

TMMi[®] dans le monde DevOps

#SuccessWithTMMi

REDACTEUR EN CHEF : ERIK VAN VEENENDAAL

TRADUCTION FRANÇAISE : ERIC RIOU DU COSQUER, FREDERIC LEVALET,
JEAN-BAPTISTE CROUIGNEAU

TMMi est conçu pour être indépendant du cycle de vie. Ce document explique comment le modèle d'amélioration des tests TMMi peut être utilisé et appliqué de manière bénéfique dans un contexte DevOps. Les domaines de processus TMMi de niveau 2 et 3 et leurs objectifs sont abordés un par un. Il est indiqué comment ceux-ci sont liés à DevOps et à quoi ressemblent généralement les pratiques. Si une pratique n'est pas censée être mise en œuvre dans un contexte DevOps, car elle n'apporte aucune valeur ajoutée, cela est également explicitement indiqué. Il convient de noter que ce document n'est pas destiné à être un programme complet de tests DevOps, mais plutôt un document qui montre « uniquement » comment les objectifs TMMi doivent être interprétés dans un contexte DevOps et qui fournit des idées et des orientations pour des études plus approfondies. TMMi dans le monde DevOps n'est pas non plus destiné à être une description remplaçant le modèle TMMi complet. TMMi dans le monde DevOps doit toujours être utilisé en conjonction avec le document de référence du modèle TMMi.

© TMMi Foundation 2011–25

Tous droits réservés. Aucune partie de cette publication ne peut être prêtée, vendue, transférée, reproduite ou transmise, en tout ou en partie, sous quelque forme ou par quelque moyen que ce soit, sans l'autorisation préalable de la Fondation TMMi, sauf dans les conditions décrites dans la documentation relative à la licence associée. Lorsque toute forme de copie est autorisée en vertu des conditions de la documentation relative à la licence associée, elle est soumise à la condition que la présente mention soit reproduite dans toutes les copies.

Les mots que nous avons des raisons de croire constituer des marques commerciales ont été désignés comme tels. Cependant, ni la présence ni l'absence d'une telle désignation ne doivent être considérées comme affectant le statut juridique d'une marque commerciale.

TMMi est une marque déposée de la Fondation TMMi (Royaume-Uni).

Contributeurs

Katalin Balla (Hongrie)

Suresh Chandra Bose (États-Unis)

Kari Kakkonen (Finlande)

Mattijs Kemmink (Pays-Bas)

Rik Marselis (Pays-Bas)

Szilard Szell (Hongrie)

Erik van Veenendaal (Pays-Bas)

Marques de révision

Cette section résume les principales révisions apportées entre les différentes versions de ce document.
Cette section est fournie à titre informatif uniquement.

Version	Notes de révision
V1.0	Version initiale du document « Testing in the DevOps world » (Les tests dans le monde DevOps)
V1.1	L'interprétation des domaines de processus, des objectifs et des pratiques du niveau 3 du TMMI dans un contexte DevOps a été ajoutée. De plus, un glossaire des termes spécifiques à DevOps a été ajouté.

Table des matières

Contributeurs.....	2
Marques de révision.....	3
Table des matières	4
1 TMMi® dans le monde DevOps	6
1.1 Introduction	6
1.2 DevOps et DevSecOps	6
1.3 Intégration du modèle de maturité des tests (TMMi).....	7
1.4 Les tests dans un contexte DevOps	8
1.4.1 Défis des tests dans un contexte DevOps	8
1.4.2 TMMi et DevOps	9
1.4.3 Relation avec le Syllabus ISTQB DevOps.....	11
2 TMMi niveau 2 géré	12
2.1 Domaine de processus 2.1 Politique et stratégie de test	12
2.1.1 SG1 Établir une politique de test	12
2.1.2 SG2 Établir une stratégie de test	14
2.1.3 SG3 Établir des indicateurs de performance du test	16
2.2 Domaine de processus 2.2 Planification des tests	17
2.2.1 SG1 Effectuer une évaluation des risques produit	17
2.2.2 SG1 Établir une approche de test	18
2.2.3 SG3 Établir des estimations de test	20
2.2.4 SG4 Développer un plan de test	21
2.2.5 SG5 Obtenir un engagement sur le plan de test	22
2.3 Domaine de processus 2.3 Surveillance et contrôle des tests	22
2.3.1 SG1 Surveiller l'avancement des tests par rapport au plan	23
2.3.2 SG2 Surveiller la qualité du produit par rapport au plan et aux attentes.....	24
2.3.3 SG3 Gérer les actions correctives jusqu'à leur terme	26
2.4 Domaine de processus 2.4 Conception et exécution des tests	26
2.4.1 SG1 Analyser et concevoir les tests en utilisant des techniques de conception de tests....	27
2.4.2 SG2 Implémenter les tests	28
2.4.3 SG3 Exécuter les tests	30
2.4.4 SG4 Gérer les incidents de test jusqu'à leur clôture	31
2.5 Domaine de processus 2.5 Environnement de test	32
2.5.1 SG1 Développer les exigences d'environnement de test	32
2.5.2 SG2 Implémenter l'environnement de test	33
2.5.3 SG3 Gérer et contrôler les environnements de test	33
3 TMMi niveau 3 défini	34
3.1 Domaine de processus 3.1 Organisation des tests	34

3.1.1	SG1 Mettre en place une organisation des tests.....	34
3.1.2	SG2 Établir des fonctions de test pour les spécialistes de test	35
3.1.3	SG3 Établir des parcours de carrière en test	35
3.1.4	SG4 Déterminer, planifier et mettre en œuvre des améliorations du processus de test	36
3.1.5	SG5 Déployer le processus de test organisationnel et intégrer des retours d'expérience...	37
3.2	Domaine de processus 3.2 Programme de formation en test	38
3.2.1	SG1 Etablir une capacité organisationnelle de formation en test	38
3.2.2	SG2 Fournir les formations en tests	38
3.3	Domaine de processus 3.3 Cycle de vie et intégration du test.....	39
3.3.1	SG1 Établir les actifs du processus de test organisationnel	39
3.3.2	SG2 Intégrer les modèles de cycle de vie de test aux modèles de développement.....	41
3.3.3	SG3 Établir un plan de test maître	41
3.4	Domaine de processus 3.4 Test non fonctionnel.....	42
3.4.1	SG1 Effectuer une évaluation des risques produit non fonctionnels.....	43
3.4.2	SG2 Établir une approche de test non fonctionnel.....	43
3.4.3	SG3 Effectuer l'analyse et la conception des tests non fonctionnels	44
3.4.4	SG4 Implémenter les tests non fonctionnels	45
3.4.5	SG4 Exécuter les tests non fonctionnels	46
3.5	Domaine de processus 3.5 Revues par les pairs	47
3.5.1	SG1 Établir une approche de revue par les pairs	47
3.5.2	SG2 Effectuer des Revues par les Pairs.....	49
	Glossaire de Termes spécifiques à DevOps.....	51
	Références	53

1 TMMi® dans le monde DevOps

1.1 Introduction

Ce document explique comment la combinaison d'une approche DevOps pour le développement logiciel et du modèle d'amélioration des processus de test TMMi peut vous aider à atteindre vos objectifs métier. Il explique comment TMMi peut être utilisé et appliqué de manière avantageuse dans un contexte DevOps. Il propose des alternatives éprouvées, basées sur DevOps, aux approches de test traditionnelles qui mettent en œuvre les pratiques TMMi. Ce document explique comment TMMi peut aider les organisations DevOps, quel que soit leur niveau de maturité dans leur parcours DevOps, en leur rappelant les pratiques de test essentielles qui sont souvent négligées à mesure que les organisations se développent et que la pression des projets augmente. Chaque domaine de processus TMMi et ses objectifs spécifiques sont abordés un par un. Il est indiqué comment ceux-ci sont liés à DevOps et à quoi ressemblent généralement les pratiques associées. Si une pratique n'est pas censée être mise en œuvre dans un contexte DevOps, car elle n'apporte aucune valeur ajoutée, cela est également explicitement indiqué.

Dans le monde DevOps actuel, où l'intégration continue, la livraison continue et l'automatisation sont les moteurs du développement logiciel, il est primordial de garantir la qualité à chaque étape. Le Test Maturity Model Integration (TMMi) fournit un cadre structuré pour évaluer et améliorer les pratiques de test. Alors que les équipes DevOps s'efforcent de réaliser des livraisons rapides sans compromettre la qualité, l'intégration de TMMi offre une feuille de route pour améliorer la maturité des tests, s'aligner sur les objectifs métier et obtenir des résultats. Cette synergie entre TMMi et DevOps crée une approche équilibrée pour accélérer le développement tout en maintenant des normes élevées de qualité et de fiabilité.

Ce document s'adresse à plusieurs publics cibles, dont les principaux sont les suivants :

- Les organisations traditionnelles matures qui souhaitent passer au DevOps tout en conservant la maturité de leurs processus
- Les organisations moins matures qui souhaitent commencer à appliquer le TMMi pour accroître leur maturité en matière de tests et passer en parallèle au DevOps
- Les organisations agiles qui disposent d'un processus de test agile structuré et souhaitent désormais passer au DevOps
- Les organisations DevOps qui connaissent le succès et sont en pleine croissance. Elles ont donc besoin d'une certaine maturité des processus tout en souhaitant conserver les avantages de l'agilité.

Ce document n'est pas destiné à être utilisé indépendamment du modèle TMMi original. Il vise à fournir des conseils à ceux qui améliorent les processus de test dans un environnement DevOps sur l'interprétation et la signification des différents objectifs et pratiques TMMi. Ce document complète le modèle TMMi original et doit être utilisé comme un document complémentaire.

1.2 DevOps et DevSecOps

DevOps est une philosophie et un ensemble de pratiques visant à combler le fossé entre les équipes de développement logiciel et les équipes d'exploitation informatique. En favorisant une culture de collaboration et de responsabilité partagée, DevOps cherche à rationaliser le développement, le déploiement et la maintenance des logiciels. L'idée centrale est de briser les silos traditionnels entre le développement et l'exploitation, afin de permettre une livraison plus efficace et efficiente des logiciels. Au cœur de DevOps se trouvent l'automatisation et les processus continus. L'automatisation joue un rôle crucial dans la réduction

des tâches manuelles, ce qui permet de réduire les erreurs et d'augmenter la vitesse. Cela inclut l'automatisation de l'intégration des modifications de code, des tests et des processus de déploiement grâce à des pratiques d'intégration continue (CI) et de déploiement continu (CD). L'accent est également mis sur la nécessité d'évoluer vers une culture DevOps caractérisée par l'autonomie, le leadership, la collaboration, l'expérimentation fréquente, des cycles de développement plus courts, une fréquence de déploiement accrue et des versions plus fiables, en étroite adéquation avec les objectifs métier et les clients.

L'approche DevOps présente de nombreux avantages, notamment une mise sur le marché plus rapide, une meilleure collaboration entre les équipes, une qualité et une fiabilité accrues des logiciels, ainsi qu'une évolutivité et une flexibilité accrues. En permettant une détection et une résolution plus rapides des problèmes, DevOps réduit également les temps d'arrêt et améliore la résilience du système. En fin de compte, DevOps concerne autant le changement culturel que la technologie et les processus. Il nécessite un changement de mentalité, dans lequel les équipes de développement et d'exploitation travaillent vers des objectifs communs, acceptent les commentaires et s'efforcent continuellement de s'améliorer. En intégrant ces éléments, DevOps aide les organisations à fournir des logiciels de manière plus efficace, plus fiable et plus sécurisée.

DevSecOps

DevSecOps, abréviation de Development, Security et Operations (développement, sécurité et opérations), est une approche qui intègre les pratiques de sécurité dans le processus DevOps. Cette méthodologie vise à garantir que la sécurité est une responsabilité partagée tout au long du cycle de vie informatique, du développement initial au déploiement et au-delà. DevSecOps vise à fournir des logiciels sécurisés plus rapidement et plus efficacement, en réduisant le risque de failles de sécurité.

1.3 Intégration du modèle de maturité des tests (TMMi)

Le framework TMMi a été développé par la Fondation TMMi comme ligne directrice et cadre de référence pour l'amélioration des processus de test, abordant les questions importantes pour les responsables de test, les ingénieurs de test, les développeurs et les professionnels de la qualité logicielle. Dans le cadre de TMMi, les tests sont définis au sens large afin d'englober toutes les activités liées à la qualité des produits logiciels. TMMi utilise le concept de niveaux de maturité pour l'évaluation et l'amélioration des processus. En outre, les domaines, les objectifs et les pratiques des processus sont identifiés. L'application des critères de maturité de TMMi permet d'améliorer le processus de test et a démontré son impact positif sur la qualité des produits, la productivité de l'ingénierie des tests et l'effort nécessaire au cycle de vie. TMMi a été développé pour aider les organisations à évaluer et à améliorer leurs processus de test.

TMMi dispose d'une architecture par étapes pour l'amélioration des processus. Il comprend des étapes ou des niveaux que l'organisation franchit à mesure que son processus de test évolue, passant d'un processus ad hoc et non géré à un processus géré, défini, mesuré et optimisé. La réalisation de chaque étape garantit que tous les objectifs de cette étape ont été atteints et que les améliorations constituent la base de l'étape suivante. La structure interne de TMMi est riche en pratiques de test qui peuvent être apprises et appliquées de manière systématique afin de soutenir un processus de test de haute qualité qui s'améliore par étapes progressives. TMMi comporte cinq niveaux qui prescrivent la hiérarchie de maturité et le cheminement évolutif vers l'amélioration du processus de test. Chaque niveau comporte un ensemble de domaines de processus qu'une organisation doit mettre en œuvre pour atteindre la maturité à ce niveau. Les domaines de processus pour chaque niveau de maturité de TMMi sont présentés dans la figure 1.

L'un des principes fondamentaux de TMMi est qu'il s'agit d'un modèle générique applicable à divers modèles de cycle de vie et environnements. La plupart des objectifs et des pratiques définis par le TMMi se sont avérés applicables à la fois aux modèles de cycle de vie séquentiels et itératifs, y compris Agile. Cependant, au niveau le plus bas du modèle, bon nombre des sous-pratiques et des exemples fournis sont (très) différents selon le modèle de cycle de vie appliqué. Il convient de noter que dans le TMMi, seuls les objectifs sont obligatoires, les pratiques ne le sont pas. TMMi est disponible gratuitement sur le site web de la Fondation TMMi et également sous forme de livre publié. Le modèle a été traduit dans de nombreuses langues, notamment en espagnol, en français et en chinois.

