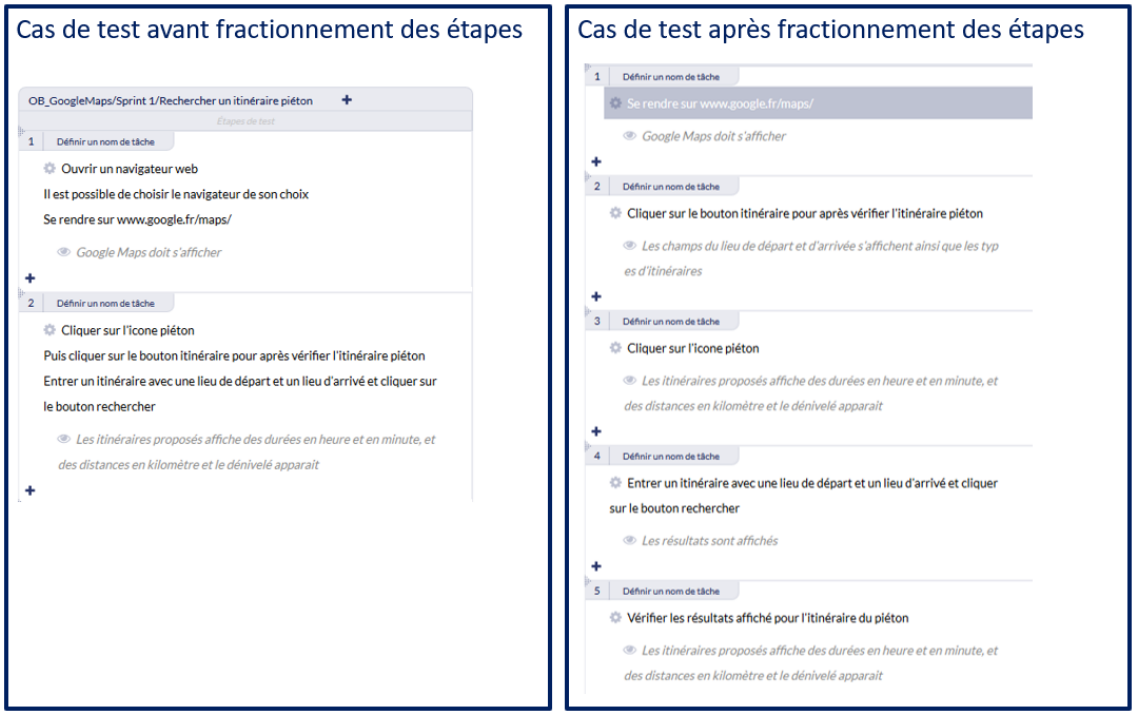


FIGURE 5.11 – Fractionnement des étapes de test

ILLUSTRATION SUR L'EXEMPLE FIL-ROUGE

Comme pour l'activité précédente, l'état de la représentation visuelle des tests permet de décider de la terminaison de cette activité : la représentation visuelle doit être simple, et chaque cas de test devrait former des chemins plutôt indépendants excepté pour les fonctions de test communes et les étapes ou fonctions de test sur lesquels des paramètres pourraient être introduits. C'est l'objet de la dernière activité de notre processus.

Dans notre exemple, nous avons supprimé les cas de test en doublon et la représentation graphique reste la même après la consultation des cas de test. Ceci s'explique par le fait que nous n'avons pas fait d'opérations particulières impactant la représentation durant cette activité.

SYNTHÈSE

L'activité de correction des cas de test se décompose en deux phases : la réorganisation des étapes et fonctions de test et la suppression des cas de test redondants. L'ensemble des bonnes pratiques présenté ici participe à corriger efficacement les étapes et fonctions de test et contribue ainsi à supprimer les obsolescences des suites de tests. Pour aller plus loin, il est possible d'appliquer des paramètres, ce qui est l'objet de la sous-section suivante.

5.1.5/ PARAMÉTRISER LES ÉTAPES ET FONCTIONS DE TEST

L'introduction de paramètres dans les étapes et fonctions de test termine notre processus de refactoring. Cette activité arrive en dernier, car elle nécessite une bonne connaissance métier du projet.

DESCRIPTION DE L'ACTIVITÉ

L'utilisation des paramètres doit être faite avec précaution, car une mauvaise utilisation peut entraîner des irrégularités et des erreurs sur le référentiel de test.

Une autre raison d'appliquer le paramétrage en fin du processus est que celui-ci est plus facile à réaliser grâce aux propositions automatiques d'application des paramètres proposés par le support outillé. Lors de la création d'un paramètre, l'outil propose une liste d'étapes ou fonctions de test qui sont de bons candidats pour appliquer le même paramètre. Dans la figure 5.12, les contenus proches sont présentés par le support outillé en lien avec le remplacement de la valeur ”voiture” dans l'action ”Cliquer sur le logo

voiture". Ici en appliquant un paramètre, par exemple "type d'itinéraire", il sera automatiquement appliqué aux contenus proches "Cliquer sur le logo piéton/avion/train".

FIGURE 5.12 – Propositions automatiques d'application des paramètres

Le problème est que si le paramétrage est appliqué avant d'atteindre un certain niveau d'uniformité, les correspondances de mots clés proposées ne sont pas pertinentes, car elles peuvent contenir des erreurs métier, des fautes d'orthographe, etc.

On cesse le paramétrage lorsque toutes les étapes et fonctions de test qui pouvaient être liées à l'aide de paramètres ont été regroupées.

BONNES PRATIQUES DE L'ACTIVITÉ

Afin d'optimiser la mise en place de paramètre, nous avons établi un ensemble de bonnes pratiques.

Bonne pratique

Veillez à ne pas confondre les données : Lors du refactoring, il est possible de confondre deux types de données, une donnée A et une donnée B qui nécessitent un traitement différent. Si nous unifions des étapes ou fonctions de test en un seul paramètre, nous traitons ces données de manière similaire, ce qui peut semer la confusion, car elles n'auraient pas dû l'être.

BONNE PRATIQUE 28 – Ne pas confondre les données

Un exemple où une confusion est possible est sur une application liée à la gestion des trajets ferroviaires traitant de la notion de PK (point kilométrique) de PR (point remarquable de référence) et de PC (point de conflit). Ce sont trois notions totalement différentes, mais dont les acronymes sont proches. Considérons les fonctions de test suivantes et leur nombre d'occurrences :

- "Vérifier que le PK est bien positionné" (apparaît 3 fois)
- "Vérifier que le point remarquable est bien positionné" (apparaît 10 fois)
- "Vérifier que le PR est bien positionné" (apparaît 40 fois)
- "Vérifier que le PC est bien positionné" (apparaît 50 fois)
- "Vérifier que le point de conflit est bien positionné" (apparaît 10 fois)

Une personne découvrant les suites de tests et connaissant peu le métier pourra penser que "PK" est une erreur dans les cas de test, car il apparaît peu de fois (3 fois ici). D'autant plus si la représentation visuelle ne met pas en avant de différences. Dans ce cas, la personne pourra faire le choix de créer la donnée "type de point", mais avec deux valeurs ("PR" et "PC", mais pas "PK"). Ceci introduit donc des erreurs dans les cas de test. D'où l'intérêt d'être vigilant lors de l'homogénéisation et l'application de paramètres.

Dans la continuité de la bonne pratique invitant à étudier avec attention les différentes données pour éviter des erreurs, nous proposons une bonne pratique visant à limiter les confusions au niveau des données.

Bonne pratique

Séparer les étapes de test manipulant des données différentes : dans le cas où des étapes ou fonctions de test sont très proches, mais où le type de données à manipuler est totalement différent, il est préférable de garder les étapes ou fonctions de test clairement séparés, quitte à développer chaque action pour éviter qu'elle ne soit confondue à l'avenir.

BONNE PRATIQUE 29 – Séparer les étapes de test manipulant des données différentes

Pour finir, bien que nous traitions ici de l'application de paramètre, nous conseillons de ne pas les utiliser de manière systématique, comme le décrit la bonne pratique suivante.

Bonne pratique

Ne pas introduire systématiquement des paramètres : Il n'est pas nécessaire d'introduire des paramètres de manière systématique. Ceci doit être fait dans le respect des règles métier et quand cela est pertinent pour le cadre du refactoring.

BONNE PRATIQUE 30 – Ne pas introduire systématiquement des paramètres

Cela est pertinent quand on souhaite éliminer le plus de redondances possible (pour éliminer les doublons), ou préparer l'automatisation. En effet, si l'automatisation est basée sur des mots clés liés à des étapes de test, l'effort est moindre pour produire une fonction avec un paramètre, que trois fonctions réalisant les mêmes stimulations sur le système. Par exemple, il est plus rapide d'avoir une fonction permettant de cliquer sur un bouton et prenant en paramètre ce bouton (exemple le clic sur le logo voiture, piéton ou autre) que d'avoir une fonction pour cliquer sur le bouton voiture, une autre pour cliquer sur le bouton piéton, etc.

ILLUSTRATION SUR L'EXEMPLE FIL-ROUGE

Dans notre exemple, il est pertinent de paramétrer le type d'itinéraire choisi, car il existe dans les cas de test le choix de l'itinéraire piéton, voiture et en transport. Dans la figure 5.13, l'action "cliquer sur l'icône voiture" est sélectionnée, mais elle a évolué par rapport à la figure 5.9. Ici voiture a été remplacé par le paramètre "type_itinéraire" avec la valeur "voiture".

L'action est toujours correcte et nous l'avons aussi appliqué dans les autres cas

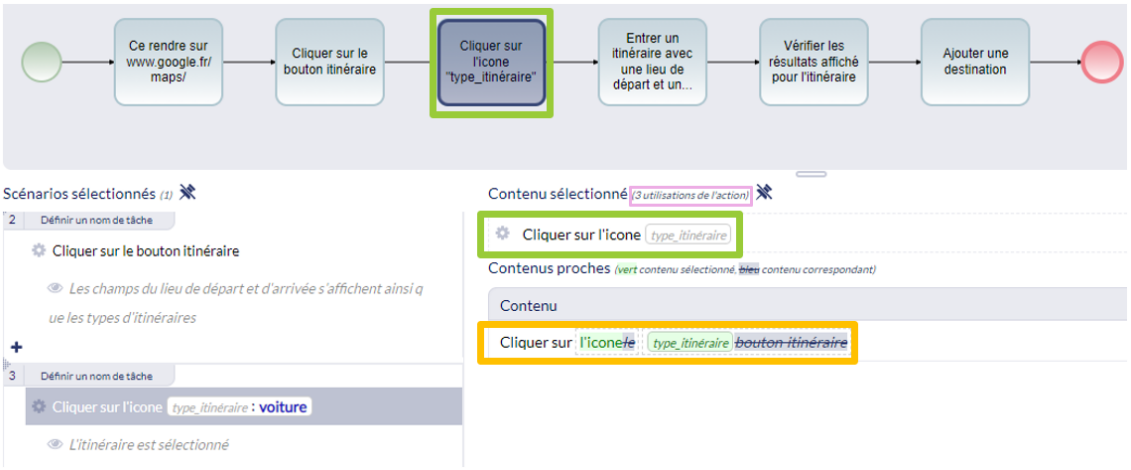


FIGURE 5.13 – Usage de paramètres

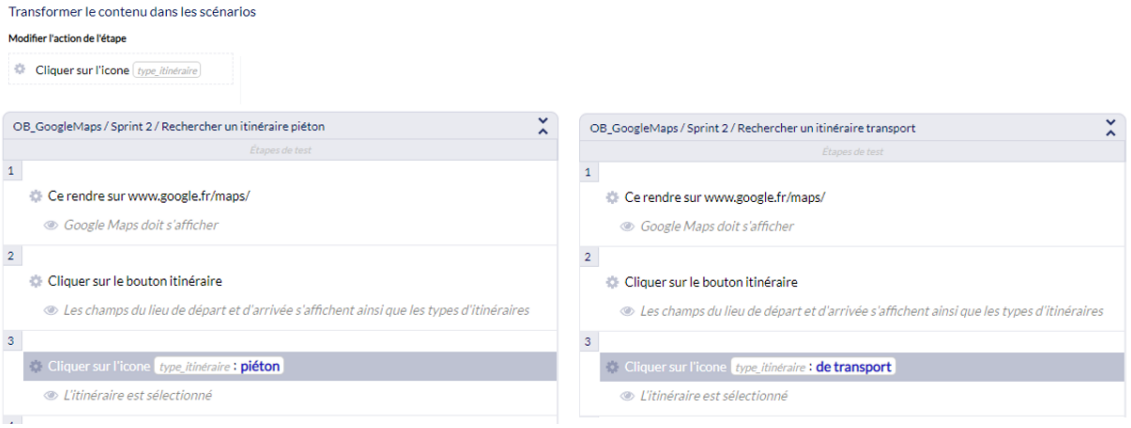


FIGURE 5.14 – Exemple de partage des paramètres entre différents cas de test

d'itinéraire. L'action *"Cliquer sur l'icône <type_itinéraire>"* est utilisée 3 fois comme spécifiée sur la figure 5.13. La figure 5.14 présente le détail des deux autres cas d'utilisations (piéton et transport). D'une part, dans le cas de l'itinéraire voiture comme initialement présentée, mais aussi dans les cas des itinéraires piéton et en transport après l'application de paramètres. Via la visualisation des cas de test, on peut s'assurer que l'usage des données est pertinent dans chacun des contextes d'appel. La représentation visuelle des suites de tests à la fin du processus de refactoring est visible dans la figure 5.15. Elle présente moins d'étapes de test qu'après l'activité d'homogénéisation car des étapes de test ont été fusionnées en fonctions de test par l'ajout de paramètres (comme pour les types d'itinéraire).

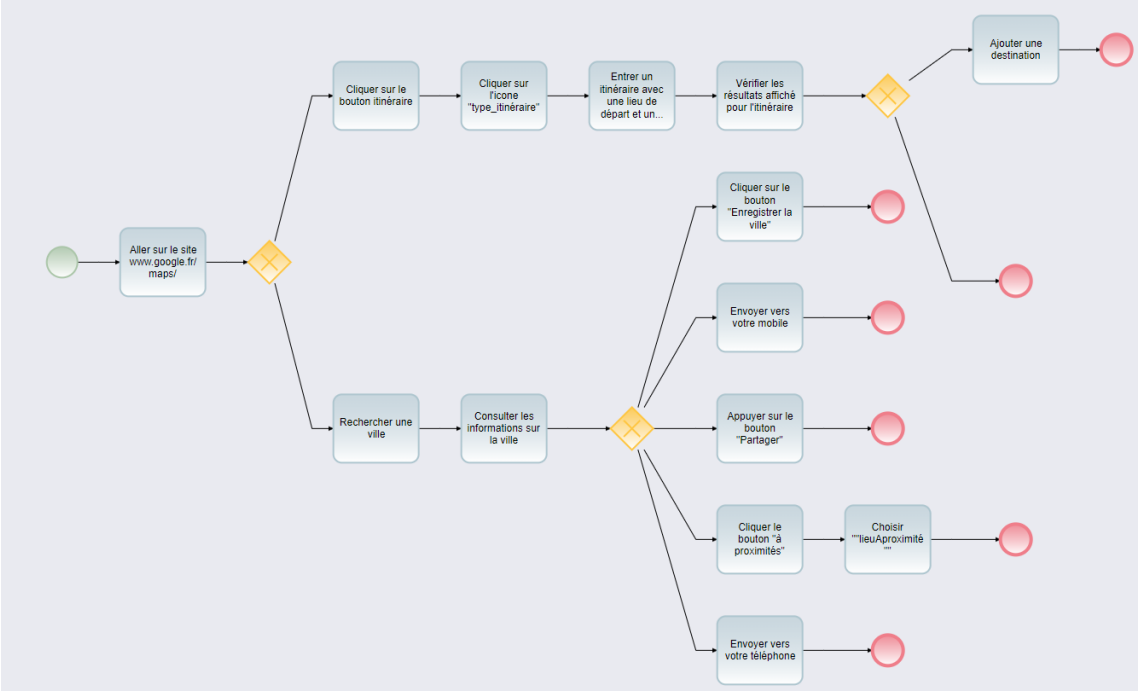


FIGURE 5.15 – Représentation visuelle des suites de tests à la fin du processus de refactoring

SYNTHÈSE

L'application de paramètres fournit un niveau supplémentaire d'homogénéisation des étapes et des fonctions de test. Toutefois, leur utilisation doit être effectuée avec une bonne connaissance des besoins du métier. Leur application peut avoir un effet positif ou négatif selon la manière dont elle est effectuée, d'où l'importance de suivre les bonnes pratiques afin de garantir une homogénéisation adéquate du référentiel.

5.2/ EXPÉRIMENTATIONS

Les expérimentations de refactoring présentées dans cette section ont été réalisées dans le contexte de la maintenance de suites de tests pour plusieurs grandes applications dans le domaine ferroviaire. La taille totale des référentiels de cas de test existants était de plusieurs milliers de cas de test manuels, documentés en langage naturel.

Nous avons appliqué notre approche de refactoring sur des sous-ensembles allant de plusieurs dizaines de cas de test à plusieurs centaines de cas de test (703 cas de test dans le plus grand sous-ensemble) en fonction de l'application et des objectifs du projet concerné.

Nous avons étudié deux types de suite de tests, celles dites fonctionnelles et celles de non-régression [3].

L'objectif du refactoring effectué se situait à plusieurs niveaux :

- réduire les redondances existantes pour diminuer le nombre de cas de test.
- identifier les étapes et fonctions de test effectuant les mêmes stimulations sur le système et les fusionner en une seule fonction de test pour faciliter la maintenance.
- paramétrer les étapes et fonctions de test pour préparer une éventuelle automatisation.

Pour vérifier l'efficacité de notre approche à répondre à ces objectifs, nous avons défini les métriques d'évaluation décrites dans la section suivante.

5.2.1/ MÉTRIQUES D'ÉVALUATION

Ces métriques concernent l'effort et les résultats obtenus :

- Le temps (en heures) nécessaire pour effectuer le refactoring avec notre approche et l'outil Orbiter. Il s'agit d'un temps mesuré.
- Le temps (en heures) nécessaire pour effectuer le refactoring sans outils. Il s'agit d'un temps estimé.
- Le décompte initial et final des cas, des fonctions de test et des étapes, afin de calculer les pourcentages de réduction.

La raison de la comparaison d'un temps mesuré (le temps de refactoring avec notre approche outillée) et d'un temps estimé (le temps sans notre approche ni outillage) provient

3. Une définition de ces termes sont définies en Annexe A.1

de l'impossibilité, durant ces travaux, de mettre en place un protocole où les méthodes sont utilisées de façon indépendante, puis comparées. Il aurait fallu pour cela deux personnes distinctes en parallèle, avec des compétences similaires et un même niveau de connaissance applicative. Ce que nous n'avions pas.

ESTIMATION DES TEMPS DE REFACTORING SANS OUTILLAGE

Notre estimation est basée, tout d'abord, sur un élément quantifiable dans les suites de tests : le nombre d'étapes et de fonctions de test dans la suite de tests : NBO

Ensuite nous considérons des temps de consultation/modification de base pour une étape ou fonction de test de complexité :

- faible : S = 2 secondes.
- moyenne : M = 4 secondes.
- élevée : C = 6 secondes.

Pour ces étapes et fonctions de test nous estimons une complexité moyenne (AC), basée sur une échelle de 1 à 5 choisie par l'utilisateur, où 1 est simple et 5 est complexe. La complexité moyenne des étapes ou fonctions de test (AC) est utilisée pour pondérer les temps de consultation et de modification de base pour les étapes ou fonctions de test (faible, moyenne et élevée) afin de tenir compte des suites de tests plus complexes.

Nous multiplions les temps consultation/modification de base pour une étape ou fonction de test par la complexité moyenne AC pour obtenir trois nouveaux temps (en secondes) :

- temps estimé de consultation/modification indice faible : $IS = AC * S$
- temps estimé de consultation/modification indice moyen : $IM = AC * M$
- temps estimé de consultation/modification indice élevé : $IC = AC * C$

Ces temps (IS, IM et IC) sont les temps estimés de consultation/modification par le testeur d'une étape ou fonction de test en tenant compte de la complexité de cette étape ou fonction de test.

Sur la base de notre expérience, nous avons considéré que les toutes les étapes ou fonctions de test ont été consultées et :

- pour 50%, elles ont été consultées rapidement (complexité faible)
- pour 50%, elles ont été consultées de façon moyenne (complexité moyenne)

En revanche, nous ne modifions pas toutes les étapes ou fonctions de test, nous considérons que la moitié est modifiée. Pour une moitié, nous estimons que parmi les étapes ou fonctions de test que nous prévoyons de corriger :

- pour 50%, elles prendront un temps moyen pour être modifiées
- pour 50%, elles prendront un temps élevé pour être modifiées

Le temps de consultation est égal à la moitié du nombre d'étapes de test initial ($NBO/2$) multiplié par le temps faible (IS), plus l'autre moitié du nombre d'étapes de test initial ($NBO/2$) multiplié par le temps moyen (IM).

Le temps de modification est calculé de la même manière que le temps de consultation, mais ajusté par un pourcentage d'objectif de refactoring (RO) et en utilisant le temps moyen (IM) et le temps élevé (IC).

L'objectif de refactoring (RO) est un pourcentage choisi par l'utilisateur, donnant une estimation de la réduction globale qu'il souhaiterait obtenir par le refactoring.

Le temps total estimé pour le refactoring manuel est la somme du temps de consultation et du temps de modification.

Ainsi, le temps de modification est égal à la moitié des étapes de test initiales ($NBO/2$) multipliée par l'objectif de refactoring (RO) multipliée par le temps moyen (IM), plus la moitié des étapes de test initiales ($NBO/2$) multipliée par l'objectif de refactoring (RO) multipliée par le temps élevé (IC). Tout cela donne la formule suivante :

Estimation du temps de refactoring manuel en minutes

$$\frac{NBO * (IS + IM + RO * (IM + IC))}{2 * 60}$$

5.2.2/ BILAN DES EXPÉRIMENTATIONS

Nous avons réalisé neuf expérimentations de refactoring dont les résultats sont présentés dans le tableau 5.16. Pour réaliser l'évaluation, le processus décrit dans la section 5.1 a été suivi.

CONTEXTE DES EXPÉRIMENTATIONS

Chaque expérimentation porte sur un ensemble de fonctions de tests manuels existants, issus du référentiel de test à notre disposition, et réalisé dans le contexte d'un projet réel

donné (c'est à dire avec un objectif de refactoring correspondant à un besoin du projet concerné). Ainsi, les expérimentations menées l'ont été dans les conditions de terrain (le centre de service Sogeti).

RÉSULTATS DES EXPÉRIMENTATIONS

#	Type de test	Nombre de cas de test	Diminution du nombre de cas de test	Nombre d'étapes de test	Réduction du nombre d'étapes de test	Nombre de fonctions de test	Diminution du nombre de fonctions de test	Temps estimé pour une reprise manuelle	Temps passé avec Orbiter	Ecart de temps entre la reprise manuelle/Orbiter
1	non-régression	11,00	0,00%	278,00	-31,65%	151,00	-48,34%	102,00	45,00	-55,88%
2	non-régression	9,00	-33,33%	334,00	-29,94%	201,00	-13,43%	45,00	20,00	-55,56%
3	non-régression	63,00	0,00%	1380,00	-7,25%	920,00	-18,80%	460,00	330,00	-28,26%
4	fonctionnel	44,00	0,00%	350,00	-0,57%	301,00	-23,87%	58,00	40,00	-31,03%
5	fonctionnel	39,00	0,00%	369,00	0,00%	313,00	-14,70%	49,00	30,00	-38,78%
6	fonctionnel	547,00	-1,28%	4105,00	-2,46%	2473,00	-5,90%	1710,00	205,00	-88,01%
7	fonctionnel	302,00	0,00%	6212,00	0,00%	1444,00	-28,74%	932,00	600,00	-35,62%
8	fonctionnel	703,00	-3,98%	16648,00	-2,04%	5440,00	-4,25%	971,00	338,00	-65,19%
9	non-régression	31,00	-90,32%	402,00	-84,58%	249,00	-92,77%	251,00	338,00	34,66%
							Totaux	**4578**	**1946**	**-57,49%**

FIGURE 5.16 – Tableau des résultats

Toutes suites de tests confondues, nous observons une diminution moyenne de 14% du nombre de cas de test, 18% du nombre d'étapes de test et 28% des fonctions de test.

