Syllabus

REQB ®

Professionnel Certifié en Ingénierie des Exigences

Praticien Agile

Version 1.1

2015

Les droits d'auteur ® pour cette édition du syllabus dans toutes les langues sont détenus par Global Association for Software Quality, GASQ.

© GASQ – Global Association for Software Quality

Aperçu des modifications

Version	Date	Commentaire
0.1	20 mai 2014	Première version du syllabus
0.2	31 Juillet 2014	Réduction de l'introduction et des chapitres rappelant les concepts généraux de l'Agile
0.3	9 septembre 2014	Nouvelle version du syllabus après la première revue
0.9	15 décembre 2014	Nouvelle version du syllabus après la deuxième revue
1.0	31 Janvier 2015	Version finale 1.0
1.1	31 Mai 2015	Quelques corrections mineures

Remerciements

Ce document a été réalisé par une équipe du « Requirements Engineering Qualifications Board » - groupe de travail au niveau avancé.

A la fin de l'écriture de ce syllabus Praticien Agile, l'équipe de base était composée de :

Salvatore Reale (Agile Stream Lead), Alain Ribault (Chair de WP), Folke Nilsson (Président REQB), Alexander Lindner.

Auteur : Salvatore Reale.

Relecteurs : Werner Henschelchen, Beata Karpinska, Alexander Lindner, Judy McKay, Folke Nilsson, Ingvar Nordström, Tal peer, Alain Ribault.

L'équipe remercie les membres des comités nationaux qui ont soumis des contributions et des suggestions.

Traduction CFTL 2016.

© GASQ – Global Association for Software Quality

Table des matières

0 Introduction 6
0.1 Objectif du Syllabus 6
0.2 Examen 6
0.3 Accréditation 7
0.4 Internationalité et traduction 7
0.5 Avantages pour le métier 8
0.6 Objectifs d'apprentissage 10
0.7 Avantages pour le métier 10
0.8 Niveau de détail 11
0.9 Organisation du Syllabus 11
1 Introduction au développement logiciel Agile (80 minutes) 12
1.1 Principes fondamentaux du développement logiciel Agile 12
1.1.1 le Manifeste pour le développement logiciel Agile 13
1.1.2 Méthodes Agile de développement logiciels 14
1.1.3 Approche de l'équipe Agile 17
1.2 Aspects généraux des exigences logicielles Agile 18
1.2.1 Planification de Release et planification d'Itération 18
1.2.2 Intégration continue 19
1.2.3 Feedback précoce et fréquent 19
1.2.4 Rétrospectives 20
1.3 Applicabilité de standards et de normes au développement Agile 20
2 Exigences en contexte Agile (90 minutes) 22
2.1 Les exigences dans l'organisation Agile 22
2.1.1 Niveau Métier 26
2.1.2 Niveau Programme 27
2.1.3 Niveau équipe 28
2.2 Attributs des exigences Agile 29
2.3 Qualité des exigences Agile 30
2.4 Ingénierie des Exigences en développement logiciel Agile 31
2.4.1 Facteurs de réussite dans les projets Agile 32
2.4.2 Facteurs de risque dans les projets Agile 32
3 Rôles en Ingénierie des exigences Agile (65 minutes) 33

© GASQ – Global Association for Software Quality

3.1 les parties prenantes 34
3.2 Le Product Owner 36
3.3 L'équipe de développement 37
3.4 Le Scrum Master 37
4 Développement des exigences Agile (240 minutes) 38
4.1 Elicitation des exigences 39
4.1.1 L'atelier d'exigences 41
4.1.2 Vision 42
4.1.3 Contrats 43
4.1.4 Cas d'utilisation 44
4.1.5 Exigences non fonctionnelles 45
4.2 Analyse des exigences 46
4.2.1 L'atelier de planification de Release 47
4.2.2 Infrastructures et implémentation des User Story 48
4.2.3 L'atelier d'analyse 50
4.2.4 Le Spike 51
4.2.5 Acceptation de l'exigence 52
4.2.6 L'atelier de Grooming 53
4.2.7 Modélisation de la solution en Agile 54
4.3 Spécification des exigences 54
4.3.1 Spécification d'exigence par la création et le raffinement collaboratif des User Story 55
4.3.2 Affecter les conditions de satisfaction aux User Story raffinées 58
4.3.3 Consacrer du temps pour l'expérience utilisateur 58
4.3.4 Attribuer les critères d'acceptation 59
4.4 Validation et vérification des exigences 59
4.4.1 La revue de Sprint 60
4.4.2 la rétrospective 60
4.5 Estimation des User Story 61
4.5.1 Valeur métier 61
4.5.2 Priorité client 62
4.5.3 Taille relative et complexité 62
4.5.4 Risque 63
4.5.5 Effort 63
5 Gestion des exigences Agile (70 minutes) 64

© GASQ – Global Association for Software Quality

5.1 Gestion de projet et des risques 64
5.2 Traçabilité des exigences Agile 66
5.3 Gestion de configuration et du changement 67
5.4 Assurance Qualité 68
6 Outils et artefacts en ingénierie des exigences Agile (80 minutes) 70
6.1.2 Le graphique Burndown de Release 72
6.1.3 Exigences émergentes : le Parking Lot 73
6.2 Outils prenant en charge l'ingénierie des exigences Agile 73
6.2.1 Vote : Le Planning Poker 73
6.2.2 Minutage 74
6.2.3 Outils de support de la Communication 75
6.2.4 Autres outils prenant en charge l'ingénierie des exigences Agile 76
7 Références 76
7.1 Tableaux/Figures 76
7.2 Normes 77
7.3 Documents REQB 77
7.4 Livres et Publications 77
7.5 Autres références 78
Index 80

© GASQ – Global Association for Software Quality

0 Introduction

0.1 Objectif du Syllabus

Ce syllabus définit le programme de formation pour devenir un professionnel certifié REQB en ingénierie des exigences (CPRE) au niveau praticien en ingénierie des exigences Agile. REQB a mis au point ce syllabus en collaboration avec le GASQ. Le périmètre du syllabus REQB couvre le processus de l'ingénierie des exigences pour tous les types de produits informatiques qu'ils soient logiciels, matériels, services, processus métier et documentation. Dans ce cadre, le syllabus praticien Agile couvre le processus de l'ingénierie des exigences pour les organisations de développement logiciel mettant en œuvre les méthodologies Agile.

REQB fournit ce syllabus comme suit :

1. Au comités nationaux (CFTL pour la France), pour traduire dans leur langue locale et accréditer les organismes de formation. La traduction peut inclure d'adapter le syllabus au langage et de modifier les références (ouvrages et publications) pour s'adapter aux publications locales.
2. Au comités d'examens, pour créer dans leur langue, des questions d'examen adaptées aux objectifs d'apprentissage définis dans ce syllabus.
3. Aux fournisseurs de formation, demandant l'accréditation REQB pour les prestataires de formation, pour produire des supports de formation. Tous les domaines présentés de ce syllabus doivent être incorporés dans les supports de formation.
4. Aux candidats à la Certification, comme matériel de préparation pour l'examen de certification (dans le cadre d'un cours de formation agréé, ou indépendant).
5. A la communauté internationale d'ingénierie des exigences, pour faire avancer la profession d'ingénieur en exigences.

0.2 Examen

L'examen pour être certifié professionnel en ingénierie des exigences - praticien Agile - est basé sur ce syllabus. Toutes les sections de ce syllabus peuvent ainsi être traitées par les questions d'examen. Les questions d'examen ne proviennent pas nécessairement d'une section individuelle ; une question peut faire référence à plusieurs sections.

Le format des questions de l'examen est à choix multiple.

Les examens peuvent être passés après avoir assisté à un cours accrédité ou dans des sessions d'examen ouvertes (sans cours au préalable). Les informations détaillées au sujet des temps d'examen se trouvent sur le site Web de REQB (www.reqb.org), sur le site Web de GASQ (www.gasq.org) ou sur le site Web d'un fournisseur d'examen.

Remarque :

Ce syllabus se réfère aux notions de base d'ingénierie des exigences décrites dans le syllabus niveau Fondation. Les questions d'examen, ainsi que la terminologie et les thèmes abordés durant la formation, peuvent faire référence au contenu du syllabus niveau Fondation [REQB_FL_SYL], donc

© GASQ – Global Association for Software Quality

une lecture de ce document [REQB_FL_SYL] ou la connaissance des concepts qu'il contient est recommandée.

0.3 Accréditation

Les fournisseurs d'un cours Professionnel certifié REQB en ingénierie des exigences - praticien Agile doivent être accrédités par le GASQ. Leurs experts examinent la documentation du fournisseur de formation pour vérifier son exactitude et sa conformité avec le contenu et les objectifs d'apprentissage du syllabus. Un cours agréé est considéré comme étant conforme au syllabus. À la fin d'un tel cours, un examen « professionnel certifié en ingénierie des exigences » (examen CPRE) peut être effectué par un Institut de certification indépendant (selon les règles ISO 17024).

Les organismes de formation accrédités peuvent être identifiés par le logo officiel REQB Organisme de formation accréditée.

0.4 Internationalité et traduction

Ce syllabus a été développé par coopération entre experts internationaux. Le contenu de ce syllabus peut donc être considéré comme une norme internationale. Le syllabus permet la formation et l'évaluation avec un même niveau à l'échelle internationale.

Certains choix ont été fait concernant la traduction française. Les termes utilisés dans le syllabus sont le reflet des termes utilisés dans les organisations. Même si certaines organisations ont traduit en français tous les termes Agile, la majorité d'entre elles utilisent des termes anglo-saxon pour certains mots. Nous avons conservé ici les termes :
Backlog, Product Owner, Grooming, User Story et Story, Release, Spike, Timeboxing, Burndown, Epic.

© GASQ – Global Association for Software Quality

0.5 Avantages pour le métier

Les objectifs et avantages du syllabus REQB Professionnel Certifié en ingénierie des exigences - praticien Agile sont présentés dans le tableau suivant.

Objectifs	Avantages
Obtenir une nouvelle qualification clé	Toute solution logicielle, matérielle ou service est basée sur les besoins des parties prenantes et vise à satisfaire les besoins métier spécifiques. Pour être en mesure de livrer une solution compatible avec les besoins du groupe cible, une ingénierie des exigences appropriée est nécessaire. Depuis 2001, date de publication du Manifeste Agile, le nombre de sociétés, organisations, groupes de pays qui adoptent les méthodes Agile pour le développement de leurs logiciels a augmenté d'une année à l'autre. Gérer les exigences et les développer en concevant la solution dans un contexte Agile devient un besoin croissant. Tous ces aspects sont couverts par un praticien Agile.
Accroître la satisfaction de vos clients	La satisfaction du client est atteinte quand la solution répond à son attente et à ses besoins. Une ingénierie des exigences éprouvée minimise les écarts entre les attentes et la perception de la conformité à ces besoins. Le premier principe du développement logiciel Agile met en priorité haute la satisfaction du client par le biais de livraisons du logiciel rapides et continues. L'ingénierie des exigences Agile fournit les moyens d'améliorer la qualité du produit final et de renforcer la fidélité des clients en leur fournissant ce qu'ils veulent.
Minimiser les coûts de développement et de suivi	Une ingénierie des exigences de qualité en développement logiciel Agile, grâce à des incréments potentiellement expédiables et par le biais de cycles de feedback fréquents, minimise les risques projet et produit, et permet d'éviter les coûts associés à retravailler suite à des divergences entre les attentes du client et la solution livrée. L'ingénierie des exigences Agile contribue à minimiser les coûts des changements et des actions correctives.
Avantage concurrentiel	La valeur de tout élément sur le marché diminue au fil du temps. Par conséquent, pour obtenir le maximum de profit, une organisation devrait être la première sur le marché, ou du moins assez tôt. L'ingénierie des exigences Agile contribue à fournir les produits et les services plus rapidement, et à satisfaire tous les besoins et les attentes.

Table 1 Objectifs du syllabus Praticien Agile, ses bénéfices and focus principaux

Le niveau praticien Agile dans le Syllabus REQB Professionnel certifié en ingénierie des exigences est adapté à toute personne impliquée dans le développement de solutions produit/métier et la maintenance, y compris les analystes Métier et Système, les équipes marketing, les représentants des clients, les responsables Produit, les concepteurs et développeurs matériel/logiciel, les testeurs,

© GASQ – Global Association for Software Quality

chefs de projet, équipes techniques et de maintenance, Scrum Master, les auditeurs IT et les représentants de l'assurance qualité.

L'objectif principal du syllabus praticien Agile est de fournir une terminologie commune et une compréhension commune des notions clés liées aux processus d'ingénierie des exigences dans un contexte Agile. Les connaissances de base fournies dans le syllabus praticien Agile intègrent les définitions et les concepts communs liés à l'ingénierie des exigences Agile et explique les processus d'ingénierie des exigences Agile ainsi que leurs résultats escomptés. Le syllabus repose sur les normes généralement reconnues et conseillées. Le tableau 1 fournit une brève description des objectifs et des focus de ce syllabus.

Objectifs	**Bénéfices**
Eléments de base Agile	Rappeler les valeurs fondamentales et les principes du développement Agile, les méthodes les plus couramment utilisées, les flux et les pratiques clés d'un projet Agile. Contient des éléments sur l'applicabilité des règles et des normes à un contexte Agile.
Exigences Agile	Comprendre les concepts de base associés aux exigences Agile, leur classification et les niveaux d'abstraction ; expliquer la signification des attributs d'exigences et le rôle des critères qualité.
Des exigences aux User Stories	Expliquer le raffinement progressif d'une exigence jusqu'à ce qu'elle se divise en User Stories qui peuvent être implémentées et complètement délivrées dans une itération.
Rôles dans l'ingénierie des exigences Agile	Expliquer comment les rôles classiques d'une organisation Agile contribuent à l'ingénierie des exigences.
Développement des exigences agile	Comprendre les changements nécessaires dans l'élicitation des besoins, l'analyse, la spécification, la vérification et validation dans un contexte Agile.
Gestion des exigences Agile	Comprendre les changements nécessaires dans la gestion des risques, la traçabilité des exigences, la gestion de configuration et du changement et l'assurance de qualité dans un contexte Agile.
Outils et artefacts	Expliquer les artefacts de base et les outils de support d'ingénierie des exigences Agile et comment ils peuvent répondre aux besoins des organisations Agile habituelles.
Exercices	Identifier les parties prenantes ; les exigences ; raffiner en exigences de qualité au moyen des User Stories ayant les propriétés INVEST ; s'assurer de la qualité des exigences ; analyser et estimer des besoins ; vérifier et valider.

Table 2 brève description des objectifs et des focus de ce syllabus

© GASQ – Global Association for Software Quality

0.6 Objectifs d'apprentissage

Les objectifs d'apprentissage de ce syllabus ont été divisés en différents niveaux cognitifs de connaissance (K-Levels). Cela permet au candidat de connaître le « niveau de connaissance » de chaque point.

Chaque section de ce syllabus a un niveau cognitif associé :

- K1 - Compétences/connaissances : Connaissance de détails précis tels que les termes, définitions, faits, données, règles, principes, théories, caractéristiques, critères, procédures. Les étudiants sont en mesure de rappeler et d'exprimer des connaissances.
- K2 - Compréhension : les étudiants sont en mesure d'expliquer ou de résumer les faits avec leurs propres mots, de donner des exemples, de comprendre les contextes, d'interpréter les tâches.
- K3 - Appliquer : Les élèves sont capables d'appliquer leurs connaissances dans de nouvelles situations spécifiques, par exemple, en appliquant certaines règles, méthodes ou procédures.
- K4 - Analyser : Les élèves sont capables d'analyser de nouveaux problèmes spécifiques et de donner des solutions appropriées basées sur leurs différentes connaissances et compétences.

Remarque :

Aucun objectif d'apprentissage K4 n'est inclus dans cette version du syllabus.

0.7 Avantages pour le métier

Après avoir réussi l'examen REQB® Professionnel Certifié en Ingénierie des Exigences - Praticien Agile, une personne peut :

- Bo01 Communiquer les modèles et les concepts fondamentaux et comprendre les rôles dans le développement logiciel Agile pour le cycle de vie d'un produit.
- BO02 Comprendre et communiquer les concepts fondamentaux des exigences et leur application dans un cycle de vie d'un produit dans un contexte Agile.
- BO03 Sensibiliser sur la signification des processus d'ingénierie des exigences et des résultats escomptés en développement logiciel Agile.
- BO04 Faire connaître les attributs et la qualité des exigences en développement logiciel Agile.
- BO05 Identifier les parties prenantes et les impliquer dans les activités d'ingénierie des exigences : Collection, ateliers, planification, estimation, acceptation.
- BO06 Comprendre et communiquer les responsabilités, les tâches, les compétences et les contributions nécessaires aux processus d'ingénierie des exigences aux personnes jouant les rôles principaux dans un projet de développement Agile.

© GASQ – Global Association for Software Quality

- BO07 Comprendre comment améliorer les processus d'ingénierie des exigences dans le contexte Agile.
- BO08 Comprendre, communiquer et appliquer les différences dans le développement des exigences entre les processus de développement/maintenance Agile et traditionnels.
- BO09 Comprendre, communiquer et appliquer les différences dans la gestion des exigences entre les processus de développement/maintenance Agile et traditionnels.
- BO10 Comprendre et appliquer les critères d'estimation pour les exigences dans le contexte Agile.
- BO11 Communiquer l'importance et la mise en pratique des artefacts et des outils Agile pour garder un projet Agile sous contrôle et faciliter la gestion et le développement des exigences.

0.8 Niveau de détail

Le syllabus REQB Professionnel certifié en ingénierie des exigences - Praticien Agile est destiné à être la base de l'examen et à uniformiser les formations à l'échelle internationale. Ce syllabus comprend les éléments suivants pour atteindre cet objectif :

- Objectifs pédagogiques généraux décrivant l'intention du syllabus de certification REQB Praticien Agile.
- Objectifs d'apprentissage pour chaque domaine de connaissances qui décrit les résultats d'apprentissage cognitif de la formation et les qualifications que le participant doit atteindre.
- Une liste d'informations à enseigner, y compris une description et les références à des sources supplémentaires telles que la littérature technique acceptée, les normes ou standards, si nécessaires.
- Une liste des termes que les participants doivent être en mesure de rappeler et de comprendre. Les termes sont décrits en détail dans le document REQB " Glossaire standard des termes utilisés dans l'ingénierie des exigences ".

Le contenu du syllabus n'est pas une description de l'ensemble du domaine de connaissance de l'Ingénierie des exigences. Il apporte un niveau de détail pertinent pour la certification Praticien Agile.

0.9 Organisation du Syllabus

Il est composé de six grands chapitres et d'un dernier pour les références.

Le titre de niveau supérieur de chaque chapitre présente le sujet couvert par le chapitre et spécifie la durée minimale qu'un cours agréé doit passer sur le chapitre.

Les objectifs d'apprentissage à respecter figurent au début de chaque chapitre.

Chaque chapitre est décomposé en sections. Un cours agréé doit passer une durée minimale sur chaque section. Les paragraphes qui n'ont pas une durée associée sont inclus dans la durée globale de la section.

© GASQ – Global Association for Software Quality

1 Introduction au développement logiciel Agile (80 minutes)

Mots clés

Manifeste agile, développement logiciel Agile, équipe Agile, livraison continue, intégration continue, équipe interfonctionnelle, équipe de développement, Extreme Programming, kanban, modèle incrémental, planification des itérations, modèle itératif, Backlog de Produit, incrément produit, Product Owner, planification de Release, rétrospective, Scrum, Scrum Master, Sprint, timebox, User Story, approche équipe intégrée

Objectifs d'apprentissage pour l'Introduction au développement logiciel Agile

1.1 Les principes fondamentaux du développement logiciel Agile (35 minutes)

AR-1.1.1 Rappeler les valeurs fondamentales et les principes du développement logiciel Agile basés sur le Manifeste Agile(K1)

AR-1.1.2 Rappeler les approches de développement logiciel Agile (K1)

AR-1.1.3 Comprendre les avantages d'une équipe interfonctionnelle et l'approche d'équipe intégrée (K2)

1.2 Aspects généraux du développement logiciel Agile (45 minutes)

AR-1.2.1 Rappeler les bases des User Story, la planification de Release et d'itérations (K1)

AR-1.2.2 Rappeler les avantages de l'intégration continue (K1)

AR-1.2.3 Rappeler les avantages des feedback précoces et fréquents (K1)

AR-1.2.4 Rappeler comment les rétrospectives peuvent être utilisées comme mécanisme d'amélioration de processus dans les projets Agile (K1)

1.1 Principes fondamentaux du développement logiciel Agile

Le travail et le rôle d'un ingénieur en exigences seront différents sur un projet traditionnel. Les processus et les pratiques de développement et de gestion des exigences seront aussi différents de que ceux d'un projet traditionnel, mais sont encore nécessaires pour parvenir à des résultats positifs dans un projet Agile.

Les Ingénieurs en exigences doivent comprendre les valeurs et les principes qui sous-tendent les projets Agile, et comment les ingénieurs en exigences font partie intégrante d'une approche équipe intégrée avec les développeurs, les testeurs et les représentants Métier.

© GASQ – Global Association for Software Quality

1.1.1 le Manifeste pour le développement logiciel Agile

Les racines du développement logiciel Agile remontent à 2001, quand un groupe de dix-sept personnes ayant une longue expérience dans les modèles légers de développement de logiciels ont convenu de quatre valeurs et 12 principes qui forment le Manifeste Agile [Agile Manifesto, 2011].

Le Manifeste Agile met l'accent sur l'importance des quatre valeurs Agile pour obtenir des succès durables et profitables :

1. Les individus et leurs interactions plus que les processus et outils
2. Un logiciel opérationnel plus qu'une documentation exhaustive
3. La collaboration avec les clients plus que la négociation contractuelle
4. L'adaptation au changement plus que le suivi d'un plan

Le manifeste Agile fournit également les douze principes Agile qui sont la référence de base pour toute méthode Agile :

1. Notre plus haute priorité est de satisfaire le client en livrant rapidement et régulièrement des fonctionnalités à grande valeur ajoutée.
2. Accueillez positivement les changements de besoins, même tard dans le projet. Les processus Agile exploitent le changement pour donner un avantage compétitif au client.
3. Livrez fréquemment un logiciel opérationnel avec des cycles de quelques semaines à quelques mois et une préférence pour les plus courts.
4. Les utilisateurs ou leurs représentants et les développeurs doivent travailler ensemble quotidiennement tout au long du projet.
5. Réalisez les projets avec des personnes motivées. Fournissez-leur l'environnement et l'assistance dont ils ont besoin et faites-leur confiance pour atteindre les objectifs fixés.
6. La méthode la plus simple et la plus efficace pour transmettre de l'information à l'équipe de développement et à l'intérieur de celle-ci est le dialogue en face à face.
7. Un logiciel opérationnel est la principale mesure d'avancement.
8. Les processus Agile encouragent un rythme de développement soutenable. Ensemble, les commanditaires, les développeurs et les utilisateurs devraient être capables de maintenir indéfiniment un rythme constant.
9. Une attention continue à l'excellence technique et à une bonne conception renforce l'Agilité.
10. La simplicité – c'est-à-dire l'art de minimiser la quantité de travail inutile – est essentielle.
11. Les meilleures architectures, spécifications et conceptions émergent d'équipes auto organisées.
12. À intervalles réguliers, l'équipe réfléchit aux moyens de devenir plus efficace, puis règle et modifie son comportement en conséquence.

Les mots soulignés montrent clairement la forte relation entre les principes Agile et l'ingénierie des exigences.

Les différentes méthodes Agile fournissent des règles et des pratiques pour mettre ces valeurs et principes Agile en action [Schwaber, 2001], [Beck, Andres, 2004], [Anderson, 2010].

© GASQ – Global Association for Software Quality

1.1.2 Méthodes Agile de développement logiciels

Il n'y a pas une approche unique de développement logiciel Agile, mais différentes méthodes. Chacune d'entre elles implémentent les valeurs et les principes du manifeste Agile de différentes manières. Selon les enquêtes les plus récentes sur l'adoption des méthodes Agile, les trois méthodes Agile les plus populaires sont Scrum, Extreme Programming et Kanban. Chacune d'entre elles est traitée dans ce chapitre.

Scrum est adopté dans plus de 50 % des cas. Après Scrum, XP, Kanban et des hybrides des deux derniers et Scrum représentent l'autre 50 %. Les autres méthodes sont adoptées à des pourcentages inférieurs.

Scrum

Scrum est un framework itératif et incrémental pour le développement de produits, d'applications ou de projets.

Il structure le développement dans les itérations appelées Sprints. Ces itérations durent de 1 à 4 semaines et se déroulent les uns après les autres. Les Sprints sont de durée déterminée – ils finissent à une date déterminée, que le travail ait été effectué ou non et ne sont jamais étendu ; ils sont timeboxés.

Au début de chaque Sprint, une équipe interfonctionnelle sélectionne les éléments (exigences client) dans une liste priorisée. La liste peut avoir changé depuis le Sprint précédent, ce qui permet une adaptation rapide aux changements. L'équipe s'engage à réaliser les éléments pour la fin du Sprint.

En cours de Sprint, les éléments choisis ne changent pas. Tous les jours l'équipe se rassemble brièvement pour échanger sur l'avancement, et mettre à jour des graphiques simples qui orientent le travail restant. Cette réunion quotidienne est appelée mêlée quotidienne ou stand-up meeting.

L'artefact Scrum le plus important est l'incrément produit. Chaque Sprint produit un incrément produit. L'incrément produit doit être de qualité suffisante pour être donné aux utilisateurs. Lorsque l'incrément produit est livré, il doit être « fini » selon une compréhension partagée de ce que "fini" signifie. Cette définition est différente pour chaque équipe Scrum, et plus l'équipe monte en maturité, la définition de « fini » s'élargit et devient plus stricte. La définition de « fini » doit toujours comprendre que l'incrément produit doit être de qualité suffisamment élevée pour être livré si l'équipe choisit de le faire immédiatement. L'incrément produit comprend les fonctionnalités de tous les incréments produit précédents et est entièrement testé afin que tous les éléments terminés continuent à fonctionner ensemble.

À la fin du Sprint, l'équipe fait une revue du Sprint avec les parties prenantes et fait la démonstration de ce qu'elle a construit. Suite à cette démonstration, l'équipe reçoit un feedback qui peut être utilisé dans le prochain Sprint. Scrum met l'accent sur l'obtention d'un produit fonctionnel à la fin du Sprint qui soit vraiment « fini ». Dans le cas des logiciels, cela signifie que le code est intégré, entièrement testé et potentiellement expédiable.

Un thème majeur dans Scrum est « inspecter et s'adapter » grâce aux rétrospectives. Puisque le développement implique inévitablement l'apprentissage, l'innovation et les surprises, Scrum met en avant de prendre un peu de temps pendant le développement, pour inspecter à la fois le produit et l'efficacité des pratiques pour ensuite adapter les objectifs produit et les pratiques des processus.