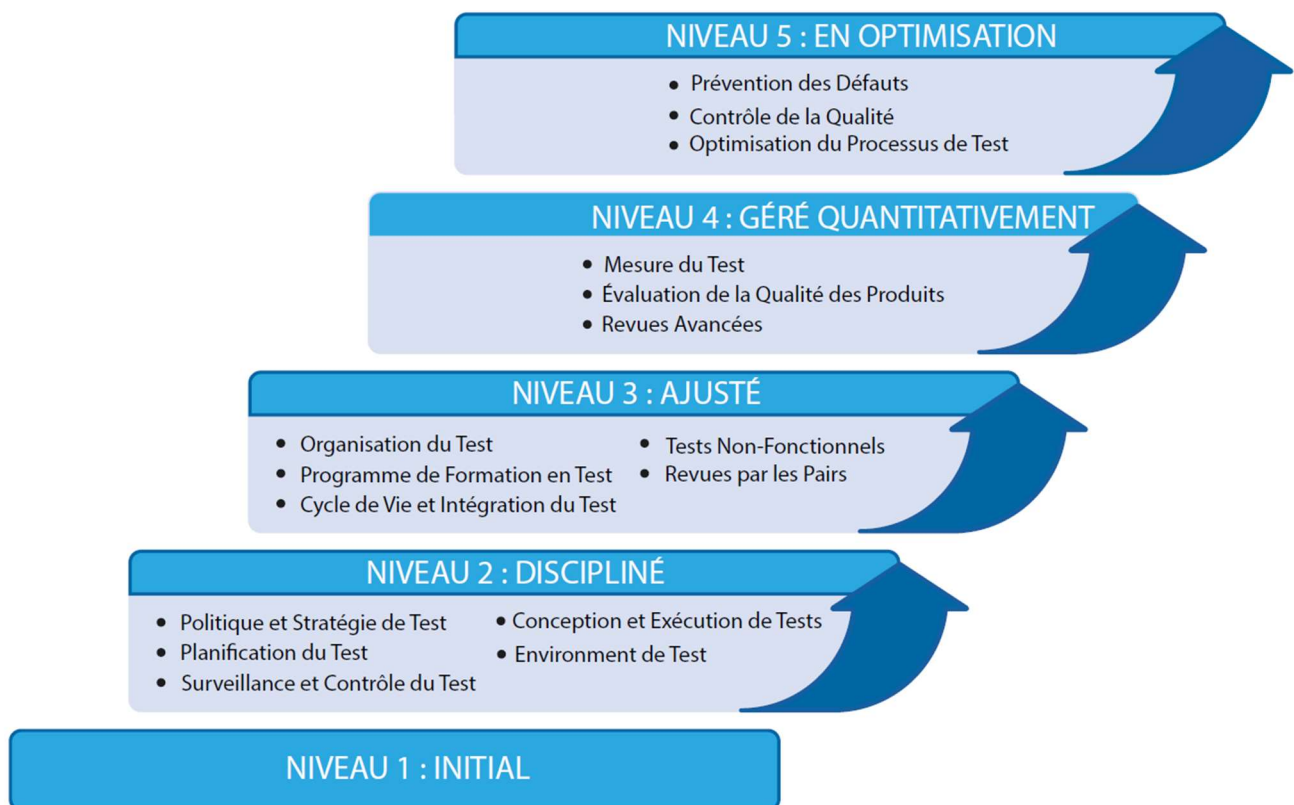


Figure 1 : Niveaux de maturité et domaines de processus TMMi

1.4 Les tests dans un contexte DevOps

1.4.1 Défis des tests dans un contexte DevOps

Les tests dans un contexte DevOps présentent des défis uniques en raison de la nature des pipelines d'intégration continue et de livraison continue (CI/CD), de la collaboration entre les équipes de développement et d'exploitation, et du rythme rapide des cycles de développement. Voici quelques-uns des principaux défis liés aux tests :

- *Tests continus* : les tests doivent être exécutés rapidement et fréquemment pour suivre le rythme des pipelines CI/CD. Cela dépend fortement des tests automatisés, ce qui nécessite des scripts et des frameworks robustes et fiables.
- *Gestion de l'environnement* : il peut être compliqué d'assurer une gestion adéquate de l'environnement de test et de fournir des environnements de test réalistes pour les tests continus. Il existe souvent un manque de parité entre les environnements de production et de test, ce qui peut entraîner des problèmes non détectés. Un autre défi consiste à gérer et à maintenir des configurations cohérentes dans plusieurs

environnements. Les environnements doivent être déployés et configurés automatiquement, généralement à la demande, puis supprimés après les tests.

- *Gestion des données de test* : si les données de test doivent également être similaires à celles de production, cohérentes et déployées à la demande, la conformité aux directives en matière de confidentialité est également indispensable.
- *Tests d'intégration et de régression* : l'intégration de plusieurs composants et services qui présentent des interdépendances complexes rend leur test difficile. Il faut également s'assurer que les nouvelles modifications ne nuisent pas aux fonctionnalités existantes, ce qui nécessite des suites de tests de régression complètes.
- *Outillage et infrastructure* : intégrer divers outils de test aux pipelines CI/CD et autres outils DevOps et s'assurer que l'infrastructure de test peut évoluer pour gérer l'augmentation de la charge et de la complexité.
- *Défis culturels et organisationnels* : favoriser la collaboration entre les équipes de développement, de test et d'exploitation, qui ont traditionnellement fonctionné en silos avec des incitations souvent contradictoires. S'assurer que les membres de l'équipe possèdent les compétences nécessaires pour gérer les tests automatisés, l'intégration continue et les pratiques de déploiement.
- *Tests de performance* : simuler les charges et les conditions réelles dans un environnement de test contrôlé, mais aussi intégrer les tests de performance dans le pipeline CI/CD afin de surveiller et d'optimiser en permanence les performances.
- *Tests de sécurité* : avec des pipelines en évolution rapide, les tests de sécurité sont souvent mis de côté. L'intégration des tests de sécurité (DevSecOps) sans ralentir le développement et en s'assurant que les vulnérabilités sont détectées tôt nécessite des outils et des stratégies spécialisés, ce qui représente un défi pour de nombreuses équipes. Les tests de sécurité doivent de préférence être intégrés tôt et tout au long du cycle de vie du développement à l'aide d'outils automatisés pour effectuer des tests de sécurité statiques et dynamiques.
- *Retour d'information et rapports* : fournir à l'équipe un retour d'information rapide, clair et exploitable sur la base des résultats des tests. Générer des rapports complets qui fournissent des informations sur la qualité de la construction, tout en conservant l'agilité du processus.
- *Pratiques Shift Left* : améliorer la qualité en réalisant les tâches de test dès que possible dans le cycle de vie. Les tests Shift Left consistent à effectuer des tests plus tôt dans le processus, par exemple au moyen de revues (de code), d'analyses statiques et de tests unitaires approfondis.
- *Pratiques Shift Right* : comme il est essentiel de déployer les changements en production aussi fréquemment que possible, une stratégie de test doit trouver un équilibre entre les activités de test dans la partie CI et l'observation de l'utilisation du système en production dans la partie CD.
- *Surveillance et boucles de rétroaction* : dans le DevOps, les tests ne s'arrêtent pas au déploiement ; la surveillance continue et la collecte de rétroactions provenant des environnements de production sont cruciales. La mise en place d'outils de surveillance efficaces et l'interprétation des données provenant de la production pour détecter les défauts et optimiser les performances sont souvent négligées ou sous-développées.

1.4.2 TMMi et DevOps

On croit à tort que les approches TMMi et DevOps sont incompatibles. Les approches DevOps et TMMi peuvent non seulement coexister, mais lorsqu'elles sont intégrées avec succès, elles apportent des avantages

substantiels. Il existe également un défi qui consiste à envisager les tests différemment, à les intégrer pleinement dans le développement et à disposer d'un pipeline CI/CD, ainsi qu'à comprendre ce que cela signifie dans le contexte d'un programme d'amélioration des « tests ». Il convient de noter que le « i » de TMMi fait référence au fait que les tests doivent faire partie intégrante du développement logiciel et ne pas être traités comme quelque chose de totalement distinct. L'utilisation du modèle TMMi permet de rappeler les pratiques de test essentielles, telles que l'analyse des risques liés aux produits, qui sont souvent « oubliées » dans un contexte DevOps. Ce document montre à l'aide d'exemples que les méthodes TMMi et DevOps peuvent fonctionner efficacement ensemble.

Lors de la mise en œuvre de TMMi, il faut tenir compte du fait que l'objectif du modèle TMMi n'est pas d'« imposer » un ensemble de pratiques à une organisation, ni d'être appliqué comme une norme à laquelle il faut « prouver sa conformité ». Utilisé de manière appropriée, le TMMi peut vous aider à identifier les domaines de test spécifiques où le changement peut apporter une valeur ajoutée compte tenu des objectifs métier. Cela est vrai quel que soit le modèle de cycle de vie appliqué. Il est important de toujours garder à l'esprit que les pratiques TMMi sont une composante attendue, mais qu'elles peuvent également être mises en œuvre par ce que l'on appelle des pratiques « alternatives » par rapport à une pratique TMMi définie. Demandez-vous toujours quel est l'objectif de la pratique, quelle est sa justification et comment elle apporte une valeur ajoutée à l'entreprise. Souvent, dans un environnement DevOps, l'objectif est déjà atteint, mais par le biais d'une pratique alternative. En règle générale, « toute » solution est conforme tant qu'elle répond aux besoins de l'entreprise ! Lorsque vous utilisez TMMi, ne soyez pas trop normatif, ce n'est pas ainsi que TMMi a été conçu à l'origine. Interprétez toujours les objectifs et les pratiques TMMi en fonction du contexte de votre situation. En général, en établissant d'abord les besoins en matière de processus dans votre contexte métier spécifique, vous pouvez prendre des décisions sur la manière de cibler et de mener à bien les priorités en matière d'amélioration des processus.

La plupart des conflits qui surgissent entre le TMMi et DevOps sont basés soit sur une vision historique de TMMi de ce à quoi devrait ressembler une « bonne pratique » lorsqu'elle est mise en œuvre, soit sur une mauvaise compréhension de la raison d'être des pratiques DevOps basées sur la manière dont elles devraient soutenir la méthode de travail et les valeurs DevOps. Les experts TMMi, y compris les évaluateurs (principaux), devront reconsidérer et éventuellement repenser les messages qui pourraient être involontairement diffusés concernant ce à quoi devrait ressembler une « bonne pratique conforme au TMMi » lorsqu'elle est mise en œuvre. Lorsque les approches DevOps sont mises en œuvre de manière appropriée avec les processus TMMi, cela se traduit par une mise en œuvre efficace des pratiques de test, et non par leur suppression. Il convient de noter qu'outre les pratiques spécifiques (de test) au sein de TMMi, il existe également des pratiques génériques. L'objectif des pratiques génériques est de soutenir l'institutionnalisation d'un domaine de processus, ce qui signifie en fait de s'assurer que l'organisation dispose d'une infrastructure en place pour soutenir le domaine de processus lorsque de nouvelles personnes arrivent ou que d'autres changements surviennent au sein de l'organisation.

Le passage du développement logiciel traditionnel au DevOps peut également inciter à alléger et rationaliser les processus tels qu'ils sont définis aujourd'hui. De cette manière, les organisations basées sur le TMMi bénéficieront de la pensée agile et lean, toutes deux présentes dans le DevOps. Les gens ont tendance à interpréter le modèle TMMi de manière erronée, créant ainsi des processus et des produits d'activité inutiles et sans valeur ajoutée. En revenant aux sources et aux objectifs d'amélioration et en utilisant le modèle TMMi comme prévu, on favorisera l'alignement des processus à valeur ajoutée réelle avec les besoins et les objectifs réels des processus. La simplification et l'allègement des processus dans un esprit agile/lean permettront d'obtenir des processus qui reflètent ce que les gens font réellement et garantiront que seules les données

utilisées sont collectées. Cet état d'esprit mettra également l'accent sur la simplification maximale, ce qui n'est généralement pas facile, mais qui sera bénéfique pour ceux qui mettent en œuvre le TMMi. Les améliorations au sein de DevOps, guidées par les mesures de performance DORA du délai de changement, de la fréquence de déploiement, du pourcentage d'échec des changements et du temps de récupération après un déploiement échoué [DORA16], [DORA24], se feront généralement par le biais de petites équipes autonomes capables d'agir rapidement, ce qui est un autre avantage que TMMi peut tirer de l'application de DevOps. Bien que TMMi soit complet, les organisations ne peuvent pas appliquer chaque élément TMMi. Pour réussir, les organisations doivent identifier leurs propres pratiques de test clés et les améliorations qui doivent être prioritaires et ciblées.

1.4.3 Relation avec le Syllabus ISTQB DevOps

TMMi et l'ISTQB ont conclu une alliance afin de promouvoir ensemble la profession de testeur de logiciels. De plus, lors de l'élaboration de ce document, le comité technique TMMi et le groupe de travail sur le programme « Quality in DevOps » de l'ISTQB ont collaboré. TMMi est un modèle d'amélioration des tests qui fournit une approche d'amélioration prédéfinie avec des priorités basées sur la structure du modèle TMMi. Ce document vise principalement à fournir une interprétation des objectifs d'amélioration de TMMi pour ceux qui travaillent dans un contexte DevOps. Il ne fournit pas une description complète et détaillée des bonnes pratiques d'ingénierie DevOps. Le syllabus « Quality in DevOps » de l'ISTQB est un document basé sur le contenu. Il vise donc à fournir une description détaillée des bonnes pratiques d'ingénierie DevOps, par exemple les pratiques d'ingénierie de la qualité (y compris les tests). Les deux documents sont donc très complémentaires : le TMMi indique ce qui doit être amélioré (quels processus), tandis que le syllabus DevOps de l'ISTQB décrit comment les choses doivent être faites (meilleures pratiques d'ingénierie).