Constatation : les suites de tests sont plus concises.
Le refactoring permet la réduction du nombre de cas de test, du nombre d'étapes de test et du nombre de fonctions de test participant à rendre plus concises les suites de tests en conservant une couverture fonctionnelle identique.

CONSTAT 12 – Les suites de tests sont plus concises

En complément, nous avons estimé que le temps de refactoring a été réduit de 57% en moyenne avec l'outil par rapport au refactoring manuel. Ces résultats varient selon les suites de tests étudiées et leur type : fonctionnels ou de non-régression.

Constatation : notre approche est deux fois plus rapide.
Nos résultats laissent supposer que notre approche de refactoring est deux fois plus rapide qu'une approche manuelle de reprise des suites de tests.

CONSTAT 13 – Notre approche est deux fois plus rapide

RÉDUCTION DU NOMBRE DE CAS DE TEST

On observe une réduction moyenne de 14% du nombre de cas de test. Avec une diminution moyenne de 31% pour les suites de tests de non-régression et 1% pour les suites de tests fonctionnels.

Pour les suites de tests de non-régression : ces résultats s'expliquent par le fait que l'objectif était de réduire le nombre de cas de test. Les tests étant plus transverses, des cas de test étaient très proches et certains parmi eux ont pu être regroupés et d'autres, supprimés. Dans un cas, nous avons supprimé 90% des cas de test. En analysant le référentiel, nous nous sommes rendu compte que cette suite de tests de non-régression, qui avait été construite sur une durée d'un an, décrivait quasiment à chaque fois les mêmes comportements, mais en ajoutant quelques étapes de test à chaque cas de test. Les cas de test étaient donc similaires entre eux à hauteur de 80%, mais utilisaient des terminologies différentes pour décrire les mêmes comportements. Pour simplifier le référentiel, il a donc fallu dans un premier temps que nous uniformisions les étapes et fonctions de test, pour nous rendre compte de la redondance, pour ensuite finalement supprimer quasiment l'ensemble des cas de test, car ils étaient identiques.

Pour les suites de tests fonctionnels : peu, voire aucun cas de test n'ont été supprimés. Chaque cas de test vérifiait un aspect précis d'une fonctionnalité et chaque cas de test était plutôt indépendant. C'est pour cette raison que pour 3 suites de tests fonctionnels sur 5 aucun cas de test n'a été supprimé et que la moyenne de réduction du nombre de cas de test est aussi faible. Des changements ont plutôt été opérés au niveau des fonctions de test et c'est ce que nous allons décrire.

Constatation : la réduction du nombre de cas de test.
La réduction du nombre de cas de test est plus fortes pour les suites de tests de non-régression. Ils comportent plus de cas de test similaires entre eux que les suites de tests fonctionnelles. Il est donc plus commun de supprimer des cas de test dans les suites de tests de non-régression que dans les suites de tests fonctionnelles. Ceci explique l'écart de diminution du nombre de cas de test entre ces deux types de suites de tests.

CONSTAT 14 – La réduction du nombre de cas de test

RÉDUCTION DU NOMBRE DE FONCTIONS DE TEST

La réduction du nombre de fonctions de test est en moyenne à 28% toutes suites de tests confondues. Pour les suites de tests de non-régression, on observe une moyenne de réduction à hauteur de 43% contre 15% pour les suites de tests fonctionnelles.

On observe ici le même phénomène que pour la diminution du nombre de cas de test. Les suites de tests de non-régression sont composées de comportements transverses, donc elles contiennent plus de redondances et les résultats obtenus en termes de réduction sont plus fort que la moyenne et que sur les tests fonctionnels[3].

3. Une définition de ce terme dans le contexte de l'approche ALME est fournie en Annexe A.1

Les tests de non-régression [3] étant écrit au fil de l'eau et sans forcement reprendre des tests existants on retrouvait donc souvent des formulations différentes, mais décrivant le même comportement. Cependant, un autre élément entre en jeu pour expliquer cette diminution : l'introduction de paramètres. Ici, nous cherchions à les uniformiser pour simplifier la mise en place de l'automatisation, nous avons donc utilisé des paramètres. Par exemple, l'introduction d'un paramètre sur l'étape de test de connexion a permis de passer de 6 étapes de test différentes à une seule fonction de test paramétrée. En appliquant ce mécanisme pour des étapes et fonctions de test identiques à différents profils, nous avons ainsi diminué le nombre de fonctions de test. Ceci explique donc le taux de diminution du nombre de fonctions de test pour les suites de tests de non-régression.

Quant aux suites de tests fonctionnelles, cette diminution est moins forte (15%), mais non négligeable. Contrairement aux tests de non-régression, les cas de test fonctionnels sont écrits pour chaque nouvelle évolution et reprennent des comportements existants et évoluent.

Constatation : la réduction du nombre de fonctions de test.
La réduction du nombre de fonctions de test est plus forte dans les suites de tests de non-régression que dans les suites de tests fonctionnelles. Ceci s'explique par le caractère transverse de ces suites de tests.

CONSTAT 15 – Réduction du nombre de fonctions de test

RÉDUCTION DU NOMBRE D'ÉTAPES DE TEST

La réduction du nombre d'étapes de test est en moyenne de 18% toutes suites de tests confondues. Pour les suites de tests de non-régression, on observe une moyenne de réduction de 38% contre 1% pour les suites de tests fonctionnels.

Comme pour les deux indicateurs précédents (nombre de fonctions de test et de cas de test) on observe toujours une diminution plus forte pour les suites de tests de non-régression et une réduction très faible pour les suites de tests fonctionnelles. Ces résultats s'expliquent de la même manière : les cas de test de non-régression comportent des doublons, d'où le besoin de supprimer certaines étapes de test.

Les cas de test fonctionnel eux sont très précis et comporte déjà chacun un nombre limité et précis d'étapes de test pour vérifier les comportements fonctionnels dans le détail. Il n'y avait donc pas besoin de supprimer des tests, mais plutôt de les corriger et uniformiser comme cela était présenté dans la section précédente.

Constatation : la réduction du nombre d'étapes de test.
La réduction du nombre d'étapes de test est plus forte dans les suites de tests de non-régression que dans les suites de tests fonctionnelles. Ceci s'explique par le caractère transverse de ces suites de tests.

CONSTAT 16 – La réduction du nombre d'étapes de test

TEMPS ESTIMÉ DE LA REPRISE MANUELLE COMPARÉ AU TEMPS PASSÉ AVEC L'OUTIL

L'écart de temps entre l'estimation de la reprise manuelle et la reprise avec Orbiter s'élève en moyenne à 40%.

On observe un gain moyen de 26% pour les suites de tests de non-régression et 52% pour les suites de tests fonctionnel.

Il faut noter que dans nos expérimentations, pour la suites de tests numéro 9 portant sur des tests de non-régression nous avons estimé passer 34% de temps supplémentaire avec notre approche. Ceci s'explique par le fait que nous avons supprimé 90% des cas de test. Dans ce cas, une reprise manuelle aurait été plus rapide, car les cas de test auraient été directement supprimés au lieu de procéder à la correction et l'uniformisation des étapes et fonctions de test.

Ce genre de cas est toutefois à la marge, et si on l'enlève de nos expérimentations pour calculer l'écart de temps entre une reprise moyenne et Orbiter, on obtient finalement 47% de moyenne de réduction de temps pour les suites de tests de non-régression ce qui est finalement très proche des suites de tests fonctionnelles (pour rappel 52%).

Constatation : gains observés.
On observe un gain plus élevé pour les suites de tests fonctionnelles que pour les suites de tests de non-régression. Ceci peut s'expliquer par le caractère plus précis et plus uniforme des suites de tests fonctionnelles : leur analyse est plus rapide. Le gain plus faible pour les suites de tests de non-régression s'explique par leur caractère transverse. Leur étude est plus longue, car les comportements décrits dans les cas de test sont plus disparates et demandent donc plus d'études et donc plus de temps.

CONSTAT 17 – Gains observés

DISCUSSION DES BIAIS DE VALIDITÉ DE L'ÉTUDE

De meilleurs résultats sont observés sur des suites de tests plus petites (moins de 65 cas de test, ce qui représente 66% des référentiels de tests examinés). Cela s'explique par le fait que sur un ensemble réduit, nous pouvons généralement examiner tous les cas de test, alors qu'à grande échelle (au moins 300 cas de test), nous n'examinons pas tous les cas de test. Un point d'attention serait de développer les expérimentations et d'essayer de couvrir des référentiels plus larges dans leur ensemble. Cela permettrait de traiter un plus grand nombre de cas de test, afin d'obtenir des résultats plus détaillés et de ne pas cibler que des sous-ensembles des larges référentiels.

En complément, nos estimations sur le temps nécessaire pour effectuer le refactoring manuel sont basées sur un calcul qui comprend un ensemble de mesures et la validation des testeurs qui connaissent bien le référentiel de test. Bien que les temps évalués semblent cohérents, ils ne remplacent pas les expériences réelles de refactoring manuel par un enregistrement du temps passé. L'absence de comparaison manuelle est due à la difficulté de faire réaliser l'expérience par deux personnes différentes (pour éviter les biais et réduire l'effet d'apprentissage) qui auraient eu le même degré de maturité et de connaissance du référentiel. Cela aurait également doublé les coûts et le temps consacré au refactoring, ce qui n'était pas souhaitable dans les contextes industriels.

5.3/ SYNTHÈSE

Dans ce chapitre, nous avons présenté notre approche de refactoring des suites de tests manuelles. Cette approche est structurée en 5 activités pour identifier et réduire les obsolescences des suites de tests et les rendre plus faciles à maintenir, plus réutilisables, plus concises et plus simples à automatiser.

Lors des expérimentations, nous avons vu que l'approche proposée permettait de réduire le nombre de cas de test, de fonctions de test et d'étapes de test pour les rendre plus concises, et ainsi faciliter leur maintenance et leur réutilisabilité futures. De plus, les cas de test sont plus réutilisables, car ils sont d'une part moins nombreux et d'autre part leurs étapes et fonctions de test sont plus uniformes. Cela correspond à la question de recherche RQ-4 présentée en introduction de ce chapitre. De plus, la comparaison entre les temps passés avec notre approche outillée et une estimation du temps requis sans outillage montre un gain de productivité important. Cela correspond à la question de recherche RQ-5.

De plus, dans un contexte d'évolution fonctionnelle de l'application, si des modifications de masse sur une suite de tests doivent être réalisées, la réutilisation de l'outil pour mettre à jour l'ensemble des étapes et fonctions de test peut être considérée comme une

alternative à une approche manuelle. Le support outillé permet un gain de temps vis-à-vis d'une approche manuelle, car elle nécessite une mise à jour individuelle de chaque étape ou fonction de test contrairement au support outillé présenté. Tous ces éléments font que l'approche proposée rend moins coûteuse à utiliser et à maintenir les suites de tests manuelles comparées à une approche non outillée.

Cependant la réutilisabilité des cas de test dépend majoritairement des pratiques mises en place en test après le refactoring. Sans une stratégie pour éviter les redondances, avec le temps la réutilisabilité du référentiel pourrait diminuer.

Un autre aspect que nous avons étudié dans notre approche est aussi comment elle pouvait faciliter l'automatisation par le refactoring des suites de test. L'usage de notre approche nous a aidés à rendre plus facile l'automatisation en réduisant le nombre de fonctions de test à implémenter grâce aux paramètres. En se basant sur des techniques de KDT, plus il y a de mots clés, plus l'implémentation des mots clés va prendre du temps. D'autant plus si certains fonctions de test ou étapes de test produisent des comportements similaires, mais sont écrits différemment. Les automaticiens n'ont pas toujours une vision métier élevée des comportements du système. Ils peuvent donc être amenés eux aussi à réaliser des développements de mots clés en doublon. De plus, la complétion de la couche d'adaptation va être plus difficile, car il faudra relier de nombreuses étapes et fonctions de test à chaque mot clé. En uniformisant les mots clés on réduit et facilite l'automatisation au niveau de l'implémentation des mots-clés et de leur liaison avec les cas de test manuels.

De futurs travaux qui pourraient être réalisés pour mieux évaluer les résultats de notre approche seraient de comparer réellement une reprise manuelle et non une estimation avec notre approche. Nous pourrions ainsi observer les différents pourcentages de résultats obtenus en termes de réduction du nombre de cas de test, fonctions de test et étapes de test. Un autre aspect serait de raffiner notre estimation de reprise manuelle. Nous avons observé que selon le type de référentiel (fonctionnel ou de non-régression) et la taille des suites de test, les estimations et résultats pouvaient beaucoup varier. Nous pourrions donc introduire de nouveaux paramètres pour mieux prendre en compte ces deux facteurs et ainsi obtenir une estimation et des résultats plus précis.

6

CONTRIBUTION ANALYTIQUE : LES TESTS INTER-ÉQUIPES DANS L'AGILITÉ À L'ÉCHELLE

L'analyse de l'état de l'art sur les pratiques de test dans des contextes d'Agilité à l'échelle (cf. section 2.2.2) a montré un besoin d'approfondissement des connaissances, notamment dans la mise en œuvre des tests inter-équipes.

Notre hypothèse de recherche est que l'approche ALME, grâce à la modélisation des parcours applicatifs au niveau de la solution métier, peut faciliter la collaboration inter-équipes pour ce niveau de test. Mais avant d'approfondir cette hypothèse, il nous a semblé nécessaire de commencer par une analyse de terrain (celle-ci n'existant pas dans l'état de l'art) qui s'est concrètement traduite par deux études complémentaires :

- *une étude qualitative* au travers de 21 entretiens semi-structurés conduits auprès de professionnels impliqués dans une organisation de type Agilité à l'échelle.
- *une étude quantitative* sous la forme d'un questionnaire en ligne diffusé auprès des communautés de l'Agilité à l'échelle d'une part et des tests logiciels d'autre part pour lequel nous avons obtenu 66 réponses.

Les questions qui ont guidé notre travail étaient les suivantes :

- Quels rôles relatifs au test interviennent dans les contextes d'Agilité à l'échelle ?
- Quelles sont les activités des testeurs et comment ont-elles évolué dans les contextes d'Agilité à l'échelle ?
- Quelles sont les pratiques de test mises en place dans les contextes d'Agilité à l'échelle ?

- Quelles sont les difficultés rencontrées par les praticiens de l'Agilité à l'échelle dans la mise en place des tests inter-équipes ?
- Quels facteurs influent sur les pratiques de test notamment au niveau inter-équipes dans les contextes d'Agilité à l'échelle ?

Dans la suite de ce chapitre, nous commençons par introduire les concepts principaux de l'Agilité à l'échelle et des tests inter-équipes. Nous présentons ensuite le protocole et les résultats de l'étude qualitative (section 6.2) et de l'étude quantitative (section 6.3). Puis nous analysons les résultats comparés des deux études (section 6.4) et nous terminons par une synthèse.

6.1/ AGILITÉ À L'ÉCHELLE ET TESTS INTER-ÉQUIPES

Comme discuté dans le chapitre état de l'art, l'Agilité ayant été fondée pour des équipes de petites tailles (7 à 12 personnes), lorsqu'une organisation souhaite mettre en oeuvre l'Agilité sur un grand système, qui requiert des dizaines de contributeurs (voir des centaines dans certains cas), alors il faut organiser les équipes dans un format appelé "Agilité à l'échelle".

Cette organisation fait l'objet de cadres (appelés frameworks) qui ont évolué ces dernières années. Les plus diffusés sont SAFe, Less, le modèle Spotity et Scrum of Scrums, que nous avons déjà cités en section 2.2.

Notre propos n'est pas de rentrer dans le détail de l'un ou l'autre de ces frameworks : ils possèdent chacun un vocabulaire, des concepts et une approche spécifique de l'Agilité à l'échelle. Mais, ils ont tous en commun l'idée suivante : en Agilité à l'échelle, plusieurs équipes Agiles contribuent en même temps à un système. Il faut donc organiser ce travail concurrent et la validation du résultat commun. Cette validation au niveau de la solution globale est l'objectif des *tests inter-équipes*.

Dans une organisation de type "Agilité à l'échelle", il y a donc le niveau équipe qui implémente sur son propre périmètre, et le niveau inter-équipes (que nous appelons niveau Solution) qui concerne tout ce qui est commun, à la fois pour l'expression du besoin global, pour la validation de la solution métier à ce niveau et aussi pour le rythme et les décisions de mise en production au niveau du système complet.

6.2/ ÉTUDE QUALITATIVE

En génie logiciel, les études qualitatives [29] sont utilisées pour étendre les connaissances sur la façon dont les pratiques et les méthodes sont interprétées et adaptées

dans des contextes spécifiques.

Ce type d'étude correspond donc bien à notre objectif de recherche : mieux comprendre les pratiques actuelles et les problématiques relatives aux tests inter-équipes dans des contextes Agiles à grande échelle.

Pour cela, nous avons mené des interviews, qui est la technique de collecte de données courante pour une étude qualitative, en nous appuyant sur deux sources en définissant la méthodologie [81, 36].

21 interviews ont été réalisés de janvier à juin 2020 sous une forme semi-structurée, c'est à dire comprenant un mélange de questions ouvertes et de questions fermées, conçues pour obtenir non seulement les informations ciblées, mais aussi des éléments inattendus. Ces entretiens ont été réalisés auprès d'un panel de praticiens de l'Agilité à l'échelle ayant des rôles différents, tels que Consultant, Scrum Master ou Test Manager, et issus de différents secteurs d'activités tels que l'assurance, le service, le transport, etc. La figure 6.1 présente le panel complet des personnes interrogées.

Comme la qualité des résultats obtenus dépend de la qualité des répondants [36], nous avons pris soin de les sélectionner selon un ensemble de critères :

- **1er critère : leurs années d'expérience dans des contextes d'Agilité à l'échelle.** Pour ce critère, nous n'avons sélectionné que des personnes ayant au moins une année d'expérience.
- **2ème critère : leur implication dans des activités de test.** Nous avons choisi d'interviewer principalement des personnes impliquées dans ou coordonnant des activités de test afin de recueillir le plus d'informations possibles sur la mise en œuvre des processus de test dans un contexte Agile à l'échelle.
- **3ème critère : leur rôle.** Nous avons privilégié les personnes ayant des rôles impliquant la connaissance des tests et ayant des responsabilités dans leur gestion. Ceci afin d'obtenir des réponses d'acteurs ayant un certain recul sur la gestion des activités de test.
- **4ème critère : l'implication des répondants dans un contexte Agile à l'échelle.** Nous avons favorisé à la fois des personnes directement impliquées dans des contextes d'Agilité à l'échelle, c'est-à-dire faisant partie d'équipes Agiles au sein d'une solution organisée dans de l'Agilité à l'échelle, et des personnes ayant un rôle plus transverse comme supervisant la solution et les activités. Cela nous a permis de connaître à la fois la perception des personnes directement impliquées dans les projets et celle des personnes en charge de la coordination des activités.
- **5ème critère : la diversité des sources.** Nous avons essayé de sélectionner des

participants issus de différents secteurs industriels, avec des tailles, des nationalités et des degrés de maturité différents dans la pratique de l'Agilité à l'échelle.

ID	Domaine organisation	Rôle	Expérience	Impliqué dans les équipes	Impliqué dans le test
P1	Service	Expert en transformation des entreprises	Entre 3 et 10 ans	non	oui
P2	Service	Scrum Master	Entre 1 et 3 ans	oui	non
P3	Service	Test Manager	Entre 1 et 3 ans	non	non
P4	Banque / Assurance	Manager	Entre 1 et 3 ans	oui	oui
P5	Service	Manager	Entre 3 et 10 ans	non	non
P6	Service	Expert en transformation des entreprises	Entre 3 et 10 ans	non	oui
P7	Transports /Tourisme	Ingénieur test/qualité/validation	Entre 3 et 10 ans	oui	oui
P8	Transports /Tourisme	Responsable d'une cellule de test	Entre 3 et 10 ans	non	oui
P9	Communication / Multimédia / Télécom	Responsable d'une cellule de test	Entre 3 et 10 ans	non	oui
P10	Service	Manager	Entre 3 et 10 ans	non	oui
P11	Banque / Assurance	Responsable d'une cellule de test	Entre 3 et 10 ans	oui	oui
P12	Banque / Assurance	Responsable d'une cellule de test	Entre 3 et 10 ans	oui	oui
P13	Banque / Assurance	Responsable d'une cellule de test	Entre 3 et 10 ans	non	oui
P14	Communication / Multimédia / Télécom	Responsable d'une cellule de test	Entre 3 et 10 ans	non	oui
P15	Service	Test Manager	Entre 3 et 10 ans	oui	oui
P16	Institution publique	Responsable d'une cellule de test	Entre 3 et 10 ans	oui	oui
P17	Banque / Assurance	Test Manager	Entre 1 et 3 ans	oui	oui
P18	Commerce / Distribution	Responsable d'une cellule de test	Entre 3 et 10 ans	oui	oui
P19	Communication / Multimédia / Télécom	Expert en transformation des entreprises	Entre 3 et 10 ans	oui	oui
P20	Service	Manager	Entre 3 et 10 ans	non	oui
P21	Service	Ingénieur test/qualité/validation	Entre 3 et 10 ans	oui	oui

FIGURE 6.1 – Panel des personnes interrogées

Pour la préparation des entretiens, nous nous sommes basés sur les travaux de Elizabeth Hove et Bente Anda [36] :

- Nous avons conçu à l'avance des tableaux que nous avons remplis après les entretiens. Ceci afin de s'assurer que les détails spécifiques soient classés et enregistrés. Dans le tableau en annexe D.2 nous présentons un exemple factice de remplissage pour une interview. Pour un nombre moyen de 33 propositions recueillies par interview, nous présentons un exemple de classification parmi les différentes thématiques.

- Nous avons élaboré un guide d'entretien pour nous aider à organiser les interviews, visible en annexe D.5. Il se compose d'une liste de questions pour orienter l'entretien de manière à couvrir les thèmes du tableau.

Pour le déroulement des interviews, nous avons aussi suivi un ensemble de recommandations qui sont les suivantes :

- Chaque entretien a été précédé d'une présentation de l'étude menée, afin d'en faire comprendre les tenants et les aboutissants.

- Nous avons évité les questions auxquelles on ne pouvait répondre que par oui ou par non, et nous avons veillé à ne pas couper brusquement les personnes interrogées dans leurs réponses, même si la réponse fournie s'écartait de la question initiale.
- Nous avons feint l'ignorance, c'est-à-dire de demander des détails déjà bien connus, ce qui a permis de dissiper toute perception que nous aurions pu avoir sur des sujets particuliers.

Les interviews ont permis d'extraire un total de 706 déclarations recevables pour l'étude. Pour être recevable, une réponse donnée (appelée "déclaration") par les personnes interrogées doit concerner le sujet de l'étude, ou y être liée, c'est-à-dire concerner des activités de test réalisées dans des contextes d'Agilité à l'échelle et plus précisément au niveau inter-équipes.