Scrum est un processus d'équipe. L'équipe Scrum inclut trois rôles, le Product Owner responsable du produit, les membres de l'équipe de développement et le Scrum Master.

© GASQ – Global Association for Software Quality

Le Backlog de Produit est un artefact indispensable dans Scrum. Le Backlog de Produit est une liste ordonnée d'idées relatives au produit, dans l'ordre que l'équipe s'attend à adresser selon la priorité donnée par la valeur Métier, la priorité des clients, les risques et ainsi de suite. C'est la source unique d'où découlent toutes les exigences. Cela signifie que tout le travail de l'équipe de développement provient du Backlog de Produit. Chaque idée de fonctionnalité, d'amélioration, de correction d'anomalie, d'exigence de documentation – tous les bouts de travail qu'elle fait– est dérivé d'un élément de Backlog de Produit. Chaque élément priorisé du Backlog de Produit inclut une description ainsi qu'une estimation.

Le Product Owner représente le client et génère, maintient et priorise le Backlog de Produit. Cette personne n'est pas le chef de l'équipe.

L'équipe de développement est faite de professionnels qui font le travail de délivrer l'incrément produit. L'équipe doit comprendre la vision et les objectifs du Sprint pour le Product Owner afin de livrer des incréments produit potentiellement expédiables. Ils s'auto-organisent pour accomplir le travail.

Scrum requiert que l'équipe de développement soit un groupe interfonctionnel de personnes qui ont parmi eux toutes les compétences nécessaires pour délivrer chaque incrément du produit.

Le Backlog de Sprint est la liste raffinée des éléments de Backlog de Produit choisis pour le développement lors d'un Sprint. Il correspond aux prévisions de l'équipe du travail qui peut être réalisé. Une fois le Backlog de Sprint en place, le Sprint commence, et l'équipe de développement développe le nouvel incrément produit tel que défini dans le Backlog de Sprint.

Les équipes Scrum n'insistent pas sur les rôles traditionnels du génie logiciel, comme programmeur, concepteur, testeur ou architecte. Toutes les personnes sur le projet travaillent ensemble pour réaliser le travail pour lequel ils se sont engagés collectivement dans un Sprint.

Scrum. Le Scrum Master a les responsabilités suivantes :

- S'assurer que l'équipe Scrum respecte les valeurs et les pratiques de Scrum
- Etre facilitateur et coacher l'équipe
- Protéger l'équipe d'agressions extérieures
- Contrôler les cycles « inspecter et s'adapter » de Scrum
- Enseigner le processus, challenger les habitudes et encourager de nouvelles pensées et comportements
- Aider les personnes extérieures à l'équipe à comprendre le processus et à comprendre les interactions avec l'équipe qui sont utiles et celles qui ne le sont pas.

Les Scrum Masters ne sont <u>pas</u> les responsables des équipes Scrum.

L'Extreme Programming

Extreme Programming (XP) [Beck, Andres, 2004] est une approche de développement Agile décrite par certaines valeurs, principes et pratiques de développement.

XP englobe cinq valeurs pour guider le développement :

Communication, simplicité, respect, feedback, courage

XP décrit un ensemble de principes comme lignes directrices supplémentaires :

Humanité, économie, bénéfice mutuel, self-similarité, amélioration, diversité, reflet, flux, occasion, redondance, échec, qualité, pas de bébé, responsabilité acceptée

© GASQ – Global Association for Software Quality

XP décrit treize pratiques primaires :

S'asseoir ensemble, toute l'équipe, espace de travail informatif, travail énergique, programmation en binôme, Stories, cycle hebdomadaire, cycle trimestriel, mou, construction en dix minutes, intégration continue, test avant la programmation, conception incrémentale

Les pratiques clés de XP incluent ce qui suit :

- Une équipe de cinq à dix programmeurs travaillent à un endroit avec un représentant du client sur place.
- Le développement se fait par versions fréquentes ou itérations, qui peuvent ou pas être livrables et qui offrent des fonctionnalités supplémentaires.
- Les exigences sont spécifiées sous forme de User Story, chacune étant une partie des nouvelles fonctionnalités de l'utilisateur.
- Les programmeurs travaillent en binôme, suivent des normes de codage strictes et font leurs propres tests unitaires. Les clients participent aux tests d'acceptation.
- Les exigences, l'architecture et la conception émergent au cours du projet.

XP a une portée normative et est généralement appliquée en petites équipes de moins de dix développeurs, où le client fait partie intégrante de l'équipe ou est facilement accessible. En outre, contrairement aux autres méthodes, XP décrit certaines pratiques strictes de codage qui ont été conçues pour produire des livrables de très haute qualité.

La plupart des approches de développement logiciel Agile utilisées aujourd'hui sont influencées par XP, ses valeurs et ses principes. Par exemple, les équipes Agile suivant Scrum intègrent souvent les pratiques XP.

Kanban

Inspiré par le mouvement du développement Lean, Kanban [Anderson, 2010] (le mot signifie « signal » en japonais) est basé exclusivement sur les principes Lean [Leffingwell, 2011]. C'est le nom d'un moyen de planification et de gestion du travail qui voit son utilisation croitre dans la communauté Agile.

Tel que défini par le Limited WIP (Work-in-Progress - Travail en cours) society, un système logiciel Kanban a les caractéristiques suivantes.

- Visualiser des unités de valeur. Cette unité de valeur pourrait être une User Story, une fonction métier minime, une exigence ou quelque chose d'autre. C'est différent d'un tableau de tâches qui généralement se focalise sur la visualisation des tâches courantes.
- Gérer le flux de ces unités de valeur, grâce à l'utilisation de limites WIP.
- Traiter ces unités de valeur à travers l'ensemble du système, de quand ils passent sous le contrôle de l'équipe jusqu'à ce qu'ils le quittent.
- En rassemblant ces trois propriétés d'un système Kanban, Kanban permet de faire circuler ces unités de valeur à travers le système à l'aide de limites WIP et de créer un flux constant travail.
- De plus, les limites WIP fournissent un mécanisme au système Kanban pour montrer quand il y a de nouvelles capacités de travail disponibles, créant ainsi un système de traction.
- Enfin, les limites WIP peuvent être ajustées et leurs effets mesurés et permettre au système Kanban de s'améliorer continuellement.

Le système de traction de Kanban tend à montrer rapidement les obstacles, les problèmes de blocage et les goulets d'étranglement dans le flux (qui peuvent résulter soit d'une contrainte de capacité, soit

© GASQ – Global Association for Software Quality

d'une non disponibilité d'une ressource à un moment donné). L'équipe peut alors améliorer ses processus. Par rapport à une approche plus traditionnelle du changement, cette approche adaptative, visible, peut diminuer les résistances et accélérer l'amélioration des capacités.

Kanban utilise trois instruments [Linz, 2014] :

- La carte kanban : La chaîne de valeur à gérer est visualisée par un tableau Kanban. Chaque colonne affiche une station, qui est un ensemble d'activités, p. ex., développement, tests. Les éléments devant être produits ou les tâches à réaliser sont symbolisées par des tickets qui vont de gauche à droite sur le tableau par le biais des stations.
- Limite de Travail en Cours: la quantité de tâches actives parallèles est strictement limitée. Ceci est contrôlé par le nombre maximal de tickets autorisé pour une station et/ou globalement par le tableau. Chaque fois qu'une station a de la capacité, un ticket est tiré de la station précédente.
- Délai d'exécution : Kanban est utilisé pour optimiser le flux continu de tâches en minimisant le délai (moyen) pour le flux de valeur complet.

Les caractéristiques de Kanban n'ont que quelques similitudes avec Scrum. Dans les deux frameworks, visualiser les tâches en cours (par exemple, sur un tableau blanc public) assure la transparence du contenu et l'avancement des tâches. Les tâches non encore planifiées attendent dans un Backlog et se déplacent sur le tableau Kanban dès qu'un nouvel espace (capacité de production) est disponible.

Les itérations ou Sprints sont facultatifs avec Kanban. Le processus Kanban permet de délivrer élément par élément, plutôt que dans le cadre d'une Release. Le Timeboxing comme un mécanisme de synchronisation est donc facultatif, contrairement à Scrum qui synchronise toutes les tâches dans un Sprint. Certains considèrent Kanban une intégration utile et un complément à Scrum, tout comme les autres méthodes telles que XP, d'où la popularité croissante des méthodes « hybrides » telles que Scrum/XP et "Scrumban".

Pour les organismes de formation : Parler des points communs (propriétés communes Agile) et des différences (particularités des méthodes) entre les méthodes Agile décrites ci-dessus.

1.1.3 Approche de l'équipe Agile

L'équipe interfonctionnelle possède des connaissances suffisantes sur les différentes technologies et composants logiciels pour pouvoir produire, à partir d'un point de vue client, des fonctionnalités significatives, soit pour un produit complet soit pour le développement d'un ensemble d'exigences. Idéalement, l'équipe interfonctionnelle est véritablement interfonctionnelle, et pas que pour le développement. Avoir une équipe interdisciplinée est la première étape pour y parvenir. Cette équipe comprend des représentants du client et d'autres parties prenantes Métier qui déterminent les caractéristiques du produit.

La taille de l'équipe est généralement de cinq à neuf personnes. Idéalement, l'équipe partage le même espace de travail, car la co-implantation facilite fortement la communication et l'interaction. Lorsque l'équipe est distribuée et ne peut pas partager le même espace de travail, les technologies de communication actuelles aident en facilitant les interactions en face à face, même quand de longues distances séparent les membres de l'équipe.

L'approche équipe interfonctionnelle se fait par le biais des Stand Up meeting quotidiens (daily Scrum), qui impliquent tous les membres de l'équipe, dans lesquels le travail en cours est communiqué et les obstacles à l'avancement mis en évidence.

© GASQ – Global Association for Software Quality

Toute l'équipe est responsable de la qualité dans les projets Agile. Les représentants Métier, par conséquent, travailleront en étroite collaboration avec les développeurs et les testeurs pour s'assurer que les niveaux de qualité désirés pour les besoins et les incréments produit ont été obtenus. Toute l'équipe participe à des consultations ou à des réunions dans lesquelles les caractéristiques du produit sont présentées, analysées ou estimées [ISTQB_FA_SYL]. Le concept de faire participer les testeurs, développeurs et les représentants Métier à toutes les discussions sur les fonctionnalités a été appelé le "pouvoir des trois » [Crispin, Gregory, 2008]. Cette approche d'équipe intégrée a un certain nombre d'avantages dont :

- L'amélioration de la communication et de la collaboration au sein de l'équipe
- Permettre aux différentes compétences au sein de l'équipe d'être exploitées au profit du projet
- Faire de la qualité la responsabilité de tous

Ces avantages favorisent une dynamique d'équipe plus efficace et efficiente.

1.2 Aspects généraux des exigences logicielles Agile

1.2.1 Planification de Release et planification d'Itération

Les User Story (story pour faire court) remplacent généralement en Agile ce qui traditionnellement est dénommé « exigences logicielles » [REQB_APPROACH], [REQB_FL_SYL].

Les User Story sont utilisées en développement logiciel Agile comme base pour définir les fonctions qu'un système doit fournir et faciliter la gestion des exigences. Elle capture le « qui », « quoi » et « pourquoi » d'une exigence d'une manière simple et concise, souvent limitée en niveau de détail par ce qui peut être écrit à la main sur une fiche de papier.

La forme habituelle pour une User Story est :

En tant que < rôle > je veux < action > de façon à < bénéfice attendu >

Par exemple : En tant que photographe débutant je veux voir affiché un classement fait par les utilisateurs de façon à mieux décider quel appareil acheter en comparant les différentes cotes.

Les User Story sont écrites par ou pour les utilisateurs Métier. La Story est le moyen principal d'un utilisateur pour influer sur le fonctionnement du système en cours d'élaboration. Les User Story peuvent également être écrites par les développeurs et les autres parties prenantes pour exprimer des exigences non fonctionnelles (p. ex., sécurité, performance, qualité).

Pour les cycles de vie Agile, deux types de planification existent, planification de Release et planification d'itération.

La planification de Release commence au lancement d'un produit, souvent quelques mois avant le début d'un projet. La planification de Release définit et redéfinit le Backlog de Produit et peut nécessiter de raffiner de plus grandes User Story (parfois appelés EPIC) en un ensemble de petites story.

La planification de Release est de haut niveau. Dans la planification de Release, le Product Owner établit et priorise les User Story, en collaboration avec l'équipe. Basés sur ces User Story, les risques projet et produit sont identifiés et une estimation de haut niveau de l'effort est effectuée.

Une fois la planification de Release terminée, la planification d'itération pour la première itération commence. La planification d'itération est faite pour une seule itération et se concentre sur la détermination du Backlog de Sprint.

© GASQ – Global Association for Software Quality

La planification d'itération est de bas niveau. Dans la planification d'itération, l'équipe sélectionne des User Story dans le Backlog de Release hiérarchisé, élabore les User Story, effectue une analyse de risque produit pour les User Story et estime les travaux nécessaires pour chaque User Story. Si une Story est trop vague, l'équipe peut refuser de l'accepter et utilise la prochaine User Story basée sur la priorité. Le Product Owner doit répondre à l'équipe sur chaque Story afin que les membres de l'équipe puissent comprendre ce qu'ils doivent implémenter et comment tester chaque Story.

Le nombre de story sélectionnées est basé sur la vélocité de l'équipe établie (la capacité de l'équipe à réaliser toutes les story d'une itération) et la taille estimée des User Story sélectionnées. Après que le contenu de l'itération ait été déterminé, les User Story sont divisées en tâches qui sont auto attribuées aux membres de l'équipe.

Les plans de Release peuvent changer en cours du projet, dont des modifications aux User Story individuelles dans le Backlog de Produit.

1.2.2 Intégration continue

La Livraison d'un incrément produit nécessite un logiciel fiable, fonctionnel, intégré à la fin de chaque Sprint. L'intégration continue adresse cela en fusionnant toutes les modifications apportées au logiciel et en intégrant régulièrement tous les composants changés, au moins une fois par jour. Gestion de la configuration, compilation, Build logiciel, déploiement et tests sont regroupés en un processus unique, automatisé et répétable. Chaque Build intégré est vérifié par un test de Build automatique qui détecte les erreurs d'intégration aussi rapidement que possible. Étant donné que les développeurs intègrent leur travail constamment, font le Build constamment et testent constamment, les erreurs dans le code sont détectées plus rapidement.

Un environnement d'intégration continue efficace est une composante de base de la livraison continue, la discipline de développement logiciel où le logiciel est construit de telle sorte qu'il peut être mis en production à tout moment. La livraison continue a pour but de réduire les coûts, le temps et le risque en livrant des modifications incrémentales aux utilisateurs [Humble, Farley, 2010]. La livraison continue est une composante fondamentale du modèle itératif Agile.

1.2.3 Feedback précoce et fréquent

Les projets Agile ont des itérations courtes permettant à l'équipe projet de recevoir un feedback rapide et continu sur la qualité du produit tout au long du cycle de développement. Une des façons de fournir un feedback rapide est l'intégration continue et la livraison continue. Quand des approches de développement non itératives sont utilisées, souvent le client ne voit pas le produit tant que le projet n'est pas presque terminé. À ce stade, il est souvent trop tard pour le l'équipe de développement de faire face efficacement à toute question que le client peut avoir. Recevoir un feedback client régulier en cours de projet, permet aux équipes Agile d'intégrer cette nouvelle information dans le processus de développement du produit. Cela maintient mettant l'accent sur les fonctionnalités de plus haute valeur métier, ou sur les risques associés, et celles-ci sont livrées au client en premier. Au travers des feedback fréquents, l'équipe Agile apprend aussi à connaître ses propres capacités. Par exemple :

- Quelle quantité de travail peut-on faire dans un Sprint ou une itération ?
- Qu'est ce qui pourrait nous aider à aller plus vite ?
- Qu'est-ce qui nous empêche de le faire ?

© GASQ – Global Association for Software Quality

Les avantages des feedback précoces et fréquents comprennent :

- Eviter les malentendus sur les exigences qui pourraient n'être décelés que plus tard dans le cycle de développement lorsqu'ils sont plus coûteux à corriger
- Clarifier tôt et régulièrement les demandes de fonctionnalités client tout au long du développement, rendant plus probable que les caractéristiques clé seront disponibles pour le client au plus tôt et que le produit reflète mieux ce que veut le client
- Découvrir (grâce à l'intégration continue) des problèmes de qualité dès le début, les isoler plus facilement et les résoudre plus tard dans le cycle de développement
- Fournir des informations à l'équipe Agile au sujet de sa productivité et de sa capacité à livrer

Ces avantages favorisent une dynamique projet cohérente [ISTQB_FA_SYL]. Voir aussi 4.4

1.2.4 Rétrospectives

Comme indiqué dans le douzième principe du manifeste Agile, inspecter et s'adapter est une pratique clé que toutes les organisations Agile et que les équipes Scrum doivent adopter au cours de leur travail. En développement Agile, une rétrospective est une réunion qui se tient à la fin de chaque itération pour discuter ce qui a réussi, ce qui pourrait être amélioré et comment intégrer les améliorations tout en conservant durablement les réussites. Les rétrospectives couvrent des sujets tels que les processus, les personnes, les organisations, les relations et les outils. Des réunions de rétrospective menées régulièrement, avec des activités de suivi appropriées, sont essentielles à l'auto-organisation et à l'amélioration continue [Derby, Larsen, 2006]. Le suivi approprié et l'enregistrement des rétrospectives permettent à l'équipe de transférer ses expériences et ses actions d'amélioration aux autres équipes et aux autres parties prenantes, dans un processus continu d'amélioration et d'adaptation.
Il existe de nombreuses approches pour tenir des rétrospectives, et il est bon d'utiliser des approches différentes pour garder leur fraicheur et l'engagement des membres de l'équipe. Au début d'une rétrospective, pour donner le ton et créer un climat de confiance, il est important de rappeler aux membres de l'équipe qu'il faut créer un climat sain. Le Scrum Master facilite généralement la rétrospective. Voir aussi 4.4.2.

Dans une rétrospective réussie, l'équipe met l'accent sur les problèmes qu'elle peut traiter et crée des plans d'action pour prendre en compte les changements. Se concentrer sur les problèmes au sein de leur propre sphère d'influence oblige les membres de l'équipe à ne pas tourner continuellement la rétrospective en session de plainte. Créer un plan d'action et agir tout de suite dans le Sprint suivant donne à l'équipe un sentiment d'amélioration continue.

Pour les organismes de formation : Fournir quelques exemples de rétrospectives, éventuellement à partir de cas réels et expliquer comment mettre l'accent sur l'inspection et l'adaptation.

1.3 Applicabilité de standards et de normes au développement Agile

Les standards et les normes des processus qui sont utiles à l'ingénierie des exigences sont répertoriés dans [REQB_FL_SYL].

Concernant l'une des principales questions qui se pose (puisque le développement Agile a été adopté par un nombre croissant d'organisations), il a été précisé au cours de la dernière décennie que :

© GASQ – Global Association for Software Quality

- Les standards tels que ISO 9000 et CMMi sont traditionnellement pensés comme reposant sur de grandes quantités de documents administratifs. Le développement Agile a prouvé qu'il peut être appliqué efficacement à ces cadres structurés en mettant l'accent sur ce qui est fait et comment les objectifs Métier sont atteints. Il est important que l'auditeur sache ce qu'est Agile et comment Agile fonctionne généralement avec le standard à auditer.
- Les autres normes et standards répertoriés dans [REQB_FL_SYL] s'appliquent habituellement bien au développement Agile quand ils décrivent des modèles qualité et des concepts généraux pour l'ingénierie des exigences et des systèmes.
- Certaines normes et standards ne sont pas applicables par les équipes des organisations/projets Agile lorsque ces documents sont basés sur des modèles de développement spécifiques tels que les modèles séquentiels, souvent appelés "modèles de développement traditionnels".

© GASQ – Global Association for Software Quality

2 Exigences en contexte Agile (90 minutes)

Mots Clés

Organisation Agile, exigence Agile, ingénierie des exigences Agile, niveau Métier, valeur Métier, engagement, criticité, EPIC, fonctionnalité, INVEST, Backlog de portefeuille, priorité, niveau programme, Backlog programme, élément d'exigence, risque, niveau équipe, thème, vision

Objectifs d'apprentissage pour les exigences dans un contexte Agile

2.1 Les exigences dans l'organisation Agile (30 minutes)

AR-2.1.1 Comprendre le processus de développement et de livraison de logiciels, les équipes et les unités d'organisation dans les organisations Agile (K2)

AR-2.1.2 Décrire le niveau équipe, le niveau programme et le niveau métier dans le workflow d'ingénierie des exigences Agile (K2)

2.2 Attributs des exigences Agile (15 minutes)

AR-2.2.1 Comprendre les principaux attributs des exigences en développement logiciel Agile (K2)

2.3 Qualité des exigences Agile (15 minutes)

AR-2.3.1 Expliquer les critères qualité les plus importants des exigences en développement logiciel Agile (critères INVEST) (K2)

2.4 Ingénierie des exigences en développement logiciel Agile (30 minutes)

AR-2.4.1 Comprendre les différences entre l'ingénierie des exigences en Agile et dans le développement traditionnel de logiciels (K2)

AR-2.4.2 Expliquer quand une approche d'ingénierie des exigences Agile peut être bénéfique et quand elle peut être problématique (K2)

2.1 Les exigences dans l'organisation Agile

Le syllabus niveau Fondation REQB [REQB_FL_SYL] définit les concepts de problème, de solution et de produit pour introduire la classification des exigences. On trouvera la liste complète des termes et définitions dans [REQB_GLO].

Après avoir donné la définition des exigences à partir de l'IEEE 610, [REQB_FL_SYL] indique que l'un des principaux objectifs des exigences est d'exprimer les besoins et les attentes du client concernant la solution prévue. Les exigences servent également de base pour l'évaluation, la planification, l'exécution et le suivi des activités projet et font souvent partie des contrats de service, commandes ou contrats. Les exigences non seulement définissent ce qui doit être fait, mais établissent les limites de la solution, la livraison, le plan de livraison et les services contractuels.

© GASQ – Global Association for Software Quality

Le résumé ci-dessus reste tout à fait valide dans un environnement Agile, où il faut prendre soin des spécifications du produit, des aspects fonctionnels et non fonctionnels ainsi que des contraintes tant métier que techniques. Encore une fois, il y a des exigences internes et externes, avec des niveaux d'abstraction différents incluant les exigences métier, les exigences de solution/système et les exigences produit/composant.

Le but de ce chapitre est de donner une vue générale de comment, la définition, le but, la classification et le niveau d'abstraction des exigences s'appliquent à une organisation de développement Agile.

La définition et le concept d'une exigence, ainsi que la plupart des normes et des règlements qui définissent ses caractéristiques et ses limites, demeurent entièrement applicables dans un environnement de développement Agile.

La réalisation d'un contexte Agile, cependant, n'est pas simple et implique beaucoup de changements, surtout au sein d'une organisation habituée à un modèle traditionnel de développement. L'environnement Agile introduit de nouveaux termes, de nouveaux rôles, des changements de rôles existant, des changements dans les pratiques de développement et la gestion des exigences, et des différences fondamentales dans la façon dont l'équipe travaille et communique.

Plusieurs auteurs en littérature Agile [Cohn, 2010] et plusieurs figures de proue dans le développement Agile ont vérifié de nombreuses fois qu'un certain nombre d'organisations ont terminé une période de transition à partir du développement de logiciels traditionnel. Dans ces transitions, certains modèles courants comme le Lean et l'Agile de logiciels ont commencé à émerger. A partir de cette évidence, REQB propose ce qui peut s'appeler le « Workflow de l'ingénierie des exigences Agile » pour décrire le processus d'ingénierie des exigences dans le nouveau mécanisme de développement et de livraison, les nouvelles équipes et leur rôle dans les unités organisationnelles et certains des principaux rôles dans le paradigme Agile. REQB propose une image qui met en lumière les pratiques concernant les exigences au sein de ce modèle.

La figure (voir Figure 1) a l'avantage de mettre en évidence, par sa division en trois couches, comment le modèle s'applique à n'importe quel type de structure organisationnelle, de simples groupes de quelques personnes qui développent des applications simples pour le marché de la consommation, à des projets complexes ou à des organisations à produits multiples, typiques des grandes organisations.

Cette représentation sert à la fois de modèle d'organisation et de processus pour décrire la définition des exigences Agile, les rôles, pratiques et artefacts, qui seront détaillés plus loin dans ce syllabus.