2 TMMi niveau 2 géré

2.1 Domaine de processus 2.1 Politique et stratégie de test

L'objectif du domaine de processus « Politique et stratégie de test » est de développer et d'établir une politique de test, ainsi qu'une stratégie de test à l'échelle de l'organisation ou du programme dans laquelle les activités de test, par exemple les types et les niveaux de test, sont définis sans ambiguïté. Afin de mesurer les performances des tests, la valeur des activités de test et d'identifier les domaines à améliorer, des indicateurs de performance des tests sont introduits.

2.1.1 SG1 Établir une politique de test

Toute organisation qui se lance dans un projet d'amélioration des tests doit commencer par définir une politique de test. La politique de test définit les objectifs généraux de l'organisation en matière de tests, ses buts et ses visions stratégiques concernant les tests et les professionnels des tests. Il est important que la politique de test soit alignée sur la politique générale de l'organisation en matière d'activité et de qualité. Les améliorations des tests doivent être guidées par des objectifs métier clairs, qui doivent à leur tour être documentés dans la politique de test (d'amélioration). Une politique de test est nécessaire pour parvenir à une vision commune des tests et de leurs objectifs entre toutes les parties prenantes au sein d'une organisation. Cette vision commune est nécessaire pour aligner les activités de test (amélioration des processus) dans toute l'organisation. Il convient de noter que les objectifs des tests ne doivent jamais être des objectifs en soi, mais qu'ils découlent de l'objectif supérieur qui consiste à établir la qualité des logiciels et des produits fonctionnels.

Ce qui précède est également vrai dans une organisation qui a adopté la culture DevOps et pratique le développement logiciel DevOps. En effet, au sein de nombreuses organisations, il y a beaucoup de discussions sur l'évolution du rôle des tests, l'indépendance des tests, l'automatisation des tests et les testeurs professionnels dans le développement logiciel DevOps. Ces points, parmi d'autres, sont généralement des questions qui doivent être abordées lors d'une discussion avec la direction et les autres parties prenantes, puis documentées dans une politique de test. Toute organisation, y compris celles qui pratiquent le DevOps, qui souhaite lancer un projet d'amélioration des tests doit identifier et définir les moteurs et les besoins métier d'une telle initiative. Pourquoi lancer un projet d'amélioration sinon ? En prenant le temps de cerner les véritables besoins métier, on peut fournir un contexte permettant de décider où concentrer les priorités d'amélioration (des tests), par exemple sur quel domaine de processus. Il convient de noter qu'une politique de test est généralement un document concis d'une page, une page web ou un tableau mural au niveau de l'organisation, et non un document au niveau du projet.

L'objectif spécifique de TMMi « Établir une politique de test, y compris ses pratiques spécifiques » s'applique pleinement aux organisations qui appliquent le développement logiciel DevOps. Cependant, dans DevOps, le développement, les tests et les opérations sont généralement couverts par le même processus ou, à tout le moins, étroitement alignés. Ainsi, une politique de test devrait également faire partie intégrante d'une politique DevOps globale, en mettant davantage l'accent sur l'ingénierie de la qualité que sur les seuls tests. Les tests dans DevOps sont organisés en équilibrant les approches Shift Left et Shift Right, en utilisant des pratiques collaboratives et une réflexion axée sur les tests. Dans DevOps, les activités de test doivent soutenir les trois approches (telles que définies dans le DevOps Handbook) : le flux, le retour d'information et l'expérimentation et l'apprentissage continu.

Comme indiqué, les objectifs des tests doivent se concentrer sur un équilibre clair entre Shift Left et Shift Right ; une approche proactive, axée sur la conception et le test d'abord, afin de mettre en place les activités de qualité et de test nécessaires avant le déploiement (au sein et autour du pipeline CI/CD) pour détecter les défauts, et des mesures réactives pour réduire les temps d'arrêt du service en production en cas de défaillance, par exemple un mécanisme de retour arrière automatique pour réduire les temps d'arrêt, un déploiement bleu-vert ou une version canary pour réduire l'impact. Si l'objectif des activités de test est de réduire la probabilité de déployer une défaillance (logiciel défectueux) en production, des stratégies de déploiement sont appliquées pour réduire l'impact de toute défaillance donnée, ce qui se traduit par une meilleure qualité globale. Afin de réduire l'impact, les activités de test du pipeline de déploiement doivent être rationalisées pour améliorer le flux et les délais, tout en maintenant les niveaux de risque globaux à un niveau bas.

Outre les tests tout au long du développement logiciel, le Shift Right (test en production) est également une pratique courante dans le DevOps, ce qui aura également une incidence sur la définition des objectifs de test tels que définis dans la politique, en mettant l'accent sur des réactions rapides aux problèmes en production, sur la base d'informations télémétriques, afin de réduire les temps d'arrêt, d'améliorer la disponibilité et, par conséquent, la qualité. Les pratiques d'ingénierie de fiabilité des sites (SRE) couvrent la surveillance de l'état des systèmes, des services et des applications, ainsi que la création de mécanismes d'automatisation en boucle fermée pour réagir aux problèmes/écarts de fonctionnement [Beyer]. Ce processus est également appelé « auto-réparation ».

En raison du nombre de changements attendus dans DevOps, les objectifs des tests se concentrent fortement sur les tests de régression, tant sur les aspects fonctionnels que non fonctionnels. Certains aspects non fonctionnels peuvent être mesurés en production (Shift Right) après le déploiement, tandis que d'autres doivent être testés avant la mise en production, par exemple la sécurité.

Exemples d'objectifs de test:

- Les tests visent à s'assurer que les changements apportent la valeur attendue (test d'acceptation) et qu'ils ne compromettent aucune fonctionnalité existante (test de régression).
- Les tests doivent soutenir les objectifs de développement de l'organisation, tels que la réduction du taux d'échec des changements ou l'augmentation de la fréquence des déploiements.

De plus, l'indépendance et la définition de processus de haut niveau par la politique de test sont généralement affectées dans le contexte DevOps. En ce qui concerne l'indépendance, le test est un rôle au sein de l'organisation de développement, souvent appelé DevTest pour représenter le double rôle. L'indépendance est donc généralement faible. Les organisations s'appuient généralement sur une équipe dite « plateforme » composée d'ingénieurs en développement logiciel spécialisés dans les tests (SDET), dont le rôle principal est de développer et de maintenir les aspects de testabilité du produit et les solutions d'automatisation des tests, qui seront utilisées par les rôles DevTest. En outre, certaines équipes de test dédiées peuvent être désignées pour des domaines plus complexes, tels que les tests non fonctionnels ou les scénarios complexes de bout en bout. Dans ce cas, l'indépendance est plus grande, mais la coopération reste étroite. En ce qui concerne la définition du processus de test de haut niveau, les tests font désormais partie intégrante du processus de développement, ou de la stratégie CI/CD, tout comme le processus de test.

Il est important de veiller à ce que la politique de test, généralement un document léger/concis, soit diffusée à l'ensemble de l'organisation/du flux de valeur/du train de publication Agile/de la tribu. Les tests étant un élément crucial du travail des équipes Agile et DevOps, et non une organisation distincte, la politique doit être connue de toutes les personnes impliquées et concernées.

2.1.2 SG2 Établir une stratégie de test

La stratégie de test suit la politique de test et sert de point de départ aux activités de test au sein des projets. Une stratégie de test est généralement définie à l'échelle de l'organisation ou du programme. Une stratégie de test type repose sur une évaluation globale des risques liés au produit et comprend une description des types et des niveaux de test à effectuer. Il ne suffit pas de déclarer, par exemple, que des tests d'intégration et d'acceptation seront effectués dans le cadre de chaque projet. Nous devons définir ce que l'on entend par tests unitaires et tests d'acceptation, comment ceux-ci sont généralement réalisés et quels sont leurs principaux objectifs. L'expérience montre que lorsqu'une stratégie de test est définie et suivie, il y a moins de chevauchements entre les différentes activités de test, ce qui rend le processus de test plus efficace. De plus, comme les objectifs et l'approche des différents types et niveaux de test sont alignés, il y a moins de lacunes, ce qui rend le processus de test plus efficace et améliore ainsi la qualité du produit.

Une stratégie de test est un document essentiel dans un environnement DevOps. Elle définit de manière générale les tests à effectuer par les équipes, principalement dans le cadre de la stratégie CI/CD, les niveaux et types de tests à exécuter et, de manière générale, leur approche. Pour chaque niveau et type de test, les objectifs, les responsabilités, les principales tâches de test continu, les critères d'acceptation/les critères de qualité sont définis. La stratégie de test suit généralement les quadrants de test Agile (voir figure 2), mais elle est construite autour du pipeline CI/CD afin de représenter le flux d'activités au sein du pipeline. Le pipeline doit être compris non seulement comme les activités exécutées par un outil CI/CD, mais aussi comme tout ce que font les équipes lorsqu'elles travaillent sur un élément du backlog. Les activités de test telles que les revues et les tests exploratoires se déroulent en dehors des outils CI/CD, mais font partie intégrante du flux/pipeline/flux de valeur. Il convient de noter que le pipeline peut être utilisé non seulement comme technologie/outil, mais aussi comme représentation visuelle des activités, des processus et des personnes [voir figure 3].

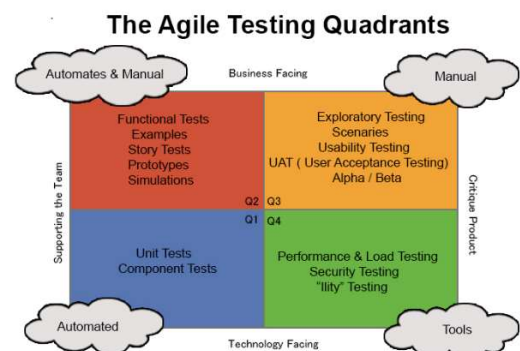


Figure 2 : Les quadrants du test agile

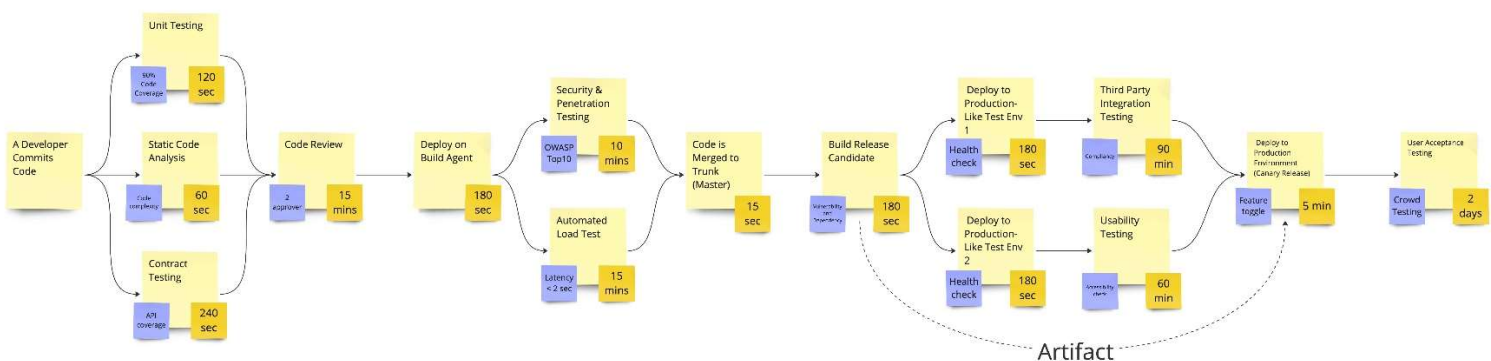


Figure 3 : Une stratégie de test représentée sous forme de pipeline de haut niveau

Une stratégie de test est également un document essentiel lors de l'application d'une méthode de travail DevOps. Le DevOps est relativement nouveau pour de nombreuses organisations. Une stratégie de test commune à l'ensemble de l'organisation guidera donc toutes les personnes impliquées et expliquera où et comment les tests sont effectués dans le pipeline. Elle évitera également à chaque projet/pipeline de réinventer la roue.

La stratégie de test doit bien sûr être parfaitement alignée sur la stratégie de développement, car les tests font partie intégrante du pipeline de développement. Voici quelques considérations (classées par thème) à prendre en compte lors de l'élaboration d'une stratégie de test DevOps :

Niveaux de test

- Les niveaux de test sont principalement représentés par des activités/tâches de test dans le pipeline CI/CD.
- La stratégie de test est souvent élaborée autour des tests d'acceptation des user stories développées, guidés par les pratiques de développement piloté par le comportement (BDD). Cela nécessite des tests continus à chaque étape du pipeline.
- La stratégie de test définit généralement un niveau de test tel que le test de contrat afin d'atténuer les risques liés à l'utilisation d'API ouvertes et de prendre en charge les tests d'intégration.

Tests de bas niveau

- Les pratiques de développement et de test de bas niveau telles que les pull request, les merge et les revues de code doivent être prises en considération.
- Les tests statiques et autres activités de test de bas niveau sont généralement imposés par l'outil CI/CD et ne peuvent souvent même pas être évités (vérification et tests automatiques dans le cadre de la pull request, par exemple).
- Sur la base du paradigme « shift left », l'accent est également mis sur les tests unitaires, l'atteinte d'un niveau convenu de couverture du code et les tests d'intégration.

Automatisation des tests

- Les tests continus sont intégrés de manière transparente dans le pipeline de livraison de logiciels et la chaîne d'outils DevOps.
- Une stratégie d'automatisation des tests est un élément crucial de la stratégie de test DevOps.

Critères d'entrée et de sortie

- Les critères d'entrée sont représentés sous la forme d'une Definition of Ready (DoR) pour les équipes DevOps. La DoR doit imposer une base de test de haute qualité qui sert de base aux tests, par exemple des critères d'acceptation définis en collaboration par l'équipe, généralement sous la forme d'une déclaration « Given-When-Then ». Il est important que les risques et les objectifs des tests soient bien définis et convenus avant d'atteindre l'état **Prêt** (Ready), afin que l'estimation couvre également les besoins en matière de tests.
- Les critères de sortie sont divisés en definition of Done (DoD) pour les équipes DevOps et en contrôle qualité intégré à chaque tâche de test dans le pipeline CI/CD. La DoD peut être définie différemment pour différents niveaux de backlog (epic, feature, story) représentant la réalisation des tests, le niveau de confiance atteint, à différents niveaux/étapes. En plus des tests d'acceptation réussis, la DoD contient généralement les activités de test attendues pour répondre aux exigences non fonctionnelles, qui servent de contrainte à tous les éléments du backlog. Dans les équipes DevOps, l'état « TERMINÉ » n'est atteint que lorsque l'élément du backlog est déjà déployé en production et fonctionne sur la base de la surveillance du système/service.