6.2.1/ COMMENT ONT ÉTÉ ANALYSÉES LES DONNÉES ?

Pour analyser les données de l'étude qualitative nous avons mis en place un processus appelé codage, relaté dans les méthodes des articles [82, 36]. Dans notre contexte, le processus de codage est basé sur la classification des déclarations issues de nos entretiens (considérées comme des valeurs) par thèmes (les variables), ce qui nous permet de transformer nos données qualitatives en données quantitatives. Cette approche est connue dans la littérature sous le nom de méthode de comparaison constante [82]. Une approche similaire est mise en œuvre par Afsoon Afzal et al. dans une étude sur les défis posés par les tests de systèmes robotiques [2].

6.2.2/ MÉTHODE D'ANALYSE DES RÉSULTATS DE L'ÉTUDE QUALITATIVE

Dans le cadre de nos recherches, nous avons identifié une série de thèmes et regroupé chacune des déclarations issues des interviews en fonction de ces thèmes (afin de suivre la méthode de codage présentée plus haut). Les thèmes identifiés sont les suivants :

- La maturité Agile est faible
- La maturité Agile est modérée
- La maturité en test est faible
- La maturité en test est modérée
- L'atelier des 3 amigos est utilisé

- Les tests inter-équipes sont maîtrisés
- Les tests inter-équipes ne sont pas maîtrisés
- Il existe un manque de personnel compétent en test
- Un Test Manager ou similaire est intégré à la solution
- Des pratiques et outils communs sont mis en place
- Le niveau de maturité des équipes est hétérogène
- Il existe des équipes de test transverses
- Des rôles sont fusionnés (testeur, BA, dev)
- Il existe un manque de disponibilité/implication/vision du métier
- Il y a de la résistance au changement
- Il existe des difficultés de synchronisation/vision hors équipe
- Les comportements et attentes du top management sont parfois non adaptés
- Il existe des difficultés techniques

Il est rapidement apparu que les tests inter-équipes étaient souvent peu maîtrisés par rapport aux tests au sein des équipes.

Nous avons donc isolé, d'une part, les déclarations liées à un bon degré de maturité dans la réalisation des tests inter-équipes, que nous avons classées sous le thème : "Maîtrise des tests inter-équipes".

Et d'autre part nous avons classé les déclarations qui traitaient d'un manque ou d'une absence de maturité dans la réalisation des tests au niveau inter-équipes dans le thème : "Faible maturité des tests inter-équipes".

Ensuite, nous avons comparé le nombre de déclarations entre chacun de ces 2 thèmes et leur contenu pour construire 4 niveaux de maîtrise dans les tests inter-équipes. Nous avons fait ce choix de comparaison parce que l'étude de toutes ces déclarations a clairement divisé les contextes d'étude en 4 niveaux de maîtrise :

- pas de maîtrise
- faible maîtrise
- maîtrise moyenne
- maîtrise élevée

Pour chacun de ces niveaux de maîtrise dans les tests inter-équipes, nous avons étudié le nombre et le contenu des déclarations des autres thèmes afin de déterminer une corrélation entre le niveau de maîtrise inter-équipes et les autres thèmes. Ceci afin d'identifier les différents facteurs qui peuvent ou non influencer la maîtrise des tests inter-équipes.

En complément, nous avons comparé les réponses fournies sur ces différents thèmes avec les résultats du questionnaire afin de vérifier si une corrélation existe entre les réponses pour les différentes sources de résultats (interviews et questionnaire).

6.3/ ÉTUDE QUANTITATIVE

Notre étude quantitative a été réalisée sous la forme d'un questionnaire en ligne destiné à obtenir de l'information auprès de nouvelles personnes à partir des réseaux sociaux professionnels. Le questionnaire a été construit pour compléter l'étude quantitative dans une approche inspirée de [36].

Ce questionnaire comprend 28 questions (cf. annexe D.4), dont une partie a pour but de vérifier si les thèmes et propos recueillis lors de l'analyse qualitative sont présents à plus large échelle. Pour vérifier cela, nous avons intégré des questions rejoignant les thématiques abordées et d'autres ouvrants de nouvelles thématiques.

Ces nouvelles thématiques, au nombre de 9, sont les suivantes :

- Il y a des difficultés d'automatisation
- L'Agilité à l'échelle n'a pas d'impact pour les tests
- L'évolution des rôles en test est bien vécue
- La gestion des tests s'est complexifiée
- L'intégration des tests dans les itérations est compliquée
- Des techniques comme le BDD et l'ATDD sont utilisées
- Le soutien de l'organisation est important pour la qualité globale de la solution

Les détails de la construction du questionnaire et des réponses obtenues sont exposés dans la section qui suit. Le questionnaire a été ouvert sur la durée du mois de juin 2020. L'information a été diffusée via notre réseau de connaissance par mail et via un groupe de travail sur LinkedIn [1] regroupant des professionnels et experts de l'Agilité à l'échelle. Nous avons obtenu 66 réponses.

1. Il s'agit du groupe SAFe francophone - https ://www.linkedin.com/groups/8521588/.

6.3.1/ Analyse des résultats de l'étude quantitative

Pour analyser les données de l'étude quantitative, nous avons mis en place un processus de codage similaire à l'étude qualitative.

Nous avons effectué le classement des réponses aux questions selon les thématiques identifiées lors des interviews. Chaque nombre de réponses est comptabilisé sur une thématique.

Ainsi pour la question : "Parmi les propositions suivantes concernant le rôle test, lesquelles sont vraies dans votre contexte ?" et pour laquelle la proposition "Un ou plusieurs Test Managers coordonnent les activités de test" a été sélectionnée 27 fois, on comptabilise ces 27 réponses dans la thématique "Intégration du Test Manager ou similaire". Chaque question et réponse est traitée selon le même principe.

6.3.2/ Détail des questions

Parmi les 28 questions, les 14 premières étaient dédiées à l'identification des répondants et leur contexte de travail :

- du profil des personnes interrogées (rôle, années d'expérience dans l'Agilité à l'échelle, nombre d'expériences, dans quels pays),
- de leurs activités (domaines d'activités, gestion et participation aux activités de test et la mise en place de la transformation vers l'Agilité à l'échelle),
- de leurs environnements de travail (technique d'Agilité à l'échelle utilisée, types d'applications, approche mise en place avant le passage à l'Agilité à l'échelle),
- de l'organisation Agile (taille, nombre et répartition des équipes Agiles).

Les questions 15 et 16 étaient dédiées à une évaluation des difficultés rencontrées de manière globale et au niveau métier dans l'Agilité à l'échelle.

Les questions 17, 18, 19, 20 et 22 portaient sur les activités et rôles en test. À noter que la question 22 apparaît plusieurs fois car les réponses qui la composent abordent différents sujets : les activités et rôles en test et la maîtrise des tests inter-équipes.

La question 21 portait sur les techniques de test.

Les questions 22 et 23 traitaient de la maîtrise des tests et des difficultés rencontrées.

La question 24 était une question ouverte sur des propositions pour répondre aux challenges évoqués sur la réalisation des tests inter-équipes.

La question 25 interrogeait sur les éléments clés pour la maîtrise des tests inter-équipes selon des choix multiples et la question 26 sur les éléments à mettre en place pour garantir la maîtrise à travers des réponses libres.

Enfin les questions 27 et 28 étaient à texte libre afin de permettre aux répondants de laisser des remarques et adresses mail.

6.4/ ANALYSE ET DISCUSSION

Pour rapprocher nos deux études, nous avons extrait des valeurs de variables quantitatives à partir de données qualitatives (issues des entretiens) [36].

Nous avons sélectionné pour cela 9 questions du questionnaire qui pouvaient être mises en relation avec les interviews. Ont été exclues les questions sur l'identification de la personne interrogée. Et sur l'ensemble des réponses aux questions du questionnaire, les réponses totalement hors périmètre représentent moins de 1% (0,9%) et ont été exclues. Nous considérons comme hors périmètre les réponses libres où les répondants ont répondu ne pas savoir ou ont répondu hors thématique.

Une fois la classification des réponses réalisées, nous avons comparé le nombre de réponses obtenues par thématiques entre les interviews et le questionnaire. Ce nombre est exprimé par un pourcentage visible dans le tableau en annexe D.1. À noter que certains pourcentages sont à 0% : ceci indique que la thématique n'a pas été abordée.

Il y a en moyenne 2,9% de différence et 2% de médiane entre les réponses apportées par thématiques lors des interviews et lors du questionnaire. Ce faible écart rend les 2 sources comparables et pouvant faire l'objet de recherche de corrélations plus approfondie.

Les plus grands écarts entre 5 à 10% s'expliquent par le fait que certaines thématiques des interviews comme "la maturité Agile/en test modérée/faible" regroupaient des éléments d'ordres génériques ne pouvant faire l'objet d'un isolement particulier. De ce fait, les propositions en lien avec ces thématiques sont plus importantes dans les interviews que dans le questionnaire. De nombreuses déclarations ont pu être extraites des interviews, contrairement au questionnaire où les réponses à fournir étaient plus ciblées, malgré la possibilité qui était laissée aux personnes sondées de s'exprimer à l'aide des réponses ouvertes pour chaque question.

Afin d'établir une première analyse, nous avons étudié indépendamment chacune des thématiques en regroupant les résultats issus des interviews et de le questionnaire (visible dans l'annexe D.2). Nous avons ensuite formulé des observations que nous présentons dans la suite, en structurant par thème.

À noter que les sommes des pourcentages peuvent être supérieures à 100%, car dans le questionnaire en ligne, de nombreuses questions permettaient le choix de plusieurs réponses.

L'ORGANISATION DES RÔLES RELATIFS AU TEST DANS LES CONTEXTES D'AGILITÉ À L'ÉCHELLE

Nous avons observé la présence de testeurs au niveau Solution pour 72% des personnes ayant répondu au questionnaire, et ce constat est confirmé par les interviews.

La présence de Tests Managers à ce niveau est aussi fréquente pour 69% des répondants du questionnaire et à 90% dans les interviews.

Enfin dans 50% des cas du questionnaire, l'implication d'automaticiens et d'équipes de tests transverses est mentionnée. Dans les interviews, les équipes de tests transverses sont aussi mises en valeur avec 60% des répondants qui ont évoqué travailler avec elles, mais en revanche peu de données ont été récoltées sur l'automatisation.

Observations:

- Au moins un testeur fonctionnel est présent au sein des équipes Agiles et c'est fréquent au niveau Solution
- La présence de Tests Managers au niveau Solution est fréquente
- La présence d'équipe de tests transverses n'est pas systématique
- La présence d'automaticiens est peu mentionnée au niveau Solution

---

Pour synthétiser, il ressort de notre étude que Test Manager et testeur sont des rôles particulièrement présents dans les contextes d'Agilité à l'échelle et que par contre, la présence d'équipes de tests transverses et d'automaticiens est moins systématique. La présence de testeur ou Test Manager ne semble pas être un facteur qui influe sur la maîtrise des tests inter-équipes au vue des interviews, leur présence étant constatée dans tous les contextes de maîtrise. Il est important de mentionner que le Test Manager a été considéré comme un rôle clé pour la maîtrise des tests inter-équipes dans le questionnaire pour 62% des répondants. L'existence d'un Test Manager peut donc être considérée comme un facteur favorisant la maîtrise des tests inter-équipes même si sa seule présence n'est pas un gage de maîtrise.

Au sujet des équipes de tests transverses, on relève que moins les tests inter-équipes sont maîtrisés, moins on évoque la présence d'équipe de tests transverses. Les équipes

de tests transverses (combinée à d'autres éléments) sont donc un facteur qui pourrait favoriser la maîtrise des tests inter-équipes. Toutefois, la présence d'équipes de tests transverses peut ne pas être un facteur, mais le résultat d'une maîtrise des tests inter-équipes. On peut supposer que dans les contextes où les tests inter-équipes sont maîtrisés, la maturité des tests est assez forte. Par conséquent, les organisations ont pris soin de mettre en place des équipes de test transverses pour vérifier les comportements de bout en bout. Ces équipes réalisent des tests au niveau inter-équipes et participent donc à la maîtrise de ces tests.

Quant aux automaticiens, ce thème n'a pas été abordé dans les interviews et les réponses au questionnaire ne l'indiquent pas comme un facteur de succès des tests inter-équipes. Néanmoins, plusieurs personnes interrogées ont mentionné l'importance de mettre en place des mécanismes d'automatisation.

Observations:

- La présence de testeur, Test Manager et d'automaticien ne semble pas être des facteurs qui influent sur la maîtrise des tests inter-équipes
- Les équipes de tests transverses (combinée à d'autres éléments) sont un facteur qui pourrait favoriser le succès des tests inter-équipes.

LES ACTIVITÉS DES TESTEURS

Dans le chapitre 2, nous avons exprimé que le rôle des testeurs évoluait avec le passage du cycle en V vers l'Agile et cette transformation s'observe tout autant avec le passage vers l'Agilité à l'échelle. Notre questionnaire révèle que seulement 13% des personnes interrogées considèrent que les compétences et les activités des testeurs n'ont pas évolué. Dans notre étude qualitative, cette tendance se confirme. Les personnes interviewées insistent sur l'importance du rôle du testeur et de sa présence au sein des équipes. Le testeur doit être polyvalent, et impliqué fonctionnellement et techniquement dans la qualité de la solution. Fonctionnellement, il interagit avec les acteurs du métier pour la construction et la validation des User Stories, notamment sur la mise en place des critères d'acceptation. Techniquement, il doit être orienté de plus en plus vers l'automatisation pour permettre le test en continu dans les chaînes d'intégration continue. Ces nouveaux profils polyvalents sont parfois compliqués à trouver comme cela a été exprimé dans certains entretiens.

Ces constats sont confirmés par les réponses au questionnaire qui indiquent majoritairement (74%) que les testeurs collaborent davantage avec les acteurs du produit. Ils

réalisent des tâches plus techniques dans 62% des cas, et pour 56% des réponses, ils effectuent une plus grande variété de tâches. On retrouve la même tendance dans les interviews avec 70% des répondants qui estiment que les rôles du testeur ont évolué. Dans les contextes où la maîtrise des tests inter-équipes est avancée, on retrouve systématiquement des profils de testeurs "multi-profils". Néanmoins, on retrouve aussi des profils polyvalents dans des contextes où les tests inter-équipes ne sont pas maîtrisés. Les testeurs ont tendance à être plus polyvalents dans les contextes qui maîtrisent les tests inter-équipes, c'est donc probablement un facteur qui, combiné à d'autres, aide à la maîtrise des tests inter-équipes.

Observations:

- Les compétences et les activités des testeurs ont évolué.
- Les testeurs collaborent davantage avec les acteurs du produit
- Les testeurs polyvalents sont probablement un facteur qui, combiné à d'autres, aide à la maîtrise des tests inter-équipes.

L'ÉVOLUTION DE L'ORGANISATION DES ACTIVITÉS ET DES PRATIQUES DE TEST

Les réponses au questionnaire montrent que seulement un quart des personnes interrogées ont estimé qu'il y avait peu d'impacts en termes d'organisation des tests et que ces changements étaient pour la plupart bien vécus. Ces résultats peuvent suggérer que pour la majorité des contextes, la transformation a eu de fortes répercussions et n'a pas nécessairement été bien accueillie. Cette tendance se confirme dans les interviews où beaucoup de résistances aux changements ont été évoquées (à hauteur de 67%), essentiellement dans des contextes où les tests inter-équipes sont peu maîtrisés. La résistance au changement est donc un facteur qui peut influencer la maîtrise des tests inter-équipes.

Observations:

- La transformation vers l'Agilité à l'échelle a eu des impacts en termes d'organisation des tests
- La transformation a eu de fortes répercussions et n'a pas nécessairement été bien accueillie
- La résistance au changement est un facteur qui peut influencer la maîtrise des tests inter-équipes.

Ensuite, si nous entrons dans le détail des activités, nous avons peu appris sur l'organisation des tests inter-équipes, l'adoption de schémas organisationnels étant freinée par les difficultés de mise en œuvre des tests. Comme nous l'avons déjà mentionné, les organisations travaillant dans des contextes d'Agilité à l'échelle ont plusieurs équipes Agiles travaillant sur une solution commune. Chaque équipe comporte, dans la plupart des cas, un représentant qualité. Il est en charge en particulier de la conception et de l'exécution des tests sur le périmètre de l'équipe. Par contre, il n'est pas en charge des tests inter-équipes. Ce sont plutôt les équipes de tests transverses qui ont cette responsabilité.

Observations:
Le questionnaire nous apprend que les pratiques de test les plus utilisées au niveau inter-équipes sont :

- les tests exploratoires (71%)
- le BDD (44%)
- les ateliers dits 3 amigos (avec PO, Testeur et Développeur) (41%)
- le test à partir de checklists (24%)
- l'approche Risk-Based-Testing (21%)
- l'ATDD (21%)
- le Model-Based Testing (15%)

La pratique dominante est donc clairement les tests exploratoires. La prédominance des tests exploratoires peut s'expliquer par l'absence de scénario de test de bout en bout qui est la première difficulté évoquée dans les tests inter-équipes (Plus de détails sur les difficultés des tests inter-équipes sont présentés plus loin dans cette section).

La mise en place de communautés de test et de bonnes pratiques en test

Pour 1/3 des répondants au questionnaire, les activités de test sont régies par de bonnes pratiques mises en place par une communauté de test. Dans les interviews, 95% des répondants parlent de communautés de test et de bonnes pratiques en test.

Cependant, même si de bonnes pratiques existent, elles ne sont pas nécessairement appliquées. Ceci s'explique par le fait que les équipes travaillant sur une même solution sont très hétérogènes. Dans les réponses au questionnaire 45% affirment que les équipes sont hétérogènes et 81% dans les interviews.

Ces résultats montrent que malgré la mise en place de communautés de test et de bonnes pratiques, les équipes restent libres de travailler comme elles le souhaitent et d'appliquer ou non les bonnes pratiques en matière de test. Ceci explique aussi le fait que 50% des répondants aux interviews évoquent un manque de vision et de synchronisation entre les équipes. Chaque équipe travaille plus ou moins en silo sans tenir compte des développements transverses. Ce mode d'organisation s'explique dans les entretiens par un manque de temps pour s'impliquer au-delà de sa propre équipe, car l'organisation ne donne pas aux équipes les moyens de le faire.

Dans les contextes maîtrisant le mieux les tests inter-équipes on observe qu'il n'y a pas de difficultés de synchronisation entre les équipes. Avoir une bonne synchronisation et une vision globale, en dehors de l'équipe, doit donc être un facteur qui contribue à la maîtrise des tests inter-équipes. Néanmoins, il faut noter que dans certains contextes ne rencontrant pas de difficultés de synchronisation les tests inter-équipes sont peu maîtrisés. Ce facteur doit donc probablement être combiné à d'autres pour permettre la maîtrise des tests inter-équipes.

Observations:

- Les activités de test sont régies par de bonnes pratiques mises en place par une communauté de test
- Les équipes restent libres de travailler comme elles le souhaitent et d'appliquer ou non les bonnes pratiques en matière de test.
- Avoir une bonne synchronisation et une vision globale, en dehors de l'équipe, est un facteur qui contribue à la maîtrise des tests inter-équipes
- À l'heure actuelle, il semble y avoir peu de mécanismes entre les équipes pour permettre une coordination des tests inter-équipes

---

LES DIFFICULTÉS RENCONTRÉES POUR LES TESTS INTER-ÉQUIPES DANS L'AGILITÉ À L'ÉCHELLE

Les interviews et les réponses au questionnaire ont mis en évidence que les tests inter-équipes n'étaient pas maîtrisés dans la majorité des contextes. Dans les interviews, 67% des personnes interrogées mentionnent des éléments de non-maîtrise des tests inter-équipes et évoquent rencontrer des difficultés. Dans les réponses au questionnaire, 91% des répondants déclarent rencontrer des difficultés.

Ces difficultés sont majoritairement dues au manque ou à l'absence de scénarios de bout en bout pour vérifier les comportements transverses de la solution. Cette difficulté est présente dans la majorité des contextes indépendamment du niveau de maîtrise des tests inter-équipes.

Ensuite, le questionnaire met en avant à parts égales que c'est la gestion de l'automatisation des tests, l'intégration des tests dans les itérations et le manque ou l'absence d'environnement pour effectuer des tests qui posent des difficultés. Les difficultés dans la mise en place et la réalisation des tests inter-équipes sont donc diverses. Le manque de scénario de bout en bout est avéré.

À l'heure actuelle pour répondre à cette problématique les acteurs ont donc plutôt tendance à effectuer des tests exploratoires (pour 71% des répondants au questionnaire) qui ne nécessitent pas de scénarios de test contrairement aux autres pratiques de test comme le BDD, l'ATDD, etc. Quant aux raisons qui pourraient expliquer le manque de scénario de bout en bout, nous pouvons évoquer la faible communication entre les équipes qui ne permet donc pas de construire ces scénarios de test transverses pour la réalisation des tests inter-équipes. Une autre cause peut être le manque de moyen alloué à la qualité et qui est abordé dans la section suivante.

Observations:

- Les tests inter-équipes ne sont pas maitrisés
- Les difficultés sont majoritairement dues au manque ou à l'absence de scénarios de bout en bout pour vérifier les comportements transverses de la solution.
- La faible communication entre les équipes freine la construction des scénarios de test transverses

---

LE SOUTIEN DE L'ORGANISATION EST UN FACTEUR POUR LA RÉUSSITE DES TESTS INTER-ÉQUIPES

Bien que les interviews n'aient pas mis l'accent sur les limites organisationnelles (à hauteur de 40%), dans les réponses au questionnaire, 67% des répondants ont mentionné des difficultés dans la gestion hiérarchique et organisationnelle. Comme évoqué dans l'analyse des thématiques D.1, le faible pourcentage dans les interviews peut s'expliquer par le fait que les personnes interrogées ont préféré éviter de mentionner les limites de leurs organisations.

On peut remarquer que dans les contextes où les tests inter-équipes sont plutôt maîtrisés, on retrouve peu, voir pas de commentaire sur les limites hiérarchiques. Néanmoins, il y a aussi des contextes ne maîtrisant pas les tests inter-équipes et qui n'ont pas mentionné de problématiques au niveau organisationnel. Par contre, dans le questionnaire, nous avons demandé quels étaient les éléments clés à la maîtrise des tests inter-équipes et les réponses mentionnaient majoritairement le soutien de l'organisation sur la qualité globale de la solution. Nous pouvons donc en déduire que c'est un facteur qui influence la maîtrise des tests inter-équipes.

Observations:

- Des difficultés dans la gestion hiérarchique et organisationnelle existent
- Le soutien de l'organisation sur la qualité globale de la solution est un facteur qui influence la maîtrise des tests inter-équipes.

DISCUSSION DES BIAIS DE VALIDITÉ

Le nombre d'interviews de l'étude qualitative est assez représentatif des études qualitatives que nous avons vu dans la littérature. Par exemple, l'étude sur les défis posés par les tests de systèmes robotiques [2] concerne 12 professionnels interviewés.

Il nous semble que le principal biais de validité provient du nombre limité de répondants pour le questionnaire en ligne (66 répondants). Ceci est du en partie au temps dont nous avons disposé pour en effectuer la promotion mais aussi au fait que les personnes récentes sur les questions de l'Agilité à l'échelle ne voulaient pas répondre sans une expérience plus affirmée.

Un autre point de discussion est le caractère national de nos études, tant pour les interviews que pour le questionnaire en ligne. Il serait intéressant de déployer ces études à

un niveau international, mais cela fait partie des perspectives ouvertes par cette thèse.