© GASQ – Global Association for Software Quality

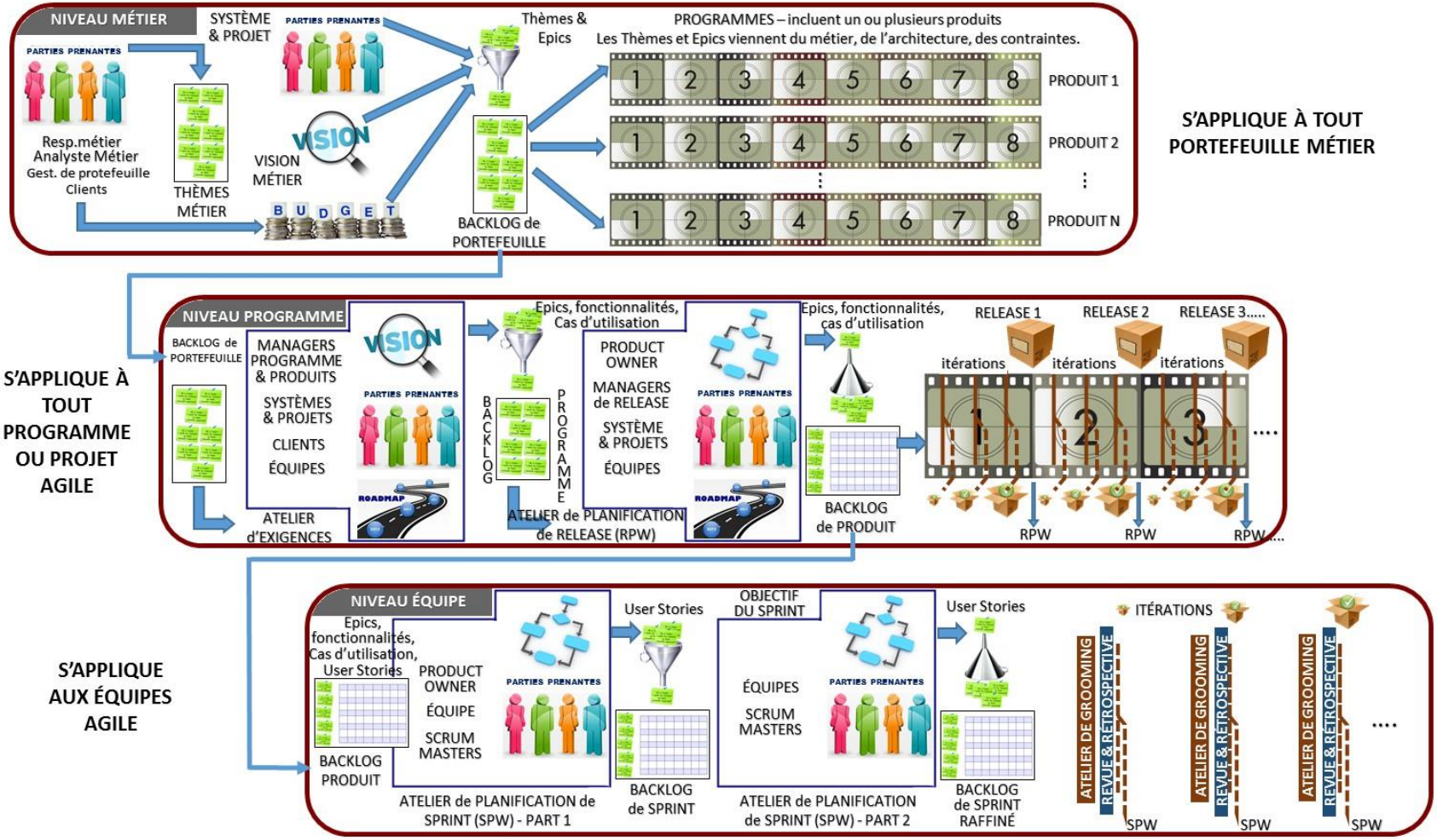


Figure 1 : Workflow d'ingénierie des exigences Agile

© GASQ – Global Association for Software Quality

La figure 1 contient les éléments suivants :

Un groupe de partie prenantes	Résultat de l'analyse et de la modélisation des tâches
Filtrage des exigences	Raffinage des exigences
Une séquence de Releases	Une feuille de route programme/produit
Un incrément potentiellement expédiable (à la fin d'une itération)	Un incrément de programme (à la fin d'une Release)

La figure 1 reprend les différents niveaux d'abstraction des exigences [REQB_APPROACH]. **La figure 2** illustre cette correspondance, la façon dont les exigences sont raffinées et comment elles évoluent au travers des niveaux d'une organisation Agile tout en augmentant le niveau de détail.

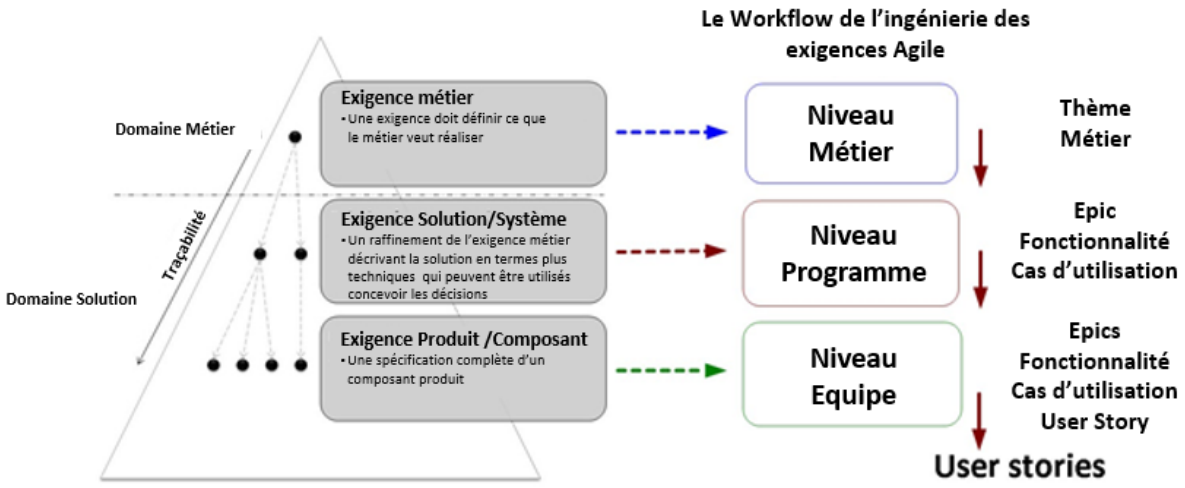


Figure 2 : Niveaux d'Abstraction des exigences mappés avec le workflow d'ingénierie des exigences Agile

Pour les organismes de formation : Donner des exemples de l'applicabilité du « Workflow d'ingénierie des exigences Agile » aux différentes tailles d'organisation.

© GASQ – Global Association for Software Quality

2.1.1 Niveau Métier

Au niveau métier, le niveau le plus élevé dans l'image, le portefeuille des produits de la société est géré. La gestion de portefeuille inclut tous les rôles qui traitent de la gestion des investissements de l'organisation en accord avec les analystes métier, qui devraient à leur tour être fonction des besoins et des perspectives du marché et des clients.

Les thèmes et la vision métier peuvent être considérés comme les résultats principaux du processus d'analyse métier concernant les besoins métier de l'organisation Agile pour le développement logiciel. En d'autres termes les thèmes de l'organisation au niveau métier décrivent les problèmes métier qui sont les éléments d'entrée du processus d'ingénierie des exigences.

L'ensemble des thèmes métier donne les objectifs d'investissement relatifs de l'organisation ou du département métier et priorisent l'investissement et le travail nécessaire pour l'organisation pour délivrer en fonction de sa stratégie métier. Ces thèmes apportent une vision à tous les programmes et sont exprimés en une série de large EPIC, qui seront affectés aux différentes Releases. Les thèmes métier sont des propositions de valeur produit clés, qui apportent une différenciation sur le marché et un avantage concurrentiel.

L'EPIC représente le plus haut niveau d'expression d'un besoin client. Les EPIC sont des initiatives de développement qui visent créer de la valeur pour un thème métier. Les EPIC sont identifiées, priorisée, estimées et maintenues dans le Backlog de portefeuille. Avant de planifier la Release, les EPIC sont raffinées en spécificités, qui à leur tour, sont converties en User Stories plus détaillées pour l'implémentation.

Les EPIC peuvent être exprimées sous forme de User Story, sous forme de listes, en une phrase ou deux, en vidéo, en prototype ou sous n'importe quelle forme utile pour exprimer les intentions de l'initiative de produit. L'objectif de l'EPIC est stratégique et non spécifique. En d'autres termes, l'EPIC doit être seulement détaillée pour permettre la discussion sur les types de fonctionnalités qu'elle nécessite.

Ces artefacts sont habituellement défendus par les gestionnaires de portefeuille, responsables de lignes de produit, spécialistes produits, ou d'autres qui ont des responsabilités fiduciaires avec les parties prenantes. Plus de détails sur le flux d'ingénierie des exigences au niveau Métier sont présentés dans les chapitres 4 et 5.

Un élément d'exigence comprend plusieurs niveaux, chacun voyant une augmentation du niveau de détail :

Thème = > Epic = > Fonctionnalité = > User Story

Ceux-ci couvrent les trois niveaux d'une organisation Agile complète qui développe des produits logiciels :

- Niveau métier
- Niveau de programme
- Niveau équipe

Tous les éléments d'exigence peuvent être facilement écrits sous forme de User Story. Les thèmes, EPIC et caractéristiques pourraient être écrits dans d'autres formats également (description courte, cas d'utilisation et ainsi de suite).

© GASQ – Global Association for Software Quality

2.1.2 Niveau Programme

Au niveau programme, le niveau intermédiaire dans l'image, le développement des fonctionnalités de haut niveau est effectué par plusieurs équipes dans un programme d'incrémentation Agile synchronisé. Il s'agit d'un cadencement standard d'itérations (1 à 4 semaines) à durée fixe (timeboxées) et de jalons qui ont une date et une qualité fixes, mais un périmètre variable. Dans les grandes organisations l'incrément de programme devrait produire des Releases ou des incréments potentiellement expédiables à des intervalles fréquents, généralement fixes, de 60 à 120-jours.

Ces incréments évaluables peuvent être livrés au client, ou pas, selon la capacité du client à absorber de nouveaux produits, ainsi que d'événements extérieurs qui peuvent influencer le calendrier. Dans les petites organisations ou les projets plus petits, souvent avec un public d'utilisateurs large (par exemple les applications web et mobiles), les Releases peuvent être plus fréquentes avec des itérations plus courtes.

Au sein de l'organisation, la gestion des produits, ou éventuellement la gestion des programmes ou la fonction d'analyste métier, est principalement responsable du maintien de la vision produits, systèmes ou applications dans leur domaine d'influence.

La vision apporte des réponses aux grandes questions sur le système, application ou produit, y compris ce qui suit :

- Qui sont les clients ?
- Quels sont les problèmes résolus ?
- Quels sont les avantages ?
- Quelles sont les fonctionnalités apportées ?
- Quelles propriétés comme la performance, la fiabilité et autres sont comprises ?
- Quelles plates-formes, normes et applications seront supportées ?

Le contenu principal de la vision est un ensemble priorisé de fonctionnalités destinées à offrir des avantages aux utilisateurs. En outre, la vision doit également contenir les diverses exigences non fonctionnelles, telles que la fiabilité, la performance, la facilité d'utilisation, la compatibilité aux standards, et ainsi de suite, qui sont nécessaires au système pour atteindre ses objectifs.

D'une manière analogue au Backlog de produit, qui contient principalement des User Story, le Backlog de programme contient l'ensemble des fonctionnalités souhaitées et priorisées qui n'ont pas encore été réalisées. Le Backlog de programme ne doit pas contenir des estimations pour les fonctionnalités, mais devrait avoir une priorité donnée en fonction du point de vue métier et de la priorité du client.

En Agile et Scrum en particulier, une EPIC est une grande User Story. Lorsque les User Story sont sous la forme d'EPIC, elles sont trop grosses pour être directement implémentées et elles devront être raffinées par l'équipe en petites story.

Les EPIC, fonctionnalités et les cas d'utilisation pourraient couvrir plusieurs produits et Release. Le processus de raffinement peut continuer jusqu'à ce que chaque User Story ait une valeur métier, une priorité, des critères d'acceptation et une estimation de l'effort indiquant qu'elle peut être réalisée en une seule itération. Les EPIC sont parfois traitées comme des cas d'utilisation ou des descriptions courtes (titres) des fonctionnalités.

Un thème est une grande collection de User Story. Les thèmes sont utilisés pour exprimer les macros fonctionnalités, les exigences de haut niveau et les objectifs d'investissements.

© GASQ – Global Association for Software Quality

2.1.3 Niveau équipe

Au niveau de l'équipe, le niveau inférieur sur l'image, les équipes Agile de 5 à 9 membres définissent, construisent et testent les nouvelles fonctionnalités et composants alignés sur les exigences fonctionnelles et non fonctionnelles, qui sont décrites dans les User Story (voir la section 1.2). Le travail est décomposé en tâches et se fait par une série d'itérations et de Releases.

Dans les plus petites organisations, il peut n'y avoir que quelques équipes de la sorte, ou juste une seule équipe. Dans certains cas extrêmes, Scrum est adopté par une seule personne qui développe des applications mobiles. Dans les grandes organisations, des groupes d'équipes Agile travaillent ensemble pour construire de gros blocs de fonctionnalités d'un produit complet, des fonctionnalités, composants architecturaux, sous-systèmes et ainsi de suite. La responsabilité de gérer le Backlog des User Story et les autres tâches que l'équipe doit remplir appartient au Product Owner avec l'assistance de l'équipe. Quand le nombre d'équipes augmente, il peut être nécessaire de diviser le travail entre plusieurs Product Owner.

Dans son travail quotidien, l'équipe est épaulée par des architectes, des ressources externes d'Assurance Qualité, des spécialistes en documentation, bases de données, gestion de configuration des sources (SCM), le personnel de construction/infrastructure, les internes de la DSI, et quiconque est nécessaire. Parfois et cela peut être bénéfique, ces personnes en support sont également membres de l'équipe, travaillant souvent dans plusieurs équipes.

Comme vu à la section 1.2, dans un projet Agile les User Story sont les objets principaux qui prennent en charge les exigences du client par le biais d'une chaîne de valeur — partant depuis l'analyse des besoins, passant par le codage et l'implémentation. La User Story prend souvent la forme suivante :

En tant que < rôle >, je veux < action > de façon à < avantage (description de la valeur métier) >

Avec ce formulation, l'équipe apprend à mettre l'accent sur le rôle de l'utilisateur et la valeur fournie par les nouvelles fonctionnalités. La User Story est prête pour être implémentée lorsque les critères d'acceptation et les tests ont été définis et quand sa priorité et l'estimation de sa taille (effort) ont été donnés.

Le Backlog de Produit est référencé par toutes les équipes impliquées dans le développement du produit. Il se compose de toutes les User Story que les parties prenantes du système et du projet ont identifiées ensemble avec le Product Owner pour l'implémentation. Une équipe se réfère toujours à un Backlog de Produit et un Product Owner qui le maintient et le priorise. Dans les projets de très grande envergure, le Backlog peut ne pas définir un produit réel, mais plutôt un de ses composants ; pour cette raison le Backlog est parfois appelé « Backlog de l'équipe ».

Le processus d'ingénierie des exigences dans une organisation Agile comprend l'identification, la priorisation, l'élaboration, la maintenance, la planification, la mise en œuvre, les tests et l'acceptation des User Story. À ce niveau, l'équipe travaille sur le raffinement des exigences du Backlog de Produit afin que chaque User Story qui entre dans le prochain Backlog de Sprint soit complètement « finie » à la fin de l'itération.

Le schéma des trois niveaux s'applique non seulement aux grandes organisations qui ont des projets de grande envergure et complexes, mais aussi aux organisations relativement petites qui développent différentes lignes de produits, diversifient les programmes par l'entremise de différentes feuilles de route de Release et destinent leurs produits à différentes catégories de clients et de marchés. Il existe également des cas ou le niveau métier et même le niveau programme peuvent ne pas convenir pour décrire le développement d'un projet Agile. Par exemple, une application interne avec une seule Release et un seul client interne ne devrait pas avoir de niveaux plus élevés.

© GASQ – Global Association for Software Quality

Les prochains chapitres de ce syllabus décrivent davantage les processus et les pratiques entourant la création et la maintenance des User Story.

Pour les organismes de formation : Fournir des exemples des différents éléments d'exigences, avec l'augmentation de leur niveau de détail : Thème = > Epic = > Fonctionnalité = > User Story.

2.2 Attributs des exigences Agile

Les attributs les plus importants d'une exigence de haut niveau, selon [REQB_FL_SYL], sont les suivants :

- Engagement
- Priorité
- Criticité

Cela vaut également pour les exigences Agile.

L'engagement pour les exigences de haut niveau (métier), c'est-à-dire, thèmes et Epics, s'exprime à travers les mots clés typiques "doit", "devrait", "pourrait", "peut " [après ISO/IEC/IEEE 29148:2011, précédemment IEEE 830]. Les mots clés « devrait », « pourrait », « peut », apparaissent uniquement dans les exigences métier. Comme les exigences sont raffinées en fonctionnalités et en User Story, l'engagement devient plus strict. Les User Story au niveau de l'équipe, utilisent le format « Je veux », vu du point de vue de l'auteur de l'exigence.

Les principales différences entre les exigences traditionnelles et Agile sont les suivantes :

- En passant des exigences de haut niveau aux User Story, l'équipe garde la perception de la vue du client et l'avantage qu'il veut obtenir.
- La vue du client n'est jamais perdue quand l'organisation Agile raffine l'exigence de haut niveau du thème d'origine en fonctionnalités et User Story.

Le [REQB_FL_SYL] explique que la priorité exprime l'importance/urgence d'une exigence. En Agile le concept reste le même, en soulignant la priorité comme « L'importance donnée par un client ou un marché spécifique pour l'exigence ». Les attributs correspondants sont la valeur métier qui met l'accent sur la vue de l'organisation/entreprise de l'exigence sur tous les aspects qui influencent les perspectives d'opportunité métier, et la priorité client qui met l'accent sur la vue du client de l'urgence de voir se réaliser l'élément. La valeur métier est particulièrement critique s'il est difficile d'obtenir la priorité du client.

En exigences Agile, la criticité est prise en compte avec tous les autres aspects de risque, donc pas seulement l'évaluation du risque des dommages qui se produiraient si l'exigence n'était pas remplie (risques produit, les risques pour le métier), mais également tous les aspects concernant les risques possibles pendant le développement (par exemple, de grands changements dans les produits existants, de nouvelles plateformes, l'intégration avec des parties tierces). En exigence Agile, la criticité est généralement dénommée « risque » et est raffinée quand l'exigence est traitée en descendant les niveaux jusqu'à celui de l'équipe.

[ISO/IEC/IEEE 29148:2011, précédemment IEEE 1233] propose des catégories supplémentaires d'attributs d'exigences : Identification, Dépendance, Source, Justification, Difficultés et Type. La plupart d'entre elles fournit des informations supplémentaires pour l'évaluation des exigences qui doit être découvertes au cours du raffinement de l'exigence. L'objectif principal de l'évaluation des exigences est d'estimer la valeur métier en utilisant les attributs obligatoires, Engagement, Priorité

© GASQ – Global Association for Software Quality

et Criticité. L'organisation Agile décidera quels sont les autres attributs nécessaires afin de mieux estimer la valeur métier et, plus tard, la taille et l'effort pour chaque exigence.

Pour les organismes de formation : Expliquer comment les principaux attributs des exigences de haut niveau sont mappés et évoluent à mesure que les exigences Agile sont raffinées.

2.3 Qualité des exigences Agile

[REQB_FL_SYL] présente une liste de problèmes potentiels qui peuvent nuire à la qualité des exigences. En comparant cette liste au Manifeste Agile, il est évident qu'appliquer les valeurs et les principes déclarés devrait aider à minimiser ces risques. Comme pour tout projet, il est toujours important de formuler et de spécifier les exigences de façon à éviter tout écart de qualité.

Comme indiqué, la définition des exigences dans un projet Agile passe par un processus de raffinement progressif, traversant les différents niveaux de description des exigences (Thèmes, EPIC, Fonctionnalités, User Story). Chaque raffinement vise à atteindre des niveaux plus fins de détail qui permettent la définition de critères de qualité vérifiables pour chaque exigence. Une classification des critères qualité employés couramment pour des exigences Agile est résumée par l'acronyme **INVEST** [Wake, 2003] :

- Indépendant – Il est plus facile de travailler avec des User Story si elles sont indépendantes. Idéalement, elles peuvent être programmées et implémentées dans un ordre quelconque. Une des caractéristiques des méthodologies Agile telles que Scrum ou XP est la possibilité de déplacer un peu les User Story, en tenant compte de leur priorité relative. Ce n'est pas toujours réalisable, mais quand les Story sont étroitement dépendantes, il peut être judicieux de les combiner en une Story ou de les répartir dans les Sprints identiques ou consécutifs.
- Négociable – une bonne story est négociable. Ce n’est pas un contrat explicite décrivant la fonctionnalité. Au lieu de cela, les détails seront discutés entre le Product Owner/représentant client et les membres de l'équipe avant et même pendant le développement. Alors que la Story est dans le Backlog de Produit, elle peut être réécrite ou même rejetée, selon le métier, le marché et les problèmes techniques.
- Valeur - Une Story doit apporter de la valeur au client. C'est surtout le problème du fractionnement des Story : Souvent, les développeurs ont tendance à ne travailler que sur une seule couche à la fois (et de le faire « bien »), mais une couche de base de données complète, par exemple, a peu de valeur pour le client, s'il n'y a pas la couche de présentation. Venir avec des Story techniques qui sont vraiment agréables à coder mais qui n'apportent aucune valeur à l'utilisateur final viole un des principes Agile, qui doit délivrer en permanence des logiciels qui apportent une valeur ajoutée.
- Estimable – Une bonne Story peut être estimée. Être estimable fait partie de la négociation, car il est difficile d'estimer une Story qui n'est pas comprise. C’est également une question de taille ; des Story de grande taille sont plus difficiles à estimer. Enfin, la facilité d’estimation d’une Story variera selon l'expérience de l'équipe. Si la taille d'une Story ne peut pas être estimée, elle ne sera jamais planifiée, décomposée en tâches ni faire partie d'une itération. La plupart du temps, une estimation impossible est causée par l'absence de renseignement, soit dans la description de la Story elle-même, soit de la part du Product Owner.

© GASQ – Global Association for Software Quality

- Taille appropriée – Les bonnes Story tendent à être petites. Les Story représentent généralement au plus quelques semaines/homme de travail. (Certaines équipes les limitent à quelques jours/homme de travail) Au-dessus de cette taille, il est difficile de comprendre ce qui est dans la périmètre de la Story. Les descriptions des Story peuvent être petites aussi, comme des éléments promettant une conversation ultérieure. Les détails peuvent être précisés lors de conversations avec le client, ou les parties prenantes impliquées. Les grandes Story peuvent être utilisées comme points de départ, mais elles doivent être découpées en petites Story afin d'être placées dans un Backlog d'itération.
- Testables – une bonne Story est testable. En fait écrire les tests au plus tôt aide à atteindre cet objectif. Si une Story n'est pas testable, il est probable qu'elle n'est pas clairement définie. Une Story n'est « finie » que si elle a été testée avec succès. Si on ne peut pas tester une Story à cause d'un manque d'information (voir ci-dessus « Estimable »), la Story n'est pas une bonne candidate pour faire partie d'un Backlog d'itération. Ceci est particulièrement vrai pour les équipes mettant en œuvre le Développement Piloté par les Tests (TDD).

Pour les organismes de formation : Donner des exemples d'exigences (peut-être à l'aide de User Story) qui répondent ou non aux critères INVEST.

Les critères qualité peuvent être appliqués non seulement à une exigence seule, mais également être utilisés pour une spécification d'exigences. Les critères qualité principaux pour les spécifications énumérées dans [REQB_FL_SYL] sont toujours valables dans un projet Agile. Ce qui change, c'est la notion de spécification d'exigences qui, dans une approche traditionnelle, est un document bien structuré. Dans un projet Agile, la spécification peut prendre une forme différente.

Validation et vérification assurent encore le niveau de qualité requis pour les exigences ou autres produits d'ingénierie des exigences dans un contexte Agile. Validation et vérification bénéficient des itérations du modèle Agile qui produisent des incréments potentiellement expédiables. Cela permet un feedback fréquent, donc la validation et la vérification sont réalisables en étapes sur le produit délivré, au lieu d'être une comparaison finale du produit avec ses spécifications. Dans les projets Agile, les pratiques communes suivantes aident à vérifier continuellement la qualité des exigences :

- TDD (Développement Piloté par les Tests)
- Intégration continue
- Automatisation des tests
- Programmation par paires.

Pour les organismes de formation : Expliquer comment les pratiques énumérées ci-dessus peuvent être bénéfiques pour la vérification de la qualité des exigences.

2.4 Ingénierie des Exigences en développement logiciel Agile

La définition générale de l'ingénierie des exigences qui est fournie dans [REQB_FL_SYL] est toujours cohérente et valide en développement logiciel Agile.

Dans un processus de développement séquentiel, il y a normalement une longue phase de collecte des exigences, pendant laquelle le produit présumé est complètement spécifié. L'idée est qu'en pensant plus longtemps, plus fort et mieux dès le début du projet, il n'y aura aucun coin d'ombre pendant la phase de développement principale du projet ; que si les exigences ont été correctement

© GASQ – Global Association for Software Quality

documentées, les utilisateurs obtiendront exactement ce qu'ils veulent. Ce n'est pas nécessairement vrai. Au mieux, les utilisateurs auront exactement ce qui a été écrit, ce qui peut ou peut ne pas être ce qu'ils veulent vraiment.

Le développement Agile met l'accent sur le raffinement progressif d'une exigence. Du peu de mots qui expriment une idée, toutes les parties concernées collaborent et discutent et définissent les détails en apprenant de plus en plus sur le produit, les utilisateurs, les concurrents, l'équipe et ainsi de suite. Dans le syllabus Fondation, vous trouverez également une liste des causes de négligence des exigences. Les méthodologies de développement Agile fournissent un ensemble de pratiques qui devraient assurer, si elles sont correctement interprétées et mises en œuvre, que le processus d'ingénierie des exigences n'est pas négligé dans l'implémentation du logiciel. Ces pratiques comprennent :

- Communication constante entre l'équipe et les parties prenantes
- Comparaison fréquente entre ce qui a été accompli et ce qui est attendu par le client
- Acceptation du changement
- Cycle perpétuel « inspecter et s'adapter » en faisant des rétrospectives
- Attribution des responsabilités à l'équipe plutôt qu'aux individus

Dans un projet Agile, la documentation écrite n'est plus le principal outil utilisé pour la spécification des exigences, mais elle peut être un outil utile uniquement lorsque l'équipe en identifie le besoin réel (par exemple, lorsque des connaissances doivent être transmises ou lorsque les clients le demandent). Les documents contiennent alors ce qui est réellement nécessaire et ne remplacera jamais, pour le développement des produits, l'échange direct d'idées et la communication en face à face.

2.4.1 Facteurs de réussite dans les projets Agile

Au cours des dernières décennies, les méthodologies Agile ont été utilisées dans différentes catégories de projets et d'organisations. Le processus de développement logiciel Agile s'est révélé plus efficace lorsque les projets et les organisations avaient les propriétés suivantes :

- L'environnement du projet nécessite de répondre rapidement et efficacement aux changements
- Les objectifs finaux du projet ne sont clairs en début du projet
- L'organisation prévoit de fréquentes versions pour le client final, les applications mobiles par exemple
- L'organisation est prête à changer et n'importe quelle résistance peut être gérée efficacement
- La communication et la collaboration entre les équipes est bien gérée et efficace même avec les grandes équipes et les équipes distribuées

2.4.2 Facteurs de risque dans les projets Agile

Dans la plupart des cas, l'ingénierie des exigences Agile ne fonctionne pas si les principes et les valeurs du développement Agile sont simplement ignorés, ou que les formations sont inadéquates et que les rôles principaux manquent de préparation. Indépendamment de cela, il existe certaines situations où même un projet de développement Agile bien exécuté peut être problématique ou moins bénéfique. Ces situations incluent ce qui suit :

© GASQ – Global Association for Software Quality

- Les contraintes de contrat sont fortes d'un point de vue exigences et solutions. Lorsque la marge de négociation et de discussion avec le client est limitée, ces contraintes peuvent miner les avantages d'un projet Agile (voir section 4.1.3).
- Les organisations où le respect des exigences légales et réglementaires à des jalons fixes est un objectif récurrent. Il s'agit d'un problème, surtout lorsqu'elles sont fixes, car des exigences stables à dates fixes nécessitent une importante documentation écrite pour la conformité.
- Les organisations où il est difficile de réunir les parties prenantes pour communiquer de façon cohérente et fiable. Par exemple, cela peut se produire lorsqu'Agile est mis en place dans les équipes de développement, mais que les parties prenantes client se fondent sur des documents d'exigences formelles. Cela peut également se produire lorsque le Product Owner n'est pas un représentant adéquat de la communauté des utilisateurs finaux (voir aussi la section 3.2).
- Les organisations ayant un processus de développement traditionnel bien établi où une forte résistance est réelle ou attendue lors d'une transition vers le développement Agile, ou lorsque la transition vers le développement Agile n'est pas sponsorisée par la direction.
- Les projets avec un développement distribué, ou les compétences sont concentrées sur un site et une équipe de développement, avec un haut niveau de compétition entre les sites et les équipes de développement et une mauvaise intégration et collaboration entre les équipes.