Types de tests : tests de sécurité

- Identifiez les caractéristiques non fonctionnelles, telles que la sécurité, les performances, la fiabilité ou la facilité d'utilisation, qui sont essentielles à la réussite, ainsi que leur approche de test.
- Compte tenu de l'importance de la sécurité, les tests statiques de sécurité des applications (SAST), les outils d'analyse dynamique de la sécurité (DAST), les tests interactifs de sécurité des applications (IAST), l'analyse de la composition logicielle (SCA), les tests de pénétration et les tests de sécurité des API peuvent être mentionnés comme faisant partie de l'approche des tests de sécurité.

Environnements de test

- Les environnements de test sont présentés pour les différentes étapes de test du pipeline CI/CD (par exemple, développement, staging, préproduction), chacune représentant un objectif différent. Ils sont généralement alignés sur les différents niveaux de test identifiés (par exemple, tests unitaires, tests d'intégration, tests système et tests d'acceptation). Comme DevOps se concentre sur l'automatisation intensive des pratiques de déploiement et de configuration pour les environnements de production, en appliquant des pratiques telles que la configuration en tant que code (Configuration-as-Code), il est évident d'utiliser les mêmes pratiques lors de la conception et de la création d'environnements de test. Avec l'utilisation d'environnements basés sur le cloud, la création d'un environnement de test à la demande et sa suppression après utilisation est une pratique courante, ce qui permet d'obtenir un environnement de test standardisé, contrôlé et stable. Lorsqu'on utilise des stratégies de déploiement telles que le déploiement bleu-vert, l'environnement de test de la dernière version de notre produit peut devenir l'environnement de production après avoir satisfait aux critères de qualité et à la DoD. Dans ce cas, l'environnement de production précédent est réutilisé comme prochain environnement de test (en changeant l'objectif entre les environnements bleu-vert).

Le document de stratégie de test allégée est souvent une bonne solution pour les organisations DevOps et un moyen de s'affranchir des plans de test détaillés (projet). L'objectif spécifique TMMi « Établir une stratégie de test », y compris ses pratiques spécifiques, est pleinement applicable aux organisations qui appliquent le développement logiciel DevOps. Comme pour la politique de test, il est important de veiller à ce que la stratégie de test soit diffusée à l'ensemble de l'organisation / chaîne de valeur / train de release Agile / tribu.

2.1.3 SG3 Établir des indicateurs de performance du test

Les objectifs métier pour l'amélioration des tests, tels que définis dans la politique de test, doivent être traduits en un ensemble d'indicateurs clés de performance des tests. La politique de test et les indicateurs de performance qui l'accompagnent fournissent une orientation claire et un moyen de communiquer les niveaux de performance attendus et atteints. Les indicateurs de performance doivent indiquer la valeur des tests et de l'amélioration des processus de test aux parties prenantes. Étant donné que les investissements dans l'amélioration des processus nécessitent un soutien à long terme de la direction, il est essentiel de mesurer quantitativement les avantages d'un programme d'amélioration afin de maintenir leur motivation. Il convient de noter que cet objectif spécifique de TMMi consiste à définir un nombre limité (par exemple, 2 ou 3) d'indicateurs de performance des tests. Il ne s'agit pas de mettre en place et de mettre en œuvre un programme de mesure complet, mais plutôt de définir un ensemble d'indicateurs clés qui vous indiquent comment la valeur des tests évolue au fil du temps et dans différents environnements de livraison.

Comme dans le cadre de l'Agile, le DevOps mettra également l'accent sur le travail d'équipe et la réflexion systémique. Cela peut se traduire par un élargissement correspondant des indicateurs aux indicateurs de performance des processus de développement et de déploiement, plutôt que de se limiter uniquement aux spécificités des tests eux-mêmes. Les principaux indicateurs de performance DevOps suivent généralement les métriques DevOps Research and Assessment (DORA). Il convient de noter qu'au lieu de la couverture (des tests), c'est le taux d'échec des changements qui est mesuré lors du déploiement en production (pourcentage de déploiements ayant échoué).

Exemples de mesures DORA :

- Indicateurs de performance de la livraison logicielle : fréquence de déploiement, délai de mise en œuvre des changements, temps de restauration du service et taux d'échec des changements
- Performances opérationnelles : indicateurs de fiabilité, par exemple MTTR et MTTF

Les indicateurs de performance doivent faire partie intégrante des activités CI/CD, et les données doivent être disponibles en temps quasi réel pour tous les membres de l'organisation. En plus des métriques DORA, les métriques de flux sont également utilisées pour soutenir la gestion/optimisation de la chaîne de valeur. Les activités de test peuvent être mesurées conformément aux métriques ci-dessus, en veillant à ce que leur contribution soit compréhensible.

Exemples de mesures de flux pour la gestion de la chaîne de valeur liée aux tests :

- Délai d'exécution des tests, taux de réussite ou d'échec des activités de test dans le cadre du CI/CD par modification du code
- Indicateurs de flux, comment le délai d'exécution des tests contribue au délai global

Le défi consiste à définir la combinaison appropriée d'indicateurs liés à l'approche DevOps et à la pensée systémique, tout en donnant une bonne indication des performances réalisées par un programme d'amélioration des tests basé sur TMMi. L'objectif spécifique de TMMi « Établir des indicateurs de performance des tests, y compris ses pratiques spécifiques » est pleinement applicable, même si les indicateurs de performance sélectionnés et appliqués peuvent avoir une portée plus large et ne pas être uniquement liés aux tests. Bien sûr, cela rendra l'analyse et l'interprétation des indicateurs de performance plus difficiles. En fait, les indicateurs de performance utilisés ne peuvent pas être appelés indicateurs de performance des tests. Cela reste acceptable dans le contexte de TMMi tant qu'ils comportent des éléments liés aux tests et qu'ils sont utilisés pour évaluer les progrès réalisés dans l'amélioration des tests.

2.2 Domaine de processus 2.2 Planification des tests

L'objectif de la planification des tests est de définir une approche de test basée sur les risques identifiés et la stratégie de test définie, et d'établir et de maintenir des plans bien fondés pour la réalisation et la gestion des activités de test.

Il convient de noter que la clé d'une planification des tests réussie réside dans la réflexion en amont (« l'activité ») et non dans la définition du plan de test associé (« le document »).

Pour les cycles de vie DevOps, deux types de planification sont généralement utilisés : la planification des versions et la planification des itérations. Le domaine de processus Planification des tests au niveau 2 de TMMi se concentre sur les activités liées aux tests lors de la planification des versions et des itérations. La planification des versions anticipe la sortie d'un produit au début d'un projet. Elle nécessite un backlog de produit défini et peut impliquer de raffiner des user stories plus importantes en une collection de stories plus petites. Cependant, il faut garder à l'esprit que DevOps et Agile fonctionnent généralement avec des délais fixes et un contenu flexible. Il est important de s'assurer que seuls les éléments du backlog qui ont été testés conformément à l'approche de test et à la DoD sont publiés. La planification de la publication fournit la base d'une approche de test et d'un plan de test couvrant toutes les itérations. Les plans de publication sont de haut niveau et se concentrent sur la stratégie ou l'approche globale plutôt que sur des tâches détaillées. Une fois la planification de la publication terminée, la planification de la première itération commence. La planification de l'itération anticipe la fin d'une seule itération et concerne le backlog d'itération.

2.2.1 SG1 Effectuer une évaluation des risques produit

Il est impossible de réaliser des tests exhaustifs, et il faut toujours faire des choix et établir des priorités. Cet objectif TMMi s'applique donc également aux projets DevOps. Pour les projets DevOps, une évaluation des risques liés au produit de haut niveau doit être effectuée sur la base d'un document de vision du produit ou d'un ensemble d'objectifs de haut niveau lors de la planification de la version. Pour chaque itération, une session plus détaillée sur les risques liés au produit doit être organisée sur la base des user stories ou d'autres

exigences pour cette itération dans le cadre de la session de planification de l'itération. Cela est parfois déjà fait lors des sessions de raffinement, où les user stories sont discutées et analysées en termes de risques. Le processus d'évaluation des risques liés au produit dans un projet DevOps aura un format beaucoup plus léger que ceux appliqués dans un projet traditionnel suivant un modèle de cycle de vie séquentiel. Parmi les techniques légères d'évaluation des risques liés au produit à utiliser, on peut citer le risques poker [Veenendaal12] et le ROAM (Resolved, Owned, Accepted, Mitigated) [Baah]. Lors de la planification de la version ou de la planification de l'incrément du programme (PI planning), les représentants métier qui connaissent les fonctionnalités de la version fournissent une vue d'ensemble des fonctionnalités à développer, avec les exigences non fonctionnelles comme contraintes, et toute l'équipe, y compris le ou les testeurs, participe à l'identification et à l'évaluation des risques.

Au cours de la planification de l'itération, l'équipe DevOps identifie et analyse les risques produit sur la base des user stories à mettre en œuvre dans l'itération à venir. Il est préférable que tous les membres de l'équipe Agile et, si possible, certaines autres parties prenantes participent à la session sur les risques produit. Il en résulte une liste hiérarchisée des éléments de risque produit, qui identifie les domaines critiques à tester. Cela permettra ensuite de déterminer l'effort de test approprié à allouer afin de couvrir chaque risque avec suffisamment de tests, et de séquencer ces tests de manière à optimiser l'efficacité et l'efficacité du travail de test à effectuer. Les tâches estimées peuvent être classées par ordre de priorité en fonction du niveau de risque produit qui leur est associé. Les tâches associées à des risques plus élevés doivent être commencées plus tôt et nécessiter un effort de test plus important. Les tâches associées à des risques plus faibles doivent être commencées plus tard et nécessiter un effort de test moindre.

Compte tenu de l'importance de la sécurité dans de nombreuses organisations, la modélisation des menaces peut, en plus de ce qui précède, être utilisée pour identifier les exigences de sécurité, repérer les menaces de sécurité et les vulnérabilités potentielles, quantifier la criticité des menaces et des vulnérabilités, et hiérarchiser les méthodes de remédiation à mettre en œuvre pour les atténuer. Les menaces de sécurité sont hiérarchisées à l'aide de lignes directrices permettant de quantifier la probabilité et l'impact de chaque menace afin d'en déterminer la gravité.

2.2.2 SG1 Établir une approche de test

Une approche de test est définie afin d'atténuer les risques produit identifiés et hiérarchisés. Pour une itération spécifique, les éléments et fonctionnalités à tester sont identifiés lors de la planification de l'itération. Cette activité s'appuie également sur les résultats de la session consacrée aux risques produit. La liste hiérarchisée des éléments à tester est généralement liée aux user stories à tester dans cette itération. Les fonctionnalités sont généralement liées, entre autres, aux différentes caractéristiques de qualité du logiciel à tester. De nouveaux risques produit peuvent apparaître au cours de l'itération, nécessitant des tests supplémentaires. Les questions telles que les nouveaux risques produit nécessitant des tests supplémentaires sont généralement discutées lors des réunions quotidiennes (daily standup meetings).

Avec DevOps, l'approche de test est définie sur la base de tests continus via le pipeline CI/CD. Le test continu consiste à tester tôt et souvent à l'aide de l'automatisation, avec un retour d'information exploitable adapté à chaque étape du pipeline de livraison. Les outils d'intégration continue (CI) et de déploiement continu (CD) sont généralement invoqués automatiquement lors de la validation du code et orchestrent les activités de développement, d'intégration, de test, de sécurité, de conformité et de déploiement. Les activités de test au sein du pipeline CI/CD s'appuient généralement sur les niveaux de test identifiés à l'aide des quadrants de test agile, et sont continuellement optimisées par les équipes.

L'approche de test définie au niveau de l'itération ou au niveau des éléments du backlog pour atténuer les risques peut couvrir, par exemple, la revue supplémentaire des user stories et des critères d'acceptation, l'effort de test proportionnel au niveau de risque, la sélection de techniques de test appropriées en fonction

du niveau et du type de risque. Elle peut également suggérer des activités de test qui ne sont effectuées qu'après le déploiement, par exemple des tests participatifs en production pour recueillir des commentaires sur l'expérience utilisateur, ou des tests de performance avec du trafic utilisateur réel à l'aide d'un lancement restreint. Dans le contexte du DevOps, le développement piloté par le comportement (BDD) et le développement piloté par les tests d'acceptation (ATDD) sont des techniques populaires pour identifier et concevoir des cas de test. L'approche de test au niveau de la version sera d'un niveau beaucoup plus élevé et devra être basée sur une stratégie de test définie au niveau du programme ou de l'organisation. Souvent, une approche de test est conservée ou affichée sur le wiki de l'équipe/du projet.

Un risque important qui est toujours présent dans le développement itératif est le risque de régression. L'approche de test doit définir comment le risque de régression est géré. En général, cela se fait en créant un ensemble de tests de régression spécifique, de préférence automatisé. Dans ce contexte, la pyramide d'automatisation des tests est utile [Cohn]. Elle montre comment maximiser la valeur de l'automatisation des tests de régression, en commençant par les tests unitaires au niveau le plus bas de la pyramide et en passant ensuite aux tests au niveau des services. Les tests d'interface utilisateur se situent tout en haut. Les tests unitaires sont rapides et fiables. La couche de service permet de tester la logique métier au niveau de l'API ou du service, où vous n'êtes pas gêné par l'interface utilisateur (UI). Plus le niveau est élevé, plus les tests deviennent lents et fragiles. L'automatisation étant un élément important du DevOps, l'automatisation des tests est au centre de l'approche de test DevOps et ne sert pas uniquement à soutenir les tests de régression. Une autre façon de gérer le risque de régression consiste à s'appuyer sur l'observabilité du système, à surveiller en permanence son état et son utilisation, et à mettre en place des mécanismes automatiques pour éviter les risques, par exemple en revenant à une version précédente stable.