Enfin, le caractère très récent du déploiement de l'Agilité à l'échelle dans les organisations fait que ce sujet est en forte évolution, y compris sur la question des tests inter-équipes. Il est probable que si nous refaisons ce questionnaire à un an d'intervalle, nous pourrons constater une évolution dans les observations.

6.5/ SYNTHÈSE

Afin de mettre en lumière les pratiques des tests logiciels dans les contextes de l'Agilité à l'échelle, nous avons mené deux études, l'une qualitative et l'autre quantitative. Elles nous ont permis de mieux comprendre les pratiques et difficultés actuelles.

Le point essentiel mis en évidence est qu'aujourd'hui les tests inter-équipes ne sont pas bien maîtrisés et que les praticiens rencontrent des difficultés dans leur implémentation.

La principale difficulté est l'absence ou le manque de scénarios de bout en bout, suivie par les difficultés de gestion de l'automatisation, d'intégration des tests dans les itérations, et enfin l'absence ou le manque d'environnements pour effectuer les tests.

Bien que des communautés de test se créent et tentent de définir des bonnes pratiques, chaque équipe reste relativement libre de mettre en œuvre ou non ces bonnes pratiques. Cette situation conduit donc souvent à des solutions composées d'un ensemble d'équipes ayant des niveaux de maturité très inégaux en matière de tests et d'Agilité. Ces équipes ont tendance à travailler en silos et, dans la plupart des cas, il n'existe pas de mécanismes bien définis pour la réalisation des tests inter-équipes.

La construction de scénarios de bout en bout rassemblant les connaissances de chaque équipe ne se fait pas, ce qui entraîne un manque de visibilité sur la solution dans son ensemble et une prédominance des tests exploratoires en l'absence de scénario de test. Ces tests de bout en bout sont menés en général par des équipes transverses qui tentent de saisir la vision métier par des techniques telles que le BDD, et les ateliers dits 3 amigos.

Par ailleurs, la qualité et la réussite des tests inter-équipes dépendent aussi largement du soutien de l'organisation. C'est elle qui fixe les objectifs de chaque équipe, et si ces objectifs ne sont pas sur la qualité, mais uniquement sur la vélocité de production, les équipes ne consacreront pas du temps à la qualité. Cela pourrait expliquer la présence de communautés de test et de bonnes pratiques, mais le manque de cohérence réelle entre les pratiques de test au sein des équipes. L'organisation est également le décideur quant aux moyens à donner à la qualité par le financement de serveurs, de spécialistes de l'automatisation, etc. Sans le soutien général de l'organisation, il est donc difficile pour

les équipes d'apporter des réponses opérationnelles aux tests inter-équipes.

La maîtrise des tests inter-équipes ne repose donc pas sur un seul facteur, mais sur un ensemble. Les organisations doivent donner les moyens aux communautés de test et aux Tests Managers de mettre en place des environnements, des outils et bonnes pratiques au sein des équipes. Ces équipes doivent être challengées non pas uniquement sur la production d'un composant, mais sur la construction d'une solution de qualité. En équilibrant le niveau de maturité des équipes, et en les sensibilisant aux enjeux des tests, les équipes pourront travailler ensemble à la réalisation de scénarios de test de bout en bout.

Il reste donc encore beaucoup de chemin à parcourir pour parvenir à la maîtrise des tests inter-équipes. Pour avancer, il faut trouver les approches et les outils de demain qui permettront de réunir les acteurs impliqués dans les solutions, et de travailler ensemble pour mieux comprendre, construire et vérifier les grands systèmes d'information. Ceci en prenant en considération la nécessité d'acquérir plus de visibilité sur le système dans son ensemble, tout en étant capable de placer ces efforts en test dans des itérations et en abordant les questions relatives à l'automatisation.

Nous pensons que la modélisation des parcours applicatifs et la scénarisation visuelle des tests de bout en bout pourraient offrir un support pour répondre à ces challenges. Aujourd'hui la plus grande difficulté dans la maîtrise des tests inter-équipes porte sur le manque de scénario de bout en bout. Si dans les grands systèmes nous étions en mesure de proposer des modèles autour desquels pourraient se réunir toutes les parties prenantes d'une solution, on pourrait espérer améliorer la communication et la collaboration entre ces différents acteurs. Il serait plus facile d'exprimer les besoins métier, car souvent des schémas sont plus compréhensibles qu'une longue documentation.

Les modèles pourraient avoir différents niveaux d'abstraction. Ils pourraient représenter la solution de manière transverse et aussi unitaire pour chacun des produits qui la compose. La vision transverse apporterait plus de visibilité aux équipes et les impliquerait plus dans le développement de la solution complète que sur le développement de produits individuels.

Cette sensibilisation pourrait être orchestrée par des Tests Managers garants de la vision transverse. Et pour faciliter la conception des tests de bout en bout, des équipes de test transverse pourraient travailler avec les équipes projet à leur conception. Elles profiteraient ainsi de leur expérience sur les produits, et les équipes bénéficieraient de la vision de bout en bout apportée par l'équipe de test transverse. Nous pensons qu'orchestrer de manière visuelle les scénarios de test d'acceptation de bout en bout participera à la validation de la solution métier.

Lors de cette thèse, nous avons eu un premier contexte d'expérimentation de l'approche

ALME dans l'Agilité à l'échelle avec la société Orange [9], mais il nous faudra répliquer et approfondir pour en tirer tous les enseignements. Cela constitue une des perspectives à l'issue de nos travaux de thèse.

III

ÉPILOGUE

7

CONCLUSION ET PERSPECTIVES

7.1/ BILAN DE LA THÈSE

Cette thèse s'inscrit dans le contexte d'une évolution profonde des pratiques des tests logiciels pour répondre aux besoins d'agilité des grandes organisations.

Nous en avons abordé les défis sur 4 plans à partir du paradigme du Model-Based Testing :

- En adaptant l'approche MBT pour en faciliter l'usage par les testeurs fonctionnels et la mise en œuvre en mode itératif et incrémental de développement du logiciel.
- En définissant un continuum d'activités dans l'automatisation des tests de leur conception à leur implémentation.
- En élaborant un processus de refactoring des tests manuels pour répondre au problème de l'obsolescence des référentiels de test.
- En identifiant les problèmes induits par le passage à l'échelle de l'Agilité, en particulier pour les tests inter-équipes.

Notre hypothèse de recherche est que le Model-Based Testing peut apporter une réponse efficiente à ces défis, à condition de revoir l'approche en profondeur : sur la notation de modélisation proposée aux testeurs, sur le cycle d'analyse et de conception des tests, sur leur implantation pour l'automatisation de l'exécution et sur la reprise des cas de test existants.

Le résultat de notre travail de thèse, développé et expérimenté en immersion dans un centre de service en tests logiciels, se traduit par une approche générale, appelée ALME - *Agile Lightweight Model-Based Testing for Enterprise IT* -, qui a été présentée dans la partie technique de cette thèse. Comme cela a été décrit, nos contributions se situent principalement sur le plan méthodologique, c'est à dire portant sur l'organisation et les bonnes pratiques de la mise en œuvre de ce Model-Based Testing revisité. Le contexte

partenarial de la thèse a facilité ce focus sur les bonnes pratiques et les réponses aux questions concrètes des testeurs.

Il est évidemment trop tôt pour savoir si cette évolution du Model-Based Testing conduira à une adoption large par les professionnels des tests dans le domaine du logiciel d'entreprise visé. Nous avons seulement pu constater, sur plusieurs projets, que cette adoption était effective ou en bonne voie.

La partie refactoring de notre travail correspond à un enjeu de taille : il y a beaucoup plus de cas de test à faire évoluer que de nouveaux cas de test à créer. Une approche qui ne répondrait qu'au contexte de nouveau logiciel, et pas au contexte du logiciel existant, passerait à coté d'une grande partie de l'activité des testeurs.

Enfin, la découverte du contexte de l'agilité à l'échelle nous a conduit à répondre au besoin d'une meilleure connaissance, pour la communauté du génie logiciel, de l'état de l'art et des problématiques spécifiques des tests inter-équipes. Cela s'est traduit pas une étude quantitative et qualitative en prise directe avec les praticiens, pour à terme apporter des réponses spécifiques aux problèmes posés.

7.2/ PERSPECTIVES

La définition et l'expérimentation de l'approche ALME ont été réalisées dans des contextes Agiles, mais pas à l'échelle. Il serait donc intéressant d'étudier si l'approche ainsi proposée s'adapte aux contextes à l'échelle et quelles sont les pratiques à ajouter pour répondre aux difficultés actuellement rencontrées dans les tests inter-équipes.

En effet, dans notre analyse qualitative et notre analyse quantitative, nous n'avons pas pu déterminer quelles étaient les approches actuelles qui permettaient de maîtriser les tests inter-équipes. Nous avons établi que les approches basées sur les représentations visuelles, telles que nous les avons proposées avec l'approche ALME, pouvaient soutenir la maîtrise des tests inter-équipes, mais nous n'avons pas eu le temps de le démontrer. De futurs travaux pourraient donc être d'étendre notre approche ALME pour répondre aux besoins en test dans l'Agilité à l'échelle et ainsi élargir son champ d'application.

L'adaptation de l'approche ALME aux contextes à l'échelle pose la question de la gestion des tests automatisés de bout en bout. En contexte Agile, nous avons présenté comment notre approche facilitait le processus d'automatisation des tests. En contexte d'Agilité à l'échelle, il serait intéressant d'étudier s'il en est de même. Comme pour l'approche ALME, le processus d'automatisation devra être complété, en particulier pour permettre une gestion des tests de bout en bout. Là encore, la spécialisation de l'approche ALME, et plus précisément du processus d'automatisation, pourrait contribuer à la maîtrise des tests inter-équipes en facilitant la conception et l'implémentation manuelle et automatisée

des tests de bout en bout.

Concernant le refactoring des suites de tests en langage naturel, de futurs travaux pourraient être d'améliorer le calcul de l'estimation établie afin de donner une estimation plus précise du temps nécessaire au refactoring manuel et au refactoring avec le support outillé en se basant sur nos abaques. Un autre aspect qui n'a pas été étudié est comment la reprise des suites de tests pourrait être incluses dans des itérations Agiles. Au lieu de réaliser le refactoring sur des suites à un point d'obsolescence avéré, on peut étudier une utilisation régulière pour anticiper ce problème. Pour cela, il faudra définir un processus de contrôle et de maintenance régulière des suites de tests qui s'inscrive dans les itérations Agiles.

Enfin, un autre aspect non abordé dans la thèse est l'usage de l'approche et du support outillé du refactoring pour l'analyse des exigences. En effet, notre état de l'art sur le refactoring des suites de tests a montré l'intérêt des travaux existants sur l'analyse des exigences et leur visualisation. De futurs travaux pourraient être d'étudier comment notre approche de refactoring pourrait s'adapter à l'analyse des exigences rédigées en langage naturel.

BIBLIOGRAPHIE

[1] Pekka Abrahamsson, Outi Salo, Jussi Ronkainen, and Juhani Warsta. Agile software development methods : Review and analysis. *arXiv preprint arXiv :1709.08439*, 2017.

[2] A. Afzal, C. L. Goues, M. Hilton, and C. S. Timperley. A study on challenges of testing robotic systems. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 96–107, 2020.

[3] S. Ali and H. Hemmati. Model-Based Testing of Video Conferencing Systems : Challenges, Lessons Learnt, and Results. In *Verification and Validation 2014 IEEE Seventh International Conference on Software Testing*, pages 353–362, March 2014.

[4] Marwah Alian, Dima Suleiman, and Adnan Shaout. Test case reduction techniques-survey. *International Journal of Advanced Computer Science and Applications*, 7(5) :264–275, 2016.

[5] Mashal Alqudah and Rozilawati Razali. A review of scaling agile methods in large software development. *International Journal on Advanced Science, Engineering and Information Technology*, 6(6) :828–837, 2016.

[6] Arilo C. Dias Neto, Rajesh Subramanyan, Marlon Vieira, and Guilherme H. Travassos. A survey on model-based testing approaches : a systematic review. In *Proceedings of the 1st ACM international workshop on Empirical assessment of software engineering languages and technologies : held in conjunction with the 22nd IEEE/ACM International Conference on Automated Software Engineering (ASE) 2007*, pages 31–36. ACM, 2007.

[7] Julian M. Bass. Artefacts and agile method tailoring in large-scale offshore software development programmes. *Information and Software Technology*, 75 :1–16, 2016.

[8] Elodie Bernard, Fabrice Ambert, and Bruno Legeard. Supporting efficient test automation using lightweight MBT. In *2020 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*, pages 84–94. IEEE, 2020.

[9] Elodie Bernard, Claude Barreau, and Fabrice Grimbert. Testing in safe : Coordinate and optimize test efforts with visual atdd - experience report at orange. In *7th User*

Conference on Advanced Automated Testing (UCAAT 2019), Bordeaux, France, oct 2019. https://ucaat.etsi.org/images/Docs/2019/02_Wednesday%2023rd%20Oct/04_SESSION%205/PDF/S5_03_UCAAT%202019_template_Elodie%20Bernard.pdf.

[10] Daniel M. Berry. Evaluation of tools for hairy requirements and software engineering tasks. In *2017 IEEE 25th International Requirements Engineering Conference Workshops (REW)*, pages 284–291. IEEE, 2017.

[11] International Software Testing Qualifications Board. Certified Tester Advanced Level Syllabus : Test Automation Engineer. *Software Testing*, page 84, 2016.

[12] International Software Testing Qualifications Board. Foundation level model-based tester - istqb® international software testing qualifications board. *Software Testing*, 2018.

[13] Aldert Boersma and Erik Vooijs. *Neil's Quest for Quality A TMAP HD STORY*. Sogeti edition, 2014.

[14] Rodrick Borg and Martin Kropp. Automated acceptance test refactoring. In *Proceedings of the 4th Workshop on Refactoring Tools*, pages 15–21. ACM, 2011.

[15] Jan Bosch and Petra Bosch-Sijtsema. Coordination between global agile teams : from process to architecture. In *Agility Across Time and Space*, pages 217–233. Springer, 2010.

[16] Fabrice Bouquet, Christophe Grandpierre, Bruno Legeard, Fabien Peureux, Nicolas Vacelet, and Mark Utting. A subset of precise UML for model-based testing. In *Proceedings of the 3rd international workshop on Advances in model-based testing*, pages 95–104, 2007.

[17] Zhenyu Chen, Baowen Xu, Xiaofang Zhang, and Changhai Nie. A novel approach for test suite reduction based on requirement relation contraction. In *Proceedings of the 2008 ACM symposium on Applied computing*, pages 390–394, 2008.

[18] Michele Chinosi and Alberto Trombetta. BPMN : An introduction to the standard. *Computer Standards & Interfaces*, 34(1) :124–134, 2012. Publisher : Elsevier.

[19] Juyun Cho. A hybrid software development method for large-scale projects : rational unified process with scrum. *Issues in Information Systems*, 10(2) :340–348, 2009.

[20] T. S. Chow. Testing Software Design Modeled by Finite-State Machines. *IEEE Transactions on Software Engineering*, SE-4(3) :178–187, May 1978.

[21] Eliane Figueiredo Collins and Vicente Ferreira de Lucena. Software test automation practices in agile development environment : An industry experience report. In *2012 7th International Workshop on Automation of Software Test (AST)*, pages 57–63. IEEE, 2012.

[22] AEVB Coutinho, Emanuela G. Cartaxo, Patrícia DL Machado, and Campina Grande SPLab-UFCG. Test suite reduction based on similarity of test cases. In *7st Brazilian workshop on systematic and automated software testing—CBSoft*, volume 2013, 2013.

[23] Siddhartha R. Dalal, Ashish Jain, Nachimuthu Karunanithi, J. M. Leaton, Christopher M. Lott, Gardner C. Patton, and Bruce M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st international conference on Software engineering*, pages 285–294, 1999.

[24] Fabiano Dalpiaz, Ivor Van der Schalk, and Garm Lucassen. Pinpointing ambiguity and incompleteness in requirements engineering via information visualization and NLP. In *International Working Conference on Requirements Engineering : Foundation for Software Quality*, pages 119–135. Springer, 2018.

[25] Comité Français des Tests Logiciels. Enquête cftl 2019 sur les pratiques des tests logiciels en france. 2019. https://www.cftl.fr/wp-content/uploads/2020/07/R%C3%A9sultats-de-lenqu%C3%AAte-les-tests-inter-%C3%A9quipes-dans-lAgilit%C3%A9-%C3%A0-l%C3%A9chelle.pdf.

[26] Arilo C. Dias-Neto and Guilherme H. Travassos. A Picture from the Model-Based Testing Area : Concepts, Techniques, and Challenges. In Marvin V. Zelkowitz, editor, *Advances in Computers*, volume 80 of *Advances in Computers*, pages 45–120. Elsevier, January 2010.

[27] Remco M. Dijkman, Marlon Dumas, and Chun Ouyang. Semantics and analysis of business process models in BPMN. *Information and Software technology*, 50(12) :1281–1294, 2008. Publisher : Elsevier.

[28] Kim Dikert, Maria Paasivaara, and Casper Lassenius. Challenges and success factors for large-scale agile transformations : A systematic literature review. *Journal of Systems and Software*, 119 :87–108, September 2016.

[29] Yvonne Dittrich, Michael John, Janice Singer, and Bjørnar Tessem. Editorial : for the special issue on qualitative software engineering research. *Information and software technology*, 49(6) :531–539, 2007.

[30] Meryem Elallaoui, Khalid Nafil, and Raja Touahni. Automatic generation of TestNG tests cases from UML sequence diagrams in Scrum process. In *2016 4th IEEE*

International Colloquium on Information Science and Technology (CiSt), pages 65–70. IEEE, 2016.

[31] David Farago. Model-based Testing in Agile Software Development. page 5, 01 2010.

[32] Hira Farman. Software Testing Basic Principles and Testing Methodologies. *International Journal of Scientific and Engineering Research*, July 2018.

[33] Avinash Gupta, Namita Mishra, and Dharmender Singh Kushwaha. Rule based test case reduction technique using decision table. In *2014 IEEE International Advance Computing Conference (IACC)*, pages 1398–1405. IEEE, 2014.

[34] R. Hametner, D. Winkler, and A. Zoitl. Agile testing concepts based on keyword-driven testing for industrial automation systems. In *IECON 2012 - 38th Annual Conference on IEEE Industrial Electronics Society*, pages 3727–3732, October 2012.

[35] Zhong Hai He, Xiang Zhang, and Xiang Yin Zhu. Design and Implementation of Automation Testing Framework Based on Keyword Driven. In *Applied Mechanics and Materials*, volume 602, pages 2142–2146. Trans Tech Publ, 2014.

[36] Siw Elisabeth Hove and Bente Anda. Experiences from conducting semi-structured interviews in empirical software engineering research. In *11th IEEE International Software Metrics Symposium (METRICS'05)*, pages 10–pp. IEEE, 2005.

[37] Marta Indulska, Jan Recker, Peter Green, and Michael Rosemann. Are we there yet ? seamless mapping of bpmn to bpel4ws. Association for Information Systems, 2007.

[38] Paul C. Jorgensen. *The Craft of Model-Based Testing*. CRC Press, 2017.

[39] Mika Katara and Antti Kervinen. Making model-based testing more agile : a use case driven approach. In *Haifa Verification Conference*, pages 219–234. Springer, 2006.

[40] Saif Ur Rehman Khan, Sai Peck Lee, Raja Wasim Ahmad, Adnan Akhunzada, and Victor Chang. A survey on Test Suite Reduction frameworks and tools. *International Journal of Information Management*, 36(6) :963–975, 2016. Publisher : Elsevier.

[41] Mateja Kocbek, Gregor Jošt, Marjan Heričko, and Gregor Polančič. Business process model and notation : The current state of affairs. *Computer Science and Information Systems*, 12(2) :509–539, 2015.

[42] Harry Koehnemann and Mark Coats. Experiences applying agile practices to large systems. In *2009 Agile Conference*, pages 295–300. IEEE, 2009.

[43] Tim Koomen, Bart Broekman, Michiel Vroon, and Leo van der Aalst. *TMap Next, for result-driven testing Hardcover*. Utn publishers edition, July 2006.

[44] Nikolai Koudelia. Acceptance Test-Driven Development. *University of Jyvaskyla department of meathematical information technology*, Master's Thesis in Information Technology (Software engineering) :102, 2011. https://jyx.jyu.fi/bitstream/handle/123456789/37392/URN:NBN:fi:jyu-201202161200.pdf?sequence=1.

[45] Anne Kramer, Robert V. Binder, and Bruno Legeard. 2016 / 2017 Model-based Testing User Survey : Results. 2016. https://www.cftl.fr/wp-content/uploads/2017/02/2016-MBT-User-Survey-Results.pdf.

[46] Anne Kramer and Bruno Legeard. 2019 Model-based Testing User Survey : Results. 2019. https://www.cftl.fr/wp-content/uploads/2020/02/2019-MBT-User-Survey-Results.pdf.

[47] Jacob Krüger, Mustafa Al-Hajjaji, Sandro Schulze, Gunter Saake, and Thomas Leich. Towards Automated Test Refactoring for Software Product Lines. In *Proceedings of the 22Nd International Systems and Software Product Line Conference - Volume 1*, SPLC '18, pages 143–148, New York, NY, USA, 2018. ACM.

[48] Craig Larman and Bas Vodde. Scaling agile development. *CrossTalk*, 9 :8–12, 2013.

[49] B. Legeard and A. Bouzy. Smartesting CertifyIt : Model-Based Testing for Enterprise IT. In *Verification and Validation 2013 IEEE Sixth International Conference on Software Testing*, pages 391–397, March 2013.

[50] Henrik Leopold, Jan Mendling, and Oliver Günther. What we can learn from Quality Issues of BPMN Models from Industry. *IEEE Software*, 33, March 2015.

[51] Maurizio Leotta, Diego Clerissi, Filippo Ricca, and Paolo Tonella. Capture-replay vs. programmable web testing : An empirical assessment during test case evolution. In *2013 20th Working Conference on Reverse Engineering (WCRE)*, pages 272–281, Koblenz, Germany, October 2013. IEEE.

[52] Frank Leymann. BPEL vs. BPMN 2.0 : Should you care ? In *International Workshop on Business Process Modeling Notation*, pages 8–13. Springer, 2010.

[53] Wenbin Li, Franck Le Gall, and Naum Spaseski. A survey on model-based testing tools for test case generation. In *International Conference on Tools and Methods for Program Analysis*, pages 77–89. Springer, 2017.

[54] Garm Lucassen, Fabiano Dalpiaz, Jan Martijn EM van der Werf, and Sjaak Brinkkemper. Improving agile requirements : the quality user story framework and tool. *Requirements Engineering*, 21(3) :383–403, 2016. Publisher : Springer.

[55] Renate Löffler, Baris Güldali, and Silke Geisen. Towards model-based acceptance testing for Scrum. *Softwaretechnik-Trends, GI*, 2010.

[56] Raluca Marinescu, Cristina Seceleanu, Hèléne Le Guen, and Paul Pettersson. A research overview of tool-supported model-based testing of requirements-based designs. In *Advances in Computers*, volume 98, pages 89–140. Elsevier, 2015.

[57] C. Medoh and A. Telukdarie. Business process modelling tool selection : A review. In *2017 IEEE International Conference on Industrial Engineering and Engineering Management (IEEM)*, pages 524–528, December 2017. ISSN : 2157-362X.

[58] Atif Memon. *Advances in Computers, Volume 101*. Academic Press, Inc., Orlando, FL, USA, 1st edition, 2016.

[59] Jan Mendling. Business process execution language for web service (bpel). In *EMISA Forum*, volume 26, pages 5–8, 2006. Issue : 2.

[60] Jan Mendling, Hajo A. Reijers, and Wil MP van der Aalst. Seven process modeling guidelines (7PMG). *Information and Software Technology*, 52(2) :127–136, 2010. Publisher : Elsevier.