Pour les organismes de formation : Donner des exemples de situations qui sont bénéfiques pour les approches d'ingénierie des exigences Agile et d'autres qui sont problématiques.

3 Rôles en Ingénierie des exigences Agile (65 minutes)

Mots Clés

Développement de l'équipe, Product Owner, Parties prenantes du projet, manager des exigences, Scrum Master, partie prenante, parties prenantes système, membre de l'équipe

Objectifs d'apprentissage pour les rôles dans l'ingénierie des exigences Agile

3.1 Les parties prenantes (20 minutes)

AR-3.1.1 identifier les parties prenantes pour l'ingénierie des exigences Agile et expliquer leurs rôles (K3)

3.2 Le Product Owner (15 minutes)

AR-3.2.1 Comprendre le rôle du Product Owner en Ingénierie des exigences Agile (K2)

3.3 L'équipe de développement (15 minutes)

AR-3.3.1 Comprendre le rôle de l'équipe de développement en Ingénierie des exigences Agile et les principales contributions (K2)

© GASQ – Global Association for Software Quality

3.4 Le Scrum Master (15 minutes)

AR-3.4.1 Comprendre le rôle du Scrum Master en Ingénierie des exigences Agile (K2)

3.1 les parties prenantes

Les parties prenantes jouent un rôle clé en l'ingénierie des exigences dans les organisations Agile. Elles fournissent les entrées nécessaires à l'ingénierie des exigences dans toutes les activités d'un projet Agile [REQB_FL_SYL] :

- Élicitation des exigences
- Analyse des exigences
- Spécification des exigences
- Vérification et validation des exigences
- Traçabilité des exigences
- Gestion de configuration et du changement
- Assurance Qualité

Selon [Leffingwell, 2011] un projet a des parties prenantes Système et des parties prenantes Projet.

Les parties prenantes Système sont celles qui :

- Utilisent directement le système
- Travaillent avec les résultats de ceux qui utilisent le système
- Seront touchées par le déploiement et le fonctionnement du système

Ces parties prenantes sont les suivantes :

- Utilisateurs et opérateurs, ainsi que les utilisateurs de rapports, données, signaux et autres sorties du système
- Managers, acheteurs et administrateurs des utilisateurs
- Personnel de support et d'assistance
- Développeurs qui travaillent sur d'autres systèmes qui intègrent ou interagissent avec le système livré
- Professionnels de l'installation et la maintenance
- Toute autre personne qui interagit avec ou dépend du système

Ces parties prenantes fournissent les exigences Système principales et, par conséquent, seront les principales personnes impliquées dans les activités de découverte des exigences.

En plus des parties prenantes Système, il est important d'identifier toutes les personnes qui peuvent avoir des intérêts substantiels à ce projet d'implémentation du système, en d'autres mots les parties prenantes du projet qui :

- Ont envie de résoudre un problème métier avec une solution métier
- Ont un intérêt direct dans le budget et le calendrier
- Ont un intérêt direct dans la solution/produit/système développé
- Seront impliqués dans le marketing, la vente, l'installation ou la maintenance du système

© GASQ – Global Association for Software Quality

Ces parties prenantes projet comprennent les sponsors projet, le management de l'organisation, la gestion de projet, la gestion de portefeuille, les cadres, le personnel de gestion financière, les responsables du contrôle de configuration, les responsables informatiques et ainsi de suite.

Pour y parvenir, l'équipe doit d'abord comprendre et puis faire la synthèse des exigences avec une vision cohérente. Le Product Owner s'emploie à aider chaque partie prenante à trouver un terrain d'entente, pour qu'elles acceptent un consensus – et dans une perspective individuelle à trouver un compromis potentiel – Lorsque cela est nécessaire, le Product Owner prend des décisions pour les parties prenantes.

Les parties prenantes Projet et Système doivent être identifiées au démarrage d'un projet et sont parfois ajoutées en cours de projet. Les différentes parties prenantes ont différents degrés de participation :

- Certaines parties prenantes devraient être tenus informées ; elles ont simplement besoin de connaître l'état du projet et être informées des décisions qui les concernent. Elles peuvent ou pas participer à ces décisions, mais peuvent influencer ceux qui le font.
- Certaines parties prenantes devraient être consultées car leur domaine d'expertise aide à la définition ou à la construction du produit. Elles devraient être impliquées dans les décisions dans leur domaine d'expertise. Ces parties prenantes sont les spécialistes en la matière, analystes marketing, architectes et concepteurs d'interface utilisateur.
- Certaines parties prenantes sont partenaires du développement ou du processus. Elles incluent d'autres responsables métier ou produit, d'autres équipes de développement, analystes métier ou d'exigences et les fournisseurs de solutions avec qui le système interagit.
- Certaines parties prenantes contrôlent les résultats ; prennent les décisions finales pour la solution. Elles peuvent inclure les cadres, les responsables des Releases, les responsables métier et les clients clés qui utiliseront la solution.

L'ensemble des parties prenantes devrait être identifié parmi ceux:

- Qui vont directement utiliser le système
- Qui vont utiliser les résultats de ceux qui utilisent le système
- Qui seront responsables du support du système
- Qui sont impliqués dans d'autres systèmes avec lequel le système développé interagit
- Qui utiliseront les interfaces
- Qui peuvent fournir des indications sur la qualité des fonctionnalités et du système (facilité d'utilisation, fiabilité, performance et capacité de prise en charge) pour le système
- Qui doivent être consultés sur le périmètre du projet
- Qui ont un impact sur le budget et le calendrier
- Qui gèrent en fin de compte les relations métier entre les équipes et le client
- Qui détermineront comment et quand le système sera livré aux clients
- Qui peuvent aider ou nuire politiquement au projet
- Qui sont les partenaires qui dépendent du système
- Qui s'occupent du processus utilisé pour développer le système

Pour les organismes de formation : Fournir des exemples de parties prenantes Système et Projet et donner un exercice sur comment les identifier et leur degré d'implication, éventuellement à partir de cas réels.

© GASQ – Global Association for Software Quality

3.2 Le Product Owner

Selon [REQB_FL_SYL] un gestionnaire exigences est une personne avec une responsabilité globale pour documenter, analyser, tracer, prioriser et coordonner l'accord sur les exigences, puis pour contrôler les changements et communiquer les exigences aux parties prenantes concernées. Dans un contexte Agile, les activités du gestionnaire d'exigences correspondent au rôle de Product Owner.

Dans un projet Agile et en particulier en ingénierie des exigences Agile, en supposant Scrum comme méthode de référence, toutes les activités concernant la gestion et le développement des exigences sont le résultat d'une collaboration et d'une communication intense entre les parties prenantes, le Product Owner et l'équipe de développement. Lorsque le rôle de Product Owner a été introduit au paragraphe 1.1.2, il a été dit que le Product Owner, au travers du Backlog de Produit (voir 6.1.1), gère et suit toutes les informations sur les exigences, leur donne une priorité, coordonne les activités de discussion, évalue et examine les exigences grâce à des ateliers et gère les changements grâce à un feedback continu.

Concernant l'ingénierie des exigences, le Product Owner est responsable des points suivants :

- Gérer et contrôler le Backlog de Produit
- Représenter les parties prenantes pour l'équipe. Les parties prenantes ne sont pas souvent pas disponibles quand besoin. Dans la plupart des cas, leur présence est limitée à des événements collaboratifs clés (p. ex., les ateliers et les revues de Sprint). Le Product Owner est dans ce cas la voix des parties prenantes.
- Piloter la définition des exigences
- Établir les priorités
- Collaborer étroitement avec l'équipe
- Évaluer et inspecter le produit délivré

Le Product Owner devrait avoir les compétences énumérées dans [REQB_FL_SYL] pour être un bon ingénieur en exigences. Les compétences les plus importantes pour un Product Owner sont les suivantes :

- Etre aptitude à la communication
- Etre un facilitateur
- Avoir un bon sens du métier
- Avoir de bonnes connaissances techniques du domaine
- Etre en capacité à prendre des décisions
- Créer la confiance, être fiable

Les plus graves dangers pour la réussite d'un Product Owner sont les suivants :

- Consacrer uniquement un temps partiel au projet
- Etre Product Owner dans plusieurs équipes (selon le niveau d'activité et d'état des projets)
- Ne pas adhérer aux principes des rôles des équipes de développement Agile, comme quand le responsable des développeurs est également le Product Owner
- Ne pas représenter le client et les autres parties prenantes de façon adéquate à cause d'un manque de connaissances ou d'expertise (par exemple, ayant une forte expérience technique sans culture client)
- Avoir plus d'un Product Owner dans une équipe (ce n'est pas autorisé dans Scrum)

© GASQ – Global Association for Software Quality

Identifier la bonne personne pour jouer ce rôle est fondamental pour la réussite globale de l'organisation Agile. Surtout dans les grandes organisations, composées d'un grand nombre d'équipes qui pourraient travailler dans un environnement distribué avec des équipes externalisées et délocalisées, le fait d'identifier avec succès plusieurs Product Owner capables de faire du réseau et de collaborer avec de nombreux parties prenantes et équipes distribuées, déterminera dans quelle mesure le projet Agile sera réussi.

Pour les organismes de formation : Fournir quelques exemples de situations réelles où un Product Owner a correctement interprété son rôle pour l'ingénierie des exigences et d'autres où il n'a pas réussi et expliquez pourquoi.

3.3 L'équipe de développement

En développement Agile, selon [Leffingwell, 2011] toute l'équipe de développement est entièrement impliquée dans la définition, l'optimisation des exigences et l'obtention de compromis, l'implémentation, les tests, l'intégration dans une nouvelle baseline et ensuite doit veiller à ce que des exigences « finies » soient livrées aux clients. C'est le seul but de l'équipe.

L'unité de base de travail pour l'équipe est la User Story. Chaque User Story est réalisée dans une itération à travers une séquence de définition, construction et de test des actions jusqu'à ce qu'elle soit « finie ».

Avant de commencer le développement d'une itération, les membres de l'équipe sont impliqués dans toutes les activités qui composent le processus de définition des exigences. Il s'agit d'ateliers d'élicitation, d'analyse et de spécification des exigences.

La présence de l'équipe de développement dans cette phase est essentielle pour :

- Comprendre, grâce à la contribution des parties prenantes, l'origine, le contenu et la raison d'une exigence
- Aider à identifier les contraintes liées à l'exigence
- Contribuer à l'émergence d'exigences fonctionnelles et non fonctionnelles
- Fournir une première estimation de la taille, en terme de quantité relative de travail de développement et de test, et la complexité de chaque User Story
- Aider à identifier les risques associés spécifiquement aux aspects techniques, liés à la fois au développement et aux tests

L'équipe est l'acteur principal du développement des exigences. La présence et le poids de ses membres dans les événements d'élicitation des exigences, d'analyse, de spécification et de validation augmente à mesure que les exigences prennent forme et que les User Story sont suffisamment raffinées pour être entièrement implémentées dans une itération, en supposant qu'elles aient les propriétés INVEST.

Pour les organismes de formation : Fournir quelques exemples de situations réelles où les membres de l'équipe de développement sont impliqués dans le développement des exigences (p. ex., la participation de développeurs experts ou testeurs).

3.4 Le Scrum Master

© GASQ – Global Association for Software Quality

Le Scrum Master est responsable de :

- Faciliter les progrès de l'équipe jusqu'à l'objectif
- Diriger les efforts de l'équipe en amélioration continue
- Appliquer les règles du processus Agile
- Éliminer les obstacles
- Éviter les changements dans le Backlog pendant l'exécution des Sprints si les changements changent l'objectif du Sprint, ajoutent des risques à l'atteinte de l'objectif et ne sont pas motivés par des raisons métier fortes

Étant données les responsabilités du Scrum Master, il est clair que ce rôle est important en ingénierie des exigences, par exemple pour protéger la cohérence entre le Backlog de Sprint et l'objectif du Sprint. Le Scrum Master aide l'équipe à développer de meilleures compétences en communication, tant internes qu'avec d'autres partie prenantes, prend en charge l'équipe mettant les bonnes personnes à résoudre les obstacles et empêche l'équipe de perdre de vue sa capacité fondamentale. Ceci est fait en mettant en commun les connaissances et l'expérience de tous les membres pour raffiner, évaluer et diviser chaque User Story en tâches élémentaires pour l'implémentation et les tests.

Il apporte également ce support au Product Owner, et donc, d'un point de vue méthodologique, le Scrum Master a un rôle de support essentiel à la fois pour la gestion des exigences et pour le développement dans un environnement Agile.

Un Scrum Master devrait avoir les compétences suivantes :

- Compétences méthodologiques (connaissance pratique de la méthode Scrum, des techniques et des outils et comment les appliquer à l'ingénierie des exigences)
- Compétences en modération et en négociation
- Capacité à argumenter et à convaincre
- Compétences linguistiques et en communication

Pour les organismes de formation : Fournir quelques exemples de situations réelles où un Scrum Master interprète correctement son rôle pour l'ingénierie des exigences et d'autres où il ne réussit pas et expliquez pourquoi.

4 Développement des exigences Agile (240 minutes)

Mots clés

Concept 3C, critères 4C + R, critères de satisfaction, atelier d'analyse, valeur métier, conditions de satisfaction, contrat, priorité client, estimation de l'effort, atelier de Grooming, Story d'implémentation, Story d'infrastructure, modèle, exigences non fonctionnelles, Backlog de portefeuille, taille relative et complexité, atelier de planification de Release, élicitation des exigences, spécification des exigences, atelier d'exigences, rétrospective, Spike, atelier de planification de Sprint, revue de Sprint, cas d'utilisation, expérience utilisateur, vision, atelier

Objectifs d'apprentissage pour le développement des exigences Agile

4.1 Elicitation des exigences (65 minutes)

© GASQ – Global Association for Software Quality

4.1.1-AR Expliquer l'élicitation des exigences et ses techniques usuelles dans l'organisation Agile (K2)

4.1.2-AR La pratique de l'atelier d'exigences (K3)

4.1.3-AR Comprendre les implications des exigences en matière de contrat et d'impact sur les organisations Agile (K2)

AR-4.1.4 Expliquer comment extraire les User Story de cas d'utilisation et des exigences non fonctionnelles (K2)

4.2 Analyse des exigences (70 minutes)

AR-4.2.1 Pratiquer l'atelier de planification de Release, l'atelier d'analyse et le Spike (CO3)

4.2.2-AR Expliquer les Story d'infrastructure et d'implémentation (K2)

4.2.3-AR Résumer l'acceptation des exigences Agile (K2)

4.2.4-AR Pratiquer l'atelier de planification de Sprint et l'atelier de Grooming (K3)

4.3 Spécification des exigences (55 minutes)

AR-4.3.1 Pratiquer le concept des 3C et le processus de raffinage d'une User Story jusqu'à ce qu'elle remplisse les critères INVEST (K3)

AR-4.3.2 Affecter et appliquer les conditions de satisfaction et les critères d'acceptation utilisateur pour raffiner les User Story (K3)

AR-4.3.3 Expliquer pourquoi et comment consacrer du temps sur l'expérience utilisateur dans une User Story (K2)

4.4 Validation et vérification des exigences (30 minutes)

AR-4.4.1 Expliquer la validation et la vérification des exigences dans la revue de Sprint (K2)

AR-4.4.2 Comprendre comment la rétrospective est utilisée pour l'inspection et l'adaptation en développement des exigences (K2)

4.5 Estimation des User Story (20 minutes)

AR-4.5.1 Estimer la valeur métier, la priorité client, la taille relative et la complexité, les risques et l'effort d'une exigence et expliquer comment ces valeurs sont attribuées au cours du raffinage de l'exigence (K3)

4.1 Elicitation des exigences

Selon [REQB_FL_SYL] sur le développement des exigences, l'élicitation des exigences comprend :

© GASQ – Global Association for Software Quality

- La collecte des besoins des parties prenantes et, en se basant sur les définitions initiales et les besoins métier, la formulation des exigences métier et l'identification des éventuelles limitations et hypothèses
- Le détail des exigences connues de haut niveau
- L'exclusion des fonctionnalités inutiles

Dans un contexte Agile, ce sont les activités qui transforment un Backlog de portefeuille en un Backlog de programme. Cela se fait en sélectionnant les thèmes métier et en les raffinant pour créer un ensemble de User Story, ou au moins d'EPIC, qui plus tard serviront de base pour l'analyse et la spécification.

La plupart des techniques habituelles pour l'élicitation des exigences [REQB_FL_SYL] sont utilisées dans un contexte Agile, notamment ce qui suit :

- Ateliers d'exigences
- Entrevues et questionnaires
- Enregistrements
- Sollicitation d'informations auprès des représentants du client sur site
- Identification sur la base de documents existants
- Réutilisation (c.-à-d., réutilisant la spécification d'un autre projet)
- Brainstormings
- Observation sur le terrain et apprentissage
- Cas d'utilisation

En outre, les suivants sont également utilisés :

- Analyse des concurrents
- Participation de la communauté

L'atelier d'exigences utilise toutes les techniques ci-dessus afin de partager les exigences, d'aider à faire émerger les exigences cachées et de discuter ce qui en ressort et de le communiquer à toutes les parties prenantes.

Certaines des techniques ci-dessus seront également utilisées dans les ateliers ultérieurs. Les ateliers ne sont pas des réunions et ne doivent pas être prises comme telles. Selon [Gottesdiener, 2002] un atelier :

- Est facilité, non conduit ou dirigé par un directeur
- Nécessite un travail préalable par les participants
- Prend des décisions fondées sur la collaboration ou sur des règles prises au départ
- Se focalise sur la découverte et la création
- Nécessite la participation active de tous les participants
- Produit des résultats spécifiques

Ce qui suit doit être posé lors de la préparation d'un atelier :

- Définir le sujet à partager pour l'atelier
    - Cibler le périmètre et faire un focus sur l'activité à produire
- Établir les principes de participation,
    - Fixer les règles de base – respect, ouverture, temps
    - Déterminer les règles et le processus de prise de décision
- Dans le cas où un intervenant ne peut pas participer, cette personne doit nommer un remplaçant qui peut jouer un rôle actif

© GASQ – Global Association for Software Quality

Remarque :

Tous les résultats et les artefacts de l'atelier doivent être enregistrés, et mis à la disposition de toutes les personnes impliquées.

Pour les organismes de formation : Fournir quelques exemples des différences entre une réunion et un atelier pour expliquer pourquoi le second doit toujours être un événement collaboratif.

4.1.1 L'atelier d'exigences

L'atelier d'exigences rassemble tous les acteurs clés pour obtenir un accord entre les Product Owner et les représentants des équipes de développement concernant la valeur métier de chaque exigence identifiée. Les buts principaux des ateliers d'exigences sont :

- Identifier les exigences relatives au programme à partir du Backlog et de la vision portefeuille
- Mettre en évidence les aspects architecturaux des exigences (niveau élevé)
- Identifier le calendrier pour la prochaine Release d'un point de vue marché et d'autres dépendances intermédiaires
- Identifier les User Story (c'est-à-dire commencer à raffiner les thèmes et les EPIC, les diviser en nouvelles EPIC, fonctionnalités, User Story)
- Estimer la valeur métier des User Story du point de vue des parties prenantes
- Identifier les conditions de satisfaction de chaque User Story du point de vue des parties prenantes

L'atelier implique généralement les fonctions/rôles suivants :

- Product Owner
- Tous les parties prenantes clé
- Les représentants de l'équipe (p. ex., le développement, le test, la gestion de la configuration)
- Les architectes Système

La première étape pour la réussite de l'atelier est de sélectionner les parties prenantes nécessaires, selon les critères énumérés à la section 3.1.

L'étape suivant le choix des parties prenantes consiste à recueillir les données nécessaires pour permettre l'élicitation des exigences (par exemple, les résultats des entrevues et questionnaires, enregistrements, cas d'utilisation). L'atelier peut alors démarrer sur les thèmes principaux de l'ordre du jour suivant :

- Donner un aperçu du Backlog de portefeuille, la vision, le marché et les concurrents
- Présenter les résultats des entrevues avec les d'utilisateurs, les questionnaires et ainsi de suite
- Démarrer une session de brainstorming où les parties prenantes identifient les éléments à mettre dans le programme (p. ex., EPIC, fonctionnalités, cas d'utilisation, User Story)
- Faire les premiers découpages des EPIC, fonctionnalités, cas d'utilisation, voire les User Story
- Faire une estimation de la valeur métier de chaque exigence identifiée
- Consolider et hiérarchiser les exigences qui en résultent

Les éléments d'exigence doivent ici être identifiés avec une vue « orientée client », et pas avec une vision de développement, test, architecture ou organisation, même si les développeurs, testeurs ou les architectes participent. Il est conseillé de commencer par un petit nombre d'éléments d'exigence centrés sur le client pour la Release. Les éléments d'exigences qui sont trop généraux ou ne sont pas valables d'un point de vue économique devront être divisés en éléments plus détaillés.

© GASQ – Global Association for Software Quality

La valeur métier est un paramètre clé pour déterminer s'il convient ou pas d'implémenter une User Story, quand le faire et le retour sur investissement attendu. Les parties prenantes donnent une valeur métier qui reflète leur vision de la User Story. Cette valeur peut être guidée par l'un des facteurs suivants :

- Nouveau métier : Toutes les fonctionnalités qui apporteront potentiellement de nouveaux clients ou de nouveaux marchés et, éventuellement, un frais flux d'argent
- Vente : Toutes les fonctionnalités qui rapporteront potentiellement de l'argent de clients existants et qui pourraient être vendues comme Add-on, mise à niveau ou plug-in
- Préservation : Toutes les fonctionnalités qui éviteront la perte de clients et la perte correspondante de recette
- Coût de l'efficacité : Toutes les fonctionnalités qui permettront d'économiser de l'argent (coûts) pour le matériel, l'équipements, les personnes et le temps
- Efficacité opérationnelle : Toutes les fonctionnalités qui permettront d'économiser de l'argent (coûts) compte tenu que toute opération peut potentiellement voir ses coûts augmenter (p. ex., installation, configuration, personnalisation)
- Capacité d'investissement progressif
- Augmentation de l'expertise
- Compatibilité

Ni l'estimation de l'effort, ni la complexité de développement ne sont mis en lumière dans cet atelier et ne doivent jamais influencer l'estimation de la valeur métier.

Le Product Owner, ou un coach externe, agit comme modérateur. Les parties prenantes peuvent se poser mutuellement des questions et clarifier les éléments d'exigence sans entrer dans trop de détails. Pour chaque exigence les parties prenantes devront s'entendre sur une valeur métier. L'outil habituel pour cette activité est le planning poker (voir la section 6.2.1). Après avoir obtenu un accord commun sur la valeur métier, la valeur est affectée à l'élément d'exigence et cet élément est alors prêt à rejoindre le Backlog de produit.

La durée figée de l'atelier dépend de la complexité et de la taille du projet.

Pour les organismes de formation : Fournir quelques exemples de situations réelles où, lors d'un atelier d'exigences, une valeur métier est correctement évaluée et d'autres où l'estimation est basée sur des paramètres incorrects comme l'effort et de la complexité.

4.1.2 Vision

Une autre source importante qui exprime les besoins principaux du client est la vision. Étant donné que les documents traditionnels peuvent ne plus exister pour spécifier le comportement prévu du système, communiquer la vision directement au équipes de développement Agile devient donc critique. Ce problème potentiel est atténué ou disparait complètement grâce à l'approche de l'atelier orientée communication et partage, où les parties prenantes s'assoient ensemble.

Pour chaque produit, la vision doit être définie à nouveau. Une vision se concrétise par des objectifs à respecter par un programme/projet spécifique. Les objectifs devraient exprimer les objectifs métier du produit et doivent être S.M.A.R.T. [REQB_FL_SYL]. Une vision résume également les risques métier relatifs à ce produit.

Indépendamment des méthodes de communication, la vision communique l'intention stratégique pour le programme/projet et devrait répondre à ces questions :

- Pourquoi construisons-nous ce produit, système ou application ?

© GASQ – Global Association for Software Quality

- Quel problème résoudra-t-il ?
- Quels fonctionnalités et bénéfices fournira-t-il ?
- A qui donne t'il ces fonctionnalités et ces avantages ?
- A quels risques notre organisation doit-elle faire face ?
- Quelle performance, fiabilité et évolutivité doit il fournir ?
- Quelles plates-formes, normes, applications et ainsi de suite, acceptera-t-il ?

Lorsque toutes les parties prenantes, les objectifs et la vision sont connus, il est possible de commencer avec l'élicitation détaillées des exigences.

4.1.3 Contrats

Outre les délais de développement et de livraison d'un produit, un contrat contient aussi généralement les éléments suivants :

- La liste des exigences priorisées
- Une brève description de la solution prévue
- Les critères d'acceptation pour chaque exigence
- La liste des livrables (p. ex., documentation, code, logiciel opérationnel)
- Autres besoins et attentes comme la technologie souhaitée être utilisée, les exigences de ressources, etc...

Les contrats peuvent être une source clé pour l'élicitation des exigences. Cela peut être contraignant dans un projet Agile.

4.1.3.1 Liste des exigences priorisées

Avoir une liste des exigences priorisée dans un contrat présente des avantages et des inconvénients pour le client et le fournisseur. Certaines exigences contractuelles peuvent n'être jamais nécessaires en cours de contrat en raison de changements dans les besoins du client et les priorités, (par exemple, changement des conditions du marché, nouveaux concurrents). Habituellement, un avocat fera valoir (via le contrat) les souhaits du client qui sont d'articuler tous les besoins possibles, et le fournisseur s'efforcera de répondre à chaque exigence prévue. Dans une approche Agile, les exigences devraient être définies de façon itérative et évolutive afin de minimiser le risque ou les dépenses de temps et d'argent dans le développement de logiciels pour des exigences inutiles en fin de compte. Cela peut être contradictoire avec le contenu d'un contrat traditionnel.

Idéalement, la liste des exigences priorisées devrait être moins importante et moins spécifique dans un contrat d'un projet de développement Agile. Cela offre des avantages à la fois pour le client et le fournisseur en permettant l'adaptation aux besoins changeants.