Les critères d'entrée (également appelés « Definition of Ready »), qui font normalement partie d'une approche de test définie (pratique spécifique 2.3), sont susceptibles d'être traités différemment dans le développement DevOps. Dans le développement de logiciels DevOps, les tests font partie intégrante du processus d'équipe et constituent une activité continue. La checklist des critères d'entrée utilisée auparavant est divisée entre la définition de l'état du prêt, par exemple en définissant les critères d'acceptation comme des scénarios de test à l'aide d'instructions « Given-When-Then », et les vérifications au sein de la machine CI/CD, par exemple les vérifications dans une pull request, l'exécution d'un smoke test sur un environnement de test fraîchement déployé avant de lancer une activité d'exécution de test plus importante, ou la vérification que les données de test nécessaires ont également été déployées. Par conséquent, une checklist ou une passerelle spécifique permettant de déterminer si les tests peuvent ou non être lancés doit être mise en œuvre dans le workflow CI/CD, et le build doit échouer en cas de non-conformité, comme pour tout autre défaut. Cela s'applique également à un composant passant d'une étape de test (par exemple, les tests unitaires) à une autre (par exemple, les tests d'acceptation) au sein d'une équipe DevOps.

Les critères de sortie des tests (pratique spécifique 2.4) font partie de ce qu'on appelle la Definition of Done (DoD), qui repose largement sur les contrôles qualité effectués dans le pipeline CI/CD. Il est important que la DoD comporte des critères spécifiques liés aux tests, par exemple pour la couverture des tests et la qualité du produit. L'itération doit aboutir à la mise en œuvre de l'ensemble convenu de user stories et répondre aux critères de sortie (de test) tels que définis dans la DoD. Dans DevOps, « TERMINÉ » signifie généralement que les modifications de code liées aux éléments du backlog sont déployées en production, après avoir passé tous les contrôles qualité au sein du pipeline CI/CD. Dans le cas où la publication se fait à la demande et est dissociée du déploiement à l'aide d'une technologie telle que Feature Toggles, il existe également une DoD au niveau de la publication qui peut s'étendre sur plusieurs itérations. Cependant, l'objectif de DevOps est de publier souvent, par petits morceaux si possible. La DoD au niveau de la version comportera généralement des critères liés à la couverture et à la qualité du produit, axés sur la validation de la valeur attendue pour les utilisateurs, sur la base d'activités de test en production, telles que le crowd testing ou le canary testing.

La pratique spécifique 2.5 Définir les critères de suspension et de reprise n'est probablement pas pertinente pour les cycles de vie DevOps. Les tests faisant partie intégrante du processus de développement logiciel DevOps, ils ne seront bien sûr pas traités comme une activité distincte et indépendante des autres activités d'itération. Lorsque des problèmes bloquants pouvant être considérés comme des menaces potentielles ou réelles pour la progression des tests se présentent, ils sont discutés lors de la réunion quotidienne. Au cours de cette réunion, l'équipe décidera des mesures à prendre, le cas échéant, pour résoudre les problèmes. Ainsi, il ne sera pas nécessaire de définir des critères formels de suspension et de reprise, les problèmes liés à cette question étant traités dans le cadre de la routine DevOps normale. La routine DevOps sert donc de pratique alternative à cette pratique spécifique.

2.2.3 SG3 Établir des estimations de test

Pour les équipes DevOps, des estimations détaillées pour les tests seront faites pendant la planification des itérations. Des estimations de test de haut niveau sont faites pendant la planification des versions et éventuellement aussi lors des sessions de raffinement du backlog. Toutes les estimations sont bien sûr faites en équipe et incluent tous les efforts nécessaires pour livrer chaque story. Il est important de s'assurer que les activités de test sont bien prises en compte pendant les sessions d'estimation. Pour ce faire, les activités de test peuvent être identifiées comme des tâches distinctes, puis chaque tâche de test individuelle peut être estimée, ou chaque user story peut être estimée, ce qui permet de rendre explicites les tests à effectuer et donc de les prendre en compte. Comme dans DevOps presque tout doit être automatisé, l'automatisation des activités liées aux tests doit également être estimée, y compris l'automatisation des tests, l'automatisation du déploiement de l'environnement de test, l'automatisation du déploiement des données de test, ainsi que les améliorations du pipeline conformément à l'approche de test identifiée pour l'itération. Lors de l'estimation, un testeur, faisant partie de l'équipe DevOps, doit participer aux sessions d'estimation. Le Planning Poker, le T-shirt sizing, le swim lane sizing (bucket sizing) et le dot voting sont toutes des techniques d'estimation couramment utilisées dans le développement de logiciels DevOps.

Le travail à effectuer pour la prochaine itération est généralement défini par les user stories. Les user stories doivent être de petite taille pour pouvoir être estimés. L'estimabilité est l'un des critères des user stories définis par INVEST et s'applique aux user stories qui doivent faire partie d'une itération. Lors de la planification de l'itération, les équipes Agile identifient et définissent généralement les tâches liées aux tests. Les tâches de test seront consignées sur un tableau des tâches avec les autres tâches de développement. Lors de la rétrospective, les équipes doivent revenir sur la précision de leurs estimations (si cela a été identifié comme un problème) et l'améliorer en s'appuyant sur les enseignements tirés de la taille des tâches individuelles réalisées au cours de l'itération.

Lors de la planification de la version, les user stories ou les epics sont généralement définies à un niveau plus élevé et ne sont pas encore détaillées en tâches spécifiques. Cela rendra bien sûr les estimations plus difficiles et moins précises. Comme indiqué, les projets DevOps n'établiront pas de structure de répartition du travail comme base pour les estimations, mais peuvent, au niveau de la version, bénéficier d'informations sur ce qu'il faut développer et comment développer l'élément du backlog, ce qui est suffisant pour atteindre la précision d'estimation nécessaire, c'est-à-dire suffisante pour permettre la hiérarchisation des éléments du backlog. Comme dans DevOps l'objectif est d'optimiser le flux de valeur, l'estimation doit soutenir la décision de ce qu'il faut développer ensuite, et non pas créer un calendrier de projet complet.

Bien qu'une équipe DevOps effectue ses estimations de manière relativement informelle, la justification des estimations doit être claire (c'est-à-dire quels facteurs sont pris en compte). Les discussions basées sur la justification favorisent un niveau plus élevé de précision des estimations. En général, dans les projets DevOps, l'estimation se concentre sur la taille (en utilisant des story points) ou l'effort (en utilisant des jours-homme idéaux comme unité d'estimation). Les coûts ne sont généralement pas abordés dans le cadre des sessions

d'estimation des projets DevOps. Il convient de noter que les coûts d'infrastructure sont généralement élevés dans le contexte DevOps, et que leurs estimations sont généralement établies lors de sessions d'estimation spécifiques, et non lors des sessions d'estimation de l'équipe. Une surveillance étroite et un contrôle des coûts d'infrastructure, par exemple l'utilisation du cloud public, sont nécessaires et peuvent être intégrés dans le pipeline CI/CD, avec des rapports automatiques. (Par exemple, chaque tâche GitHub Action exécutée est présentée en minutes vCPU utilisées).

Cet objectif spécifique de TMMi et ses pratiques spécifiques sont donc pleinement applicables, à l'exception de la pratique spécifique 3.2 Définir le cycle de vie des tests. L'un des principes de base du développement logiciel itératif et DevOps est de travailler par petits morceaux. Par conséquent, les tâches qui ont été identifiées sont généralement suffisamment détaillées pour servir de base à l'estimation (des tests). Dans le contexte DevOps, il est donc nécessaire de définir également un cycle de vie pour les activités de test afin de servir de base supplémentaire à l'estimation.

2.2.4 SG4 Développer un plan de test

Rappelons que la planification des tests consiste à réfléchir en amont (« l'activité ») et non à définir le plan de test associé (« le document »). Dans le DevOps, la plupart des activités de planification des tests, telles que définies par cet objectif spécifique de TMMi, seront réalisées lors de la planification des versions et des itérations, ainsi que lors des sessions de raffinement des éléments du backlog. Cependant, le résultat de ces activités ne sera généralement pas documenté dans un plan de test, en particulier pour la planification des itérations où elles pourraient être reflétées sur le tableau des tâches. Les tests étant effectués dans le cadre de la planification des versions et des itérations, les « calendriers » qui en résultent incluront également les activités de test. Plutôt qu'un calendrier détaillé tel que celui élaboré avec des cycles de vie séquentiels, le calendrier d'un projet DevOps ressemble davantage à un classement des user stories (éléments du backlog) et des tâches qui reflète les priorités de la version et de l'itération, par exemple en fonction de la valeur métier souhaitée. Gardez à l'esprit que l'exécution des tests se déroule principalement dans le pipeline CI/CD, déclenchée par la validation du code. Des tests manuels supplémentaires ou des activités réalisées en production peuvent être planifiés sur le tableau d'itération. Les cartes mentales sont souvent utilisées ici comme technique d'aide. Le tableau des tâches reflétera les priorités d'itération. Ainsi, aucun calendrier (de test) explicite n'est établi ; il est prévu que des priorités claires de publication et d'itération soient définies pour les user stories, respectivement les tâches à effectuer, y compris les tâches de test.

Au niveau du projet, le recrutement du personnel chargé des tests fait partie de la constitution de l'équipe ; le besoin en ressources de test ou en ressources polyvalentes est identifié dès le départ. À mesure que les projets évoluent ou se développent, on peut facilement oublier les raisons qui ont motivé la sélection initiale d'une personne pour une équipe ou un projet donné, qui comprend souvent des besoins spécifiques en matière de compétences ou d'expérience liés à l'équipe ou au projet. Cela justifie de noter les compétences requises des personnes, en fournissant des informations complémentaires sur les raisons pour lesquelles elles ont été sélectionnées pour l'équipe ou le projet. Une fois les équipes DevOps définies, la dotation en personnel de test est plus ou moins fixée. Au cours de la planification des itérations, l'identification des ressources et des compétences (supplémentaires), par exemple pour les tests non fonctionnels, nécessaires à la réalisation des tests dans une itération peut, si nécessaire, être discutée afin de s'assurer que l'équipe dispose de ressources, de connaissances et de compétences suffisantes pour effectuer les tests requis.

Une session initiale sur les risques du projet doit faire partie de la planification de la version et de l'itération. L'identification (et la gestion) des autres risques du projet pendant l'itération fait

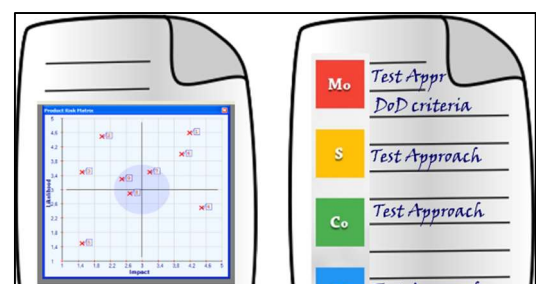


Figure 4 : Plan de test d'une page

partie des réunions quotidiennes et des réunions régulières de scrum-of-scrums, et est généralement documentée au moyen d'un registre des obstacles. Il est important que les problèmes de test soient également consignés dans le registre des obstacles. Les obstacles doivent être discutés lors de la réunion quotidienne jusqu'à ce qu'ils soient résolus, ou lors des réunions scrum-of-scrums si l'équipe n'a pas le niveau ou l'autorité nécessaires pour les résoudre.

Bien qu'aucun plan de test spécifique et détaillé ne soit généralement élaboré et documenté, la pratique spécifique 4.5 Établir le plan de test reste pertinente dans un contexte DevOps. Le résultat des discussions qui ont lieu dans le cadre de la planification des tests est toutefois susceptible d'être consigné sous une forme légère, éventuellement sous la forme d'une carte mentale ou d'un document d'une page (voir figure 4), d'un quadrant de tests agiles au niveau de l'équipe et des activités de test dans le pipeline CI/CD.

2.2.5 SG5 Obtenir un engagement sur le plan de test

Dans le cadre du DevOps, le processus d'élaboration et de mise en place d'une approche et d'un plan de test est un exercice collectif, éventuellement dirigé par un professionnel des tests (qui fait partie des membres de l'équipe). La qualité du produit relève de la responsabilité de l'équipe. Ainsi, à condition que l'équipe suive le processus correct et que les activités de test convenues soient intégrées dans le pipeline CI/CD, l'engagement envers l'approche (de test) et le plan (de test) est déjà un résultat implicite de la planification des versions et des itérations, car il s'agit d'un effort d'équipe. Cela représente bien sûr une différence considérable par rapport à la manière de travailler dans un environnement traditionnel, où le testeur prépare généralement le plan de test et doit ensuite obtenir un engagement explicite. Les ressources d'infrastructure nécessaires au pipeline CI/CD pour mettre en œuvre le plan de test sous la forme d'environnements de test déployés et de tests automatiques exécutés doivent être prises en considération et fournies par les parties prenantes.

Dans un contexte DevOps, l'équipe (y compris le product owner) doit comprendre et approuver la liste hiérarchisée des risques produit et les mesures d'atténuation des risques à mettre en œuvre. La compréhension et l'engagement peuvent par exemple être obtenus au moyen d'une brève présentation suivie d'une discussion lors de la planification de la version ou de l'itération, expliquant les risques produit, l'approche de test et sa justification à l'équipe.

Au cours de la session d'estimation (voir SG 3 Établir des estimations de test), la charge de travail est estimée. Les ressources (de test) d'une équipe sont fixées par la configuration de l'équipe DevOps. Les user stories qui sont estimées et sélectionnées pour être développées prennent les ressources disponibles comme point de départ (contrainte). Ainsi, là encore, le rapprochement entre les niveaux de travail et de ressources n'est pas une activité significative, mais il est essentiel de se concentrer en permanence sur les ressources techniques, telles que la disponibilité du cloud computing/stockage/réseau, qui doivent être fournies aux équipes. La pratique spécifique 3.2 Rapprocher les niveaux de travail et de ressources est donc une pratique qui n'est généralement pas pertinente dans un contexte DevOps. Les réunions quotidiennes et les réunions Scrum-of-Scrum seront utilisées pour traiter tout problème de ressources pendant l'itération et pour (ré)attribuer immédiatement les ressources appropriées ou supprimer un livrable d'une itération afin de le reporter dans une itération future ou dans la planification des versions.

2.3 Domaine de processus 2.3 Surveillance et contrôle des tests

L'objectif de la surveillance et du contrôle des tests est de fournir une compréhension de l'avancement des tests et de la qualité du produit afin que des mesures correctives appropriées puissent être prises lorsque l'avancement des tests s'écarte considérablement du plan et que la qualité du produit s'écarte considérablement des attentes.