[61] Michael Mlynarski, Baris Güldali, Stephan Weißleder, and Gregor Engels. Model-based testing : achievements and future challenges. In *Advances in Computers*, volume 86, pages 1–39. Elsevier, 2012.

[62] Myint Myint Moe. Comparative Study of Test-Driven Development (TDD), Behavior-Driven Development (BDD) and Acceptance Test–Driven Development (ATDD). *International Journal of Trend in Scientific Research and Development*, pages 231–234, 2019.

[63] Nils Brede Moe, Daniela Cruzes, Tore Dyb\a a, and Edda Mikkelsen. Continuous software testing in a globally distributed project. In *Global Software Engineering (ICGSE), 2015 IEEE 10th International Conference on*, pages 130–134. IEEE, 2015.

[64] Adel Hamdan Mohammad and Tariq Alwada'n. Agile software methodologies : Strength and weakness. *International Journal of Engineering Science and Technology*, 5(3) :455, 2013.

[65] Michael zur Muehlen and Jan Recker. How Much Language Is Enough ? Theoretical and Practical Use of the Business Process Modeling Notation. In Zohra Bellahsène and Michel Léonard, editors, *Advanced Information Systems Engineering*, Lecture Notes in Computer Science, pages 465–479, Berlin, Heidelberg, 2008. Springer.

[66] Anastasija Nikiforova and Janis Bicevskis. Towards a Business Process Model-based Testing of Information Systems Functionality.

[67] Chun Ouyang, Wil MP van der Aalst, Marlon Dumas, and Arthur HM Ter Hofstede. Translating bpmn to bpel. 2006. Publisher : BPM Center.

[68] Tuomas Pajunen, Tommi Takala, and Mika Katara. Model-based testing with a general purpose keyword-driven test automation framework. In *2011 IEEE Fourth International Conference on Software Testing, Verification and Validation Workshops*, pages 242–251. IEEE, 2011.

[69] Harrie Passier, Lex Bijlsma, and Christoph Bockisch. Maintaining Unit Tests During Refactoring. In *Proceedings of the 13th International Conference on Principles and Practices of Programming on the Java Platform : Virtual Machines, Languages, and Tools*, page 18. ACM, 2016.

[70] Leandro Sales Pinto, Saurabh Sinha, and Alessandro Orso. Understanding Myths and Realities of Test-suite Evolution. In *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, FSE '12, pages 33 :1–33 :11, New York, NY, USA, 2012. ACM.

[71] Michael Puleio. How not to do agile testing. In *AGILE 2006 (AGILE'06)*, pages 7–pp. IEEE, 2006.

[72] Dudekula Mohammad Rafi, Katam Reddy Kiran Moses, Kai Petersen, and Mika V. Mäntylä. Benefits and limitations of automated software testing : Systematic literature review and practitioner survey. In *2012 7th International Workshop on Automation of Software Test (AST)*, pages 36–42. IEEE, 2012.

[73] Mazedur Rahman and Jerry Gao. A reusable automated acceptance testing architecture for microservices in behavior-driven development. In *2015 IEEE Symposium on Service-Oriented System Engineering (SOSE)*, pages 321–325. IEEE, 2015.

[74] C.V. Ramamoorthy, S.-B.F. Ho, and W.T. Chen. On the Automated Generation of Program Test Data. *IEEE Transactions on Software Engineering*, SE-2(4) :293–300, December 1976.

[75] Kuda Nageswara Rao, G. Kavita Naidu, and Praneeth Chakka. A study of the Agile software development methods, applicability and implications in industry. *International Journal of Software Engineering and its applications*, 5(2) :35–45, 2011.

[76] Jan Recker. Opportunities and constraints : the current struggle with BPMN. *Business Process Management Journal*, 2010. Publisher : Emerald Group Publishing Limited.

[77] Sandeep Reddivari, Shirin Rad, Tanmay Bhowmik, Nisreen Cain, and Nan Niu. Visual requirements analytics : a framework and case study. *Requirements engineering*, 19(3) :257–279, 2014. Publisher : Springer.

[78] Benedetta Rosadini, Alessio Ferrari, Gloria Gori, Alessandro Fantechi, Stefania Gnesi, Iacopo Trotta, and Stefano Bacherini. Using NLP to detect requirements defects : An industrial experience in the railway domain. In *International Working Conference on Requirements Engineering : Foundation for Software Quality*, pages 344–360. Springer, 2017.

[79] Bernhard Rumpe. Agile test-based modeling. *arXiv preprint arXiv :1409.6616*, 2014.

[80] Ken Schwaber. *Agile project management with Scrum*. Microsoft press, 2004.

[81] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4) :557–572, 1999. Publisher : IEEE.

[82] Carolyn B. Seaman. Qualitative methods in empirical studies of software engineering. *IEEE Transactions on software engineering*, 25(4) :557–572, 1999. Publisher : IEEE.

[83] Sandeep Sivanandan. Agile development cycle : Approach to design an effective Model Based Testing with Behaviour driven automation framework. In *20th Annual International Conference on Advanced Computing and Communications (ADCOM)*, pages 22–25. IEEE, 2014.

[84] Monique Snoeck, Isel Moreno-Montes de Oca, Tom Haegemans, Bjorn Scheldeman, and Tom Hoste. Testing a selection of BPMN tools for their support of modelling guidelines. In *IFIP Working Conference on The Practice of Enterprise Modeling*, pages 111–125. Springer, 2015.

[85] Sean Stolberg. Enabling agile testing through continuous integration. In *2009 agile conference*, pages 369–374. IEEE, 2009.

[86] Megan Sumrell. From waterfall to agile-how does a QA team transition. In *Proceedings of the Agile Conference (AGILE)*, pages 291–295, 2007.

[87] David Talby, Arie Keren, Orit Hazzan, and Yael Dubinsky. Agile software testing in a large-scale project. *IEEE software*, 23(4) :30–37, 2006. Publisher : IEEE.

[88] Haruto Tanno and Xiaojing Zhang. Test script generation based on design documents for web application testing. In *2015 IEEE 39th Annual Computer Software and Applications Conference*, volume 3, pages 672–673. IEEE, 2015.

[89] Masoumeh Taromirad and Raman Ramsin. MBT in agile/lightweight processes : a process-centred review. *IET Software*, 13(5) :327–337, 2019. Publisher : IET.

[90] Suresh Thummalapenta, Saurabh Sinha, Nimit Singhania, and Satish Chandra. Automating test automation. In *2012 34th International Conference on Software Engineering (ICSE)*, pages 881–891. IEEE, 2012.

[91] Dan Turk, Robert France, and Bernhard Rumpe. Limitations of agile software processes. *arXiv preprint arXiv :1409.6600*, 2014.

[92] Mark Utting, Alexander Pretschner, and Bruno Legeard. A taxonomy of model-based testing approaches. *Software Testing, Verification and Reliability*, 22(5) :297–312, 2012.

[93] Aashish Vaidya. Does dad know best, is it better to do less or just be safe ? adapting scaling agile practices into the enterprise. *PNSQC. ORG*, pages 1–18, 2014.

[94] CollabNet VersionOne. 13th annual state of agile report. Technical report, 2019.

[95] Leonardo Villalobos-Arias, Christian Quesada-López, Alexandra Martinez, and Marcelo Jenkins. Model-based testing areas, tools and challenges : A tertiary study. *CLEI Electronic Journal*, 22(1), 2019.

[96] Stephen A. White. Introduction to BPMN. *Ibm Cooperation*, 2(0) :0, 2004.

[97] Laurie Williams and Alistair Cockburn. Agile software development : it's about feedback and change. *IEEE computer*, 36(6) :39–43, 2003.

[98] Arezoo Yazdani Seqerloo, Mohammad Javad Amiri, Saeed Parsa, and Mahnaz Koupaee. Automatic test cases generation from business process models. *Requirements Engineering*, July 2018.

[99] S. Yoo and M. Harman. Regression testing minimization, selection and prioritization : a survey. *Software Testing, Verification and Reliability*, pages n/a–n/a, 2010.

[100] Michael zur Muehlen and Danny T. Ho. Service process innovation : a case study of BPMN in practice. In *Proceedings of the 41st Annual Hawaii International Conference on System Sciences (HICSS 2008)*, pages 372–372. IEEE, 2008.

IV

ANNEXES

A

TERMINOLOGIE

A.1/ GLOSSAIRE DES TERMES ALME

Ce glossaire propose la définition des termes et acronymes utilisés dans l'approche ALME.

Action / action de test – Une action de test est une stimulation réalisée sur le système sous test.

ALME – Agile Lightweight Model-Based Testing – Une approche du Model-Based Testing destinée à être mis en œuvre au sein de projets de développement logiciel en Agile pour les tests fonctionnels des applications des Systèmes d'Information d'entreprise.

ATDD visuel – Acceptance Test Driven Development – Réalisé en utilisant la scénarisation visuelle des tests d'acceptation via une modélisation graphique sous la forme de flots d'activités.

Cas de test de non régression / Test de non-régression– Un cas de test de non-régression traite un aspect fonctionnel précis de l'application, mais en étant plus transverses dans le contrôle de la fonctionnalité, et avec des cas de test moins détaillés que pour un cas de test fonctionnel.

Cas de test fonctionnel / Test fonctionnel– Un cas de test fonctionnel traite un aspect fonctionnel précis de l'application.

Donnée de conception – Les données de conception de test sont utilisées dans les modèles. Leurs valeurs sont abstraites.

Donnée d'implémentation – Les données d'implémentation de test sont utilisées dans des tables. Leurs valeurs sont concrètes.

Étape de test – Une étape de test regroupe une action de test et un résultat attendu.

Fonction de test – Une fonction de test est une étape de test existant plusieurs fois dans le même cas de test ou partagée entre plusieurs cas de test.

Mots clés – Un mot-clé est une combinaison d'une ou de plusieurs actions sur un système sous test. L'implémentation d'un mot-clé permet l'exécution d'un script de test.

Parcours applicatifs – Dans l'approche ALME, nous appelons « parcours applicatif » une modélisation sous la forme d'un flot d'activités visant la scénarisation de cas de test.

Refactoring – On appelle « refactoring » dans cette thèse l'activité consistant à modifier un cas de test manuel, écrit en langage naturel, pour faire apparaître des étapes de test et des paramètres de ces étapes de test, destinés à être utilisée dans plusieurs cas de test.

Scénarisation visuelle des tests – Il s'agit des activités de conception abstraite des tests fondée sur la modélisation des parcours applicatifs à tester sous la forme de workflows.

Script de tests automatisés – Un script de tests automatisés regroupe un ensemble de mots-clés permettant de stimuler automatiquement des actions sur le système sous test.

Suites de tests fonctionnels – Les suites de tests fonctionnelles regroupent des cas de test sur un aspect fonctionnel précis de l'application, avec un niveau de détail élevé dans les cas de test.

Suites de tests de non-régression – Les suites de tests de non-régression regroupent des cas de test de non-régression sur un aspect fonctionnel précis de l'application, mais en étant plus transverses dans le contrôle de la fonctionnalité, et avec des cas de test moins détaillés que pour les suites de tests fonctionnels.

Workflow – Un workflow est une modèlisation graphique de flot d'activités utilisée dans l'approche ALME la scénarisation visuelle des tests.

A.2/ GLOSSAIRE DES TERMES DES TESTS LOGICIELS UTILISÉS DANS LA THÈSE

Cette section propose la définition de termes des tests logiciels utiles pour la thèse, à partir de la norme ISO/IEEE/IEC 29119-1 [1] ou du glossaire de l'ISTQB [2]. La norme ISO/IEEE/IEC 29119-1 constitue le standard international pour les tests logiciels. Elle contient plusieurs documents portant sur la terminologie, le processus, les techniques et l'automatisation. Le document sur lequel est fondé le présent glossaire est celui sur la terminologie des tests logiciels, nommé 29119-1. Il a été révisé en 2020. Le glossaire de l'ISTQB constitue une référence dans la profession et est plus complet que la norme ISO 29119-1. Lorsqu'un terme n'apparaît pas dans la norme ISO, nous utilisons alors la définition de l'ISTQB.

Cas de test – Un cas de test est un ensemble de pré-requis, d'entrée (y compris des actions, le cas échéant) et de résultats attendus, élaboré pour guider l'exécution d'un test afin d'atteindre les objectifs de test. (Source ISO)

Cas de test abstrait / Test abstrait – Un cas test avec des conditions préalables abstraites, des données d'entrée, des résultats attendus, des conditions postérieures et des actions (le cas échéant). (Source ISTQB)

Un cas de test abstrait manipule des données abstraites(données de conception) et il n'est pas directement exécutable sur le système.

Cas de test concret / Test concret – Un cas test avec des valeurs concrètes pour les conditions préalables, les données d'entrée, les résultats attendus et les conditions ultérieures et une description détaillée des actions (le cas échéant).(Source ISTQB)

1. https ://www.iso.org/fr/standard/45142.html
2. https ://glossary.istqb.org/en/search/

Un cas de test concret manipule des données concrètes (données d'implémentation) et il est exécutable sur le système.

Condition de test – Aspect testable d'un composant ou d'un système, tel qu'une fonction, une transaction, une caractéristique, un attribut de qualité ou un élément structurel identifié comme base de test. (Source ISO)

Objectif de test – L'objectif de test est la raison pour laquelle le test est effectué. (Source ISO)

Résultat attendu – Un résultat attendu est la description d'éléments issus de l'état du système après la réalisation d'une action sur celui-ci. Généralement, les éléments mentionnés sont ceux qui devraient avoir subit une modification du fait de l'action.(Source ISO)

Solution d'automatisation Réalisation/mise en œuvre d'une architecture d'automatisation des tests, c'est-à-dire une combinaison de composants mettant en œuvre une mission spécifique d'automatisation des tests. Les composants peuvent inclure des outils de test commerciaux, des cadres d'automatisation des tests, ainsi que du matériel de test. (Source ISTQB)

Suite de tests – Une suite de tests est un ensemble de cas de test ou de procédures de tests. (Source ISO)

Système sous test – Le terme "système sous test" décrit un produit de travail qui est l'objet des tests. (Source ISTQB)

B

ANNEXE CHAPITRE 3

Cette annexe a pour objectif d'apporter du détail sur la seconde expérimentation menée dans le chapitre 3. Des compléments notamment au niveau des réalisations sont apportés pour les phases de découverte et formulation visuelle des parcours applicatifs.

B.1/ EXPÉRIMENTATION 1

Cette section apporte du détail sur l'expérimentation 1 du chapitre 3.

B.1.1/ DÉCOUVERTE ET FORMULATION VISUELLE DES PARCOURS APPLICATIFS

La figure B.1 présente une représentation visuelle d'un ensemble d'US pour une itération. Chaque sous-parcours représente une US, il y a donc 9 US représentées dans cette figure. Cette représentation avait pour objectif de décrire les différents besoins métier lier aux "actions". Dans notre contexte, "les actions" sont des actions à réaliser pour des agents, comme par exemple suivre une formation, participer à un atelier de spécialisation, etc. Elles nécessitent donc un pilotage qui peut être fait par différents profils. Les profils ont été anonymisés en des profils A1 ou A2. Visuellement, les actions réalisables par chaque profil sont coloriées en bleu pour le profil A1, en violet pour le profil A2, et en jaune pour les activités identiques aux deux profils. Ces deux profils peuvent donc afficher le pilotage des actions et leurs listes dans un tableau. Ces activités sont séparées pour chacun de ces profils dans la représentation graphique, car les éléments affichés ne sont pas les mêmes selon le profil. Pour la création par contre le mécanisme est le même d'où le fait que l'action est commune sur le modèle. Ensuile, seul le profil A2 peut associer un point de veille à une action, des compétences et accepter ou refuser une action. Pour finir, les profils A1 et A2 peuvent clôturer et ré-ouvrir une action. Ce type de représentation visuelle regroupant les US était réalisé en début de sprint par les testeuses fonctionnels et l'analyste métier, et était partagé avec les parties prenantes de la

solution. Cela permettait notamment une bonne coordination entre testeur et développeur pour organiser les développements au mieux pour permettre le test au plus tôt.

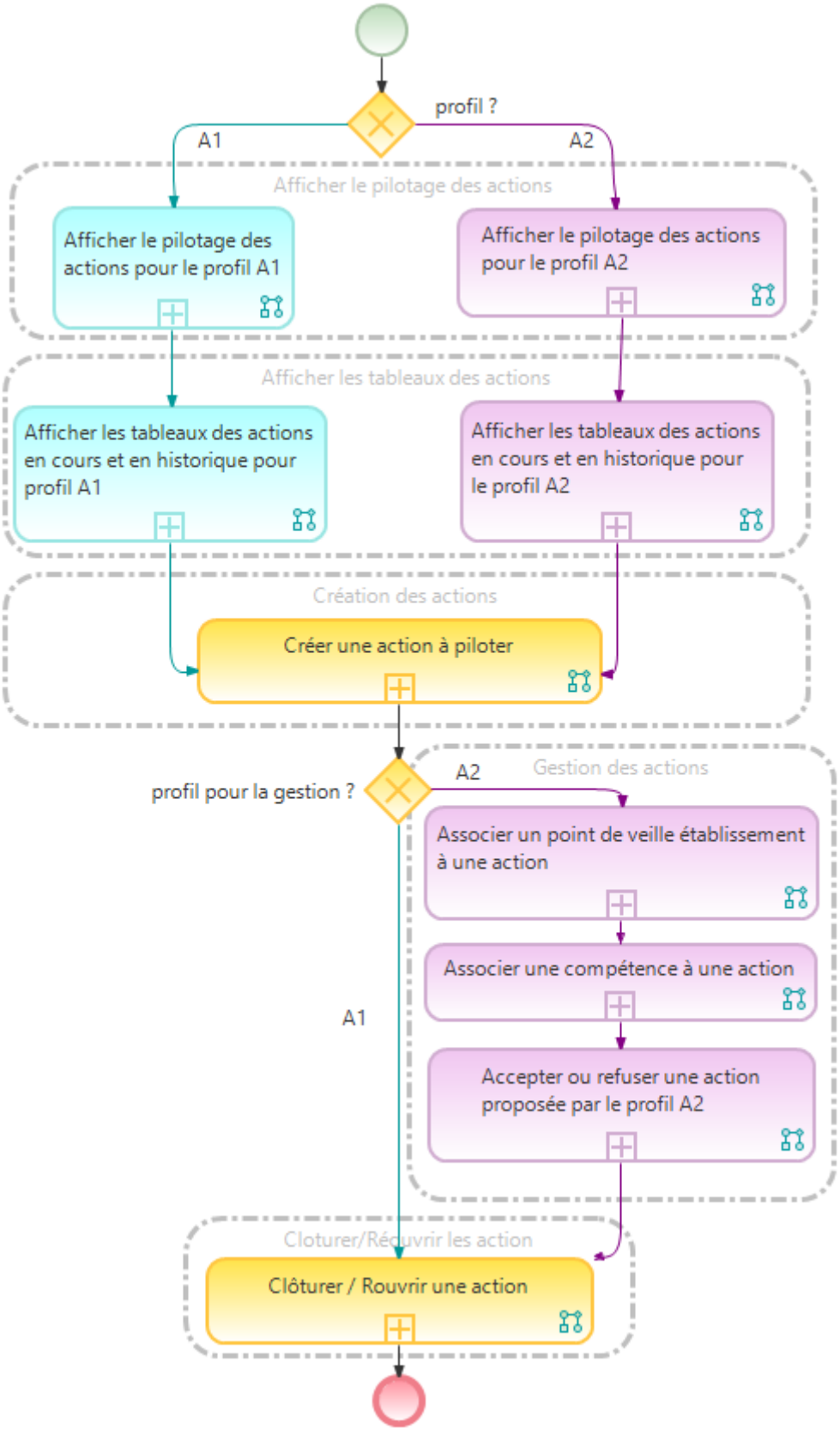


FIGURE B.1 – Exemple de représentation visuelle pour le contexte 2

C

ANNEXE CHAPITRE 4

Cette annexe a pour objectif d'apporter du détail sur les deux expérimentations menées dans le chapitre 4. Des compléments notamment au niveau des réalisations sont apportés pour les phases de découverte et formulation visuelle des parcours applicatifs, de conception détaillée des tests, d'implémentation des tests, de la complétion de la couche d'adaptation et de la production de scripts.

C.1/ EXPÉRIMENTATION 1

Cette section apporte du détail sur l'expérimentation 1 du chapitre 4.

C.1.1/ DÉCOUVERTE ET FORMULATION VISUELLE DES PARCOURS APPLICATIFS

Durant la phase de découverte et de formulation visuelle des parcours applicatifs, nous avons créé trois parcours applicatifs principaux : que nous appelons parcours A, B et C. Afin de mettre en valeur les comportements communs existant entre chacun de ces 3 parcours principaux, nous avons représenté dans la figure C.1 un aperçu de la façon dont chacun des parcours principaux enchaîne des sous-parcours communs et indépendants. En bleu apparaissent les sous-parcours communs, en vert les sous-parcours indépendants liés au parcours principal A, en orange ceux du parcours principal B et en violet ceux du parcours principal C.

Ces parcours décrivent le processus d'évaluation avec des changements pour chaque objectif de test. Après la phase de connexion, de préparation de l'évaluation, le choix de la zone d'évaluation et de l'agent, chaque parcours principal passe par une phase d'évaluation d'un point clé. Les parcours A et C ne font qu'évaluer un point clé et se termine. Ils ont chacun une manière indépendante d'évaluer leurs points clés respectifs et n'entraînent pas comme pour le parcours B le contrôle de l'axe de progrès et de la synthèse.