4.1.3.2 Courte Description de la solution prévue

Toute(s) description(s) de la solution prévue(s) doit être évitée dans les contrats. Il est préférable d'inclure un résumé du périmètre, la vision et la motivation métier du projet ou du produit dans le préambule du contrat, ainsi que le prix et les bases du paiement. Le fournisseur doit se référer au périmètre et à la vision lorsqu'il détermine les exigences. Cela se traduira par moins de contraintes sur les solutions contractuelles et laissera plus de place à la discussion des problèmes avec le client pour parvenir à un consensus.

© GASQ – Global Association for Software Quality

4.1.3.3 Critères d'acceptation pour chaque exigence

Les critères d'acceptation dans un contrat peuvent être cruciaux pour des travaux sous-traités. L'ambiguïté autour de ces zones est une source possible de conflits et de litiges. En revanche, une description détaillée des critères d'acceptation pour chaque exigence et peut-être de la solution prévue, est un écueil potentiel pour le client et le fournisseur. Si cela se produit, le client perdra la souplesse nécessaire pour modifier les exigences ou le fournisseur pourra rencontrer des contraintes qui diminueront la qualité du logiciel et la communication avec le client.

Dans un projet Agile, il est préférable de définir un cadre d'acceptation dans le contrat tel que ci-dessous :

- Dans une approche itérative, à chaque itération, l'acceptation peut être basée sur la conformité à un ensemble préalablement convenu de tests d'acceptation.
- Avec Scrum, l'acceptation peut être basée sur la conformité avec la définition de « Fini » qui inclut l'utilisation de test automatisé.

4.1.3.4 Liste des produits livrables

La sous-traitance suppose impliquer une faible transparence et peu de confiance. C'est également fréquent pour un logiciel opérationnel livré après une longue période pendant laquelle l'équipe de développement a « avancé tranquillement », offrant peu de visibilité sur ses activités. Les contrats requièrent souvent un plan d'assurance qualité classique ou une liste de livrables qui définissent la longue checkliste des documents devant être fournis. Le concept ici est que plus il y a de documentation plus le client obtiendra ce qu'il veut.

Une approche plus pratique consiste à limiter la liste des livrables à ceux qui sont vraiment nécessaires, en se concentrant sur ceux qui offrent une valeur réelle au client. Cela permet au fournisseur de se concentrer sur la fourniture d'une plus grande valeur et d'optimiser le temps pour l'ingénierie des exigences dans le cadre du projet.

4.1.3.5 Autres Besoins et Attentes

La plupart de ces besoins et attentes sont les exigences non fonctionnelles qui doivent être traitées comme toute autre exigence.

Le Product Owner devrait jouer un rôle actif dans toute négociation concernant le contrat qui peut conduire à une amélioration dans le modèle de contrat. Toutes les exigences requises dans le contrat doivent être traitées pour apporter de l'avantage au client tout d'abord, grâce à une plus grande flexibilité globale, tout en permettant au fournisseur de mettre l'accent sur la valeur du logiciel livré. Ce n'est pas toujours possible parce que la discussion contractuelle est souvent limitée. Lorsque c'est possible, le rôle de Product Owner est important pour négocier avec le client.

Pour les organismes de formation : Proposer des exemples d'évolution de contrat traditionnel vers des contrats qui prennent en charge efficacement l'ingénierie des exigences dans le contexte Agile.

4.1.4 Cas d'utilisation

Les cas d'utilisation sont une méthode alternative pour exprimer les fonctionnalités d'un système. Si une organisation Agile fonctionne bien avec des cas d'utilisation, il peut ne pas y avoir de raison de changer. Cependant, les cas d'utilisation sont censés avoir une plus grande envergure qu'une User

© GASQ – Global Association for Software Quality

Story habituelle. Les équipes Scrum travaillent généralement plus efficacement avec des unités de travail qui sont plus petites qu'un cas d'utilisation habituel. En général un cas d'utilisation unique ne respectera pas les critères INVEST, en particulier la propriété « taille appropriée », car il décrit plus de fonctionnalités qu'un Sprint unique peut implémenter.

Les Story raffinées, c'est-à-dire, celles qui sont prêtes à être incluses dans un Sprint, sont très granulaires et souvent ne donnent pas le contexte aux concepteurs — quel est l'utilisateur, quel est le contexte de fonctionnement, et quel est leur objectif global en ce moment ? [Cockburn, 2007]

Les cas d'utilisation peuvent être utiles dans un projet de développement Agile en jouant le rôle d'Epic du système, de fonctionnalités ou de grandes User Story (ou en les remplaçant). Ces cas d'utilisation devront être affinés en User Story « de taille appropriée ». De cette façon, les cas d'utilisation peuvent fournir le contexte et les User Story sont les éléments de Backlog finaux à implémenter.

[Cockburn, 2007] propose quelques conseils pour l'application des cas d'utilisation dans les projets Agile :

- Gardez-les légers — sans détail de conception, ni de spécifications d'IHM et ainsi de suite
- Ne les traitez pas comme des exigences figées. Comme les User Story, ce ne sont que des instructions de fonctionnement du système prévu
- Ne vous inquiétez pas de leur maintenance ; ce ne sont principalement que des outils de réflexion
- Modélisez-les officieusement — utiliser des tableaux blancs, des outils simples et ainsi de suite

Lorsque des cas d'utilisation sont utilisés, les User Story peuvent être extraites comme ci-après:

- Mettre en œuvre un sous-ensemble des scénarios
  - Chemin principal en premier, suivis des chemins d'exception
- Mettre en œuvre un sous-ensemble d'actions d'un scénario
  - Actions clés d'abord, puis secondaires
- Séparer les sous-objectifs communs d'un ensemble de cas d'utilisation
  - Mettre en œuvre chaque sous-objectif comme un bloc de construction

Pour les organismes de formation : Donner des exemples de cas d'utilisation et de User Story qui peuvent être extraits de ces cas d'utilisation.

4.1.5 Exigences non fonctionnelles

Les exigences non fonctionnelles (ENFs) décrivent les attributs de qualité non fonctionnelle de l'ensemble du système, ou de ses composants ou fonctions spécifiques. Les exigences non fonctionnelles peuvent décrire différents aspects de la solution. Selon ISO/IEC 25000 (précédemment ISO 9126), certains types de caractéristiques de qualité non fonctionnelles sont les suivants :

- Fiabilité
- Facilité d'utilisation
- Efficacité
- Maintenabilité
- Portabilité

© GASQ – Global Association for Software Quality

Chacun d'entre eux possède des caractéristiques secondaires.

Les exigences non fonctionnelles peuvent être considérées comme des contraintes sur la fonctionnalité car elle peut restreindre la conception des solutions pour assurer le niveau de performance, pour garantir la compatibilité avec des plateformes spécifiques, pour permettre aux utilisateurs d'être en mesure d'utiliser la fonctionnalité dans un temps limité, pour assurer qu'un nombre prévu de transactions peut être traité dans un temps limité prévu, et ainsi de suite.

Une ENF peut s'appliquer à une ou plusieurs Story et une Story peut être limitée par une ou plusieurs ENFs.

Un ENF peut elle-même être écrite sous la forme d'une User Story. En général, une ENF rédigée comme une simple déclaration peut être décrite sous la forme de User Story. Cela peut orienter de façon bénéfique l'équipe de développement sur l'exigence et faciliter l'estimation et l'implémentation.

Voici un exemple d'exigence non fonctionnelle au format traditionnel :

« Tous les logiciels open source utilisés dans le projet doivent être approuvés par le client »

Format de la User Story 1 : En tant que *représentant du client, je tiens à approuver tous les logiciels open source utilisés dans le projet, de façon à ce qu'il n'y ait pas d'explosion des licences.*

Cette première User Story ajoute des informations concernant la raison de cette exigence. La User Story pourrait également être écrite :

Format de la User Story 2 : En tant que *représentant du client, je tiens à approuver tous les logiciels open source utilisés dans le projet, pour que la réutilisation des logiciels open source déjà approuvés et fiables soit maximisée.*

Selon ce qui motive la User Story, l'équipe de développement peut identifier différentes tâches pour la même ENF. Dans le premier cas, la sélection de logiciel open source se fondera sur la politique de licence ; dans le second cas, il se fondera sur la réutilisation de logiciels fiables.

Une ENF peut être persistante, c'est-à-dire qu'elle peut rester en place pendant une longue période dans différentes versions ou s'appliquer à des produits indépendants similaires. Souvent ces exigences, surtout celles qui ont trait au rendement, sont vérifiées par le biais de tests automatisés et vont être revérifiées tout au long de la vie du produit pour que l'ENF soit uniformément couverte. Dans certains cas les ENFs sont incorporées dans la définition de « fini » et deviennent applicables à chaque User Story ou à une itération complète.

Pour les organismes de formation : Donner des exemples d'ENF et de User Story qui peuvent être extraites de ces ENFs.

4.2 Analyse des exigences

En développement des exigences, l'analyse des exigences comprend [REQB_FL_SYL] :

- Élaborer les besoins métier après les exigences système/solution et descendre vers les exigences produit/composant, en se basant sur les acquis pendant le processus
- Résoudre les conflits entre les exigences avec l'assistance des parties prenantes
- Établir les limites de la solution
- Formuler les exigences en accord avec la définition du périmètre

© GASQ – Global Association for Software Quality

- Développer des modèles de solution métier

Dans un contexte Agile, l'analyse des besoins signifie toutes les actions qui transforment un Backlog de programme en un Backlog de produit. Ces activités comprennent la sélection des éléments d'exigence qui sont sous la forme d'Epic de haut niveau, de fonctionnalités, de cas d'utilisation et des grosses User Story et de leur raffinage afin de créer les User Story priorisées dans le Backlog de produit. Une fois raffinées, ces Story sont prêtes à être estimées. Ces activités se déroulent au cours de l'atelier de planification de Release (voir Figure 2 Chapitre 2), les ateliers d'analyse et les Spikes.

4.2.1 L'atelier de planification de Release

L'atelier de planification de Release implique en général les fonctions/rôles suivants :

- Le Product Owner
- Le facilitateur de l'atelier (peut être le Product Owner)
- Une sélection de parties prenantes qui peuvent contribuer activement à fournir des informations pour le raffinement et l'estimation des risques des User Story. Ils fournissent une première vérification et validation pour s'assurer que les exigences sont comprises correctement et approuvées. Les participants pourraient inclure :
  - Les Parties prenantes représentant les clients (p. ex., gestion des produits)
  - Des représentants d'autres projets en adhérence
- Des représentants des équipes (développement, test, gestion de la configuration)
- Un architecte système
- Des Scrum Master

Les principaux objectifs de l'atelier sont :

- Présenter et s'entendre sur un classement de tous les éléments (Epics, fonctionnalités, cas d'utilisation, User Story) qui sont candidats pour le Backlog de produit, en se basant sur la valeur de la priorité et la valeur métier.
  Les éléments ont été préparés par le Product Owner à la suite de l'atelier d'exigences.
- Identifier et planifier un atelier d'analyse détaillée si nécessaire
- Compléter le raffinement des User Story dans l'ordre de classement
- Assigner une valeur de risque, basée sur le risque projet/métier, à tous les éléments dans le Backlog de produit
- Estimer la taille des User Story, du point de vue de l'équipe et du Product Owner
- Vérifier le timeboxing de la Release pour identifier les membres de l'équipe Scrum et construire les équipes
- Décider du focus de l'atelier de planification de Sprint du premier Sprint

Utiliser une approche de planification traditionnelle et essayer de capturer toutes les exigences en détail dans le Backlog de produit est sujet à erreurs lorsqu'il s'agit de produits innovants. C'est également indésirable dans un contexte Agile où les fonctionnalités détaillées du produit sont découvertes au cours du développement par le biais de feedback client précoces et fréquents et des commentaires des utilisateurs. Cela ne signifie pas que l'équipe ne peut pas prendre une décision d'investissement éclairée à un stade précoce dans un projet Agile. La date de lancement et le budget du projet sont souvent décidés avant le premier Sprint.

En se basant sur cette première ébauche de Backlog de produit préparée par le Product Owner, les participants peuvent évaluer si une analyse plus approfondie est nécessaire afin de procéder aux

© GASQ – Global Association for Software Quality

étapes suivantes : raffinement des User Story, estimation de valeur du risque et de la taille de la User Story. Si une analyse complémentaire est nécessaire, l'atelier de planification de Release est reporté et un atelier d'analyse est prévu. L'atelier de planification de la Release reprend lorsque les résultats de l'analyse sont suffisants pour continuer.

Après avoir terminé le raffinement des User Story (voir 4.3.1.1) ayant la priorité et la valeur métier la plus élevée dans le Backlog de produit, l'équipe doit procéder comme suit :

- Estimer la valeur du risque. L'atelier de planification de Release est le bon endroit pour estimer le risque parce que les gens nécessaires sont présents. La valeur du risque comprend tous les risques liés à l'exigence analysée, y compris le risque global pour le produit, le risque métier (voir 4.1.2) et les risques techniques. Cette valeur est un nombre qui résume les résultats de l'analyse des risques pour une exigence. Une valeur de risque peut être utilisée comme une information additionnelle pour prioriser les User Story dans les Sprints pour aborder au début les Story au risque le plus élevé.
- Estimer l'effort ou la complexité des Story. Dans cet atelier, une estimation de la complexité est fortement recommandée pour donner une idée préliminaire du niveau de complexité de l'implémentation de la User Story. Une estimation de l'effort plus raffinée doit être faite au cours de la réunion de planification de Sprint. Il est important de se rappeler que ce sont des estimations de taille, pas de la valeur métier. L'estimation est généralement faite dans une unité de mesure appelée points de Story. Le Planning Poker est souvent utilisé pour faire cette estimation. Le Planning Poker est discuté section 6.2.1.

Une fois de plus, les User Story qui sont trop grosses doivent être découpées dans de plus détaillées. Cela peut être fait en découpant la Story en se basant sur différents critères comme expliqué dans 4.3.1.1.

Il est important de noter que les critères permettant d'identifier le contenu du premier Sprint ne sont pas que des numéros. Bien qu'il soit logique de prioriser le Backlog de produit en se basant sur les User Story qui ont la plus grande valeur métier, la priorité et la valeur de risque la plus élevée, ce n'est pas la seule façon de définir le contenu d'un Sprint. Un bon jugement est de mise et les User Story doivent être évaluées afin de déterminer quelles conditions préalables sont nécessaires. Il peut être également logique de déplacer certaines Story de valeur inférieure vers le haut du Backlog de produit en raison des opportunités et des besoins des projets (p. ex., infrastructures et implémentation des User Story comme discuté section 4.2.2). Cette hiérarchisation logique peut se faire sans compromettre l'objectif du Sprint.

4.2.2 Infrastructures et implémentation des User Story

L'analyse des exigences est une étape supplémentaire vers l'achèvement du Backlog de produit. Deux grandes catégories d'exigences doivent être considérées et décrites sous la forme de User Story : les Story d'implémentation (ou les Story d'aptitude) et les Story d'infrastructure.

Une Story d'implémentation est un élément additionnel, idéalement rédigée sous la forme d'une User Story, qui peut être nécessaire lorsque de nouvelles fonctionnalités doivent être ajoutées pour pouvoir mettre en œuvre des éléments de fonctionnalité. Par exemple une Story d'implémentation serait nécessaire pour un nouveau pilote qui permet au logiciel composé de plusieurs User Story et qui s'exécute sur une plateforme spécifique, d'être compatible avec une plate-forme supplémentaire. Généralement il s'agit d'un élément du Backlog de produit qui pourrait apporter une

© GASQ – Global Association for Software Quality

valeur faible ou même nulle, mais qui est si important qu'il doit être inclus afin d'être en mesure de délivrer la valeur métier des User Story liées.

Une Story d'infrastructure est un élément du Backlog de produit destiné à créer l'infrastructure nécessaire pour que le Sprint atteigne le statut de « fini ». Il s'agit habituellement d'un élément de priorité haute qui doit être géré dans un Sprint comme n'importe quelle autre User Story. Un exemple d'une Story d'infrastructure serait celle qui décrit la mise en place de l'environnement d'intégration continue ou une qui explique comment l'équipement de test ou le framework d'infrastructure d'automatisation de test doit être mis en place. Cela peut être écrit sous la forme d'une User Story ou non.

L'introduction de ces éléments dans le Backlog de produit peut se faire au cours de l'atelier de planification de Release dès qu'ils sont connus. Ceci est utile parce qu'ils sont estimés comme n'importe quelle autre User Story (mais peuvent n'avoir aucune valeur métier).

À la fin de la planification de la Release, le Backlog de produit comprend une variété d'éléments :

- Nouvelles fonctionnalités client (" Permettre à tous les utilisateurs de placer un ouvrage dans un panier d'achat ").
- Objectifs d'amélioration de l'ingénierie (" Retravailler le module de traitement des transactions pour le rendre évolutif ")
- Travaux de recherche ou exploratoires (" Etudier des solutions pour accélérer la validation des cartes de crédit ")
- Les défauts connus (" Diagnostiquer et résoudre les erreurs de script de traitement des commandes ")

Le raffinement des exigences peut s'avérer complexe dans plusieurs cas, en particulier lorsque :

- Le contenu du produit est innovant
- Les clients ne sont pas en mesure de définir l'exigence sans ambiguïté
- Plusieurs clients ont une vue différente et incompatible sur une même exigence
- Différentes parties prenantes ne peuvent pas se mettre d'accord sur les conditions de satisfaction
- Le niveau INVEST de raffinement d'une exigence ne peut pas être atteint en raison de la méconnaissance de certains aspects fonctionnels ou technologiques, ou parce que les interfaces avec d'autres produits, les systèmes ou les infrastructures doivent être étudiées

Dans tous ces cas, il est nécessaire de pousser l'analyse pour atteindre le niveau de compréhension de l'exigence, tant du point de vue de la valeur métier que d'un point de vue technique. Cette analyse devrait permettre de compléter le raffinement des exigences, en clarifiant tous les aspects qui ne sont toujours pas clairs.

Il existe deux méthodes principales pour mener cette analyse :

- L'atelier d'analyse
- Le Spike

Dans les deux cas, l'objectif de l'analyse est qu'à la fin de l'activité, chaque exigence analysée acquiert les propriétés exprimées par les critères de « 4C + R » :

- Correct : Précis, reflète des besoins réels
- Clair : Une seule interprétation, pas d'ambiguïté
- Cohérent : Aucun conflit avec d'autres exigences
- Complet : Aucun élément manquant

© GASQ – Global Association for Software Quality

- Pertinent : Nécessaire pour atteindre les objectifs et les buts métier

4.2.3 L'atelier d'analyse

L'atelier d'analyse est nécessaire dans les cas suivants :

- L'équipe peut avoir besoin d'effectuer une analyse approfondie du comportement implicite, parce que les clients et les parties prenantes métier n'ont pas fourni suffisamment de détails sur ce comportement.
- Plusieurs clients voient la même exigence avec des différences substantielles concernant son contenu fonctionnel et sa priorité. Il est nécessaire de comprendre si ces différences sont compatibles et éventuellement prendre des décisions qui vont maximiser la valeur métier et minimiser les risques produit.
- L'impact de l'exigence sur les produits ou sous-systèmes existants n'est pas tout à fait clair.
- L'importance de l'impact sur l'infrastructure du projet n'est pas clair. Cela doit être clarifié afin d'avoir une estimation fiable.

Les principaux objectifs de l'atelier d'analyse sont d'examiner tous les aspects suivants d'une exigence :

- Fonctionnalités du produit : User Story, scénarios ou cas d'utilisation et modèles d'analyse liés
- Attributs Qualité : Contraintes sur la fonctionnalité (par exemple, facilité d'utilisation, maintenabilité, fiabilité, sécurité, performance, sécurité)
- Interfaces externes : Intégration / combinaison de deux sous-systèmes ou de composants système
- Technologie ou outils : Prouver la faisabilité technique pour répondre aux exigences du produit, intégrer un nouvel outil dans le projet de développement pour satisfaire les contraintes de conception ou d'implémentation
- Définir les conditions de satisfaction des exigences (User Story) si elles manquent encore.

Le principal objectif de l'atelier d'analyse n'est pas d'effectuer une analyse exhaustive, préliminaire, comme dans les modèles séquentiels, mais plutôt à clarifier les exigences jusqu'à ce qu'elles puissent être correctement évaluées au cours de l'atelier de planification de Release. Il est important de raffiner les éléments de priorité les plus élevés pour obtenir les caractéristiques INVEST, estimer l'effort et planifier les tâches individuelles qui seront réalisées dans les prochains Sprints.

Au cours de l'atelier d'analyse, Il est important de comprendre si des tâches d'analyse/conception spécifiques doivent être prévues pendant les prochains Sprints.

Une fois de plus la communication et la collaboration sont des facteurs clés ; au cours de l'atelier, il est important de dessiner des schémas, diagrammes, et tout ce qui est nécessaire sur des paperboards ou des tableaux blancs, en s'assurant qu'ils sont visibles par tout le monde. Les carnets de notes individuels ne doivent pas être utilisés sauf si absolument nécessaire.

Le résultat de l'atelier doit toujours être enregistré, en utilisant des photos lisibles sur les murs de la salle ou tout ce qui est utile, et être stocké pour un accès futur. Les participants ne quittent pas la pièce tant que tous les résultats n'ont pas été enregistrés.

© GASQ – Global Association for Software Quality

4.2.4 Le Spike

Le Spike est une Story ou une tâche visant à répondre à une question ou à collecter des informations, plutôt que de produire un produit livrable. Se référant au [Leffingwell, 2010], les Spike, une autre invention de XP, sont un type spécial de Story utilisé pour éliminer les risques et l'incertitude dans une User Story ou d'autres facettes du projet. Les Spike peuvent être utilisés pour plusieurs raisons :

- L'équipe n'a peut-être pas la connaissance d'un domaine nouveau et les Spike peuvent être utilisés pour la recherche fondamentale et familiariser l'équipe avec la technologie ou le domaine nouveau.
- La Story peut contenir des risques techniques importants, et l'équipe peut avoir à faire quelques recherches ou faire des prototypes pour gagner de la confiance dans une approche technologique qui leur permettra d'engager les User Story dans le futur.
- La Story peut contenir un risque fonctionnel important. Alors que le but de la Story est connu, on ne sait pas comment le système a besoin d'interagir avec l'utilisateur pour obtenir explicitement un bénéfice.
- Il est clair qu'une interface ou une couche spécifique seront significativement impactées, et l'équipe a besoin d'avoir une connaissance raisonnable de cet impact.

Les Spike peuvent être classés comme Spike Techniques ou Spike Fonctionnels.

Les Spike techniques peuvent être utilisés dans les cas suivants :

- Quand les ateliers d'analyse servent à identifier différentes approches techniques pour la solution, les Spike techniques sont alors utilisés pour vérifier rapidement ces mêmes approches techniques.
- Lorsque des ateliers d'analyse sont utilisés pour déterminer un achat ou un développement, un Spike technique peut servir à vérifier rapidement et à valider ces options d'achat.
- Les Spike techniques peuvent être utilisés pour évaluer les impacts potentiels sur la performance ou la charge d'une nouvelle User Story.
- Les Spike techniques peuvent être utilisés pour toute raison quand l'équipe a besoin de devenir plus confiante dans une approche désirée avant d'affecter la nouvelle fonctionnalité à une itération.

Les Spike fonctionnelles peuvent être utilisés dans les cas suivants :

- Lorsque l'incertitude quant à la façon dont un utilisateur peut interagir avec le système est importante, un Spike fonctionnel peut-être être utilisé pour permettre à l'équipe de faire plus de recherche.
- Les Spike fonctionnels les mieux réussis utilisent souvent un certain degré de prototypage, que ce soient des maquettes d'interface utilisateur, des trames, des flux de pages, ou toute technique consistant à obtenir un meilleur feedback du client ou les parties prenantes (voir 4.3.3).

Pour les organismes de formation : Donner des exemples pratiques, éventuellement à partir de cas réels, quand des ateliers d'analyse doivent être faits plutôt que des Spike.

© GASQ – Global Association for Software Quality

4.2.5 Acceptation de l'exigence

Selon [REQB_FL_SYL], l'acceptation de l'exigence (également appelée agrément ou approbation de l'exigence) est l'élément suivant de l'analyse des exigences. C'est l'accord formel que le contenu et le périmètre des exigences sont exacts et complets.

Dans un contexte Agile :

- Cela est partiellement réalisé dans l'atelier d'exigences, où un accord sur les exigences de haut niveau (exigence métier) a été obtenu.
- Cela est partiellement réalisé au cours de l'atelier de planification de Release, où les exigences ont été acceptées à différents niveaux du développement de la solution comme accord formel pour l'inclusion dans le Backlog de produit.
- Cela est partiellement réalisé par l'accord sur les exigences détaillées (exigences solution/système et les exigences composants/fonction) avec l'équipe de développement pendant l'atelier de planification de Sprints (voir 4.2.5.1).

Dans un projet Agile, l'acceptation des exigences n'est pas de s'assurer que les exigences sont stables et que toutes les modifications sont gérées via des requêtes de changement formelles. L'acceptation des exigences dans Agile signifie s'entendre sur la vision du produit, les exigences métier, le contenu du Backlog de produit et le contenu de chaque Sprint. Les changements du contenu du Backlog de produit, la priorisation et les estimations sont toujours possibles dans des ateliers récursifs. Seul le contenu du Sprint en cours est protégé contre les changements qui pourraient changer l'objectif du Sprint.

Le risque de malentendu entre le client et le fournisseur sur le périmètre du projet est minimisé grâce aux possibilités de feedback à chaque revue de Sprint.

À la fin de chaque atelier, le résultat est le (nouvel) accord entre les parties prenantes participants.

4.2.5.1 Atelier de planification de Sprint

Chaque Sprint commence avec un événement limité en temps appelé l'atelier de planification de Sprint. L'équipe Scrum collabore pour sélectionner et comprendre le travail à accomplir dans le Sprint à venir. Toute l'équipe participe à la réunion de planification de Sprint. En partant du Backlog de produit priorisé, le Product Owner et les membres de l'équipe de développement discutent de chaque élément pour arriver à une compréhension partagée de l'élément et de ce qui est à faire pour le réaliser en conformité avec la Définition de « Fini ».

Dans Scrum, la réunion de planification de Sprint est décrite comme ayant deux parties :

Partie 1 : Déterminer quels travaux achever dans le Sprint.

- Le Product Owner informe (les) l'équipe(s) de la vision produit et du Backlog de produit priorisé et définit l'objectif du Sprint.
- Le Product Owner et (les) l'équipe(s) discutent de la définition de « Fini » et, après tout accord, le valide.
- L'équipe estime l'effort de chaque User Story proposée par le Product Owner comme partie du Sprint.
- L'équipe vérifie l'effort estimé, la durée du Sprint et la disponibilité de l'équipe. Les Story sont ajoutées ou supprimées de l'ensemble sélectionné pour le Backlog de Sprint afin de correspondre à la vélocité de l'équipe.