Certaines choses sont fondamentalement différentes dans le contexte DevOps par rapport aux projets traditionnels en matière de surveillance et de contrôle. Bien que la surveillance et le contrôle soient des éléments essentiels dans le cadre DevOps, cela ne signifie pas pour autant que l'objectif soit de s'en tenir à un plan rigide. Du point de vue de la surveillance et du contrôle des tests, cela signifie que nous ne sommes pas guidés par un plan, mais que nous examinons en permanence l'avancement et les résultats des tests, et que nous adaptons notre plan et notre approche en conséquence, par exemple lorsque de nouveaux risques produit apparaissent.

Le plan de test, qui est généralement réparti entre des critères d'acceptation affinés, des éléments de backlog (y compris leur niveau de risque) et des tâches, est une entité vivante qui doit être continuellement révisée et mise à jour à mesure que de nouvelles informations sont disponibles ou que des commentaires sont fournis. Les objectifs des activités de test sont définis par l'ensemble des critères d'acceptation pour l'élément du backlog donné, l'approche de test de l'équipe étant mise à jour pour l'élément du backlog donné, et au niveau du backlog, des critères spécifiques de définition du terminé sont définis. Les projets DevOps doivent également garder à l'esprit la « vue d'ensemble » et assurer le suivi et le contrôle au niveau CI/CD ainsi qu'au niveau de l'itération.

Il est important de noter que, comme les tests sont un processus entièrement intégré au processus global de l'équipe DevOps, la surveillance et le contrôle des tests font également partie intégrante des mécanismes globaux de surveillance et de contrôle de l'équipe DevOps. Par conséquent, les testeurs ne rendent pas compte à un responsable des tests comme dans les projets traditionnels, mais à l'équipe.

2.3.1 SG1 Surveiller l'avancement des tests par rapport au plan

Les équipes DevOps utilisent diverses méthodes et outils pour surveiller et enregistrer la progression des tests, par exemple la progression des tâches et des stories sur le tableau des tâches Agile et les burndown charts. Ces informations peuvent ensuite être communiquées au reste de l'équipe à l'aide de supports tels que des tableaux de bord wiki ou envoyés par courriel, ainsi que verbalement lors des réunions quotidiennes. Les équipes peuvent utiliser des burndown charts pour suivre la progression de l'ensemble de la version et de chaque itération. Un burndown chart montre généralement la progression par rapport à la vélocité attendue et au backlog des fonctionnalités à implémenter (il montre donc les performances de l'équipe). De plus, une surveillance continue à l'aide de tableaux de bord en ligne sera utilisée dans DevOps pour surveiller l'infrastructure, l'application et le réseau de manière automatisée tout au long du pipeline CI/CD. Les ressources de l'environnement de test sont surveillées en permanence par rapport à celles convenues lors de la planification de l'itération et pour l'utilisation du pipeline CI/CD. Une autre pratique courante consiste à suivre les problèmes liés à l'environnement de test via le tableau des tâches, par exemple en utilisant des autocollants marqués « environnement de test bloqué » sur les cartes de story ou en créant une colonne séparée sur le tableau où toutes les stories bloquées par les environnements attendent d'être débloquées. Il s'agit de rendre visible l'impact d'un environnement de test qui bloque la progression et d'attirer l'attention de l'équipe sur la résolution de ce problème avant de travailler sur toute autre tâche.

Pour fournir une représentation visuelle instantanée et détaillée de l'état actuel de l'équipe et du CI/CD, y compris l'état des tests, les équipes utilisent généralement des tableaux de tâches. Les cartes des stories, les tâches de développement, les tâches de test et les autres tâches créées lors de la planification des itérations sont enregistrées sur le tableau de tâches. Pendant l'itération, la progression est gérée via le déplacement de ces tâches sur le tableau de tâches vers des colonnes telles que « à faire », « en cours » et « terminé ». Afin de préserver la fluidité, des Work in Process (WIP) sont souvent appliquées à différents états sur un tableau Kanban, contrôlant ainsi le comportement de l'équipe afin qu'elle termine les tâches (en cours) plutôt que de commencer à travailler sur de nouvelles tâches. Les équipes DevOps utilisent généralement des outils pour

gérer leurs cartes des stories sur des tableaux de tâches, ce qui permet d'automatiser les tableaux de bord et la vue d'ensemble de l'état d'avancement. Cependant, certaines équipes ne créent pas de tâches spécifiques pour les activités individuelles, mais peuvent simplement utiliser la carte de story et y ajouter des commentaires pour les tests, en référant des outils ou des wikis où les tests peuvent être documentés. Les tâches de test sur le tableau des tâches sont généralement liées aux critères d'acceptation définis pour les user stories et aux exigences non fonctionnelles qui constituent une contrainte pour chaque élément du backlog. Lorsque les scripts d'automatisation des tests et les tests exploratoires pour une tâche de test obtiennent le statut « réussi », la tâche passe dans la colonne « terminé » du tableau des tâches.

La Definition of Done (DoD) sert de critère de sortie permettant de mesurer les progrès. La DoD doit également être liée aux activités de test et indiquer tous les critères qui doivent être satisfaits avant que le développement et le test d'une user story puissent être considérés comme « terminés ». Il convient de noter que les critères de test liés aux tâches de test ne constituent qu'une partie de ce que l'équipe s'engage à accomplir dans le cadre de la Definition of Done. Les critères de Definition of Done sont généralement appliqués à plusieurs niveaux, par exemple au niveau de la story et au niveau épique. L'ensemble de l'équipe examine régulièrement l'état du tableau des tâches, souvent lors des réunions quotidiennes, afin de s'assurer que les tâches avancent à un rythme acceptable. Si certaines tâches (y compris les tâches de test) n'avancent pas ou avancent trop lentement, cela déclenche une discussion au sein de l'équipe afin d'analyser les problèmes qui pourraient bloquer la progression de ces tâches.

La réunion quotidienne implique tous les membres de l'équipe DevOps, y compris les testeurs. Lors de cette réunion, ils communiquent leur état actuel et leurs progrès réels au reste de l'équipe. Tout obstacle susceptible de bloquer le développement ou la progression des tests est communiqué lors des réunions quotidiennes, afin que toute l'équipe en soit informée et puisse agir en conséquence. De cette manière, la gestion des risques du projet est intégrée à ces réunions quotidiennes. L'équipe peut décider de réaliser régulièrement un exercice ROAMing afin de faciliter le suivi de l'état des risques liés au projet [ROAM]. Tout risque projet, y compris ceux liés aux tests, par exemple le manque de disponibilité des environnements de test, peut être communiqué et traité lors de la réunion quotidienne. (Remarque : le suivi des risques produit fait partie du suivi de la qualité des produits et est donc abordé ci-après avec l'objectif spécifique SP 2 Surveiller la qualité du produit par rapport au plan et aux attentes.) Les réunions quotidiennes pour la gestion quotidienne des tâches et les pratiques de l'équipe DevOps liées au suivi des tâches sont des techniques éprouvées qui répondent à l'intention des pratiques spécifiques TMMi dans le domaine du processus de surveillance et de contrôle des tests.

À la fin de l'itération, une revue de sprint est organisée par rapport aux objectifs de sprint convenus. L'équipe présente ce qui a été réalisé, par exemple les stories ou les epics qui ont été accomplis. L'intégration via le pipeline CI/CD est bien sûr continue. Les résultats des tests, par exemple par rapport à la Definition of Done feront partie de la revue d'itération. Des démonstrations sont organisées avec les parties prenantes afin de discuter de la valeur métier et de la qualité du produit livré. Les parties prenantes sont représentées par le Product Owner lors de la planification de l'itération, des revues d'itération (démonstrations) et des rétrospectives. Le Product Owner participe aux discussions sur le backlog du produit et fournit des commentaires et des contributions à l'élaboration des user stories et à la conception des tests tout au long de l'itération. Les autres parties prenantes sont impliquées à la fin de chaque itération lors de la revue d'itération. Aucun suivi spécifique de la participation des parties prenantes n'est nécessaire, car la représentation des parties prenantes par un Product Owner est intégrée dans la méthode de travail Agile.

2.3.2 SG2 Surveiller la qualité du produit par rapport au plan et aux attentes

Pour le contrôle de la qualité des produits, on utilise en grande partie les mêmes mécanismes que pour le contrôle de l'avancement (voir SG1 ci-dessus). Le contrôle de la qualité des produits à l'aide d'un tableau de bord en ligne n'est pas seulement effectué par les équipes pendant le développement et les tests, c'est aussi un processus continu dans lequel la qualité des produits logiciels est surveillée en production. Les rôles opérationnels surveillent en permanence la qualité du produit à l'aide de l'observabilité, généralement définie comme des indicateurs de niveau de service (SLI). Toute régression ou déviation des SLI déclenche des actions correctives afin de préserver la qualité et l'expérience utilisateur. Dans le cadre du DevOps, la surveillance des risques produits se concentre sur la revue de la liste des risques produits, y compris les risques/menaces liés à la sécurité, lors de réunions régulières plutôt que sur la revue de toute documentation détaillée sur les risques. Les risques produit nouvellement identifiés ou modifiés, par exemple à la suite de tests exploratoires, seront discutés et les mesures de test requises seront convenues. Si nécessaire, un exercice ROAMing supplémentaire peut être demandé [ROAM]. L'état des différents risques produit est généralement indiqué à l'aide de graphiques sur le tableau de bord.

Les équipes DevOps utilisent des métriques basées sur les défauts similaires à celles utilisées dans les méthodologies de développement traditionnelles, telles que les taux de réussite/échec des tests, les taux de découverte des défauts, les défauts trouvés et corrigés, afin de surveiller et d'améliorer la qualité du produit. Le nombre de défauts détectés et résolus au cours d'une itération et ceux détectés lors des activités d'intégration continue, ainsi que le nombre de défauts non résolus susceptibles d'être intégrés au backlog de la prochaine itération, doivent être surveillés lors des réunions quotidiennes. Afin de surveiller et d'améliorer la qualité globale du produit, de nombreuses équipes DevOps utilisent également des enquêtes de satisfaction client pour obtenir des commentaires sur la conformité du produit aux attentes des clients.

Les critères de sortie des tests, par exemple pour la couverture des tests et la qualité du produit, font partie de la Definition of Done (DoD). Le respect des critères de sortie convenus est généralement surveillé à l'aide du tableau des tâches, selon lequel une story ne peut être considérée comme « terminée » que si elle répond aux critères DoD, qui incluent généralement le critère selon lequel l'élément du backlog donné est déployé en production (il est « terminé » lorsqu'il est déployé, et non l'inverse). En général, des critères de sortie spécifiques sont définis pour les tests continus. Les défauts sont surveillés en continu dans le cadre des tests continus au sein du pipeline CI/CD. Le respect des caractéristiques de qualité attendues du produit en production est surveillé à l'aide de l'observabilité. Des contrôles de qualité automatisés (basés sur des seuils définis) sont souvent mis en place à différentes étapes du pipeline DevOps afin de bloquer le déploiement lorsque la qualité du produit ne répond pas aux attentes. Ces contrôles peuvent évaluer des mesures telles que le taux de réussite des tests, les benchmarks de performance, la couverture du code et les résultats de l'analyse statique du code.

Les performances et la fiabilité sont souvent des caractéristiques de qualité non fonctionnelles importantes et font donc partie du contrôle de la qualité des produits. Des tests de performance (par exemple, des tests de charge et de résistance) peuvent être intégrés dans le pipeline afin de surveiller les performances du système par rapport aux benchmarks attendus. Des alertes sont déclenchées lorsque les seuils de performance sont dépassés, afin d'avertir les équipes qui doivent alors résoudre immédiatement les problèmes. Des outils de surveillance (des journaux) peuvent également être utilisés pour suivre en temps réel les taux d'erreur, les fuites de mémoire et les pannes du système en production. L'identification précoce de ces problèmes peut contribuer à garantir que le produit répond aux attentes en matière de fiabilité.

La réunion quotidienne est le mécanisme utilisé pour effectuer des contrôles quasi continus de la qualité des produits. Des démonstrations sont organisées avec les parties prenantes afin de procéder à la validation et

discuter de la valeur métier et de la qualité du produit livré. La qualité des produits est vérifiée et validée par rapport aux critères de qualité définis par la DoD.

Dans le cadre du domaine de processus « Planification des tests », les pratiques spécifiques relatives aux critères d'entrée, de suspension et de reprise sont susceptibles de ne pas être pertinentes. Pour plus d'informations, reportez-vous à l'objectif spécifique 2 « Établir une approche de test » du domaine de processus « Planification des tests », où une explication est donnée sur les raisons pour lesquelles ces critères sont généralement susceptibles de ne pas être pertinents dans un environnement DevOps.

2.3.3 SG3 Gérer les actions correctives jusqu'à leur terme

Les équipes DevOps remarqueraient très rapidement des problèmes tels que des écarts par rapport aux attentes dans un burndown chart et/ou un manque de progression des tâches (de test) et des stories sur le tableau des tâches. Plus important encore, si la qualité du logiciel en production se dégrade, cela est immédiatement remarqué et déclenche généralement des actions correctives, telles que le retour à une version stable connue. Ces problèmes, ainsi que d'autres, par exemple ceux qui bloquent la progression des tests, sont communiqués lors des réunions quotidiennes, afin que toute l'équipe en soit informée. Cela déclenche alors une discussion au sein de l'équipe, au cours de laquelle les problèmes sont analysés. L'équipe, en collaboration avec le propriétaire du produit, décidera des mesures correctives à prendre. La gestion des mesures correctives dans les projets basés sur DevOps relève principalement de la responsabilité de l'équipe auto-organisée. L'équipe peut définir et mettre en œuvre des mesures correctives appropriées, ou signaler tout problème comme « obstacle » au Scrum master. Les mesures correctives pour la qualité des produits sont parfois automatisées, et les causes profondes sont surveillées afin d'éviter qu'elles ne se reproduisent. Cela fait partie de la culture d'amélioration continue de l'équipe DevOps. Le Scrum Master a généralement la responsabilité d'aider l'équipe à gérer les problèmes jusqu'à leur résolution. Les événements typiques pour discuter et gérer les mesures correctives sont les réunions quotidiennes et les rétrospectives. Les mesures correctives qui ont été convenues peuvent être gérées jusqu'à leur résolution en tant que « tâches » ou « éléments du backlog » via le backlog du produit ou (au sein de l'itération) via le tableau des tâches.