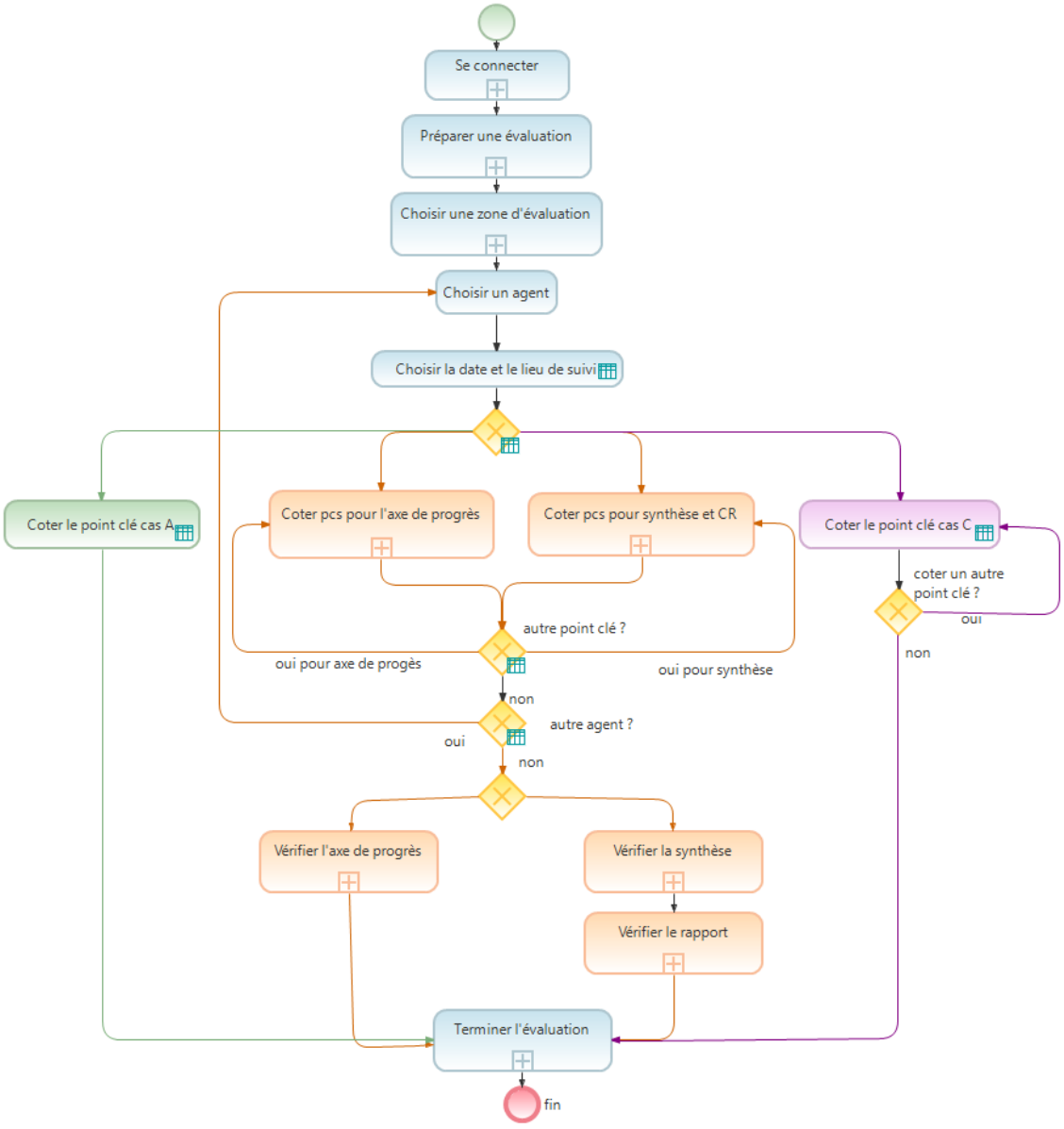


FIGURE C.1 – Aperçu du contenu des parcours principaux

La plupart des sous-parcours communs ne contiennent qu'une seule tâche et sont dédiés à une fonctionnalité spécifique de l'application, traitant d'un cas passant. En effet, chaque fois qu'un comportement (par exemple la connexion) est utilisé plus d'une fois pour les tests, nous créons un nouveau parcours et nous appelons ce nouveau parcours là où il est nécessaire. La création d'un parcours d'une tâche prend moins de cinq minutes et la même chose pour effectuer un changement.

C.1.2/ CONCEPTION DÉTAILLÉE DES TESTS

Pour la conception détaillée des tests les exigence à traiter étaient qu'un type d'évaluation S.A.M.I autorise les notations S, A, M et I, tandis qu'un type de notation E.F.G autorise les notations E, F et G. Pour chaque type d'évaluation, nous vérifions que les notations appropriées sont visibles et sélectionnables. Une fois la conception terminée nous avons généré les cas de test.

Dans la figure C.1, le parcours principal A, qui consiste à tester l'évaluation de la manière la plus élémentaire possible, génère un seul cas test en 10 étapes. Il n'utilise aucune donnée particulière, d'où la production d'un seul cas de test. L'objectif de ce cas test était de produire un scénario passant simple, mais de le variabiliser au niveau des scripts automatisés pour vérifier le bon fonctionnement de l'application avec des combinaisons de différents profils, métiers, groupes de veille, etc. Ainsi ce cas de test était exécuté un peu plus de 80 fois. L'objectif de test étant finalement de faire une variabilisation au niveau des données pour tester un grand nombre de cas, il ne contient qu'une seule donnée de conception, car c'était la seule nécessaire pour effectuer une évaluation de base. En revanche, il utilise 11 données d'implémentation dans ses mots-clés afin de permettre l'exécution des scripts. Ces données sont nécessaires à l'exécution, car il est obligatoire d'effectuer des saisies sur les différentes interfaces, mais leurs valeurs n'influencent pas le comportement métier attendu. Au total, le script était variabialisé avec 80 combinaisons différentes.

Le parcours principal B génère deux cas de test de 55 et 61 étapes. En effet, deux cas complexes sont vérifiés : d'une part, l'avancement global et, d'autre part, une synthèse. Pour ce faire uniquement des données de conception ont été utilisées (et des données propres aux mots clés, comme le Webdriver) et aucune donnée d'implémentation. Ceci s'explique par le fait que l'ensemble des données à saisir lors de la phase d'évaluation devait être vérifié dans l'avancement global et la synthèse. Ces cas étant très ciblés ils ne sont exécutés qu'une fois avec les données issues de la conception.

Pour terminer le parcours principal C génère six cas de test comportant en moyenne 17 étapes chacun. Cela s'explique par l'utilisation d'une approche combinatoire des notations et des groupes de veille. Dans le cas C, nous effectuons des actions quasi similaires

à celles du cas A, sauf que nous produisons 6 cas de test, contrairement au cas A où un seul cas de test est produit. Le parcours C reprend les mêmes données d'implémentation que le parcours A à l'exception des types de cotation qui sont utilisées comme donnée de conception et non d'implémentation. Ainsi, c'est le contrôle des différents types de cotation qui fait varier le nombre de cas de test produits. En effet, dans le cas C, on vérifie la possibilité de coter les agents sur les 6 types de cotations possibles, soit 6 cas de test. Le nombre de steps est plus grand ici que sur le parcours A car pour chaque type de cotation, un ensemble de notation est possible et l'on teste chacune d'elle. Par exemple pour une cotation de type A.B.C.D, on choisi la notation A, puis B, puis C et enfin D pour vérifier qu'il est possible pour chaque cotation de choisir l'ensemble des notations qui la compose. Dans le parcours principal A, on choisit par défaut une cotation et sa plus haute notation.

C.1.3/ COMPLÉTION DE LA COUCHE D'ADAPTATION

Pour la construction du dictionnaire de mots-clés, nous avons entre autres des fonctions pour choisir l'agent à évaluer nommé "choisirAgentAEvaluer" en prenant comme paramètre le Webdriver et le nom de l'agent à évaluer, "selectionnerPointCleEvaluation" en prenant comme paramètre le Webdriver et le nom du point clé à évaluer. Comme recommandé, nos mots clés sont nommés au plus près du comportement métier attendu afin de faciliter la complétion de la couche d'adaptation par le testeur fonctionnel et de permettre une meilleure lecture des scripts ainsi que le respect de la vision métier. Il existe également des mots clés de granularité plus élevée, tels que "effectuerNotationPointsObserves" que nous appelons dans une fonction de granularité faible.

Au sujet de la construction des mots-clés, un extrait de code du mot-clé C.2 "effectuerCotation" est présenté. Lorsque le testeur fonctionnel choisit la cotation "A", un clic sur le bouton correspondant au choix de la cotation "A" est effectué dans l'application via la ligne "Evaluation.divCotationA(driver).click() ;".

Le codage des mots clés (1400 lignes de code) et la construction du répertoire d'objets représentent 60% du processus d'automatisation. Ce pourcentage et le nombre élevé de lignes de code sont dus à la difficulté de créer un code Sélénium stable et à la grande diversité des comportements possibles de l'application. L'instabilité du code Selenium se produit dans les cas où les identifiants ne pouvaient pas être inclus dans le code. Cela a conduit à l'écriture de fonctions qui parcourent le HTML en utilisant certaines balises telles que les balises "div". L'écriture de ce code est assez fastidieuse, car elle nécessite un bon chemin du DOM pour permettre une maintenance plus facile du code d'automatisation. De plus, la lenteur d'exécution de l'application entraîne souvent des temps d'attente plus ou moins longs pour éviter que les scripts échouent.

```
public static void effectuerCotation(String ChoixCotation,WebDriver driver) {
    if (ChoixCotation.equals("A")) {
        if (Evaluation.divCotationA(driver).getAttribute( s: "style").isEmpty()) {
            Evaluation.divCotationA(driver).click();
            System.out.println("Cotation : A");
            Fonctions.sleepATime(CONSTANT_SHORT_TIME);
            if(Evaluation.boutonOui(driver) != null){
                Evaluation.boutonOui(driver).click();
                Fonctions.sleepATime(CONSTANT_VERYSHORT_TIME);
            }
        }
    }
```

FIGURE C.2 – Exemple de code d'un mot-clé

Ensuite, la diversité de l'application était essentiellement axée sur la variabilité des éléments qu'elle présentait. Pour la réalisation d'une évaluation, le choix d'un agent conduit à l'affichage de points clés particuliers avec des choix de notation particuliers, avec des points observés spécifiques au point clé choisi et aussi des notations particulières. Selon les cotations, l'affichage n'est pas le même. Le code d'automatisation a donc dû s'adapter à ces différents affichages afin de ne pas échouer.

C.2/ EXPÉRIMENTATION 2

Cette section apporte du détail sur l'expérimentation 2 du chapitre 4.

C.2.1/ DÉCOUVERTE ET FORMULATION VISUELLE DES PARCOURS APPLICATIFS

Dans cette expérimentation des parcours de vérifications ciblés ou de contrôle de cas passants et non passants ont été construits. La figure C.3 présente la différence entre ces parcours.

On retrouve à droite le parcours de complétion du besoin pour le cas passant nominal, où on effectue les saisies nécessaires à la complétion de l'étape sans effectuer de vérifications ciblées. À gauche, le parcours ciblé pour la complétion du besoin. Ces deux parcours partagent des étapes communes comme ”Entrer le nom du devis” et d'autres saisies.

La réalisation de différents parcours pour exprimer le même comportement métier a facilité la gestion des parcours de bout en bout. En effet, dans un premier temps la testeuse fonctionnelle a essayé de mutualiser les cas passants et non passants au sein d'un même

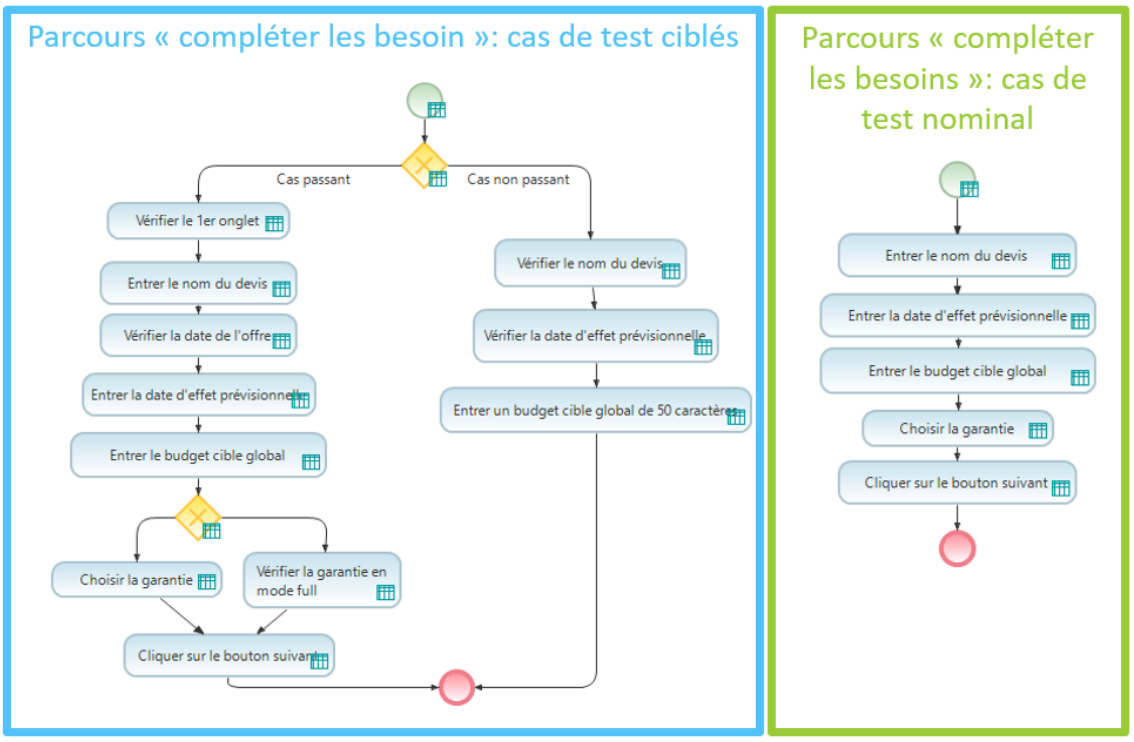


FIGURE C.3 – Comparaison entre les parcours ”compléter les besoins” ciblé vs nominal

parcours pour les cas ciblés et nominaux, mais a rencontré des difficultés pour générer les tests de bout en bout. Ces difficultés étaient dues à la gestion de la cohabitation entre les cas passants et les cas d'erreur. En effet, dans notre expérimentation, 6 modèles sont liés entre eux. Chacun d'eux manipule des données communes, mais vérifie des propriétés différentes. La complexité avec les cas d'erreur est que si par exemple le 1er modèle vérifie 2 cas d'erreur avec 2 données différentes (données A = 1 et données B = 2), si le second modèle a besoin des données précédentes, mais avec valeurs différentes (données A = 2 et données B = 1), alors la génération des cas de test échouera dans le second modèle. Cela s'explique par le fait que les valeurs des données définies dans le premier modèle ne couvrent pas les contraintes de valeurs attendues par le second. Il est alors nécessaire de réussir à faire cohabiter les cas passants et non passants pour permettre la génération des tests. Cela est possible, mais cela entraîne une grande complexité au niveau des tables et des représentations visuelles, d'où le choix de séparer les parcours comme préconisé, afin de gagner du temps et faciliter la maintenance.

Grâce à cette segmentation, il a été facile de construire des cas de test de non-régression et de couvrir de multiples fonctionnalités. La représentation visuelle des parcours applicatifs a permis de mieux comprendre les exigences métier. En effet, chaque parcours étant représenté par une séquence visuelle de tâches simples, telles que ”Saisir un chiffre d'affaires”, puis ”Contrôler le montant de la franchise”, il était simple de représenter toutes les étapes lors de la signature d'un contrat.

C.2.2/ CONCEPTION DÉTAILLÉE DES TESTS

Les exigences à couvrir lors de la conception détaillée des tests concernaient le chiffre d'affaire. Selon le chiffre d'affaires, une franchise différente s'applique, d'où le choix de faire figurer cette donnée dans les parcours applicatifs en tant que donnée de conception. Dans notre cas, 90% des données sur le périmètre choisi étaient nécessaires aux tests. Et nous n'avons donc pas inclus les 10% de données liées à la récupération d'informations client et sans effet sur le processus de souscription.

Afin de couvrir toutes les exigences, 36 cas de test ont été générés par l'outil, 16 sont issus des parcours ciblés pour vérifier dans le détail les exigences et 10 sont issus des parcours passants nominaux pour vérifier des comportements métier de bout en bout. Il est normal de trouver plus de cas de test issus des parcours ciblés que des parcours nominaux, car contrairement aux cas passants, les cas de test ciblés traitent des cas d'erreurs. Les 10 cas de test produits pour les cas nominaux sont le nombre minimum pour couvrir l'ensemble des propriétés que nous souhaitons vérifier. C'est à dire la variabilisation des chiffres d'affaires ainsi que les profils pouvant réaliser ces actions métier. Ainsi l'outil permet de générer un nombre "minimum" de cas de test afin de favoriser une couverture maximale des exigences. C'est un avantage important pour faciliter l'automatisation en limitant le nombre de cas de test à automatiser et à maintenir par la suite.

C.2.3/ IMPLÉMENTATION DES TESTS

L'implémentation des tests se compose de la définition de jeux de données. La figure 4.9 présente un ensemble de jeux de donnnés, dans lesquels est traité la souscription en tant qu'agent avec un chiffre d'affaires inférieur à 1 000 000 €. Les valeurs choisies pour le chiffre d'affaires sont 150 000, 300 000, 500 000, etc. Le choix de traiter de 5 implémentations sur ce cas s'explique par le fait de vouloir varier le montant max garanti. Ce montant provient d'une autre application et peut avoir comme valeur 100 000, 250 000, 450 000, 600 000 et 1 000 000 €. Ce montant garanti n'est pas soumis à des règles de gestion, mais dans le cadre de la vérification des informations affichées sur la nouvelle offre, il était important de vérifier que ce montant était bien reçu par l'application, peu importe sa valeur.

C.2.4/ COMPLÉTION DE LA COUCHE D'ADAPTATION

La complétion de la couche d'adaptation n'a pas concernée l'ensemble des phases présenté dans la figure 4.7, l'automatisation couvre le lanceur, le besoin, l'entreprise puis les garanties. Le devis et la souscription n'apparaissent pas dans le répertoire d'objet, car ces deux interfaces n'ont pas été incluses dans le périmètre d'automatisation.

Dans le détail des données est affiché par exemple l'espace entreprise qui regroupe les données propres aux entreprises comme le chiffre d'affaires, le code NAF, le numéro de Siret, etc. Cette gestion a facilité la liaison entre les éléments des parcours applicatifs, les données et les objets utilisés dans l'application. Cela est visible dans la figure 4.10 où le nom des données dans le dictionnaire est proche, voire identique aux termes utilisés dans les tâches du parcours applicatif. Pour finir, quand cela était possible les objets ont été identifiés par leur id et sinon par leurs propriétés les plus fortes (texte, lien, etc).

D

ÉTUDES DES PROPOSITIONS PAR THÉMATIQUE POUR L'ANALYSE QUANTITATIVE ET L'ANALYSE QUALITATIVE

Cette annexe présente les analyses individuelles menées sur chacune des thématiques identifiées pour l'analyse qualitative et l'analyse quantitative. Ces études ont pour objectif d'avoir une meilleure compréhension des résultats obtenus indépendamment, pour ensuite les utiliser dans une analyse plus transverse et comprendre les pratiques en test au niveau inter-équipes dans l'Agilité à l'échelle.

D.1/ POURCENTAGE DE RÉPONSES PAR THÉMATIQUE

La figure D.1 présente le pourcentage de répartition des propositions par thématiques pour les interviews et l'enquête. Pour chaque proposition dans les interviews, et pour chaque réponse de l'enquête, nous avons compté leurs nombres dans chaque thématique et établi un pourcentage sur le nombre total de propositions obtenu dans chacune des sources de recueil. Nous distinguons dans notre tableau les valeurs 0% et NA (Non applicable).

Dans l'enquête, nous utilisons NA dans le cas où aucune proposition ne peut couvrir la thématique. Dans les entretiens et dans l'enquête, nous obtenons 0% dans le cas où aucune proposition ne permet de couvrir la thématique.

Thématiques	Interviews	Enquête	% de différences entre l'enquête et les interviews
La maturité Agile est faible	7,2%	9,2%	2,0%
La maturité Agile est modérée	5,4%	NA	5,4%
La maturité en test est faible	11,8%	3,7%	8,1%
La maturité en test est modérée	19,1%	9,0%	10,1%
L'atelier des 3 amigos est utilisé	3,4%	1,7%	1,7%
Les tests inter-équipes sont maîtrisés	2,5%	0,0%	2,5%
Les tests inter-équipes ne sont pas maîtrisés	5,2%	3,8%	1,4%
Il existe un manque de personnel compétent en test	3,0%	NA	3,0%
Un test Manager ou similaire est intégré à la solution	7,5%	14,4%	6,9%
Des pratiques et outils communs sont mis en place	6,5%	4,2%	2,3%
Le niveau de maturité des équipes est hétérogènes	4,7%	5,5%	0,8%
Il existe des équipes de test transverses	2,8%	2,5%	0,3%
Des rôles sont fusionnés (testeur, BA, dev)	3,1%	12,5%	9,4%
Il existe un manque de disponibilité/implication/vision du métier	4,2%	4,3%	0,1%
Il y a de la résistance au changement	3,8%	6,1%	2,3%
Il existe des difficultés de synchronisation/vision hors équipe	2,8%	NA	2,8%
Les comportements et attentes du top management sont parfois non adaptés	2,7%	6,0%	3,3%
Il existe des difficultés techniques	2,3%	2,0%	0,3%
Il y a des difficultés d'automatisation	0,6%	2,0%	1,4%
L'Agilité à l'échelle n'a pas d'impact pour les tests	0,0%	1,1%	1,1%
L'évolution des rôles en test est bien vécue	0,0%	1,3%	1,3%
La gestion des tests c'est complexifié	0,0%	1,4%	1,4%
L'intégration des tests dans les itérations est compliquée	0,0%	2,0%	2,0%
Des techniques comme le BDD et l'ATDD sont utilisées	1,0%	1,7%	0,7%
Le soutien de l'organisation sur la qualité globale de la solution	0,3%	2,8%	2,5%

FIGURE D.1 – Pourcentage de proposition par thèmatique pour les interviews et l'enquète

D.2/ ANALYSE DES THÉMATIQUES

Afin d'établir une première analyse, nous avons étudié indépendamment chacune des thématiques en regroupant les résultats issus des interviews et de l'enquête.

Nous précisons dans chacune des analyses des thématiques comment nous avons classifié nos résultats pour cette thématique. Pour chaque question la somme des pourcentages mentionnée peut être supérieure à 100%, car l'enquête autorisait le choix de plusieurs réponses. Notez que ces pourcentages ne sont pas en relation avec le tableau D.1. Dans le tableau D.1 les pourcentages présentés concernent le nombre total de propositions. Alors que dans la section qui suit les pourcentages concernent soit les réponses obtenues aux questions de l'enquête.

Pour les interviews, les pourcentages sont exprimés selon le nombre de propositions associé à une thématique. Par exemple, sur 100 propositions qui traitent de maturité Agile faible ou modérée, si 30 propositions traitent de la maturité Agile faible et 70 de maturité Agile modérée ont exprimera respectivement 30% et 70% pour ces deux thématiques.

Nous présentons dans chacun des paragraphes suivants l'ensemble des thématiques identifiées ainsi que le protocole mis en place pour classer chacune des propositions dans une thématique ou une autre.

Nous avons établi pour chaque paragraphe un ensemble de critères permettant de définir si une proposition appartient à une thématique ou à une autre. Dans le cas où les propositions couvrent plus d'une thématique, nous les scindons pour couvrir l'ensemble des thématiques abordées.

LA MATURITÉ AGILE EST FAIBLE

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient l'absence de formation à l'Agilité, des personnels qui découvrent et ne maîtrisent pas leur rôle "Agile" dans le contexte, et pas ou peu d'implication pour les préceptes Agiles. Cela se traduit par des expressions comme "personne n'est formé", "découverte", "pas mature" utilisé dans les propositions sur l'Agilité.

Dans l'enquête, nous avons classé les réponses aux questions dans cette thématique lorsqu'elles mentionnaient le "manque de connaissance sur l'Agilité" à l'échelle et "la mise en place des préceptes Agile". C'est le cas pour certaines réponses aux questions 15 et 16 de l'enquête. Ainsi si l'on étudie les résultats obtenus pour ces deux questions dans l'enquête, un manque de connaissance sur l'Agilité à l'échelle est évoqué à 46% et la difficulté de mise en place des préceptes Agile est évoquée à 35%. Dans les interviews, on retrouve les mêmes constats, les personnes interrogées précisent dans certains cas

que la solution n'est pas toujours 100% Agile, que les règles de l'Agilité sont difficiles à mettre en place. Certaines organisations tentent de reproduire des organisations de type cycle en V, mais dans un contexte d'Agilité à l'échelle. Dans les interviews, on observe que pour 43%, la maturité Agile est plutôt faible et ceci s'explique par la difficulté à comprendre ce qu'est l'Agilité à l'échelle, et comment mettre en place ses préceptes.

La maturité Agile est modérée

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées exprimaient que les préceptes Agiles étaient respectés. Cela se traduit par la mention des cérémonies Agiles comme le dealy meeting, etc. Nous avons aussi inclus les propositions qui traitaient de la collaboration entre les acteurs Agiles comme la mention d'un Scrum master accompagnant les activités au sein des équipes.