© GASQ – Global Association for Software Quality

- Les équipes s'affectent les Story jusqu'à ce que chacune d'elle soit assignée.
- L'équipe confirme le calendrier du Sprint.

Participants invités pour la partie 1 :

- Tous les membres de l'équipe
- Le Scrum Master
- Le Product Owner (Responsable de la réunion)
- Les autres parties prenantes sont facultatives

Partie 2 : Déterminer comment le travail sera réalisé.

- Les membres de l'équipe définissent les tâches pour chaque User Story du Sprint.
- Les responsables de tâche estiment l'effort pour chaque tâche. Si une tâche prend plus d'un jour, elle doit être divisée en tâches plus petites.
- L'équipe vérifie que chaque bout d'activité est pris en compte (selon la définition de « fini »), y compris les analyses et conceptions supplémentaires, réunions, l'apprentissage des nouvelles technologies et les travaux d'infrastructure.
- Si l'équipe pense que le Backlog de Sprint est trop grand, des éléments de Backlog sont supprimés en accord avec le Product Owner.
- Si l'équipe pense que le Backlog de Sprint est trop petit, des éléments du Backlog les plus importants sont déplacés du Backlog de produit vers le Backlog de Sprint, en accord avec le Product Owner. L'objectif du Sprint est ajusté si nécessaire.
- L'équipe s'engage sur l'objectif du Sprint.

Participants invités pour la partie 2 :

- Tous les membres de l'équipe
- Le Scrum Master (Responsable de la réunion)
- Le Product Owner (doit être disponible pour toute question)

Le Planning Poker peut être utilisé pour les estimations des partie 1 et 2 (voir 6.2.1).

Pour les organismes de formation : Prévoir un exercice d'atelier de planification de Sprint pour une application simplifiée.

4.2.6 L'atelier de Grooming

Au début de chaque Sprint, les éléments les plus raffinés, ceux qui seront inclus dans le Sprint et insérés dans le Backlog de Sprint, sont placés en haut du Backlog de Produit. À la fin de chaque Sprint, les éléments qui sont réalisés et sont acceptés comme terminés, sont ensuite marqués du statut de « Fini ». Ceux non terminés ou non approuvés dans la revue de Sprint doivent être reportés, suivant la procédure déjà décrite pour la création du Backlog de Sprint du prochain Sprint.

Pendant un Sprint, il est nécessaire de réitérer les activités menées avant l'atelier de planification de Sprint, qui consistent à raffiner un sous-ensemble de Story du Backlog de produit (à l'aide de INVEST). Ces Story raffinées seront la base de la discussion du contenu du nouveau Sprint dans le prochain atelier de planification de Sprint.

© GASQ – Global Association for Software Quality

Cette activité est réalisée dans l'atelier de Grooming, qui est structuré en partie comme l'atelier de planification de Release même si la participation est limitée à ceux qui peuvent aider à raffiner la partie du Backlog de produit ciblé pour le prochain Sprint telle qu'identifiée par le Product Owner .

La participation à cet atelier de Grooming est requise pour :

- Le Product Owner
- Le Scrum Master
- L'équipe de développement
- Les acteurs concernés par les points en discussion

Les méthodes de préparation et de développement seront sensiblement similaires à celles des ateliers de planification de Release, des ateliers d'analyse et de Spike.

Pour les organismes de formation : Expliquer, avec des exemples réels, pourquoi l'atelier de Grooming est l'un des principaux événements collaboratifs dans Scrum.

4.2.7 Modélisation de la solution en Agile

La modélisation de la solution comme décrite dans [REQB_FL_SYL] est toujours valable dans un contexte Agile. Les types de modèles comprennent les diagrammes de contexte, les scénarios, les modèles de données, les tableaux d'évènements, les diagrammes d'état et les règles métier.

Dans un projet Agile, la modélisation est utilisée pour la conversation, la conception et pour faire se comprendre et s'entendre l'équipe et les parties prenantes, avant d'être utilisée pour documenter. En ce sens ces diagrammes sont dessinés à la main sur un paperboard ou un tableau blanc avant d'être reproduits avec un outil pour être inclus dans un document. Le document peut être nécessaire pour apporter une valeur métier, mais ne doit pas être utilisé s'il n'apporte pas de valeur et n'est juste qu'une partie de la documentation traditionnelle créée.

4.3 Spécification des exigences

En développement des exigences, la spécification des exigences comprend la production des documents de spécification des exigences.

Dans un projet Agile, l'équipe devrait "favoriser le coté opérationnel du logiciel plus que l'exhaustivité de la documentation", mais cela ne signifie pas que la documentation n'est pas nécessaire. L'équipe devrait aller au plus simple et ne documenter que ce qui a un sens d'être fait.

Dans un contexte Agile, la spécification des exigences ne signifie pas produire un document exhaustif, mais plutôt d'enregistrer, de conserver et de rendre disponible, toutes les évidences et les artefacts créés lors des conversations entre les participants aux ateliers. Les documents papier sont les bienvenus, limités à ce que les équipes considèrent nécessaires pour permettre l'implémentation. D'autres documents papier sont bienvenus lorsqu'il est nécessaire de documenter ce qui a été fait (pas ce qui va être fait), par exemple pour la documentation client ou pour répondre

© GASQ – Global Association for Software Quality

aux obligations. La spécification des exigences est un processus continu qui démarre avec des ateliers d'exigences avant et pendant les Sprints.

Une spécification d'exigences fournie au client par le fournisseur est une bonne source d'information pour toutes les activités du processus d'ingénierie des exigences. Mais même si ce document est fourni, le processus de développement des exigences Agile doit fonctionner comme décrit dans cette section.

4.3.1 Spécification d'exigence par la création et le raffinement collaboratif des User Story

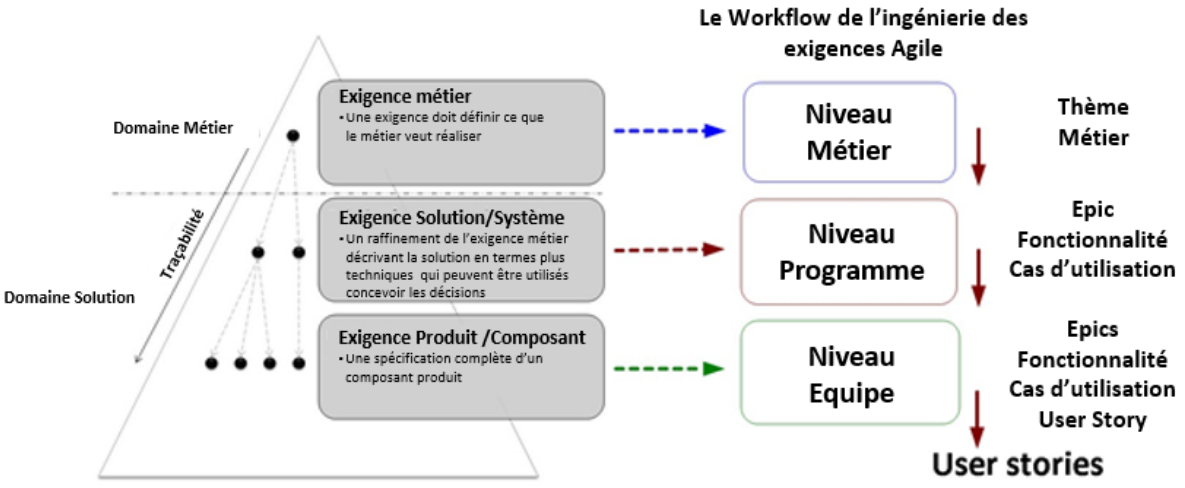
Quand on passe d'une méthodologie traditionnelle à l'Agile, les gens veulent savoir comment obtenir toutes les informations nécessaires pour comprendre les exigences et les besoins de l'utilisateur. Ils veulent aussi savoir comment spécifier, de manière suffisamment complète, la solution qui doit être mise en œuvre lorsque l'exigence est seulement exprimée en quelques mots dans une User Story. Souvent, il semble que les exigences Agile ont considérablement moins d'informations que les exigences traditionnelles.

En réalité, selon le concept de 3c [Jeffries, 2001], une User Story se compose de trois éléments :

- Carte : La carte est le support physique décrivant la User Story et ses avantages (qui, quoi et pourquoi). Elle identifie également l'exigence, sa criticité (valeur métier, priorité, risque), et l'effort attendu pour bien la réaliser. L'information doit être suffisamment précise pour être utilisée dans le Backlog de produit.
- Conversation : L'élément Conversation de la User Story, permet un dialogue continu autour de l'exigence, pour définir comment elle sera implémentée, testée, comment le logiciel sera utilisé, comment les aspects non fonctionnels attendus seront garantis, quelles solutions techniques seront utilisées et quelles contraintes doivent être gérées. Indépendamment de comment les résultats de la conversation sont documentés et enregistrés, toutes ces informations forment la composante fondamentale de la spécification d'exigences.
- Confirmation : Les critères d'acceptation, discutés dans la conversation, sont utilisés pour confirmer que la Story est « finie ». Ces critères d'acceptation peuvent couvrir plusieurs User Story. Des tests positifs et négatifs doivent servir à couvrir les critères. Lors de la confirmation, divers participants jouent le rôle d'un testeur. Cela peut inclure les développeurs tout comme des spécialistes en performance, sécurité, interopérabilité et en autres caractéristiques qualité. Pour confirmer qu'une Story est « finie », les critères d'acceptation définis doivent être testés et démontrés comme satisfaits.

Les équipes Agile varient en termes de documentation des User Story. Quelle que soit l'approche, la documentation devrait être concise, nécessaire et suffisante.

© GASQ – Global Association for Software Quality



Dans le chapitre 2, nous avons vu (**Figure 2**, à nouveau ci-dessus) comment les exigences sont entrées dans un Backlog de produit à la fin d'un processus de raffinement qui commence à partir de la définition d'un portefeuille. Le processus de raffinement inclut de comprendre les thèmes métier généraux, en sélectionnant les paramètres appropriés (comme indiqué au point 2.1.2) afin de déterminer quelles fonctionnalités la Release d'un produit doit contenir, et ensuite diviser les exigences dans des User Story de différentes tailles.

Les User Story qui atteignent ce niveau maximal de raffinement (résumé par l'acronyme INVEST) sont déplacées vers la partie supérieure du Backlog de produit et deviennent la cible du premier Sprint prévu.

Les éléments de spécification qui ne respectent pas tous les critères INVEST, surtout pour la taille, doivent être encore raffinés. Ce processus peut prendre plus d'une étape de raffinement car les User Story qui en résultent peuvent elles-mêmes être des EPIC à raffiner plus tard.

On trouvera des exemples de raffinement d'éléments de spécification dans [Cohn, 2004], [Cohn, 2010] chapitre 13 et [Cohn, 2012].

Pendant l'élicitation des exigences et l'analyse des besoins, le processus de raffinement est basé sur des conversations entre le Product Owner, les parties prenantes représentant directement ou indirectement les clients, les équipes de développement et les autres parties intéressées représentant le métier en support des équipes de développement. Ce raffinement des User Story a généralement lieu dans le cadre d'ateliers impliquant tous ceux qui ont les connaissances pour contribuer. L'objectif de ces ateliers est d'atteindre un niveau de raffinement pour un nombre suffisant de User Story et ainsi passer à l'étape suivante.

Tout ce qui se dégage de ces conversations commence à former les spécifications d'exigences. Ces spécifications ne sont pas nécessairement écrites dans un document papier ou électronique, mais seront collectées dans tout format jugé convenable. Les formats appropriés fournissent les fonctionnalités suivantes :

- Stockage de ce qui a été dit et décidé, afin que tout le monde puisse facilement trouver la référence et comprendre le contenu
- Capacité à modifier ces spécifications chaque fois que les conditions l'exigent
- Traçabilité entre l'exigence le besoin réel

© GASQ – Global Association for Software Quality

Au cours des ateliers, le résultat des discussions sur les exigences peut être collecté de plusieurs façons :

- Sur de grandes feuilles de papier accrochées aux murs, où, par exemple, des modèles sont dessinés, et qui sont stockées à la fin de l'atelier ou simplement photographiées pour être facilement distribuées
- Sur les modèles dessinés sur un tableau blanc, dont une copie est imprimée ou une photo prise comme enregistrement
- Sur des tickets affichés sur un mur pour représenter chaque User Story
- Avec un outil qui permet le partage des idées entre équipes distribuées
- Avec d'autres outils ou médias qui favorisent la collaboration et le partage de l'information

Il est important de noter une différence fondamentale entre un document de spécification des exigences et les techniques de spécification des exigences utilisées dans un projet Agile.

Le document de spécification tente de recueillir tous les éléments nécessaires pour comprendre les exigences dans un seul document, qui sera soumis à examen et approbation. Dans cette approche, la qualité de l'examen du document n'est pas sous le contrôle de l'auteur et une interprétation subjective est susceptible de se produire lorsqu'il n'y a aucun partage d'informations générales.

Dans un projet Agile, la spécification des exigences repose solidement sur ce partage, les détails se forment dans le cadre des conversations pendant les ateliers, et les détails étape par étape sont identifiés et partagés autant que nécessaire pour passer à l'étape suivante.

Pour les organismes de formation : Donner des exemples, éventuellement à partir de cas réels, de documentation des exigences dans un environnement Agile (Murs, wikis, etc.) et montrer comment une User Story est composée des éléments 3C.

4.3.1.1 Eléments communs nécessitant de découper une User Story

Ce qui suit est une liste d'éléments habituels qui peuvent indiquer qu'une Story devrait être subdivisée en plusieurs Story :

1. Etapes de workflow – un workflow complet se compose de plusieurs étapes. L'équipe peut identifier les étapes précises qu'un utilisateur réalise pour accomplir un workflow spécifique et ensuite faire une User Story raffinée pour chaque étape.
2. Variations de règles métier - A première vue, certaines Story semblaient assez simples. Toutefois, les règles métier sont parfois plus complexes ou plus étendues que le premier coup d'œil le laissait croire.
   Dans ce cas, il peut être utile de diviser la Story en plusieurs Story pour gérer la complexité de la règle métier.
3. Effort majeur- Parfois une Story peut être divisée en plusieurs parties où l'effort principal portera sur l'implémentation de la première partie. Dans de nombreux cas, l'infrastructure devrait être construite pour permettre la première Story ; l'ajout ultérieur de nouvelles fonctionnalités devrait être relativement trivial.
4. Simple/complexe - Quand l'équipe discute d'une Story et que la Story semble devenir de plus en plus large (« Et quid ce sujet ? Avez-vous pensé à ça ?"), Arrêter et demander, "Quelle est la version la plus simple qui peut éventuellement fonctionner ?" Utiliser cette version simple comme Story et puis faire des Story avec toutes les variantes et complexités.
5. Variations de données - les variations de données et sources de données sont une autre source de grandeur et de complexité. Les Story peuvent être isolées pour ne gérer que certains types de données.

© GASQ – Global Association for Software Quality

6. Méthodes d'entrée des données - Parfois la complexité est dans l'IHM plutôt que dans la fonctionnalité elle-même. Dans ce cas, il peut être judicieux de scinder la User Story pour la construire avec l'interface utilisateur la plus simple possible et construire une interface utilisateur plus riche plus tard.
7. Attributs Qualité - Parfois, la mise en œuvre initiale n'est pas difficile, et la majeure partie de l'effort porte sur la rapidité d'exécution, la fiabilité et la précision ou l'évolution. Toutefois, l'équipe peut apprendre beaucoup de l'implémentation de base qui apporte la valeur à un utilisateur. Dans ce cas, il peut être judicieux de décomposer la Story en User Story successives traitant des différents aspects non fonctionnels.
8. Opérations - Des mots tels que « gérer » ou « contrôler » sont un cadeau que la User Story peut couvrir en plusieurs fois, ce qui peut offrir un moyen naturel de diviser la Story.
9. Scénarios de Cas d'utilisation - Si des cas d'utilisation ont été développés pour représenter des interactions utilisateur/système ou système/système complexes, alors la Story peut souvent être répartie selon les différents scénarios de cas d'utilisation.
10. Eclater un Spike - Dans certains cas, une Story peut être trop grande ou trop complexe, ou peut-être son implémentation est mal comprise. Dans ce cas, l'équipe doit monter un Spike technique ou fonctionnel pour la comprendre et ensuite diviser la Story en fonction de ce résultat. (Voir 4.2.4).)

4.3.2 Affecter les conditions de satisfaction aux User Story raffinées

En spécification des exigences Agile, un élément clé consiste à affecter à chaque User Story les conditions qui satisfont l'exigence. Cela anticipe et permet la définition des critères d'acceptation, en fournissant les conditions selon lesquelles on peut considérer la User Story satisfaite du point de vue de l'utilisateur.

Lors de la discussion sur les conditions de satisfaction, souvent des critères qui ne doivent pas être satisfaits, car ils ne sont pas requis, seront également analysés. Le suivi de ces questions permettra de réduire le risque d'implémenter des parties de fonctionnalités qui ne sont pas nécessaires (déchets) et permettra à l'équipe de se focaliser sur ce qui est réellement nécessaire. Il s'agit d'un avantage que la spécification écrite dans un document officiel ne présente pas.

Il est important d'avoir les conditions de satisfaction déjà définie pour implémenter une User Story et parvenir à ses propriétés INVEST. Ces conditions devraient être discutées et recueillies lors d'un atelier d'exigences, un atelier de planification de Release et parfois lors d'un atelier de Grooming.

Pour les organismes de formation : Donner des exemples d'affectation des conditions de satisfaction, requises et pas requises. Par la suite ce sera utilisé en montrant la différence entre une condition de satisfaction et un critère d'acceptation (4.3.4).

4.3.3 Consacrer du temps pour l'expérience utilisateur

Un problème courant en développement logiciel Agile consiste à déterminer comment intégrer la conception visuelle complète du produit dans des itérations rapides. Lorsque les équipes tentent de résoudre des interactions utilisateur complexes tout en essayant de coder et de tester le système en même temps, ils peuvent finir souvent par recoder pendant de nombreuses itérations. Cela est une perte car cela retarde les boucles de feedback, ce qui complique la productivité et retarde la livraison de la valeur. [Leffingwell, 2011]

Un certain nombre de pratiques pour aborder ces éléments de conception aident les itérations à chasser l'incertitude et les risques grâce à un feedback rapide :

© GASQ – Global Association for Software Quality

- Fournir des prototypes de papier, des prototypes HTML simples, pour revue par l'utilisateur
- Conduire des Spikes d'expérience utilisateur qui peuvent être utilisés pour développer des interfaces utilisateur et les faire tester par les utilisateurs réels ou leur représentant
- Créer une petite autorité de conception d'expérience utilisateur centralisée, qui fournira les normes de conception de base et les maquettes préliminaires pour chaque interface utilisateur. Dans ce cas les équipes dans les Sprints s'appuieront pour l'implémentation sur des experts qui proviennent d'une équipe centralisée. L'autorité centralisée fournira des conceptions HTML, des feuilles de style CSS, un contrôle des logos, des maquettes, des lignes directrices d'ergonomie et autres artefacts qui fournissent une intégrité conceptuelle de l'expérience utilisateur pour l'ensemble de la solution.

Prototypes papier, résultats de Spike d'expérience utilisateur et artefacts de conception centralisée d'expérience utilisateur font partie de la spécification des exigences.

4.3.4 Attribuer les critères d'acceptation

Les critères d'acceptation des User Story constituent la base pour les tests fonctionnels destinés à assurer que chaque nouvelle User Story se comporte comme prévu et finalement répond aux besoins des utilisateurs et des Product Owner. En général, ce qui suit concernant les critères d'acceptation est vrai [Leffingwell, 2011] :

- Ils sont rédigés dans la langue du domaine métier.
- Ils sont mis au point lors d'une conversation entre les développeurs, les testeurs et le Product Owner. Le résultat de la conversation est enregistré et sauvegardé.
- Bien que n'importe qui peut écrire des tests, le Product Owner, comme proxy du client/responsable métier, est le principal responsable des tests d'acceptation.
- Ce sont des tests boîte noire car ils vérifient uniquement que les sorties du système remplissent les conditions de satisfaction, sans regarder comment le résultat est atteint.
- Les tests d'acceptation sont mis en œuvre au cours de la même itération que l'implémentation de la User Story.

Cela signifie que de nouveaux tests d'acceptation sont créés pour chaque nouvelle User Story. Si une Story ne passe pas ses tests, l'équipes ne reçoit pas de crédit pour la Story. La Story est reportée à la prochaine itération, où le code ou le test ou les deux, sont remaniés jusqu'à ce que les tests passent.

Pour les organismes de formation : Donner des exemples de critères d'acceptation. Montrant la différence entre une condition de satisfaction (4.3.2) et des critères d'acceptation.

4.4 Validation et vérification des exigences

Dans le développement des exigences, la vérification et la validation des exigences inclut le contrôle de la qualité des exigences et des documents de spécification des exigences.

Selon [REQB_FL_SYL], les défauts résultant d'exigences de faible qualité sont plus coûteux à corriger dans les phases finales du projet que les autres types de défauts. En outre, plus les défauts sont détectés tard, plus le coût de correction est élevé. Par conséquent, l'utilisation de la vérification (" nous produisons le produit correctement ") et la validation (" nous produisons le bon produit ") des exigences sont des activités clés.

© GASQ – Global Association for Software Quality

En développement logiciel Agile, la validation et la vérification des exigences s'appuie principalement sur le mécanisme de feedback continu qui est intrinsèque aux Sprints, car en Agile avec Scrum, chaque Sprint est tenu de livrer un incrément de produit potentiellement expédiable. En pensant en termes Lean et Agile, il est important d'avoir une vision globale et systémique. Les Story (exigences), l'implémentation (code) et la validation (tests d'acceptation, tests unitaires et autres) ne sont pas des activités distinctes, mais plutôt un raffinement continu de la profondeur de compréhension.

Le logiciel doit être systématiquement testé contre la régression pour s'assurer qu'il continue à fonctionner. Cela se fait exécutant et en automatisant (dans la mesure du possible) des tests d'acceptation. De nombreuses équipes écrivent les tests d'acceptation des Story en utilisant la méthode « Tests en premier », avant de développer le code. C'est ce qu'on appelle le développement piloté par les tests d'acceptation (ATDD). Les tests d'acceptation servent à enregistrer les décisions prises pendant la conversation afin que l'équipe comprenne les spécificités du comportement de la carte User Story. Le code suit logiquement par la suite. [ISTQB_FA_SYL]

4.4.1 La revue de Sprint

À la fin de chaque Sprint, l'équipe a produit un morceau de logiciel, codé, testé et utilisable qui est démontré lors de la réunion de revue de Sprint. Au cours de cette réunion, l'équipe Scrum montre ce qu'elle a accompli au cours du Sprint. Typiquement, cela prend la forme d'une démonstration des nouvelles fonctionnalités au Product Owner, à l'équipe, au Scrum Master, aux clients et parties prenantes intéressées. La revue de Sprint est intentionnellement informelle et interdit souvent la présentation de diapositives.

Au cours de la revue de Sprint, le projet est évalué par rapport à l'objectif qui a été déterminé au cours de l'atelier de planification du Sprint. Idéalement, l'équipe a terminé chaque élément de Backlog de produit introduit dans le Sprint, mais le plus important est que l'objectif global du Sprint soit atteint. Avant le début de la réunion, pour chaque Story du Backlog de Sprint, la définition de « fini » est vérifiée afin de s'assurer que la Story a obtenu le statut terminé.

Le résultat de la revue de Sprint doit toujours être enregistré. Les participants ne quittent pas la pièce tant que tous les résultats n'ont été enregistrés.

4.4.2 la rétrospective

Telle que présentée dans la section 1.2.4, la rétrospective est l'événement « inspecter et s'adapter » qui arrive à la fin de chaque Sprint. Après la revue de Sprint, quand l'équipe sait si le Sprint a réussi ou pas, c'est-à-dire, s'il a été approuvé ou non par le Product Owner et les parties prenantes, les équipes se retrouvent ensemble pour regarder ce qui s'est bien passé et ce qu'il est nécessaire d'améliorer au cours des Sprints, pour essayer de comprendre les raisons des bonnes et des mauvaises choses et pour essayer de renforcer les bonnes pratiques et de corriger les erreurs.

Un temps raisonnable de cette réunion devrait être consacrée aux exigences, par exemple :

- En regardant le résultat du Sprint, avons-nous implémenté les exigences comme prévu par les parties prenantes ?
- Si non, qu'avons-nous mal fait, et en particulier, avons-nous raffiné les exigences correctement ?

© GASQ – Global Association for Software Quality

- Si non, quand et comment avons-nous raffiné incorrectement et en particulier avons-nous analysé l'exigence correctement ?
- Si non, pourquoi et où avons-nous analysé incorrectement et en particulier avons-nous compris correctement l'exigence ?

Cela aide l'équipe à déterminer les points forts et faibles du développement des exigences. L'objectif est d'identifier les actions correctives et de les mettre en place avec une urgence qui correspond à la gravité du problème. Si nécessaire, de nouveaux points concernant des éléments du Backlog de produit sont soulevés.

Les participants invités à la rétrospective incluent :

- Tous les membres de l'équipe
- Le Scrum Master (Responsable de la réunion)
- Le Product Owner (facultatif, mais recommandé si le but du Sprint n'a pas été atteint)

Dans le cas où plus d'une équipe a été impliquée, il est utile de faire une réunion avec une seule équipe la première partie de la rétrospective, puis une deuxième partie où toutes les équipes peuvent s'asseoir ensemble (peut-être en n'impliquant que quelques représentants des équipes) pour discuter de problèmes communs, parfois avec le Product Owner.

Pour les organismes de formation : Fournir quelques exemples sur la façon d'identifier les problèmes liés à une mauvaise ingénierie des exigences pendant les rétrospectives et comment mettre en place des actions correctives.

4.5 Estimation des User Story

Dans les sections précédentes, plusieurs valeurs ont été discutées devant servir à estimer une User Story. Cette section retrace les caractéristiques de chacune de ces estimations, y compris quand les faire et quand les réévaluer quand elles sont raffinées.

Une User Story, ou plus généralement une exigence, peut être estimée à en fonction de :

- Valeur métier
- Priorité des clients
- Complexité
- Taille relative
- Risque
- Effort

4.5.1 Valeur métier

La valeur métier est le premier critère pour évaluer une exigence. Les parties prenantes métier évaluent l'avantage pour le métier qui sera gagné suite à l'implémentation de l'exigence. Cette évaluation se fondera sur la vision propre de la partie prenante et de sa visibilité dans le métier.