2.4 Domaine de processus 2.4 Conception et exécution des tests

L'objectif de la conception et de l'exécution des tests est d'améliorer la capacité du processus de test pendant la conception et l'exécution des tests en établissant des spécifications de conception des tests, en utilisant des techniques de conception des tests, en exécutant un processus de test structuré et en gérant les incidents de test jusqu'à leur clôture.

Bien que l'objectif sous-jacent (« atténuer les risques et tester le logiciel ») soit le même pour un projet DevOps et un projet séquentiel traditionnel, l'approche adoptée pour tester est généralement très différente. Dans DevOps, la flexibilité et la capacité à répondre au changement sont des points de départ importants. De plus, l'analyse et la conception des tests, la mise en œuvre des tests et l'exécution des tests ne sont pas des phases de test successives, mais sont plutôt effectuées en parallèle, de manière chevauchante et itérative, soutenues par des cycles de rétroaction rapides. Le niveau de détail de la documentation des tests établie est une autre différence clé, qui repose principalement sur la documentation dans le cadre du code d'automatisation des cas de test. Avec les tests manuels, des techniques généralement basées sur l'expérience et les défauts sont utilisées dans les projets DevOps, afin de compléter l'automatisation des tests. En outre, les tests manuels peuvent être utilisés dans le cadre de tests participatifs, après le déploiement du logiciel en production et sa mise à disposition d'un groupe de personnes présélectionnées, qui suivent certaines instructions/chartes de

test et fournissent des commentaires supplémentaires au développement sur des caractéristiques de qualité spécifiques, généralement l'utilisabilité ou l'accessibilité. Les techniques de conception de tests basées sur les spécifications peuvent également être applicables et utilisées, en particulier dans le cadre de l'analyse collaborative des exigences, en répertoriant des exemples de scénarios possibles afin d'améliorer les critères d'acceptation des User Stories. Avec un accent plus important mis sur les tests unitaires et d'intégration, les techniques boîte blanche telles que les tests d'instructions et de décisions sont également beaucoup plus populaires, tout comme les tests de contrat par rapport aux API. Une dernière différence majeure est le niveau de risque de régression résultant d'un taux de changement élevé, qui nécessite davantage de tests de régression aux différents niveaux de test. Idéalement, les tests de régression sont hautement automatisés, déclenchés et exécutés automatiquement dans le pipeline CI/CD, en plusieurs étapes de test. Il existe de nombreuses différences, mais en fin de compte, dans le contexte du domaine de processus « Conception et exécution des tests », il s'agit toujours de créer des tests, d'atténuer les risques produits, d'exécuter des tests et de détecter les défauts.

2.4.1 SG1 Analyser et concevoir les tests en utilisant des techniques de conception de tests

Dans DevOps, l'analyse et la conception des tests, ainsi que l'exécution des tests, sont des activités qui se soutiennent mutuellement et qui s'inscrivent généralement dans le cadre du mode de travail agile de l'ensemble de l'équipe produit. Dans les projets DevOps, les testeurs font partie d'une équipe qui crée et affine de manière collaborative les user stories. Des revues informelles fréquentes sont effectuées pendant l'élaboration des exigences, y compris les critères d'acceptation pour chaque user story. Ces critères sont définis en collaboration entre les représentants métier, les développeurs et les testeurs. En général, le point de vue unique du testeur permet d'améliorer la user story en identifiant les détails manquants, les scénarios alternatifs et en les rendant testables. L'analyse des tests n'est donc pas une activité distincte explicite, mais plutôt une activité implicite que les testeurs effectuent dans le cadre de leur rôle au sein du développement collaboratif des user stories.

Sur la base de l'analyse des user stories, les conditions de test sont identifiées. Du point de vue des tests, la base de test est analysée pour déterminer ce qui peut être testé : ce sont les conditions de test [Veenendaal24]. Si les critères d'acceptation définis sont suffisamment détaillés et clairs, ils peuvent très bien remplacer les conditions de test traditionnelles. Dans le contexte DevOps, des techniques telles que le développement piloté par le comportement (BDD) et le développement piloté par les tests d'acceptation (ATDD) utilisant la syntaxe Gherkin peuvent souvent aider à définir des scénarios et à identifier les conditions à tester, en les documentant comme critères d'acceptation de la user story donnée. De même, avec les exigences non fonctionnelles, les conditions de test spécifiques sont souvent définies sous la forme de critères d'acceptation. Les conditions de test sont ensuite traduites en tests, qui sont nécessaires pour couvrir les critères d'acceptation définis et convenus. Les conditions de test intéressantes peuvent également être consignées dans des chartes de test à utiliser avec les tests exploratoires. Bien sûr, ce qui précède ne s'applique pas seulement au niveau des stories, mais aussi au test des epics et des features. Les conditions de test au niveau des epics et des features sont généralement plus abstraites que celles au niveau des user stories et couvrent plusieurs user stories ou représentent des exigences non fonctionnelles. Les techniques de conception de tests basées sur les spécifications sont généralement utiles pour dériver des conditions de test à partir des user stories et des critères d'acceptation. Cependant, dans le contexte DevOps, ces techniques de conception de tests sont le plus souvent utilisées de manière implicite plutôt qu'explicite, les testeurs, forts

de leur expérience, maîtrisant ces techniques et étant capables de les utiliser avec souplesse en fonction du contexte.

Avec le principe « test first » appliqué avec DevOps, les tests qui couvrent l'ensemble des conditions de test seront identifiés (et éventuellement automatisés) avant, ou au moins parallèlement au développement du code. Cette approche commence parfois dès le raffinement du backlog. Pour les tests unitaires automatisés, une approche telle que le développement piloté par les tests (TDD) peut être envisagée. Pour les niveaux de test supérieurs, comme déjà indiqué, le développement piloté par le comportement (BDD) et le développement piloté par les tests d'acceptation (ATDD) sont des approches populaires qui sont également étroitement liées à l'automatisation des tests.

Pour la plupart des tests manuels, les tests seront identifiés/affinés au fur et à mesure que l'équipe progressera dans l'exécution des tests. Le plus souvent, les tests ne sont pas documentés de manière aussi détaillée que dans les projets traditionnels, mais plutôt sous la forme d'idées de tests, en utilisant des tests exploratoires. Pour les domaines complexes et critiques, une approche plus traditionnelle de la conception des tests et du développement des cas de test (à l'aide de techniques formelles de conception de tests) peut être le meilleur moyen de couvrir les risques. Cependant, même avec cette approche, la quantité de documentation sera limitée par rapport aux tests effectués dans un environnement traditionnel à cycle de vie séquentiel. La hiérarchisation des tests suit la hiérarchisation des user stories qu'ils couvrent. La hiérarchisation des user stories est basée sur la valeur métier ; la priorité la plus élevée correspond à la valeur métier la plus élevée. Il est important que les tests soient établis de manière à couvrir les risques fonctionnels et non fonctionnels, mais dans le domaine du DevOps en particulier, il est également important de couvrir le risque de régression. La priorité des tests est mise en œuvre en ajoutant des tests aux tâches du pipeline CI/CD.

Les données de test spécifiques nécessaires pour prendre en charge les conditions de test et l'exécution des tests sont identifiées. Cependant, dans DevOps, contrairement aux environnements traditionnels, les données de test nécessaires ne sont généralement pas spécifiées au préalable dans un document de spécification de test. Les données de test sont plutôt fournies dès que les outils et/ou fonctionnalités nécessaires sont disponibles. Les données de test sont créées immédiatement afin de permettre un démarrage quasi instantané de l'exécution des tests manuels. Pour les tests automatisés, cependant, les données devront généralement être spécifiées à l'avance. Les données de test sont généralement gérées de la même manière que les environnements de test et peuvent être déployées dans ces environnements à la demande. Les outils de préparation des données de test peuvent anonymiser, randomiser et masquer les données de production selon les besoins, conformément aux réglementations en matière de confidentialité et au RGPD.

La traçabilité entre les exigences, les conditions de test et les tests doit être établie et maintenue. Les équipes doivent clairement indiquer qu'elles ont couvert les différentes user stories et les critères d'acceptation dans le cadre de leurs tests. L'équipe a donc besoin d'outils pour organiser et gérer ses exigences (scénarios d'utilisation) de manière à attribuer des identifiants à chaque scénario d'utilisation, afin que ces identifiants puissent être utilisés pour garantir que les tests sont effectués conformément aux critères convenus. En général, les tests et les scénarios d'utilisation liés dans un outil offrent des possibilités de mesure de la couverture en temps réel. La traçabilité entre les tests et les scénarios est visible grâce au statut et au journal des tâches de test du pipeline CI/CD, générés automatiquement.

2.4.2 SG2 Implémenter les tests

L'implémentation des tests consiste à mettre en place tout ce qui est nécessaire pour commencer l'exécution des tests. En général, l'élaboration de la documentation de test pour soutenir l'exécution des tests est réduite

au minimum. Au lieu de cela, des scripts de test automatisés (de régression) sont développés et classés par ordre de priorité. La préparation des tests de régression commence dès que possible, parallèlement aux autres activités de test.

Dans un environnement DevOps, les procédures de test détaillées pour les tests manuels ne sont pas courantes. Au sein d'une équipe compétente, les tests seront très probablement documentés à un niveau d'abstraction beaucoup plus élevé, par exemple sous forme de chartes de test avec des tests exploratoires. Cela suffira pour ceux qui exécutent les tests, car on attend d'eux qu'ils possèdent le niveau de connaissances techniques et du domaine requis pour effectuer les tests. Les personnes qui exécutent les tests travaillent au sein de l'équipe, elles ont donc une meilleure compréhension à la fois de ce qui est codé et de la manière dont cela est codé, ainsi que de l'adéquation des fonctionnalités à l'objectif visé. Étant donné que le niveau de changement au sein des itérations et entre celles-ci est généralement élevé, l'élaboration de procédures de test détaillées entraînerait également un travail de maintenance important. En général, une grande partie des tests manuels est effectuée à l'aide de tests exploratoires. Dans le cadre des tests exploratoires, aucune procédure de test détaillée n'est élaborée, mais plutôt des chartes de test d'une page de haut niveau, décrivant les idées de test et les conseils à l'intention du testeur. Cependant, la plupart des équipes DevOps utilisent des tests automatisés. Les approches typiques utilisées sont le TDD, le BDD et l'ATDD. Bien que le BDD ne soit pas en soi une automatisation des tests, il peut servir de base à l'automatisation des tests d'acceptation. Les scénarios écrits en BDD (dans une syntaxe telle que Gherkin) sont généralement liés à des frameworks d'automatisation des tests. L'utilisation d'une ou plusieurs de ces approches permet de créer des scripts de test automatisés, par exemple pour détailler les scénarios Gherkin, dans le cadre de l'activité d'implémentation des tests. Les scripts de test servent également de documentation de test, avec suffisamment de commentaires pour être compris, et en utilisant de bonnes pratiques de codage. En général, des suites de tests sont créées pour regrouper les scripts de test développés. Ces suites sont ensuite liées à des tâches de test dans le pipeline CI/CD.

Les données de test spécifiques, telles que spécifiées lors de l'activité d'analyse et de conception des tests, sont de préférence créées dans un format redéployable. Les données de test sont parfois configurées sous forme de code, les données de test étant alors définies au format de fichiers de schéma et de règles. Un test d'admission est spécifié pour être mis en œuvre dans le pipeline CI/CD. Ce test, parfois appelé test de confiance ou test fumigatoire, est utilisé pour décider, au début de chaque tâche d'exécution de test dans le pipeline, si l'objet du test est prêt pour des tests détaillés et supplémentaires. Le test fumigatoire est destiné à fournir un retour d'information instantané à l'équipe de développement et aux parties prenantes en cas d'échec du test. En outre, une liste de contrôle automatisée de l'état de santé du déploiement est souvent définie pour chaque environnement/étape de test au sein du pipeline CI/CD. Les tâches liées à la préparation des données de test, aux modifications de l'environnement de test ou à la mise en œuvre de l'automatisation des tests sont estimées et planifiées sur le tableau des tâches de l'équipe lors de la planification des itérations.

En ce qui concerne la pratique spécifique 2.4 Élaborer un calendrier d'exécution des tests, l'exécution automatisée des tests est déclenchée par toute modification du code dans le pipeline CI/CD. Les scénarios de déclenchement sont identifiés et codés (web-hooks) afin de lancer l'exécution d'un test en cas de modification. Les exécutions de tests manuels sont planifiées conformément à la planification des itérations, gérées comme des tâches de test via le tableau des tâches et discutées lors des réunions quotidiennes.

2.4.3 SG3 Exécuter les tests

Les tests sont exécutés conformément aux tâches spécifiées dans le pipeline CI/CD à chaque modification et pour les nouvelles stories développées par l'équipe conformément au plan d'itération. L'ordre des tâches de test est défini de manière à fournir un retour d'information rapide. Si les ressources disponibles sont suffisantes, les tâches de test peuvent être exécutées en parallèle afin d'amplifier le retour d'information. Les coûts du cloud et la durabilité sont essentiels pour optimiser le pipeline CI/CD. Les défauts/incidents sont signalés et des journaux de test sont générés. Dans un projet DevOps, les logiciels sont livrés en production aussi souvent que possible, en fonction des modifications apportées au code. Les tests font partie intégrante du processus dans le cadre duquel le travail est effectué en coopération afin de fournir un produit de qualité. Les tests fumigatoires sont exécutés dans le cadre des tâches de test définies dans les pipelines CI/CD avant que des ensembles de tests plus importants ne soient exécutés. En cas d'échec, l'exécution du pipeline n'est pas poursuivie jusqu'à la tâche suivante et un retour d'information est transmis au développement. Si la cause de l'échec est identifiée comme un défaut, celui-ci est signalé.