Dans l'enquête, nous n'avons pas posé de questions qui permettaient de remonter des propositions rencontrant la thématique d'une maturité Agile modérée. Nos résultats sont donc fondés sur les interviews, ou dans 43% des contextes étudiés une maturité Agile plutôt faible est relevé, les autres (57%) relèvent plutôt d'une maturité Agile modérée.

La maturité en test est faible

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées exprimaient un manque de maturité en test. Cela se traduit par des propositions où elles mentionnent le manque ou l'absence de formation aux tests, d'implication pour les tests, d'intégration des tests, etc. Ces expressions peuvent être négatives vis-à-vis du test, par exemple ”pas bien implanté au niveau du test”, ”pas d'outillage de test”, ”pas de test”, etc.

Dans l'enquête, nous avons classé les réponses aux questions dans cette thématique lorsqu'elles correspondaient à une maturité faible en test. Les réponses que nous considérons comme relevant d'une maturité faible en test sont celles par exemple où on mentionne un manque de prise en compte des tests, le fait que le test a perdu de l'importance, le manque de sensibilisation aux activités de test, etc. Parmi les questions où nous avons le plus relevé de réponses sur cette thématique, on trouve les questions 20 et 23 de l'enquête. Dans le détail, les réponses obtenues sont les suivantes :

- 15% des répondants évoquent que la qualité a perdu de l'importance ;
- 29% des répondants énoncent rencontrer des difficultés dans les tests inter-équipes, car les équipes manquent de sensibilisation aux activités et enjeux du test ;

- 18% rencontre des difficultés pour la mise en œuvre des approches de test.

Dans les interviews, la faible maturité des tests en Agile se reflète par une faible appétence pour le test. Cette faible appétence s'exprime par exemple par le fait que les développements et la production priment sur la qualité du produit livré, et les tests sont relégués au second plan. Ce manque d'appétence s'explique parfois par un manque de connaissance sur les enjeux du test ce qui coïncide avec une partie des réponses obtenues dans l'enquête. Néanmoins, la maturité en test faible reste minoritaire, car elle représente 38% des propositions sur l'ensemble des propositions liées à la maturité en test.

LA MATURITÉ EN TEST EST MODÉRÉE

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées décrivaient des techniques et pratiques de test qu'ils mettaient en place dans leur contexte. Cela se traduit par des expressions avec les mots "tests 2 à 2", "TDD", "taux de couverture du code assuré", "test de perf", etc. Nous avons aussi classé les propositions dans cette thématique qui traitaient de l'organisation des tests, comme la définition de critère d'acceptance. Nous avons aussi inclus les propositions mentionnant positivement le test tel qu'"effort de test pris en compte".

Dans l'enquête, nous avons classé les réponses aux questions dans cette thématique lorsqu'elles correspondaient à une maturité modérée en test. Nous avons considéré les réponses comme relevant d'une maturité modérée en test quand des testeurs, des automaticiens étaient intégrés aux équipes. D'autant plus dans les réponses où le testeur est mentionné comme un rôle reconnu. En classant les propositions dans les interviews, on relève que 62% des contextes décrivent un niveau de test modéré. Dans l'enquête, 72% des répondants ont affirmé que les testeurs sont intégrés dans chacune des équipes Agiles, et pour 51% des répondants au cœur des équipes Agile.

L'ATELIER DES 3 AMIGOS EST UTILISÉ

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient l'atelier des 3 amigos comme étant mis en place au sein de la solution. Il en va de même pour l'enquête. Ainsi on apprend dans l'enquête que l'atelier des 3 amigos ou la coordination de rôles clés au sein des équipes font parties, parmi d'autres, des éléments clés sur lesquels reposent les tests inter-équipes. C'est aussi l'une des techniques de test les plus employées au niveau inter-équipes (à 41%).

Au sein des interviews cette technique a plutôt été mentionnée comme une technique mise en place, plutôt que comme un élément clé. On note que la majorité des répondants,

62%, mentionnent la mise en place d'atelier des 3 amigos ou proche. On ne parle pas par contre de pratiques clés, car les interviews ont été orientés pour comprendre les pratiques actuelles et ont été peu orientés sur les solutions à mettre en place.

TEST INTER-ÉQUIPES SONT MAÎTRISÉS ET TEST INTER-ÉQUIPES NE SONT PAS MAÎTRISÉS

Dans les interviews, les propositions relevant d'une faible maîtrise des tests inter-équipes sont d'abord le fait de ne pas mentionner les tests inter-équipes ou leur maîtrise. D'exprimer clairement que c'est une activité qui pose des difficultés et qui n'est pas maîtrisée. Par exemple, on va trouver des expressions comme "Le test inter-équipes n'est pas mis en place".

Au contraire, les propositions concernant les tests inter-équipes et parlant de leur maîtrise sont classées dans la thématique des tests inter-équipes maîtrisés. Ici nous avons regroupé ces deux thématiques, car l'une et le contraire de l'autre, mais dans les interviews nous avons classé les propositions selon si elles appartenaient à une thématique ou à une autre. Dans l'enquête, deux questions traitent de la maîtrise ou non-maîtrise des tests inter-équipes (questions 22 et 23). Les réponses correspondant à cette thématique sont par exemple "Les activités de test sont peu maîtrisées au niveau inter-équipes". Comme pour les interviews, ce sont des réponses où la maîtrise (ou non-maîtrise) des tests inter-équipes est mentionnée. Que cela soit au niveau de l'enquête ou des interviews, le constat est similaire : les tests inter-équipes sont peu maîtrisés et représentent une difficulté pour plus de 91% des répondants à l'enquête.

Dans les interviews, on relève que sur le panel des propositions traitant de la maîtrise des tests inter-équipes, 33% traitent de la maturité sur les tests inter-équipes et les 67% restantes indiquent une non-maîtrise des tests inter-équipes. Le traitement de ces propositions par niveau de maîtrise nous donne 52% où les tests inter-équipes ne sont pas maîtrisés du tout, 28% de maîtrise moyenne et 10% de maîtrise élevée. Dans l'enquête, on note des résultats moins marqués avec 39% qui relèvent peu de maîtrise au niveau inter-équipes. Si on considère que les personnes qui maîtrisent les tests inter-équipes rencontrent peu de difficultés, les 9% de répondants qui ne rencontrent pas de difficultés (dans l'enquête) sont cohérents avec les 10% de répondants maîtrisant les tests inter-équipes dans les interviews.

IL EXISTE UN MANQUE DE PERSONNEL COMPÉTENT EN TEST

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient le manque de personnel compétent en test.

Nous avons classé les propositions dans ce thème quand les personnes interrogées mentionnaient les mots ”manques”, ”difficultés”, ”compétent”, lorsqu'elles parlaient du recrutement.

Cette thématique n'a été relevée que dans les interviews et non reportée dans l'enquête. En effet, environ un répondant sur deux (48%) a mentionné des difficultés à rencontrer des profils de testeur s'adaptant aux contextes d'Agilité à l'échelle. La problématique était surtout liée à des difficultés de recrutements ce qui a fait que la question n'a pas été reportée dans l'enquête.

UN TEST MANAGER OU SIMILAIRE EST INTÉGRÉ À LA SOLUTION

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient qu'un Test Manager ou similaire est intégré à la solution. Cela se traduit par des expressions comme ”introduit le rôle de Test Manager”, ”une personne coordonne les tests” , etc.

Dans l'enquête, différentes réponses couvrent cette thématique. Ce sont notamment les réponses qui exprime la présence d'un Test Manager ou similaire au sein de la solution (questions 17, 19 et 25). Ce sont par exemple les réponses comme ”Un ou plusieurs Test Manager coordonnent les activités de test” ou ”oui : c'est un rôle clé (le Test Manager)”.

Les propositions mettent en avant l'intégration du Test Manager, dans les interviews c'est le 3^{ième} thème le plus abordé. Pour 90% des interviewés, un Test Manager est présent sur les solutions. Le constat est le même dans l'enquête avec 87% des répondants qui ont affirmé avoir un Test Manager au sein des solutions. Son rôle est d'aider à la transformation, d'accompagner les équipes en test sur l'outillage et les méthodes de test. Il est aussi présent pour représenter le test dans les cérémonies Agile (50% des répondants en accord avec cela dans l'enquête). Par rapport aux contextes traditionnels, son rôle a évolué avec une présence plus marquée au sein des équipes (dans l'enquête seuls 5% notent que les compétences et activités du Test Manager n'ont pas changées). Dans l'enquête environ 50% des répondants précisent que c'est un rôle clé.

Que cela soit dans l'enquête ou dans les interviews, très peu de contextes n'ont pas de Test Manager ou similaire (cela représente 5% des personnes interrogées pour l'enquête, et 10% pour les interviews).

DES PRATIQUES ET OUTILS COMMUNS SONT MIS EN PLACE

Dans les interviews nous avons classé les propositions dans la thématique de la mise en place de pratiques et outils communs lorsque les personnes interrogées évoquaient les pratiques en test et outils communs au sein de leur contexte. Nous avons isolé les

propositions contenant ”outil commun”, ”communauté”, ”accompagnement”, etc. quand elles décrivaient les tests.

Dans l'enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent la mise en place de pratiques et outils en communs. C'est le cas pour les questions 22 et 25. Par exemple, on trouve la réponse ”Les activités de test sont régies par de bonnes pratiques mises en place par une communauté de test”. Nos résultats montrent que la mise en place de pratiques et d'outils en commun est le 5^{ième} thème le plus abordé dans les interviews. 95% des personnes interviewées ont décrit la mise en place de communauté à petite ou large échelle.

Dans l'enquête cette tendance est moins forte avec 33% des répondants qui précisent que les activités de test sont régies par de bonnes pratiques mises en place par une communauté de test. Cet écart peut s'expliquer par la formulation de la réponse, dans l'enquête on demande si les activités de test sont régies par des bonnes pratiques ce qui laisse à penser qu'elles sont mises en place. Dans les interviews, bien que ces communautés soient décrites, il est peu précisé que les équipes appliquent les bonnes pratiques.

LE NIVEAU DE MATURITÉ DES ÉQUIPES EST HÉTÉROGÈNE

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées décrivaient des niveaux de maturité hétérogène au sein de leurs équipes Agiles. Cela se traduisait par l'usage d'expression comme ”différents niveaux de maturité”, ”pluridisciplinatié”, ”pas organisées” pour parler des équipes. Ainsi que les cas où les personnes interrogées exprimaient que ce n'était pas l'ensemble des équipes qui étaient matures en termes de test, développement et autres.

Dans l'enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent un niveau de maturité hétérogène des équipes. C'est le cas pour les questions 15, 22 et 23. Un exemple de réponse existe dans la question 15 avec la proposition : ”différentes approches apparaissent des différentes équipes”. Des niveaux hétérogènes de maturité des équipes sont mentionnés dans 81% des interviews. La maturité est hétérogène en test, sur l'application des approches et des techniques différentes. Et elle l'est aussi en termes de maturité Agile dans la compréhension et l'adhésion à la transformation.

Cette tendance est moins tangible dans l'enquête avec le fait que 43% des répondants expriment que cette désorganisation est une difficulté globale au sein de la transformation et un peu moins au niveau inter-équipes à 32%. La tendance plus faible dans l'enquête peut s'expliquer par le fait que le niveau de maturité des équipes était un point abordé parmi d'autres alors que lors des interviews les personnes interrogées sont entrées dans le détail des points d'hétérogénéité.

IL EXISTE DES ÉQUIPES DE TEST TRANSVERSES

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient des équipes de test transverses au sein de leurs organisations. Cela se traduit par des expressions où les personnes interrogées évoquent des équipes indépendantes pour le test comme ”équipe indépendante envoyait les bugs”, ”les transverses (équipes) travaillaient sur d’autres projets”, etc.

Dans l’enquête, une question couvre cette thématique, c’est la question 17 où l’une des réponses concerne la présence d’équipe de test transverse. 67% des personnes interrogées évoquent la présence d’équipes de test transverses dans les interviews et 51% dans l’enquête.

Les interviews nous apprennent que les tests transverses sont souvent menés pour vérifier les comportements de bout en bout mettant en jeu plusieurs produits de la solution. Chaque contexte où on retrouve de la maîtrise des tests inter-équipes a une équipe de test transverse.

DES RÔLES SONT FUSIONNÉS (TESTEUR, BA, DEV)

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient des fusions de rôles au sein de leur organisation concernant les tests. Cela peut se traduire par des testeurs qui réalisent des activités plus techniques ou fonctionnelles. Par exemple, on trouve des propositions comme ”fusionner les testeurs fonctionnels”, ”le Test Manager a un rôle fonctionnel”, ”moins de cassure entre les différents rôles”, etc.

Dans l’enquête, une question couvre cette thématique, c’est la question 18 où des réponses concernent la fusion des rôles. Dans ces réponses, on trouve par exemple que les testeurs réalisent des tâches plus variées, techniques ou bien fonctionnelles. Dans les interviews 71% des personnes interrogées mettent en avant que les rôles ont évolué et sont devenus plus « multi-compétences ». C’est-à-dire qu’un testeur ne participe plus uniquement aux activités de test nominales comme la conception et exécution de test. Il est un acteur à part entière des différentes cérémonies Agile, et travaille activement à la définition et la formulation du besoin, la conception des US, et parfois participe à des activités techniques comme l’automatisation des tests.

De manière proche, l’enquête révèle les mêmes constats, avec 87% des répondants qui affirment que les compétences et les activités en test ont changé, avec 54% qui précisent que les tâches sont plus fonctionnelles, 62% que les tâches sont plus techniques et enfin 74% que les testeurs collaborent davantage avec les acteurs du produit.

Il existe un manque de disponibilité/implication/vision du métier

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient des difficultés sur cette thématique. Cela se traduisait par des expressions avec les combinaisons des mots ”difficultés”, ”difficile”, ”entrant” (entrant ici qualifie les entrants métier : exigence, US, etc.), ”métier”, etc. Cela concerne aussi les propositions qui relève de la faible disponibilité des intervenants métier, par exemple ”le métier n’est pas disponible”.

Dans l’enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent le manque de disponibilité, d’implication et de vision métier. Cela est évoqué dans des réponses aux questions 15, 16 et 23 de l’enquête. Par exemple, certaines réponses relèvent le manque d’investissement ou d’implication.

Dans les interviews, 67% des personnes interrogées ont mis en avant un manque de disponibilité ou d’implication du point de vue métier menant à un manque de vision sur la solution dans son ensemble. Souvent, ceci est expliqué par le fait que les systèmes à vérifier dans les contextes d’Agilité à l’échelle sont des solutions très larges et complexes.

De ce fait, de nombreux niveaux et interlocuteurs existent ce qui peut parfois compliquer la transmission de l’information et donc la vision globale sur la solution. De plus, chaque acteur métier sur la solution a souvent plusieurs rôles et n’a pas toujours la disponibilité pour traiter des sujets métiers avec d’autres acteurs. Cet ensemble d’éléments conduit au manque de scénarios de bout en bout. Ce point est capital et est relayé dans l’enquête. En effet, le manque ou l’absence de scénario de bout en bout est la 1ère difficulté rencontrée dans la mise en place des tests inter-équipes (Mis en avant par 53% des répondants).

Il y a de la résistance au changement

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient des difficultés dans l’acceptation de l’Agilité à l’échelle. Cela se traduit par des expressions contenant les mots ”résistances” ”accepter”(quand employé à la forme négative), ”bloquer”, ”gestion du changement dans la difficulté”, etc.

Dans l’enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent la résistance au changement dans le contexte d’Agilité à l’échelle. C’est le cas des questions 15, 16 et 23 où des réponses couvrent cette thématique en utilisant l’expression ”résistance au changement”. Dans les interviews, 67% des personnes évoquent des problématiques de résistance au changement. Cette résistance est générale et pas particulièrement ciblée sur le test. C’est plutôt le passage du cycle en V vers de l’Agi-

lité qui pose problème. Pour de grands groupes, les cellules de test ont été dissipées pour être distribuées sur des équipes Agile ce qui a été compliqué. Dans l'enquête, la résistance au changement se traduit principalement au niveau métier et sur la mise en place globale de l'Agilité à l'échelle.

Il existe des difficultés de synchronisation/vision hors équipe

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées mentionnaient des difficultés de synchronisation et de vision hors équipe. Cela se traduit par des expressions comme "manque de synchronisation" pour parler des pratiques au sein de l'organisation. Cela regroupe aussi les propositions qui mettent en avant le manque de visibilité au sein de l'organisation pour certains acteurs, par exemple "le responsable de socle n'avait pas de visibilité sur la synchronisation". Dans l'enquête, aucune question ne traite de cette thématique.

Dans les interviews ont apprend que 52% des personnes interrogées travaillent dans des contextes où elles ont rencontrées des difficultés de synchronisation et de vision hors équipe. Ceci se traduit par le fait que les équipes Agile ont tendance parfois à travailler en silo. Dans les interviews on retrouve souvent le terme de « cloisonnement » pour parler des équipes Agile. Elles réalisent leurs développements, testent et déploient sans forcément avoir de visibilité avec les travaux des autres équipes mêmes si leurs activités peuvent être liées. De plus, la gestion de beaucoup d'équipes Agile est un véritable challenge notamment pour maintenir une visibilité sur les actions de chaque équipe sans dépenser trop de temps dans cette activité.

Les comportements et attentes du top management sont parfois non adaptés

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient des comportements ou attentes non adaptés du top management. Ce que nous considérons comme comportement et attentes non adaptés du top management ce sont toutes les actions du top management qui ne se prête pas aux préceptes de l'Agilité à l'échelle. C'est par exemple la non-prise en compte de la qualité, cela se traduit par des expressions comme "les responsables de socle n'agissent que sur les objectifs, donc s'ils ne voient pas d'intérêt, c'est de la contrainte (le test)".

Dans l'enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent des comportements ou attentes non adaptés du top management. Ce sont les questions 15 et 16 dont des réponses couvrent cette thématique avec l'évocation des limites organisationnelles. Dans le détail, les interviews présentent les attentes non

adaptées du top management souvent par un manque d'implication pour la qualité et est exprimé par 38% des répondants. Ce manque d'implication s'explique par une volonté de réduire les coûts de mise en production, et donc la réduction des activités de test participe à cette réduction de coût (à court terme).

Le fonctionnement des grandes organisations est très hiérarchique et basé sur des objectifs. Ces objectifs vont être ciblés sur la production rapide de nouvelles applications, en respectant un équilibre coût/qualité/délai, mais souvent l'équilibre est peu respecté et les objectifs sont ciblés sur les délais et les coûts. Ainsi si les plus hautes sphères de l'organisation ne définissent pas des objectifs sur la qualité, les responsables plus bas dans cette hiérarchie subissent la pression de leurs dirigeants et développent leur produit en respectant les objectifs définis et ne vont pas au-delà. Ceci n'est pas une généralité, mais une pratique existante dans les grands groupes.

Un autre aspect qui traduit les attentes et comportements non adaptés du top management est tout simplement aussi un manque de connaissance sur les enjeux du test dans les contextes Agile. Plutôt habitué au cycle en V où le test se déroule en fin de développement et où l'effort consacré à celui-ci peut-être variable, l'intégration du test tout au long de la construction de la solution est pour certains encore un point d'incompréhension.

Enfin si l'on compare les résultats obtenus dans les interviews et dans l'enquête, cette thématique est moins évoquée dans les interviews, seulement 38%, contre 67% dans l'enquête pour les difficultés rencontrées dans la mise en place de l'Agilité de manière générale. Cette différence s'explique par le fait que dans les interviews les personnes interrogées étaient peut-être moins enclins à mettre en avant les limites organisationnelles de leurs entreprises que dans l'enquête, qui elle offrait plus d'anonymat.

Il existe des difficultés techniques

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient des difficultés techniques. Cela se traduit par l'usage d'expression comme "dette technique", "outillages très techniques". Les propositions classées dans cette thématique pouvaient aussi correspondre à l'expression d'un manque ou l'absence d'environnement pour réaliser les tests. Des expressions comme "non-disponibilité des environnements" correspondent à cette thématique. Dans l'enquête, les questions qui couvrent cette thématique sont celles où des réponses évoquent des difficultés techniques. C'est principalement la question 23 qui traite de cette thématique avec la réponse "manque ou absence d'environnement permettant d'effectuer les tests".

Dans les interviews les difficultés techniques sont abordées par 38% des répondants.

Elles se traduisent pour certains par une dette technique existante qui rend compliquée la réalisation de nouveaux développements ainsi que leurs vérifications. Dans certains cas, ce sont les chaînes d'intégration continue complexes qui posent des difficultés. En effet, plus les solutions vont être large, plus l'infrastructure mise en place risque d'être complexe avec de nombreux outils. Dans ces contextes, il faut parvenir à intégrer des tests dans une organisation déjà complexe.

Enfin le dernier point de peine évoqué dans les interviews est souvent le manque ou l'indisponibilité d'environnements adaptés pour les tests. Les environnements vont être créés en priorité pour le développement et ensuite seulement pour les tests. De même la maintenance de ces environnements de tests n'est pas considérée comme prioritaire, ainsi en cas de panne des environnements de développements, ce sont les environnements de test, quand ils existent, qui sont réquisitionnés. Et en cas de panne d'environnement de test, la remise en service n'est souvent pas une priorité. Dans l'enquête, ces constats se confirment avec 44% des répondants mettant en avant le manque ou l'absence d'environnement permettant d'effectuer les tests.

IL Y A DES DIFFICULTÉS D'AUTOMATISATION

Dans les interviews, nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient des difficultés d'automatisation. Cela se traduit par des propositions où l'absence de test automatisé est exprimée, par exemple "pas de test d'intégration continue automatisé".

Dans l'enquête, la question qui couvre cette thématique est celle où une réponse évoque des difficultés d'automatisation. C'est la question 23 qui couvre cette thématique avec la réponse "Gestion de l'automatisation des tests" pour parler des difficultés rencontrées dans les tests inter-équipes. Les difficultés d'automatisation sont la 2^{ième} difficulté évoquée dans la réalisation des tests inter-équipes dans l'enquête (à 47%). Dans les interviews on retrouve cette thématique, mais de manière plus faible avec 19% des personnes interrogées relevant que l'automatisation pouvait être une difficulté, mais qui ne touche pas uniquement le niveau inter-équipes.

L'AGILITÉ À L'ÉCHELLE N'A PAS D'IMPACT POUR LES TESTS

Dans les interviews cette thématique n'a pas été abordée, elle l'a été uniquement dans l'enquête. La question 20 couvre cette thématique avec une réponse qui évoque que l'Agilité à l'échelle n'a pas d'impact pour les tests. Seul 23% des personnes ayant répondu à l'enquête estiment qu'il y a eu peu d'impact et pas de réorganisation avec le passage vers l'Agilité à l'échelle. Cette faible tendance se confirme dans les interviews où tous

les acteurs ont affirmé avoir été impacté par cette transformation. Même si les impacts étaient limités (légers changements dans l'organisation, dans les rôles et activités en test, etc), ils existaient, ne serait-ce qu'au niveau de la réorganisation des équipes de test.

L'évolution des rôles en test est bien vécue

Dans les interviews cette thématique n'a pas été abordée. Ceci s'explique par le fait que les questions étaient orientées sur les impacts de la transformation et que les personnes interrogées ont plutôt mentionné les points de peine et la résistance au changement plutôt que les éléments qui n'avait pas changé.

Dans l'enquête, la question qui couvre cette thématique est celle où une réponse évoque que l'évolution des rôles en test est bien vécue. C'est la question 20 qui couvre cette thématique avec une réponse qui exprime cela. Ainsi nos résultats de l'enquête montrent que peu de personnes évoquent le fait que l'évolution des rôles en test est bien vécue (26%). Comme pour les interviews ceci s'explique par le fait que les personnes ont plutôt été confrontées à de la résistance au changement qu'à une acceptation majoritaire.