La valeur métier est estimée au cours de l'atelier d'exigences, selon les critères d'évaluation convenus entre les participants. La méthode d'évaluation doit être une valeur relative, non absolue, donc si deux conditions sont évaluées respectivement à valeur métier de 5 et 10, il est entendu que le second a le double de la valeur du premier, sans donner un montant précis pour les deux nombres. Il est

© GASQ – Global Association for Software Quality

parfois utile d'entamer l'évaluation avec une exigence que les parties prenantes considèrent comme de grande valeur – par exemple, la valeur BIG en utilisant la technique de Planning Poker (point 6.2.1) – puis d'évaluer les exigences suivantes par rapport à la première et à celles qui suivent.

Quand une exigence est raffinée, la valeur métier devrait être répartie entre tous les éléments dérivés du raffinement. Il existe deux situations possibles :

- L'exigence a été raffinée en plusieurs User Story qui sont implémentées dans un ordre quelconque, mais il n'y a aucune valeur métier tant que la dernière n'est pas « Finie ». Dans ce cas, la valeur métier de l'ensemble est attribuée à la dernière User Story, tandis que les autres obtiendront une valeur de 0.
- L'exigence a été raffinée dans plusieurs User Story ; chacune d'elle offre une valeur métier une fois achevée, peut-être dans une certaine séquence. Dans ce cas, la valeur initiale doit être distribuée parmi les User Story raffinées, la somme de toutes étant la valeur initiale.

Les cas réels sont habituellement un mélange des deux. Lorsqu'une exigence implique la réalisation de ses raffinements en séquence pour fournir de la valeur, souvent le terme « saga » indique cette séquence.

4.5.2 Priorité client

La priorité Client nous renseigne à quel point une exigence est importante pour le client. Les représentants client évalueront l'avantage de l'implémentation de l'exigence pour le métier, selon leur propre vision et leur visibilité, et cela se traduira par une priorité.

La priorité client est estimée au cours de l'atelier d'exigences, selon les critères d'évaluation convenus entre les participants. Comme pour la valeur métier, la méthode d'évaluation doit produire une valeur relative, non absolue.

Dans le cas où plusieurs représentants du client seront présents et que l'on sait que leurs évaluations vont diverger, il est préférable de déplacer cette estimation avant l'atelier, de garder les différentes estimations et d'éviter la discussion ici. Le Product Owner gèrera les priorités des clients avec ses correspondants métier afin de trouver une clé pour interpréter les différents points de vue sur les opportunités métier et les pièges.

La priorité de la User Story n'est pas nécessairement identique à la priorité des clients. Selon la stratégie, la priorité de la User Story peut être une combinaison de la valeur métier, de la priorité des clients et des risques.

Quand une exigence est raffinée, la priorité client est la même pour tous les éléments dérivés du raffinement.

4.5.3 Taille relative et complexité

Une estimation de la taille relative est généralement faite au cours de la réunion de planification de Release. Cette estimation est utilisée pour fournir une première compréhension de l'effort de développement qui est nécessaire. La méthode d'évaluation doit être une valeur relative, non absolue.

L'estimation de la « taille » d'une User Story inclut habituellement une estimation de l'effort de l'équipe de développement. Il est parfois plus facile de discuter de la taille en termes de complexité pour que les parties prenantes non-développeur puissent mieux comprendre l'estimation. Les

© GASQ – Global Association for Software Quality

techniques de vote telles que le Planning Poker offrent la possibilité à chaque estimateur d'expliquer son estimation aux autres.

Quand une exigence est raffinée, la taille devrait être répartie entre tous les éléments dérivés du raffinement, chacun d'eux ayant une valeur de taille. La somme des tailles des User Story dérivées du raffinement peut différer de la taille de l'élément d'origine avant le raffinement. Cela parce qu'une nouvelle estimation est possible après le raffinement.

4.5.4 Risque

Une estimation du risque est faite généralement au cours de la réunion de planification de la Release et parfois au cours de la réunion de planification de Sprint, notamment en matière de risques technique. Une estimation du risque lors de l'atelier d'exigences est habituellement prématurée parce qu'il n'y a pas suffisamment d'informations sur l'exigence. Lorsque l'estimation du risque a été faite, il est important que les bonnes parties prenantes soient présentes. Le Product Owner doit comprendre quand procéder à une estimation du risque en fonction du public à impliquer. Chaque User Story doit avoir sa propre valeur de risque. Lorsqu'une Story est raffinée, sa valeur de risque doit être mise à jour.

L'analyse et l'identification des risques fait partie de la gestion des exigences (section 5.1).

Selon les critères définis avant l'atelier, la méthode d'évaluation peut donner une valeur relative ou une valeur absolue.

4.5.5 Effort

La valeur de l'effort est plus complexe à estimer que les autres valeurs. Le contenu du Sprint dépend de cette estimation et la capacité de l'équipe pour atteindre l'objectif de Sprint repose sur la fiabilité de ces chiffres. Agile contribue à cette estimation de la manière suivante :

- Les estimations sont faites pour des efforts de courte durée. Les User Story doivent être implémentées dans un Sprint ; les tâches individuelles qui composent une User Story demandent généralement des heures plutôt que de jours.
- À tout moment, les progrès pour chaque activité sont connus. L'équipe adapte rapidement la précision de ses estimations.
- Les données statistiques sur la vélocité de l'équipe de développement sont disponibles en peu de temps. Ces données peuvent servir comme référence pour les estimations ultérieures. La vélocité est exprimée en points de Story de tâches complétées par unité de temps. Une fois que les points de Story pour un ensemble de Story est connu, cette information est utilisée pour prévoir le temps nécessaire pour effectuer les tâches du projet.
- Le mécanisme « inspecter et s'adapter » avec la rétrospective permet à l'équipe d'affiner ses estimations de l'effort.

L'unité d'estimation est idéalement le Jour développeur Idéal (heures pour des itérations très courtes). Dans les premiers Sprint, il se fondera sur le temps prévu pour réaliser les éléments du Sprint. Par la suite, il se basera sur la vélocité de l'équipe et la valeur en points de Story.

Pour les organismes de formation : Donner des exemples d'estimations de l'effort, fondés sur le nombre de Jour/Heure Développeur Idéal et montrer des exemples de User Story qui doivent être

© GASQ – Global Association for Software Quality

raffinés à cause d'un effort trop élevé. Montrer comment la vélocité l'équipe peut être utilisée après quelques Sprints.

5 Gestion des exigences Agile (70 minutes)

Mots Clé

Changement, gestion du changement, demande de changement, gestion de configuration, collaboration continue, feedback continu, métrique, risque produit, risque projet, assurance qualité, contrôle qualité, risque, gestion des risques, atténuation des risques, traçabilité

Objectifs d'apprentissage pour la gestion des exigences Agile

5.1 Gestion de projet et des risques (15 minutes)

AR-5.1.1 Comprendre les problèmes habituels, les risques projet et les risques produit liés à l'ingénierie des exigences Agile (K2)

5.2 Traçabilité des exigences Agile (15 minutes)

AR-5.2.1 Comprendre comment la traçabilité des exigences peut être réalisée dans les projets Agile et quand des pratiques de traçabilité spécifiques devraient être ajoutées (K2)

5.3 Gestion de configuration et du changement (15 minutes)

AR-5.3.1 Expliquer comment la gestion du changement fonctionne en Agile (K2)

5.4 Assurance Qualité (25 minutes)

AR-5.4.1 Rappeler les facteurs qui influencent les impératifs de qualité des produits et des processus d'ingénierie des exigences (K1)

AR-5.4.2 Comprendre les critères d'utilisation des métriques en ingénierie des exigences Agile. (K2)

Selon [REQB_FL_SYL], la gestion des exigences est principalement une collection d'activités de gestion et de support qui sont nécessaires pour s'assurer que le processus de développement des exigences est exécuté correctement pendant le cycle de vie du produit.

5.1 Gestion de projet et des risques

Une processus d'ingénierie des exigences pauvre est un risque intrinsèque pour n'importe quel modèle de développement, car cela peut entraîner des exigences non précises, contradictoires, ne remplissant pas les critères et qui ne satisfont pas les besoins des parties prenantes [REQB_FL_SYL].

© GASQ – Global Association for Software Quality

En Agile, Scrum en particulier, organise les activités avant, pendant et après le développement à travers une série d'événements et de pratiques, axées sur le développement des exigences. Sans une approche systématique de développement des exigences, un projet Agile ne peut même pas démarrer.

Les problèmes les plus courants associés à l'ingénierie des exigences Agile sont les suivants :

- Vision métier peu claire et manque d'objectifs métier à atteindre au sein de la solution
- Mauvaise implication des parties prenantes pendant l'élicitation et le raffinement des exigences
- Exigences contradictoires entre les différents clients où la priorité ne peut pas être établie sur la base d'objectifs métier clairs
- Raffinement insuffisant des exigences
- Manque d'analyse, ce qui entraîne souvent des estimations imprécises de l'impact des exigences et de leur changement sur d'autres produits en cours de développement
- Estimations de taille ou d'effort qui sont trop loin de la réalité (souvent découlant d'exigences peu claires) résultant dans des buts de Sprint n'ont atteints

Ces problèmes énumérés ci-dessus peuvent être perçus comme un risque. Pour faire face à ces risques – et d'autres risques également – un processus de gestion des risques est nécessaire.

Dans un projet Agile, les risques sont identifiés, estimés et affectés aux User Story au cours des ateliers et puis gérés pendant les Sprints.

[REQB_FL_SYL] classe deux principaux types de risques : Les Risques produit et les Risques projet.

Les risques projet sont les risques qui gênent la capacité du projet à réaliser ses objectifs. Les risques qui sont particulièrement pertinents dans une organisation Agile comprennent :

- Manque de confiance avec les méthodes et les pratiques Agile
- Résistance de l'organisme à la transition de l'approche traditionnelle vers l'Agile
- Faible expérience du Product Owner et du Scrum Master
- Parties inter-fonctionnelles non couvertes adéquatement par l'équipe
- Équipes allouées à temps partiel
- Intégration continue non implémentée
- Définition de « Fini » négligée
- Temps limités des Sprints prolongés
- Pratiques de rétrospectives pauvres incluant l'oubli de rétrospectives

Les risques produit sont des zones de défaillance potentielles (évènements ou dangers futurs indésirables) dans le logiciel ou le système. Ce sont les risques qualité Produit. Les risques qui sont spécifiquement liés aux pratiques Agile comprennent :

- Peu d'automatisation de test et entretien insuffisant de l'environnement de test
- Refactorisation jamais faite
- Pas de participation des parties prenantes ou participation des mauvaises parties prenantes dans la revue de Sprint

L'atténuation des risques est généralement gérée comme toute autre activité en Agile. Cela peut être traduit en User Story supplémentaires à ajouter au Backlog de produit, des éléments devant figurer dans le Backlog d'obstacles, des contrôles systématiques à inclure dans la définition de « Fini », les pratiques à utiliser au cours du Sprint et ainsi de suite. De cette manière tout risque peut être géré et contrôlé par les mécanismes de Scrum.

© GASQ – Global Association for Software Quality

5.2 Traçabilité des exigences Agile

La traçabilité est la capacité de relier des artefacts d'un projet à d'autres. Une matrice de traçabilité des exigences est l'artefact qui est le souvent créé pour enregistrer ces relations. Elle commence avec les exigences individuelles et les trace à travers n'importe quel modèle d'analyse, modèle d'architecture, modèle de conception, code source ou les cas de test qui sont maintenus.

L'avantage d'avoir une telle matrice est que cela rend plus facile la réalisation d'une analyse d'impact relative à une modification d'exigence parce que les aspects du système susceptibles d'être touchés sont identifiés.

Dans une matrice de traçabilité des exigences, les informations de traçabilité sont souvent maintenues manuellement. La création d'une matrice de traçabilité, comme de tout autre artefact ou document en développement Agile, doit être justifiée en fournissant une valeur ajoutée réelle. La valeur réelle est déterminée en équilibrant les éléments suivant :

- Avantages
- Risque de ne pas en avoir compte tenu de la complexité du contexte
- Coût total de possession
- Contribution à la traçabilité déjà fournie par des artefacts et des outils

Concernant ce dernier point, il est important de noter que dans un processus Agile, les exigences sont raffinées sur place, non transférées. En fait, à des fins de traçabilité amont et aval, le responsable du Backlog s'appuie littéralement sur une matrice des exigences dès le début et qui se développe d'une manière qui garde la traçabilité intacte.

À mesure que le projet progresse :

- La traçabilité n'a pas besoin d'être maintenue comme une activité distincte, parce que chaque fonction du système est construite à partir d'une description générale dont l'implémentation est détaillée et approuvée comme « finie » lorsqu'elle répond aux critères d'acceptation préalablement convenus.
- Si les exigences doivent changer pour supprimer des fonctionnalités, de nouvelles Story sont introduites pour substituer le comportement qui n'est plus nécessaire.

Il est important de vérifier si les artefacts standards du projet Agile peuvent répondre aux besoins de traçabilité pour le produit et à l'organisation du fournisseur. Dans l'affirmative, des matrices de traçabilité supplémentaires ne sont pas nécessaires et n'ajouteraient pas de valeur.

Dans des situations simples, ce qui est le cas de nombreuses équipes Agile, les matrices de traçabilité ne sont pas faites, parce que le coût total pour maintenir ces matrices, même en utilisant des outils spécifiques pour le faire, l'emporte sur les avantages qu'elles apportent. Sensibiliser les parties prenantes sur les coûts réels et les avantages permet à l'équipe prendre les bonnes décisions au sujet de la quantité de documents nécessaires au suivi de traçabilité.

Il y a des cas où le maintien d'une matrice de traçabilité est logique. Cela inclut :

- Des outils automatisés existent. Certains outils de développement donneront automatiquement la traçabilité comme effet secondaire des activités de développement normales. Peut-être pas la traçabilité complète, mais une grande partie du travail de traçabilité peut et devrait être automatisé. Avec l'automatisation le coût total de possession

© GASQ – Global Association for Software Quality

de la traçabilité baisse, augmentant ainsi les chances qu'elle apportera une valeur réelle pour le travail de l'équipe.

- Domaines complexes. La nécessité d'une traçabilité est grande pour faire face à la complexité du domaine.
- Grandes équipes ou équipes géographiquement dispersées. Bien que la taille de l'équipe soit généralement motivée par une plus grande complexité – Complexité du domaine métier, technique ou organisationnelle – cela entraine aussi souvent un besoin accru de traçabilité. Quand une grande équipe est impliquée, il devient difficile même pour les membres les plus expérimentés de l'équipe de comprendre les nuances détaillées de la solution car les connaissances peuvent être réparties entre plusieurs personnes. Les informations de traçabilité donnent la vision nécessaire pour évaluer efficacement l'impact des changements proposés. Notez que la grande taille d'une équipe et la distribution géographique ont tendance à aller ensemble.
- Conformité à la réglementation. La traçabilité peut être exigée par des règlements. Dans ce cas, elle doit être fournie.

5.3 Gestion de configuration et du changement

La gestion de configuration est une pratique clé dans le développement Agile. C'est une exigence fondamentale pour l'intégration continue (voir aussi 1.2.2) et l'automatisation des tests. Concernant spécifiquement l'ingénierie des exigences, la gestion de configuration s'applique aux :

- Exigences comme artefacts de Backlog et artefacts individuels
- Résultats des ateliers
- Résultats de l'analyse des exigences (voir 4.2.3)
- Modèles

La gestion du changement dans un projet Agile n'est pas substantiellement différente de celle d'un processus traditionnel en ce qui concerne les activités de gestion des changements dans les exigences [REQB_FL_SYL].

Les changements des plans de Release peuvent être déclenchés par des facteurs internes ou externes. Les facteurs internes incluent les capacités de livraison, la vélocité et des questions techniques. Les facteurs externes incluent la découverte de nouveaux marchés et débouchés, de nouveaux concurrents ou menaces sur le métier qui peuvent changer les objectifs des Release ou des dates butoirs. En outre, les plans d'itération peuvent changer lors d'une itération. Par exemple, une User Story particulière pourrait s'avérer plus complexe que prévue au cours de son estimation [ISTQB_FA_SYL].

La gestion du changement en Agile comprend les activités suivantes :

- Identifier ou découvrir un besoin de changement
- Le mettre dans le Parking Lot jusqu'à ce qu'il soit analysé
- Estimer tout changement en terme de valeur métier, puis le déplacer comme nouvel élément du Backlog de produit en le reliant avec l'exigence initiale
- Au cours de la prochaine planification de Sprint ou du prochain atelier de Grooming, évaluer le changement (y compris l'évaluation du risque, le coût et l'effort dû au changement) afin de décider si et quand implémenter le changement
- Planifier le changement (si le changement a été approuvé pour l'implémentation)

© GASQ – Global Association for Software Quality

- Implémenter le changement et le suivre comme partie de l'objectif du Sprint

La grande différence dans le processus de gestion du changement en Agile, par rapport au développement traditionnel réside dans le fait que le mécanisme de demande de modification n'est pas obligatoire et que le comité de contrôle des changements n'est pas nécessaire. Un projet Agile est conçu pour s'adapter aux changements par le biais de rôles et d'événements qui ont déjà été définis. Quand une demande de changement arrive au Product Owner, elle est prise en compte en mettant en œuvre l'évaluation habituelle de la valeur et de la priorité de la nouvelle demande en respect de ce qui est déjà dans le Backlog. Dans le cas où la modification affecte une exigence qui est déjà implémentée ou une nouvelle exigence, le positionnement du changement dans le Backlog pourrait impliquer l'infaisabilité d'un autre élément du Backlog à temps pour la Release en exécution. Le changement d'une exigence qui n'est pas encore implémentée peut résulter dans un simple remplacement ou un changement dans les estimations, mais cela pourrait entraîner un impact pour les éléments de Backlog existants.

Pour les organismes de formation : Montrer avec des exemples la différence entre le processus de gestion du changement dans des projets traditionnels (avec demande de changement et comité de contrôle des changements) et la gestion du changement dans un contexte Agile.

5.4 Assurance Qualité

[REQB_FL_SYL] répertorie certains des facteurs qui peuvent influer sur le niveau de la qualité désirée d'un produit. Ces facteurs sont valables pour les projets Agile. Ils comprennent :

- Les critères qualité du client ou des attentes du processus d'Assurance Qualité
- Le type de produit qui est fabriqué (p. ex., le public cible ou de la complexité du produit – un produit destiné à un public restreint pourrait avoir certains attributs Qualité de niveau inférieur à un produit du marché de masse)
- L'environnement dans lequel le produit est développé (par exemple, l'environnement technique pourrait limiter la capacité à atteindre le niveau souhaité des attributs qualité)
- Domaine (par exemple, la complexité du domaine métier, le niveau d'innovation, la fréquence des changements dans le métier)
- Juridiques, de sécurité et facteurs environnementaux (p. ex., les règlements demandant un niveau qualité plus élevé de à atteindre)
- Pressions sur le temps et les coûts qui réduisent la possibilité d'exécuter les processus d'ingénierie des exigences

L'approche contrôle qualité dans un environnement Agile a quelques différences fondamentales avec les environnements de développement traditionnels. L'approche traditionnelle suppose que le produit est réalisé dans des phases successives, qui exigent le transfert des travaux d'une phase à l'autre et que la Release du produit final ne peut avoir lieu qu'à la fin de la séquence de transfert. Lorsque les produits de la chaîne sont modifiés, il y a une rupture potentielle qui doit être gérée avec beaucoup de prudence. De plus, le contrôle qualité de tout ce qui est produit pendant l'ingénierie des exigences doit s'appuyer sur les critères qualité préétablis pour les documents basés sur des modèles et les revues formelles à différents stades d'avancement. Il est nécessaire de s'appuyer sur la qualité de la documentation jusqu'à ce que la qualité du produit final puisse être vérifiée.

© GASQ – Global Association for Software Quality

Dans un projet Agile il n'y a pas de normes précises, de modèles ou de revue de documents à appliquer pour obtenir un certain niveau de qualité attendue. Un projet Agile base sa qualité sur les pratiques fondamentales du développement Agile qui proviennent des 12 principes Agile (1.1.1) :

- Une collaboration continue entre l'équipe de développement (y compris les testeurs) et les parties prenantes, basée sur une conversation directe plutôt que sur l'échange de documents, définissant les conditions de satisfaction des exigences et les critères d'acceptation des livraisons incrémentales pour s'assurer de la testabilité des exigences.
- L'intégration continue, au cours des itérations, des livraisons continues par l'équipe de développement. L'utilisation de tests automatisés et l'utilisation du Développement Piloté par les Tests visent à maintenir chaque incrément livré à un haut niveau de qualité.
- Un feedback continu, à la fin et peut-être aussi pendant l'itération, par le Product Owner et d'autres intervenants, comme principal mécanisme pour vérifier et valider les exigences et la qualité du produit en création.

Dans le contexte Agile, avec ces pratiques en place pour assurer le niveau de qualité requis, la vérification et la validation sont prévues et exécutées depuis le début du projet jusqu'à la fin.

Un évènement collaboratif incontournable qui conduit l'amélioration en Agile est la rétrospective décrite à la section 4.4.2. L'adaptation et l'amélioration continue sont gérées par un suivi approprié des conclusions et les enregistrements des rétrospectives. Ceux-ci permettent à l'équipe de transférer leurs expériences réalisées et les actions d'amélioration qui en résulte aux autres équipes et aux autres parties prenantes.

Le développement Agile conçoit l'utilisation de métriques dans le seul but de maintenir toutes les parties prenantes, en plus de l'équipe de développement, informées des progrès de la Release et de chaque Sprint, à travers des graphique Burndown. Des métriques sont également suivies pour tous les obstacles qui arrivent et les défauts qui sont trouvés et résolus.

Les métriques ne sont jamais un simple instrument de contrôle défini et normalisé par un processus ou un plan qualité. Dans Agile les métriques sont fournies à toutes ou parties des parties prenantes afin d'améliorer la transparence et l'efficacité du travail. Les métriques adoptées peuvent alors être changées selon les besoins, l'état de la Release et leur utilité réelle.

Dans la (section 2.3) les propriétés INVEST ont été introduites comme checkliste clé des attributs qualité habituels des exigences afin d'être utilisées au cours du raffinement des User Story. En parallèle, les questions suivantes devraient également être posées lors de l'évaluation de la qualité des exigences au cours d'un point de contrôle qualité :

- L'exigence est-elle sans ambiguïté ?
- L'exigence est-elle faisable ?
- L'exigence est-elle comprise ?
- Est-ce que l'origine de l'exigence est identifiable ?
- Est-ce que l'exigence est testable ?

© GASQ – Global Association for Software Quality

6 Outils et artefacts en ingénierie des exigences Agile (80 minutes)

Mots clés

Exigences émergentes, parking lot, Planning Poker, technique Pomodoro®, graphique Burndown , timeboxing

Objectifs d'apprentissage pour outils et artefacts en ingénierie des exigences Agile

6.1 Artefacts en support de l'ingénierie des exigences Agile (35 minutes)

AR-6.1.1 Utiliser le Backlog de produit, le graphique Burndown pour fournir une vue générale de la progression, plusieurs mesures et activités relatives à l'ingénierie des exigences (K3)

AR-6.1.2 Expliquer comment gérer les nouveaux besoins avec le Parking Lot (K2)

6.2 Outils de support à l'ingénierie des exigences Agile (45 minutes)

6.2.1-AR Utiliser le Planning Poker pour voter dans un environnement Agile (K3)

6.2.2-AR Expliquer le timeboxing et l'utilisation de minuteurs (K2)

6.2.3-AR Comprendre la finalité des outils de support de la communication et expliquer les facteurs importants à considérer en choisissant un outil pour l'Ingénierie des exigences Agile (K2)

6.1 Artefacts supportant l'ingénierie des exigences Agile

6.1.1 Le Backlog de produit

Comme vu dans les chapitres précédents, la première étape fondamentale en Scrum pour le Product Owner est d'articuler une liste priorisée des fonctionnalités appelées Backlog de produit. Selon [ScrumGuide, 2013] le Backlog de produit est une liste ordonnée de tout ce qui pourrait être nécessaire au produit et qui est la source unique d'exigences pour tout changement à apporter au produit. Le Product Owner est responsable du Backlog de produit, y compris de son contenu, disponibilité et de sa priorisation. Ce classement est basé sur les critères suivants :

- Identifier les valeurs des parties prenantes les plus importantes
- Minimiser les risques en mettant les risques les plus élevés au plus tôt
- Minimiser le gaspillage des fonds et des ressources
- Déterminer la meilleure performance financière
- Rendre le modèle incrémental efficace en délivrant de la valeur par étapes

Le Backlog de produit évolue comme le produit et l'environnement dans lequel il sera utilisé. Le Backlog de produit change constamment afin d'identifier ce que le produit a besoin pour être approprié, compétitif et utile.

© GASQ – Global Association for Software Quality

				New Estimates at Sprint ...				
Priority	Item	Details (wiki URL)	Initial Estimate	1	2	3	4	5
1	As a buyer, I want to place a book in a shopping cart (see UI sketches on wiki page)	...	5	0	0	0		
2	As a buyer, I want to remove a book in a shopping cart	...	2	0	0	0		
3	Improve transaction processing performance (see target performance metrics on wiki)	...	13	13	0	0		
4	Investigate solutions for speeding up credit card validation (see target performance metrics on wiki)	...	20	20	20	0		
5	Upgrade all servers to Apache 2.2.3	...	13	13	13	13		
6	Diagnose and fix the order processing script errors (bugzilla ID 14823)	...	3	3	3	3		
7	As a shopper, I want to create and save a wish list	...	40	40	40	40		
8	As a shopper, I want to to add or delete items on my wish list	...	20	20	20	20		
	...		...	...				
			537	580	570	500		

Figure 3 : Exemple de Backlog de produit [ScrumPrimer 2.0]

Au début de chaque Sprint, le travail restant pour chaque élément dans le Backlog est estimé et écrit dans la colonne liée à ce Sprint. Les éléments terminés (finis) ont zéro en estimation pour tous les Sprints restants, les éléments en cours de progression ont une estimation pour le travail restant. Les éléments qui ne sont pas encore commencés donneront le montant total qui peut être révisé, basé sur des raffinements futurs et les dernière activités d'analyse.

Chaque ligne dans le Backlog peut inclure des informations supplémentaires, par exemple :

- L'Epic / Fonctionnalité / Cas d'utilisation (élément d'exigence) qui a généré la User Story / l'élément de Backlog
- Les liens vers les artefacts, documents, résultats d'ateliers, quels que soient les détails supplémentaires fournis sur l'élément. Un outil très commun, utilisé pour stocker, organiser et échanger des données est une simple page Wiki.
- Valeur métier et la priorité
- Valeur du risque
- Notes
- Statut

Le Backlog de produit constitue un moyen d'avoir un aperçu immédiat de la progression du produit qui est toujours en évolution, en changement et en raffinement. Les informations liées à chaque élément permettent à l'équipe de suivre les changements, les spécifications et les résultats des analyses.