La gestion des tests s'est complexifiée

Dans les interviews le fait que la gestion des tests s'est complexifiée n'est pas un point qui a été abordé. Les personnes interrogées ont plutôt évoqué les évolutions et les changements plutôt que de la complexité. Dans les interviews, les difficultés abordées à ce sujet étaient plutôt liées au fait qu'avec le passage vers l'Agilité à l'échelle les équipes de tests ont été réparties dans différents projets laissant moins de visibilité au responsable des tests dans les organisations sur les personnels en test.

Dans l'enquête, la question qui couvre cette thématique est celle où une réponse évoque que la gestion des tests s'est complexifiée. C'est la question 20 qui couvre cette thématique. Cependant le fait la gestion des tests s'est complexifié reste minoritairement abordé dans l'enquête, à hauteur de 28%.

L'intégration des tests dans les itérations est compliquée

Cette thématique a été uniquement abordée dans l'enquête dans les difficultés rencontrées sur les activités de test au niveau inter-équipes. 47% des répondants ont répondu avoir été confronté à des difficultés dans l'intégration des tests dans les itérations.

DES TECHNIQUES COMME LE BDD ET L'ATDD SONT UTILISÉES

Dans les interviews nous avons classé les propositions dans cette thématique lorsque les personnes interviewées évoquaient l'usage du BDD et l'ATDD.

Dans les interviews, la pratique de techniques comme l'ATDD et le BDD est peu évoquée (29%). Parmi les personnes mettant en place ces techniques, elles sont plutôt appliquées au sein des équipes et pas forcément au niveau inter-équipes. Parmi les personnes interrogées, une a évoqué l'importance de mettre en place ce genre de technique pour améliorer la qualité des produits. Dans l'enquête, on retrouve ceci avec 40% des répondants qui affirment que des techniques comme le BDD et l'ATDD sont des éléments clés dans la maîtrise des tests inter-équipes.

LE SOUTIEN DE L'ORGANISATION EST IMPORTANT POUR LA QUALITÉ GLOBALE DE LA SOLUTION

Dans les interviews les éléments qui relevaient d'un soutien de l'organisation étaient par exemple d'exprimer que l'organisation avait un rôle à jouer pour assurer la qualité de la solution. Ce point est très peu abordé dans les interviews (5%). Ceci s'explique par le fait que les interviews avaient plutôt pour objectifs de comprendre les pratiques actuelles que de se projeter vers les solutions possibles comme cela était fait dans l'enquête. Cependant le soutien de l'organisation sur la qualité globale de la solution est considérée comme un point clé dans l'enquête (à 69%),

D.3/ MODÈLE DU TABLEAU UTILISÉ POUR LES INTERVIEWS

La figure D.2 présente le modèle du tableau utilisé pour les interviews.

D.4/ LES QUESTIONS DU QUESTIONNAIRE EN LIGNE

Maturité Agile faible	Maturité Agile modérée	Maturité en test faible	Maturité en test modéré	3 amigos	Test inter-équipes maitrisé	Test inter-équipes non maitrisé	Manque de personne compétent en test	Intégration du test Manager ou similaire	Mise en place de pratique et outil commun	Maturité des équipes hétérogènes	Equipes de test transverse	Fusion de rôles (testeur, BA, dev)	vision du métier
proposition 1	proposition 4	proposition 5	proposition 14	proposition 20	proposition 21	proposition 23	proposition 24	proposition 25	proposition 26	proposition 27	proposition 28	proposition 29	proposition 30
proposition 2		proposition 6	proposition 15		proposition 22								
proposition3		proposition 7	proposition 16										
		proposition 8	proposition 17										
		proposition 9	proposition 18										
		proposition 10	proposition 19										
		proposition 11											
		proposition 12											
		proposition 13											
Compte des propositions													
3	1	9	6	1	2	1	1	1	1	1	1	1	1

Résistance au changement	Difficulté de synchronisation/vision hors équipe	Attentes non adapté du top management	Difficultés techniques	Difficultés automatisation	L'Agilité à l'échelle n'a pas d'impact pour les tests	L'évolution des rôles en test est bien vécue	La gestion des tests c'est complexifié	Intégration des tests dans les itérations	Utilisation de BDD et ATDD	Le soutien de l'organisation
proposition 31		proposition 33			proposition 34		proposition 35			
proposition 32										
Compte des propositions										
2	0	0	1	0	0	1	0	1	0	0

FIGURE D.2 – Modèle du tableau utilisé pour les interviews

Étude : les tests inter-équipes dans l'Agilité à l'échelle

1. Quels sont/ont été vos rôles dans les activités liées à l'Agilité à l'échelle ?

- [ ] Tests: Consultant / Analyste / ingénieur qualité / Automaticien
- [ ] Tests techniques: Testeur de charges / Performance
- [ ] Manager: Test manager / Test Lead / Responsable d'une cellule de tests
- [ ] Coaching: Test / Agile / Transformation
- [ ] Agile Master / Scrummaster
- [ ] Produit : Product owner / Product manager / Proxy product owner
- [ ] Développeur
- [ ] Autre (veuillez préciser)

2. Depuis combien de temps effectuez-vous des activités liées à l'Agilité à l'échelle ? (cumul)

- ○ Moins de 1 ans
- ○ Entre 1 et 3 ans
- ○ Entre 3 et 10 ans
- ○ Plus de 10 ans

3. Sur combien de périmètres/produits différents avez vous travaillé ?

4. De manière générale, quelles méthodes avez vous utilisées pour l'implémentation de l'Agilité à l'échelle ? (plusieurs choix possibles)

- [ ] SAFe®
- [ ] Nexus
- [ ] Scrum of scrum
- [ ] Scrum @ scale
- [ ] LeSS
- [ ] Spotify
- [ ] RAD
- [ ] Lean
- [ ] Autre (veuillez préciser)

5. Sur le contexte choisi, dans quel pays avez-vous mené votre activité ?

6. Pour le contexte choisi, quel était le domaine d'activité de celui-ci ?

- [ ] Agroalimentaire
- [ ] Banque / Assurance
- [ ] Chimie / Parachimie
- [ ] Commerce / Négoce / Distribution
- [ ] Édition / Communication / Multimédia
- [ ] Électronique / Électricité
- [ ] Industrie pharmaceutique
- [ ] Machines et équipements / Automobile
- [ ] Études, conseils et services
- [ ] Textile / Habillement / Chaussure
- [ ] Transports / Logistique / Tourisme
- [ ] Ingénierie logicielle
- [ ] Institution publique
- [ ] Autre (veuillez préciser)

7. Sur le contexte choisi, pour quelle entreprise avez-vous travaillé ? (Vous pouvez ou non préciser le nom de l'entreprise concernée et décider si vous voulez ou non mentionner ce nom dans l'étude)

Nom de la société

Le nom de la société peut être mentionné dans l'étude (Oui/Non)

8. Sur le contexte choisi, quels étaient le type d'application et le contexte produit auquel vous avez participé?
(Dans le cas d'un système d'information, vous pouvez cocher plusieurs produits et types d'application)

- [ ] Pas d'activité sur un contexte produit en particulier, intervention globale sur un système d'information regroupant divers produits et technologies
- [ ] Application web
- [ ] Application mobile
- [ ] Client lourd
- [ ] Produit d'E-Commerce
- [ ] Produit ERP
- [ ] Autre (veuillez préciser)

9. Sur le contexte choisi, quelle était la taille moyenne d'une équipe Agile ?

10. Sur le contexte choisi, quel était le nombre d'équipes Agile travaillant sur la solution ?

11. Les équipes Agiles étaient elles réparties dans plusieurs pays ?

- ( ) Oui
- ( ) Non

12. Sur le contexte choisi, quel était l'approche/le framework utilisé pour l'implémentation de l'Agilité à l'échelle ?

- [ ] SAFe®
- [ ] Nexus
- [ ] Scrum of scrum
- [ ] Scrum @ scale
- [ ] LeSS
- [ ] Spotify
- [ ] RAD
- [ ] Lean
- [ ] Autre (veuillez préciser)

13. Avant de passer en Agilité à l'échelle, quelle approche était en place ?

- [ ] Cycle en V
- [ ] Agile
- [ ] Autre (veuillez préciser)

14. Dans le contexte choisi, avez-vous :

	Oui/Non
Géré des activités de test	
Participé à des activités de test	
Participé à la transformation vers l'Agilité à l'échelle	

15. De manière générale, quelles difficultés avec vous rencontrés dans la mise en place de l'agilité à l'échelle ?(plusieurs choix possibles)

- [ ] Pas de difficultés
- [ ] Manque de connaissance sur l'Agilité à l'échelle
- [ ] La mise en place des préceptes Agile
- [ ] La résistance au changement
- [ ] Le manque d'investissement
- [ ] La désorganisation : différentes approches apparaissent des différentes équipes
- [ ] La gestion hiérarchique et limites organisationnelles
- [ ] Autre (veuillez préciser)

16. Quelles sont les difficultés que vous avez rencontrées dans l'Agilité à l'échelle au niveau métier (plusieurs choix possibles)?

- [ ] Le manque de connaissance vis à vis de l'Agilité à l'échelle
- [ ] La résistance au changement
- [ ] Le manque d'investissement
- [ ] La création et l'estimation des user story sont difficiles
- [ ] Écart entre la planification à long et à court terme
- [ ] Reconnaître l'importance du rôle du PO.
- [ ] Autre (veuillez préciser)

17. Parmi les propositions suivantes concernant le rôle test, lesquels sont vraies dans votre contexte ? (plusieurs choix possibles)

- [ ] En moyenne, il y a au moins un testeur dans chaque équipe Agile
- [ ] Des automaticiens transverses sont déployés sur la solution globale selon les besoins(soit au niveau transverse, soit dans les équipes)
- [ ] Une équipe de test transverse existe pour vérifier les développements transverses à plusieurs équipes (avec des personnes issues des équipes ou complètement indépendantes)
- [ ] Un ou plusieurs Test managers coordonnent les activités de test
- [ ] Autre (veuillez préciser)

18. Les compétences et les activités des testeurs ont elles évoluées? (plusieurs choix possibles)

- [ ] Non: les compétences et les activités n'ont pas changé
- [ ] Oui: les testeurs effectuent une plus grande variété de tâches
- [ ] Oui: les testeurs effectuent des tâches plus techniques (automatisation, contrôle de build automatisé, etc).
- [ ] Oui: les testeurs effectuent des tâches plus fonctionnelles (atelier avec le Product Owner(PO), étude des besoins, etc.).
- [ ] Oui: les testeurs collaborent davantage avec les acteurs du produit(développeur, scrum master, PO, etc.)
- [ ] Autre (veuillez préciser)

19. Dans le contexte choisi, un test manager(ou similaire) est-il présent ? Si oui, ses compétences et ses activités ont elles évoluées par rapport à des contextes Agile traditionnels. (plusieurs choix possibles)

- [ ] Pas de Test manager ou rôle similaire
- [ ] Non: les compétences et les activités n'ont pas changé
- [ ] Oui: le Test Manager est plutôt appelé "Test coach".
- [ ] Oui: il collabore avec le PO et la scrum team pour une meilleure compréhension des exigences.
- [ ] Oui: il représente les tests dans les différentes cérémonies et ateliers. Il partage les bonnes pratiques auprès des dirigeants et parties prenantes à la solution.
- [ ] Oui: C'est un rôle clé
- [ ] Autre (veuillez préciser)

20. Le passage vers de l'Agilité à l'échelle a-t-il entraîné une réorganisation des personnels en test ? Si oui quels ont été les impacts de cette réorganisation? (plusieurs choix possibles)

- [ ] Pas de réorganisation ou peu d'impacts
- [ ] L'évolution des rôles en test a été majoritairement bien vécue
- [ ] La gestion du test dans l'organisation s'est complexifiée, car les testeurs sont distribués.
- [ ] Le test, la qualité de manière générale ont perdu de l'importance.
- [ ] Les testeurs sont au cœur des équipes et leur rôle est reconnu
- [ ] Autre (veuillez préciser)

21. Quelles techniques utilisez-vous pour les activités de test inter-équipes dans le cadre des tests transverses/de bout en bout/d'intégration système? (plusieurs choix possibles)

- [ ] BDD
- [ ] ATDD
- [ ] Atelier des 3 amigos
- [ ] Checklists
- [ ] Tests exploratoires
- [ ] Model-Based-Testing
- [ ] Risk-Based-Testing
- [ ] Autre (veuillez préciser)

22. Parmi les propositions suivantes, lesquelles sont vraies vis-à-vis des activités de test dans votre contexte. (plusieurs choix possibles)

- [ ] La maturité et les techniques de test employés au niveau des équipes sont très hétéroclites.
- [ ] Les activités de test sont majoritairement maîtrisées au niveau des équipes
- [ ] Les activités de test sont peu maîtrisées au niveau inter-équipes
- [ ] Les activités de test sont régies par des bonnes pratiques mises en place par une communauté de test
- [ ] Autre (veuillez préciser)

23. Quelles difficultés avez-vous rencontrées sur les activités de test au niveau inter-équipes pour vérifier des comportements transverses (test de bout en bout/tests transverses/d'intégration système)? (plusieurs choix possibles)

- [ ] Pas de difficultés
- [ ] Manque de sensibilisation aux activités et enjeux des tests
- [ ] Mise en œuvre d'approches de test
- [ ] Manque d'implication
- [ ] Désorganisation : différentes approches dans différentes équipes
- [ ] Gestion de l'automatisation des tests
- [ ] Intégration des tests dans les itérations
- [ ] Manque ou absence de scénarios de bout en bout.
- [ ] Manque ou absence d'environnements permettant d'effectuer les tests.
- [ ] Autre (veuillez préciser)

24. Quelles solutions avez-vous mises en œuvre/auriez-vous souhaité mettre en œuvre pour répondre à ces challenges?

25. D’après vous, sur quels éléments clés reposent les tests transverses inter-équipe (bout en bout, intégration, système...) – Plusieurs choix possibles

- [ ] Aucun élément en particulier
- [ ] L'atelier des 3 amigos, ou la coordination de rôles clés au sein des équipes.
- [ ] La présence de Test managers ou d'autres rôles pour coordonner les activités de test
- [ ] La mise en place d'une communauté de test pour proposer des bonnes pratiques,outils communs et des formations.
- [ ] Le soutien de l'organisation sur la qualité globale de la solution
- [ ] Des techniques de test comme le BDD ou l'ATDD
- [ ] Autre (veuillez préciser)

26. Selon vous, que faudrait-il mettre en place pour que les activités de tests soient plus efficace dans un contexte d’agilité à l’échelle ?

27. Avez-vous d’autres commentaires à ajouter qui pourraient être utiles à cette enquête ?

28. Laissez ici une adresse mail pour recevoir en primeur les résultats.

D.5/ GUIDE D'ENTRETIEN

Quelles sont vos pratiques en termes d'Agilité dans votre organisation ?

- Avez-vous l'impression que les personnes faisant partie de la solution comprennent les pratiques Agiles et leur rôle au sein de la solution ?
- Vos collaborateurs ont-ils suivi des formations à l'Agilité ?
- Mettez-vous en place certaines cérémonies Agiles ?

Quelles sont vos pratiques en termes de test dans votre organisation ?

- Avez-vous l'impression que les personnes faisant partie de la solution comprennent les activités de tests et leur rôle au sein de la solution ?
- Vos collaborateurs ont-ils suivi des formations aux tests ?
- Vos collaborateurs se sentent-ils impliqué dans la qualité de la solution ?
- Techniques ?
- 3 amigos ?

Quelles sont les pratiques mises en place au niveau des tests inter-équipes ?

- Trouvez vous que vous maitrisez les tests inter-équipes
- Rencontrez-vous des difficultés dans ce domaine

Trouvez-vous que les personnels sont assez formés aux tests ?

- Rencontrez-vous des difficultés dans leur recrutement ?
- Trouvez-vous que les rôles en test ont évolué ?

Un test Manager ou similaire est intégré à la solution ?

- Parlez moi de son rôle, de ses activités

Des pratiques et outils communs sont mis en place ?

- Lesquels ?

Comment sont organisées les équipes au sein de la solution ?

- Comment est le niveau de maturité des équipes ?
- synchronisation/vision hors équipe

Existe-t-il des équipes de test transverses ?

Que pensez-vous de l'implication du métier dans la solution

- Quelle est la vision métier que les acteurs de la solution ont sur celle-ci
- Vision globale ?

Avez-vous rencontré des difficultés dans la mise en place de l'Agilité à l'échelle ?

- Il y a de la résistance au changement ?

Document réalisé avec LaTeX et :
le style LaTeX pour Thèse de Doctorat créé par S. Galland — https://github.com/gallandarakhneorg/tex-templates
la collection de paquets `tex-upmethodology` — http://www.arakhne.org/tex-upmethodology/

SPIM
école doctorale **sciences pour l'ingénieur et microtechniques**

Titre : Approche Agile du Model-Based Testing pour les tests fonctionnels des SI d'entreprise

Mots-clés : Model-Based Testing, Automatisation, Génie Logiciel, Processus de test logiciel, Méthodes Agiles , Refactorisation de test

Résumé :

La transformation digitale et Agile des grandes organisations induit un besoin de transformer en profondeur les pratiques du test logiciel. La mise en production des évolutions des systèmes informatiques avec des délais de plus en plus courts, sur des systèmes de plus en plus complexes, remet en cause les pratiques traditionnelles du test fonctionnel fondées sur une forte composante manuelle (analyse, conception et exécution). Les nouvelles approches et les nouveaux outils de test doivent s'adapter aux courtes itérations de l'Agile, ainsi qu'à l'expansion des systèmes développés, composés d'un ensemble de produits dans des contextes d'Agilité à l'échelle, tout en garantissant la qualité des systèmes en production. L'adaptation à l'Agilité signifie que les approches et outils devront être efficaces dans la conception et l'implémentation des tests pour différents objectifs de couverture fonctionnelles. Ils devront permettre de tester un petit ensemble de nouveaux récits utilisateurs tout en maintenant opérationnel le référentiel des tests de bout en bout reprenant les principaux cas d'utilisation des applications. Ceci de manière à gérer les évolutions du système et à étendre le référentiel des tests de régression automatisés. Du point de vue de l'exécution, les approches et outils devront supporter l'exécution de tests manuels et automatisés avec une adaptation transparente des premiers aux seconds. L'exécution systématique de l'ensemble des tests manuels n'est pas possible d'où la forte nécessité à automatiser les tests. La mise en place de processus d'automatisation est un point clé dans la réussite des projets, mais cette activité reste complexe et coûteuse. Elle permet de garantir la qualité de l'application à travers les différentes itérations, notamment dans les contextes Agile, ou les livraisons sont régulières et/ou le temps pour tester manuellement croît au fil des sprints. Automatiser un ensemble de cas de test permet d'alléger la charge de test manuel et facilite la détection de régressions. Un autre enjeu des tests logiciel dans la transformation Agile sera de parvenir à maintenir les suites de tests manuels grandissantes au fil des itérations. Ces suites de tests sont généralement décrites en langage naturel et, avec le temps, deviennent désorganisées et plus difficiles à utiliser et à maintenir. Il existe donc un réel défi à proposer des approches et des outils permettant de maintenir ces suites de tests afin de les rendre plus utilisables et plus faciles à maintenir. Pour répondre à ces besoins cette thèse propose une approche nommée ALME (ALME - Agile Lightweight Model-Based Testing for Enterprise IT) qui s'appuie sur les concepts fondamentaux du Model-Based-Testing (MBT, test à partir de modèles) refondés pour répondre aux enjeux du test des grands systèmes d'information dans le contexte d'un cycle de vie de développement logiciel itératif et incrémental. Cette refondation porte en particulier sur l'articulation entre la modélisation de processus métier et la gestion de règles métier pour le test pour la génération des tests fonctionnels. Les contributions de ces travaux de recherche couvrent les phases d'analyse, de conception et d'implémentation des tests fonctionnels et sont les suivantes :

- Étendre l'état de l'art scientifique en MBT, sur l'automatisation des tests, le refactoring, le test dans l'Agilité et l'Agilité à l'échelle.
- Étendre l'état de l'art sur les pratiques en test dans l'Agilité à l'échelle par une analyse qualitative et quantitative des pratiques actuelles et des solutions aux problèmes rencontrés par les équipes de développement du logiciel.
- Définir et mettre en application :
  - l'approche ALME.
  - une approche de gestion de l'automatisation.
  - une approche de refactoring des suites de tests manuels.

Ces travaux de recherches ont été conduits au sein de l'institut FEMTO-ST,UMR 6174 et de la société Sogeti dans le cadre d'une thèse CIFRE démarrée en novembre 2017.

Title: Agile Model-Based Testing Approach for Functionnal Testing of enterprise IT Systems

Keywords: Model-Based Testing, Automation, Software Engineering, Software testing process, Agile Methods, Test refactoring

Abstract:

The digital and Agile transformation of large organizations is leading to a need to transform software testing practices in depth. Putting IT system developments into production with increasingly short delays, on increasingly complex systems, challenges the traditional practices of functional testing based on a strong manual component (analysis, design and execution).
New approaches and new test tools must therefore adapt to the short iterations of Agile, as well as to the expansion of the systems developed, composed of a set of products in Agile at scale contexts, while ensuring the quality of the systems in production.
Adapting to Agile means that approaches and tools will need to be effective in the conception and implementation of tests for different functional coverage objectives. They will have to allow testing a small set of new user stories while keeping the end-to-end test repository of the main application use cases operational. This is in order to manage the evolution of the system as a whole and to extend the repository of automated regression tests.
In terms of execution, the approaches and tools will have to support both manual and automated test execution with seamless adaptation from the first to the second. Systematic execution of all manual tests is not possible, hence the strong need to automate tests. The implementation of automation processes is a key point in the success of projects, but this activity currently remains complex and expensive. It allows to guarantee the quality of the application through the different iterations, especially in Agile contexts, where the deliveries are regular and/or the time to test manually increases with the sprints. Automating a set of test cases reduces the burden of manual testing and facilitates the detection of regressions.
Another challenge for software testing in the Agile transformation will be to manage to maintain the growing manual test suites over iterations. These test suites are usually described in natural language and, over time, become disorganized and more difficult to use and maintain. There is therefore a real challenge in proposing approaches and tools to maintain these test suites in order to make them more usable and easier to maintain, in contexts where it is necessary to be able to act quickly and efficiently.
To meet these needs, this thesis proposes an approach called ALME (ALME - Agile Lightweight Model-Based Testing for Enterprise IT) which is based on the fundamental concepts of Model-Based-Testing, redesigned to meet the challenges of testing large information systems in the context of an iterative and incremental software development lifecycle. In particular, this overhaul focuses on the articulation between business process modeling and business rule management for the generation of functional tests. The contributions of these researches cover the phases of analysis, conception and implementation of functional tests and are the following:

- Extending the scientific state of the art in Model-Based-Testing, on test automation, refactoring, testing in Agile and Agile at scale.
- Extend the state of the art on Agile testing practices at scale through a qualitative and quantitative analysis of current practices and solutions to problems encountered by software development teams.
- Define and implement the ALME approach.
- Define and implement an automation management approach.
- Define and implement a manual test suite refactoring approach.

These researches were conducted within the FEMTO-ST Institute - UMR 6174 and the Sogeti company within the context of a CIFRE thesis started in November 2017.

UBFC UNIVERSITÉ BOURGOGNE FRANCHE-COMTÉ
Université Bourgogne Franche-Comté
32, avenue de l'Observatoire
25000 Besançon, France