Il n'y a qu'un seul Backlog de produit par Product Owner ; Cela signifie que le Product Owner doit prendre des décisions de priorisation pour l'ensemble des exigences.

Une feuille de calcul est souvent utilisée pour gérer le Backlog de produit. Les équipes peuvent faire leurs propres feuilles de calcul et les personnaliser selon les besoins, télécharger une feuille de calcul sur Internet ou utiliser l'une des fonctionnalités de Backlog de Produit disponibles dans de nombreux logiciels commerciaux.

© GASQ – Global Association for Software Quality

6.1.2 Le graphique Burndown de Release

Un graphique Burndown de Release (graphique d'avancement de la Release), détenu par le Product Owner, montre la quantité de travail restant en début de chaque itération.

L'axe horizontal du graphique montre les Sprints ; l'axe vertical de gauche montre la quantité de travail restant. Dans le Backlog de produit chaque élément de Backlog contient la quantité de travail restante en points de Story. Ces valeurs sont mises à jour en permanence. Le graphique affiche le Backlog au fil du temps à partir de la quantité de travail restant en début de la Release jusqu'à zéro lorsque la Release est terminée.

L'axe vertical de droite montre la valeur ajoutée de chaque Sprint. Cette mesure fournit des indications utiles sur le travail restant et fournit également une bonne indication sur où la valeur métier va être distribuée pendant les Sprints et quelle proportion d'exigences est traitée.

Plusieurs mesures sont incluses dans le graphique, ou peuvent être produites facilement si nécessaire. Par exemple :

- Valeur métier apportée à chaque Sprint (par rapport à l'objectif de Sprint)
- Changement de valeur métier de la Release à chaque Sprint
- Déviation de la valeur du Sprint (en points de Story)

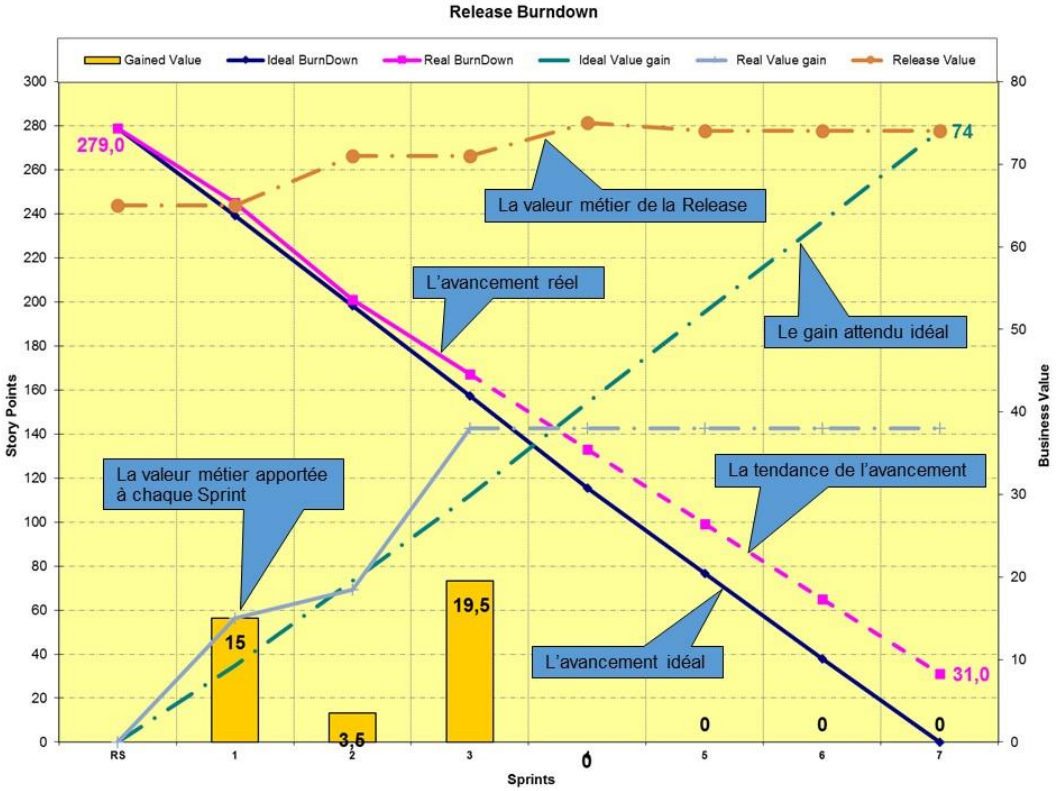


Figure 4 : Exemple de Graphique Burndown de Release

Pour les organismes de formation : Montrer par des exemples pratiques comment utiliser le Backlog de produit, le graphique Burndown de Release pour fournir un aperçu de la progression générale, plusieurs mesures et activités liées à l'ingénierie des exigences.

© GASQ – Global Association for Software Quality

6.1.3 Exigences émergentes : le Parking Lot

Certaines exigences fonctionnelles et non fonctionnelles émergent au cours du développement d'un produit. Cela peut se produire pour plusieurs raisons, par exemple :

- Pendant la revue de Sprint, en regardant ce qui a été fait, le client ou son mandataire a eu connaissance d'un aspect non fonctionnel qui doit être examiné et implémenté
- Lors de la construction d'une User Story, l'équipe réalise qu'il y a une interdépendance avec autre fonctionnalité qui nécessite une fonctionnalité supplémentaire
- Lors de la construction d'une User Story, l'équipe réalise qu'il y a des interdépendances avec une autre fonctionnalité qui implique une nouvelle exigence non-fonctionnelle, comme la facilité d'utilisation

Ces nouvelles exigences, une fois approuvées pour le développement du produit, doivent être gérées comme tout : Les entrer dans le Backlog de produit, les raffiner grâce à l'analyse par l'équipe et finalement les inclure dans un Sprint. L'identification des nouvelles exigences peut être supportée efficacement par le Parking Lot (Parc de stationnement), qui, comme son nom l'indique, est un référentiel temporaire où les exigences émergentes sont « parquées » et attendent d'être examinées, approuvées et incorporées dans le Backlog de produit.

Le Parking Lot permet de suivre les points importants, les idées et les questions qui ont besoin de plus d'investigation ou qui seraient mieux examinées à une date ultérieure.

Un exemple de ceci serait une idée qui émerge au cours d'une mêlée quotidienne qui nécessite d'être discutée avec une autre équipe pour comprendre si cela peut aider, ou une nouvelle idée qui émerge pendant un raffinement de Backlog de produit que le Product Owner veut examiner. L'équipe peut les ajouter au Parking Lot. Le Parking Lot est un outil de facilitation simple que se fait généralement avec une affiche et quelques post it. Il permet à une réunion de se poursuivre efficacement tout en notant des éléments qui nécessiteront un suivi en dehors de la réunion.

À la fin d'une réunion, 10 à 15 minutes devrait être dépensés pour revoir les éléments mis dans le Parking Lot pour identifier :

- Les éléments qui doivent être abordées maintenant
- Les éléments qui doivent être abordés mais ultérieurement (afin qu'ils restent dans le parking lot)
- Les éléments qui ne doivent plus être abordés (Les supprimer du parc de stationnement ou passer à un état de « Fini »)

6.2 Outils prenant en charge l'ingénierie des exigences Agile

6.2.1 Vote : Le Planning Poker

Le Planning poker est l'outil le plus largement utilisé pour voter dans un environnement Agile. Normalement, si un groupe de parties prenantes doit fournir une estimation, habituellement la personne qui a la vision la plus claire de la User Story et du métier sera la première à exprimer une estimation. Malheureusement, cela influencera fortement les estimations des autres. Le Planning Poker est une excellente technique pour éviter ce problème étant donné que tous les membres votants de l'équipe affichent leurs estimations en même temps. Les différences entre les estimations

© GASQ – Global Association for Software Quality

sont alors discutées et un nouveau vote est fait. Une première description et d'autres liens expliquant le Planning Poker se trouvent dans [Planning Poker].

Une technique comme le planning poker est extrêmement utile pour voter les estimations relatives aux exigences Agile telles que :

- La valeur métier et la priorité : Des estimations élevées peuvent placer l'exigence à un niveau élevé dans le Backlog de produit, ce qui signifie qu'il sera abordé dans les Sprints au plus tôt. Des estimations faibles peuvent rétrograder une exigence dans le Backlog de produit qui pourrait entrainer son retrait total.
- Risque : Des estimations élevées de risque positionnent l'exigence comme devant être discutée plus tard, faisant qu'elle sera peut-être raffinée et divisée et puis placée à un endroit plus élevé du Backlog de produit. Des estimations basses (risque faible) peuvent rétrograder l'exigence dans les dernières lignes du Backlog de produit.
- Complexité, taille et effort : des estimations élevées peuvent faire que l'exigence sera par la suite discutée, raffinée et divisée. Des estimations faibles peuvent promouvoir une exigence à une position plus élevée dans le Backlog de produit.

Pour les organismes de formation : En se basant sur un exemple d'un Backlog de produit, fournir un exercice pour une séance de Planning Poker.

6.2.2 Minutage

Le Timeboxing est un point clé du développement Agile. Cette boite de temps est une période préalablement convenue au cours de laquelle une personne ou une équipe travaille régulièrement à la réalisation d'un objectif. Plutôt que de laisser le travail se poursuivre jusqu'à ce que le but soit atteint, et évaluer le temps utilisé, l'approche de limitation du temps consiste en un arrêt du travail lorsque le délai est atteint et une évaluation de ce qui a été accompli.

Dans le développement traditionnel, il est courant de voir la durée se prolonger jusqu'à ce qu'une certaine quantité de travail soit terminée, ayant pour résultat des glissements de calendrier. En Agile, le temps n'est plus une variable ; la règle de limitation de la durée du travail est que les travaux doivent cesser à la fin du temps convenu précédemment ; aucun temps supplémentaire ne peut être donné. Lorsque la limite a été atteinte, la progression est revue : Est-ce que le but a été atteint ou partiellement atteint s'il comprenait plusieurs tâches ? Seul le travail terminé est considéré comme fait. Ce concept est appliqué à pratiquement toutes les activités d'un projet Agile : une réunion, un atelier, une itération. Toutes les pratiques clé d'ingénierie des exigences, telles que les ateliers, évènements d'analyse, Spikes ou revues de Sprint, sont limitées en temps et à la fin du temps, ce qui a été réalisé est évalué et un nouveau calendrier est créé si nécessaire.

Selon les rythmes humains ultradiens, le corps passe entre des niveaux d'énergie bas et hauts sur des cycles de 90 à 120 minutes. Toutes les heures et demi ou à peu près, un humain a besoin de prendre une pause, brève mais régulière, pour garder l'attention et un énergie haute. Au début de chaque plage de temps, toutes distractions, comme le courriel et le téléphone, doivent être stoppées. Un sablier devrait être retourné. À la fin du temps, l'équipe peut décider de retourner immédiatement le sablier s'il faut maintenir l'attention, ou l'équipe peut décider de prendre cinq ou dix minutes pour consulter ses e-mails, passer un appel ou tout simplement regarder à l'extérieur.

© GASQ – Global Association for Software Quality

Avec la technique Pomodoro®, les membres des équipes travaillent par incréments de 30 minutes. Au début de chaque incrément, un minuteur de cuisine en forme de tomate est mis à 25 minutes. L'équipe travaille diligemment au cours de cette période, sans être distraite par le courriel, le téléphone et autre. Lorsque la minuterie s'éteint, les membres de l'équipe font une pause de cinq minutes. Au cours de ces cinq minutes, ils peuvent se promener, s'étirer, partager des Story et autre, mais ne doivent pas faire des choses comme parler de travail ni vérifier d'emails. A chaque quatrième Pomodoro, une pause plus longue de 15 à 30 minutes devrait être prise.

Cette technique ou toute autre technique similaire utilisant des minuteries de cuisine de formes différentes, différents durées entre deux pauses, mais gardant un maximum d'une heure de concentration, est une bonne façon de gérer des événements limités en temps avec un haut niveau d'attention et d'énergie.

6.2.3 Outils de support de la Communication

Une communication efficace entre l'équipe et le Product Owner, entre le Product Owner et les parties prenantes, l'équipe et les parties prenantes et entre les différentes équipes au sein d'un projet ou entre plusieurs projets avec des dépendances, est le pilier de n'importe quel environnement Agile.

Les mécanismes de communication deviennent de plus en plus complexes alors que la complexité du développement augmente et qu'il faut une grande organisation. Ces dernières années, la montée des organisations distribuées géographiquement, des équipes offshores et des activités externalisées, a augmenté la complexité de la communication, suscitant des interrogations sur la faisabilité d'une conversation en face à face. L'organisation d'ateliers réunissant tous les participants dans une seule pièce est souvent impossible et les outils habituels – papiers sur murs, tableaux blancs, post it, photos des murs – qui permettent une documentation naturelle du travail dans un groupe, peuvent devenir insuffisants, lorsque tout le monde n'est pas présent ou lorsque l'analyse devient complexe ou concerne des applications critiques. Le risque de mauvaise interprétation et d'incompréhension augmente lorsque la communication en face à face diminue.

Deux catégories d'outils supportent la communication : Les outils réunissant les personnes impliquées et les faisant communiquer, tels que les systèmes de conférence audio-vidéo, et les outils permettant à des gens concernés de mettre à disposition de leurs collègues les résultats de leurs discussions et de leurs conversations sur les exigences de manière simple, partagée et compréhensible pour tout le monde.

Un certain nombre de techniques peuvent être mises en place et mises à disposition par le biais d'outils conçus pour faciliter la communication dans les situations les plus problématiques en Agile, distribuant les résultats de l'analyse, sans sacrifier la discussion pour la documentation. Certaines des techniques les plus courantes comprennent :

- Les diagrammes d'activités (Diagrammes de flux)
- Les exemples de rapports
- Le pseudo-code
- Les tables de décision et les arbres de décision
- Les machines à états finis
- Les diagrammes de séquence de message
- Les diagrammes entité-relation
- Les cas d'utilisation

© GASQ – Global Association for Software Quality

Il existe plusieurs outils, commerciaux et open source, qui assistent le développement Agile et encouragent la communication par rapport à la documentation, même dans les organisations dispersées géographiquement. Quelles que soient les techniques choisies par l'équipe pour échanger des informations, il est très important de choisir les outils les plus appropriés afin d'éviter d'avoir un outil remplaçant la communication. L'outil ne doit pas devenir un moyen d'écrire un document technique détaillé. La conversation entre les parties prenantes doit rester l'objectif principal, et les techniques fournies par ces outils doivent être un moyen de promouvoir la compréhension et la diffusion de la communication entre les gens.

6.2.4 Autres outils prenant en charge l'ingénierie des exigences Agile

[REQB_FL_SYL] répertorie certaines catégories d'outils qui prennent en charge les activités d'ingénierie des exigences.

Dans le développement de logiciels traditionnels, dans de nombreux cas, ces outils sont utilisés pour prendre en charge la documentation. Ils peuvent aussi être les outils utilisés par quelques spécialistes qui jouent, séparément pour la plus grande part, le rôle d'analyste, d'architecte et de concepteur et établissent les spécifications détaillées des besoins et des solutions à mettre en œuvre. Dans un environnement Agile des outils spécifiques devraient être utilisés uniquement lorsque l'équipe et les autres parties prenantes le juge nécessaire.

Le Backlog de produit est déjà l'outil principal dans Agile pour l'élicitation des exigences, la gestion des exigences et, avec le Backlog de Sprint, de gestion de projet. En fait, les Backlog en Agile peuvent être suivis sur des feuilles de calcul simples, mais des feuilles de calcul peuvent également être structurées et contenir plein d'informations pour gérer la traçabilité des exigences et des ressources. Dans le cas des équipes géographiquement dispersées, les équipes utilisent souvent des plates-formes de gestion permettant une distribution facile, sous contrôle de configuration, des informations telles que le Backlog de produit et les autres artefacts typiques d'un projet Agile [Larman, Vodde, 2010].

Même les outils de modélisation peuvent être utiles dans un environnement Agile, mais leur utilité n'est pas la production automatique d'un document de spécification à partir d'un modèle, mais la création d'un modèle plus clair, compréhensible et distribuable par rapport à des posters sur lequel ce même modèle a été dessiné au cours de la discussion dans un atelier d'analyse.

Dans tous les cas, un outil ne remplacera jamais la conversation directe et les échanges d'informations entre l'équipe et les autres parties prenantes. Le Scrum Master a pour tâche de s'assurer que l'utilisation d'un outil est de faciliter la communication, mais pas de la remplacer.

7 Références

7.1 Tableaux/Figures

Tables

Table 1 Objectifs du syllabus Praticien Agile, ses bénéfices and focus principaux 8
Table 2 brève description des objectifs et des focus de ce syllabus 9

© GASQ – Global Association for Software Quality

Chiffres

Figure 1 : Workflow d'ingénierie des exigences Agile 24
Figure 2 : Niveaux d'Abstraction des exigences mappés avec le workflow d'ingénierie des exigences Agile 25
Figure 3 : Exemple de Backlog de produit [ScrumPrimer 2.0] 71
Figure 4 : Exemple de Graphique Burndown 72

7.2 Normes

Voir les normes énumérées au Syllabus de niveau Fondation REQB

7.3 Documents REQB

[REQB_APPROACH] REQB ® Approach to Requirements Engineering, Version 1.0

[REQB_FL_SYL] REQB ® Syllabus niveau Fondation, Version 2.1

[REQB_GLO] REQB ® Glossaire Standard des termes utilisés dans l'ingénierie des exigences, Version 1.3

7.4 Livres et Publications

[Anderson, 2010] David Anderson, "Kanban: Successful Evolutionary Change For Your Technology Business," Blue Hole Press, 2010.

[Beck, Andres, 2004] Kent Beck and Cynthia Andres, "Extreme Programming Explained: Embrace Change, 2nd Edition", Addison-Wesley The XP Series, 2004

[Cohn, 2004] Mike Cohn, "User Stories Applied: For Agile Software Development", Addison-Wesley Professional, 2004

[Cohn, 2010] Mike Cohn, "Succeeding With Agile – Software Development Using Scrum", AddisonWesley, 2010

[Crispin, Gregory, 2008] Lisa Crispin and Janet Gregory, "Agile Testing: A Practical Guide for Testers and Agile Teams," Addison-Wesley Professional, 2008.

[Derby, Larsen, 2006] Esther Derby and Diana Larsen, "Agile Retrospectives – Making Good Teams Great", The Pragmatic Bookshelf, 2006

[Gottesdiener, 2002] Ellen Gottesdiener, "Requirements by Collaboration: Workshops for Defining Needs", Addison-Wesley, 2002

© GASQ – Global Association for Software Quality

[Humble, Farley, 2010] Jez Humble and David Farley, “Continuous Delivery”, Addison-Wesley, 2010

[ISTQB_FA_SYL] ISTQB Foundation Level Agile Tester Syllabus, Version 1.0

[Larman, Vodde, 2010] Craig Larman and Bas Vodde, “Practices for Scaling Lean and Agile Development - Large, Multisite, and Offshore Product Development with Large-Scale Scrum”, Addison-Wesley, 2010

[Leffingwell, 2011] Dean Leffingwell, “Agile Software Requirements. Lean Requirements Practices for Teams, Programs, and the Enterprise”, Addison-Wesley Agile Software Development Series, 2011

[Linz, 2014] Tilo Linz, “Testing in Scrum: A Guide for Software Quality Assurance in the Agile World,” Rocky Nook, 2014.

[Schwaber, 2001] Ken Schwaber and Mike Beedle, “Agile Software Development with Scrum,” Prentice Hall, 2001.

7.5 Autres références

Les références suivantes pointent vers des informations disponibles sur Internet et ailleurs. Même si ces références ont été vérifiées au moment de la publication de ce syllabus, REQB ne peut être tenu responsable si ces références ne sont plus disponibles.

[Agile Manifesto, 2011] Various contributors, Manifesto for Agile Software Development, 2001, www.agilemanifesto.org

[Cirillo, 2007] Francesco Cirillo, The Pomodoro Technique®, 2007
http://pomodorotechnique.com

[Cockburn, 2007] Alistair Cockburn, A user story is to a use case as a gazelle is to a gazebo, 2007
http://alistair.cockburn.us/A+user+story+is+to+a+use+case+as+a+gazelle+is+to+a+gazebo

[Cohn, 2012] Mike Cohn, Two Examples of Splitting Epics, 2012
http://www.mountaingoatsoftware.com/blog/two-examples-of-splitting-epics

[Jeffries, 2001] Ron Jeffries, Essential XP: Card, Conversation, Confirmation, 2001
http://xprogramming.com/xpmag/expCardConversationConfirmation

[Konrad, Over, 2005] Mike Konrad and James W. Over, Agile CMMI: No Oxymoron, 2005,
http://www.drdobbs.com/agile-cmmi-no-oxymoron/184415287

[Leffingwell, 2010] Dean Leffingwell, The user story primer, 2010
http://scalingsoftwareagilityblog.com/wp-content/uploads/2010/01/user-story-primer_1.pdf
[Planning Poker] Wikipedia, Planning Poker, 2014, http://en.wikipedia.org/wiki/Planning_poker

© GASQ – Global Association for Software Quality

[ScrumGuide, 2013] Ken Schwaber, Jeff Sutherland, The Scrum Guide – The Definitive Guide to Scrum, 2013, https://www.scrum.org/Portals/0/Documents/Scrum%20Guides/2013/ScrumGuide.pdf#zoom=100

[ScrumPrimer 2.0] Pete Deemer, Gabrielle Benefield, Craig Larman, Bas Vodde, The Scrum Primer 2.0, 2012, www.scrumprimer.org

[Wake, 2003] Bill Wake, INVEST in Good Stories, and SMART Tasks, 2003 http://xp123.com/articles/invest-in-good-stories-and-smart-tasks

© GASQ – Global Association for Software Quality

Index

approche équipe intégrée 12
assurance qualité.....9, 44, 64
Assurance Qualité.....34
atelier d'exigences.....38, 41
atelier d'analyse 38, 47, 48, 49, 50, 76
atelier de grooming 38, 39, 53, 58, 67
atelier de planification de Release 38
atelier de planification de Sprint .38, 39, 47, 52
atelier d'exigences.... 39, 40, 47, 52, 58, 61, 62, 63
atténuation des risques.....64, 65
Backlog de portefeuille.....22, 26, 38, 40, 41
Backlog de produit ... 27, 47, 48, 49, 52, 54, 55, 56, 61, 65, 67, 70, 71, 72, 73, 74, 76
Backlog de Produit.....12, 15, 18, 53
Backlog de programme 27
Backlog de Sprint . 15, 18, 28, 38, 52, 53, 60, 76
besoins métier 26, 40, 46
collaboration continue 64, 69
complexité37, 38, 39, 42, 48, 57, 66, 68, 74, 75
Complexité.....61
Concept 3C 38
conditions de satisfaction.....38, 39, 41, 49, 50, 58, 69
contrôle qualité 64
critères 4C + R.....38
critères d'acceptation.....59
criticité.....22, 29, 30, 55
définition de fini 14
demande de changement 68
Demande de changement 64
développement logiciel Agile 58, 60
élément d'exigence 22
élicitation des exigences. 37, 38, 39, 41, 43, 56, 76
engagement.....20, 22, 29
EPIC..... 18, 22, 26, 30, 40, 41, 45, 47, 56, 71
équipe Agile 12, 17, 20
équipe de développement ... 12, 13, 14, 19, 33, 36, 37, 44, 52, 54, 62, 63, 69
équipe interfonctionnelle.....12, 14, 17
estimation de l'effort.....27, 38, 42, 48, 62, 63
exigence Agile.....22
exigences non fonctionnelles 18, 27, 38
expérience utilisateur 59
Extreme Programming..... 12, 14, 15, 77
feedback 8, 12, 14, 15, 19, 31, 36, 47
fonctionnalité 15, 22, 26
gestion de configuration.....9, 28, 34, 64, 67
gestion de projet..... 35
gestion des exigences 18, 23, 63, 64, 76
gestion des risques 64, 65
gestion du changement 64, 67, 68
graphique Burndown 69, 70, 72
identification des risques..... 63
incrément produit..... 12
ingénierie des exigences.....6, 11, 13, 20, 22
ingénierie des exigences Agile ..8, 9, 22, 23, 24, 25, 33
intégration continue 12, 16, 19
INVEST. 9, 22, 30, 37, 39, 45, 49, 50, 53, 56, 58, 69
Kanban 12, 14, 16
livraison continue 12, 19
manager des exigences..... 33
Manifeste Agile 13, 14, 20, 30
métrique 64, 69
modèle 10, 12, 13, 21, 38, 47, 50, 54
modèle incrémental..... 12, 70
modèle itératif 12, 19
modélisation de la solution 54
nouvelle exigence 68
organisation Agile 26, 28, 29, 30, 44, 65
Parking Lot 67, 70
parties prenantes Projet 34, 35
parties prenantes système 33
planification d'itération 18
planification de Release..... 12, 18
planification d'itération 18
Planning Poker 42, 48, 53, 62, 63, 70, 73
principe Agile 13, 69
priorité 15, 19, 22, 27, 28, 29, 30, 36, 38, 39, 43, 47, 50, 55, 74
Priorité 29, 62
Product Owner....12, 14, 15, 18, 19, 28, 30, 33, 35, 36, 38, 41, 42, 44, 47, 52, 53, 54, 59, 60, 61, 69, 70, 71, 73, 75
rétrospective.. 12, 20, 32, 38, 60, 61, 63, 65, 69

© GASQ – Global Association for Software Quality

revue de Sprint 38
risque 15, 18, 19, 22, 29, 30, 37, 47, 50, 51, 55, 63, 64, 70
Risque 61, 74
S.M.A.R.T 42
Scrum.....44, 45, 52, 65, 70
Scrum Master . 9, 12, 14, 15, 20, 33, 34, 38, 47, 53, 54, 60, 61, 65, 76
spécification des exigences ..32, 37, 38, 54, 55, 57, 58, 59
Spécification des exigences 34
Spike 38, 39, 47, 49, 51, 58, 59, 74
Sprint 12, 14, 17, 19, 20, 28, 30, 36, 38, 39, 45, 47, 48, 49, 50, 52, 53, 55, 59, 60, 65, 68, 71, 72
Story d'implémentation.....38, 48
Story d'infrastructure 48
Story d'infrastructure 38
taille relative 38
Technique Pomodoro®..... 70
thème..... 22, 26
timebox..... 12, 70
traçabilité..... 34, 64, 66
User Story 12
valeur métier 15, 22, 27, 28, 29, 38, 39, 41, 42, 47, 48, 49, 50, 54, 55, 61, 62, 67, 72, 74
valeurs Agile..... 13
vision..... 22

© GASQ – Global Association for Software Quality