L'exécution des tests est effectuée conformément aux priorités définies lors de la planification de l'itération. Certains tests peuvent être exécutés à l'aide d'une procédure de test documentée, mais en général, de nombreux tests sont exécutés à l'aide de tests exploratoires et basés sur des sessions comme cadre. Les sessions de tests exploratoires sont basées sur des chartes de test prédéfinies et sont limitées dans le temps. Les tests peuvent également être exécutés en production, avant que le logiciel ne soit mis à la disposition de tous les utilisateurs. Dans le cas des tests participatifs, un groupe sélectionné de bêta-testeurs donne son avis sur certains aspects qualitatifs du produit avant sa mise sur le marché. Cela nécessite que les fonctionnalités puissent être activées et désactivées (commutation de fonctionnalités) en fonction de conditions préalables prédéfinies, telles que l'utilisateur ou la situation géographique.

Avec le développement DevOps, il est de plus en plus nécessaire d'organiser et de structurer les tests de régression. Les tests de régression, en particulier au niveau des tests unitaires et d'intégration, font partie d'un processus d'intégration continue. Il s'agit essentiellement d'un processus automatisé de « construction et de test » qui a lieu à chaque modification du code et qui détecte les défauts rapidement et à un stade précoce. L'intégration continue permet d'exécuter régulièrement des tests de régression automatisés et de fournir rapidement à l'équipe un retour d'information sur la qualité du code et la couverture de code obtenue. Les tests sont également exécutés conformément aux tâches spécifiées dans le pipeline CI/CD, déclenchées par une modification du code. Ces scripts de test automatisés fournissent des résultats de test et des journaux de test. Les journaux de test agrégés fournissent généralement des vues par epic, user story, cas de test, suite de tests et tâche de test, ainsi que pour l'ensemble du pipeline.

Les écarts entre les résultats réels et attendus sont signalés comme des défauts. La gestion des défauts fait généralement partie intégrante du système de développement et offre une traçabilité complète de l'exécution et des modifications du code. En outre, les journaux de test sont fournis automatiquement par tous les systèmes afin de fournir un enregistrement chronologique des détails pertinents concernant l'exécution des tests.

Avec le développement et le test de nouvelles user stories par l'équipe, les projets Agile font généralement l'objet d'une discussion pour déterminer si tous les défauts détectés doivent effectivement être consignés. En particulier, lorsque l'équipe est regroupée au même endroit, les testeurs doivent discuter avec les développeurs, et les défauts qui peuvent être corrigés immédiatement et inclus dans la prochaine version n'ont pas besoin d'être consignés, ils doivent simplement être corrigés. Certaines équipes n'enregistrent que les défauts qui échappent aux itérations, d'autres les enregistrent s'ils ne peuvent pas être corrigés le jour

même, d'autres encore n'enregistrent que les défauts hautement prioritaires. Si tous les défauts détectés ne sont pas enregistrés, des critères doivent être disponibles pour déterminer quels défauts doivent être enregistrés et lesquels ne doivent pas l'être.

Dans les équipes DevOps, il est également considéré comme une bonne pratique d'enregistrer les données pendant l'exécution des tests afin de déterminer si les éléments testés répondent aux critères d'acceptation définis et peuvent effectivement être considérés comme « terminés ». L'enregistrement des données de test doit être pris en charge par l'automatisation utilisée. Cela vaut également pour l'application de techniques basées sur l'expérience, telles que les tests exploratoires. Voici quelques exemples d'informations qu'il peut être utile de documenter : la couverture des tests (ce qui a été couvert et ce qui reste à tester), les observations faites pendant les tests (par exemple, le système et le scénario utilisateur testés semblent-ils stables), la liste des risques (quels risques ont été couverts et lesquels restent parmi les plus importants), les défauts détectés et autres problèmes, ainsi que les questions ouvertes éventuelles. Les informations enregistrées doivent être saisies et/ou résumées dans un outil de gestion de l'état d'avancement (par exemple, des outils de gestion des tests, des outils de gestion des tâches, un tableau des tâches), de manière à permettre à l'équipe et aux parties prenantes de comprendre facilement l'état d'avancement de tous les tests effectués. Bien sûr, l'équipe n'a pas seulement besoin de connaître l'état d'avancement des tâches de test isolées, elle doit également connaître l'état d'avancement global de la user story. Un élément du backlog est terminé lorsqu'il répond à la Definition of Done. Lorsqu'une équipe DevOps utilise le déploiement continu comme pratique, la Definition of Done inclut généralement également le déploiement réussi comme critère.

2.4.4 SG4 Gérer les incidents de test jusqu'à leur clôture

Le terme « incident » dans DevOps est généralement utilisé pour désigner un incident réel en production qui affecte les utilisateurs. Par conséquent, nous utiliserons ici le terme « défaut » dans le reste de ce paragraphe pour désigner un incident de test dans le contexte de TMMi. Les équipes DevOps étant également responsables des opérations, elles doivent réagir rapidement. La gestion des incidents dans le cadre du DevOps est un processus de gestion des services informatiques (ITIL) : « Pratique visant à minimiser l'impact négatif des incidents en rétablissant le fonctionnement normal des services aussi rapidement que possible. »

Comme dans DevOps, presque tout est codé, tous les défauts détectés sont en fait des défauts dans des logiciels. S'il s'agit d'un problème d'environnement, c'est un défaut dans l'Infra as Code de l'environnement, s'il s'agit de données de test erronées, c'est un défaut du code d'automatisation des tests, s'il s'agit d'un autre problème de fiabilité d'exécution, cela peut être un défaut du code du pipeline, etc. Comme indiqué dans le paragraphe précédent, dans les environnements DevOps, tous les défauts détectés ne seront pas consignés. Cet objectif spécifique ne s'applique qu'aux défauts qui sont consignés et qui doivent donc être gérés jusqu'à leur résolution. En principe, la gestion des incidents dans DevOps est simple. Lorsque l'incident est consigné sur le tableau des tâches, il est visualisé comme un défaut empêchant la réalisation d'une story et de ses tâches. La méthode de travail DevOps standard sera appliquée comme pour tout autre obstacle ou tâche bloquant la progression. Elle sera discutée lors des réunions et attribuée à l'équipe pour être résolue.

Si la décision est prise de reporter un défaut détecté à une autre itération, celui-ci devient simplement une autre option (ou modification) souhaitée pour le produit. Ajoutez-le au backlog et hiérarchisez-le en conséquence. Lorsque la priorité sera suffisamment élevée, il sera repris par l'équipe lors de la prochaine itération.

Il est recommandé de surveiller le nombre de défauts détectés et résolus au cours d'une itération, ainsi que le nombre de défauts non résolus susceptibles d'être ajoutés au backlog de la prochaine itération, lors des réunions quotidiennes et des rétrospectives.

2.5 Domaine de processus 2.5 Environnement de test

L'objectif de l'environnement de test est d'établir et de maintenir un environnement adéquat, y compris des données de test, dans lequel il est possible d'exécuter les tests de manière gérable et reproductible.

Le domaine de processus Environnement de test permet de mettre en place et de maintenir un environnement de test adéquat, y compris des données de test génériques, dans lequel il est possible d'exécuter les tests de manière gérable et reproductible. Bien entendu, un environnement de test adéquat est également indispensable dans un projet de développement logiciel DevOps. Grâce à l'utilisation des technologies cloud pour chaque étape du pipeline CI/CD, un environnement peut être mis en place à la demande dans un état contrôlé. En raison des cycles courts d'une itération, l'environnement de test doit être très stable et disponible. Les problèmes dans l'environnement de test, par exemple dans le pipeline CI/CD, auront toujours un impact immédiat sur la progression et les résultats de l'itération. Il est donc également primordial de gérer correctement la configuration et les modifications de l'environnement de test et des données de test.

2.5.1 SG1 Développer les exigences d'environnement de test

La spécification des exigences relatives à l'environnement de test est effectuée dès le début du projet, puis de manière continue sur la base d'un apprentissage continu. Il est important de comprendre le flux de valeur du produit, l'environnement de production, le paysage technique et la chaîne d'outils de développement utilisés afin de déterminer les besoins en matière d'environnement et d'élaborer ensuite les exigences qui guideront l'implémentation de l'environnement. La spécification des exigences est revue afin de garantir son exactitude, sa pertinence, sa faisabilité et sa représentation fidèle d'un environnement opérationnel « réel ». La spécification précoce des exigences présente l'avantage de laisser plus de temps pour acquérir et/ou développer l'environnement de test requis sur la base de la virtualisation, de la conteneurisation et d'autres aspects natifs du cloud qui sont souvent utilisés dans le contexte DevOps. Les environnements dans le pipeline de construction (la partie CI du pipeline) sont généralement déployés à petite échelle et de manière simple, puis développés en continu en fonction des attentes croissantes en matière de performances et de fiabilité. Cependant, dans le pipeline de déploiement (la partie CD du pipeline), il peut être nécessaire de procéder à un test de bout en bout des processus métier, ce qui nécessite un environnement de test beaucoup plus complexe.

Les besoins des environnements de test sont identifiés et spécifiés lors de sessions de raffinement, conformément aux pratiques et normes Infrastructure-as-Code et Configuration-as-Code définies par les playbooks CI/CD ou par une équipe de plateforme. La priorité de l'environnement de test et les exigences en matière de données sont basées sur la priorité correspondante des éléments du backlog. Parfois, un cahier des charges technique distinct, assorti de critères d'acceptation, est spécifié pour définir des exigences environnementales plus complexes. Les exigences sont généralement rédigées dans des langages conformes à l'infrastructure/configuration en tant que code, conformément aux pratiques et à la terminologie CI/CD de l'organisation. Les exigences sont mises en correspondance avec le flux de valeur ou les étapes du pipeline CI/CD. Les chaînes d'outils de développement sont alignées sur la technologie opérationnelle et de déploiement afin de garantir qu'elles sont nécessaires et complètes.

2.5.2 SG2 Implémenter l'environnement de test

L'implémentation du pipeline de déploiement et de l'environnement de production et de test doit commencer dès que possible afin de disposer d'une version initiale des environnements dès le début de la première itération. Les environnements de test sont des environnements de type production comprenant des données de test génériques permettant d'effectuer les niveaux de test pertinents dans les étapes de test CI/CD données. Les équipes DevOps efficaces réutilisent également l'automatisation de la production pour les environnements de test, en les déployant à la demande. Toutes les tâches liées à l'implémentation de l'environnement doivent être traitées dans le backlog comme n'importe quel autre élément du backlog.

Dans le contexte DevOps, l'infrastructure en tant que code est souvent utilisée pour définir et mettre en œuvre les composants de l'environnement, et la configuration en tant que code pour définir et mettre en œuvre les variables et les configurations de l'environnement, à l'aide de scripts et de pipelines automatisés. Il est souvent important de disposer d'une capacité de provisionnement d'environnement à la demande, en utilisant des services en libre-service dans les pipelines CI/CD, également pour les sessions de tests exploratoires. Les données de test génériques pour les environnements de codage sont créées à l'aide d'outils automatisés de génération et de gestion des données de test. Un test fumigatoire est généralement exécuté dans le pipeline après le déploiement d'une instance d'environnement et avant que l'environnement ne soit utilisé pour des activités supplémentaires.

2.5.3 SG3 Gérer et contrôler les environnements de test

La gestion et le contrôle des environnements de test et de production auront lieu tout au long d'un projet DevOps. Les environnements sont gérés et contrôlés afin de permettre l'exécution ininterrompue des tests à l'aide des outils disponibles pour permettre la création, la configuration, l'allocation et l'utilisation des environnements et des données. La gestion des systèmes, effectuée sur les environnements de test pour prendre en charge le provisionnement, la configuration, l'allocation, l'utilisation et la mise hors service, comprend les pratiques suivantes :

- gérer l'infrastructure par le biais du code, de l'automatisation et de la surveillance
- gérer l'accès à l'environnement de test en fournissant des informations de connexion
- fournir une assistance technique sur les problèmes perturbant le déroulement des tests pendant leur exécution
- fournir une surveillance et une télémétrie, qui peuvent être utilisées pour analyser les résultats des tests et les causes des défauts.

Le provisionnement, l'allocation, la modification et l'utilisation des données de test sont également gérés et automatisés dans la mesure du possible, afin de soutenir le processus de test.

La création et l'attribution des environnements de test sont automatisées, coordonnées et mesurées afin d'atteindre une disponibilité et une efficacité maximales. Le provisionnement en libre-service ou à la demande (spin-up/down) et la surveillance des instances d'environnement, en particulier pour les environnements persistants, sont mis en œuvre. Les problèmes qui surviennent lors du provisionnement ou de l'utilisation des environnements de test sont consignés et gérés jusqu'à leur résolution par les équipes DevOps. Les incidents liés à l'environnement de test sont consignés lorsqu'un problème est observé, soit automatiquement dans le cadre de tests échoués, soit par les utilisateurs de l'environnement.

En conclusion principale pour le domaine de processus « Environnement de test », les objectifs et pratiques spécifiques restent tous applicables et ne changent pas dans leur essence.

3 TMMi niveau 3 défini

3.1 Domaine de processus 3.1 Organisation des tests

L'objectif du domaine de processus Organisation des tests est d'identifier et d'organiser un groupe de personnes hautement qualifiées chargées des tests. Outre les tests, le groupe de test gère également les améliorations apportées au processus de test et aux actifs du processus de test de l'organisation, sur la base d'une compréhension approfondie des forces et des faiblesses du processus de test et des actifs du processus de test actuels de l'organisation.

3.1.1 SG1 Mettre en place une organisation des tests

L'organisation des tests est un domaine de processus souvent mal compris. Beaucoup interprètent cela comme signifiant que le TMMi exige un groupe de test indépendant, voire un département qui effectue des tests indépendants. Les équipes DevOps étant conçues pour être interfonctionnelles et axées sur l'optimisation des flux et l'amplification du retour d'information, une organisation de test indépendante irait à l'encontre de ce concept. Ainsi, le groupe de professionnels des tests hautement qualifiés mentionné ici est généralement réparti dans des équipes interfonctionnelles et différents rôles. En règle générale, un centre de compétences en tests ou une guilda de tests se concentre sur l'amélioration des processus de test et de la profession au sein de l'organisation, y compris les compétences et les parcours professionnels, tandis que les équipes Scrum permanentes se concentrent sur l'optimisation de la création de valeur. Les deux constituent ainsi des mécanismes d'amélioration.

Les tâches et responsabilités typiques d'un centre de compétences en tests consistent à établir, gérer et améliorer les processus de test standardisés dans toute l'organisation et à déléguer des membres à des équipes DevOps interfonctionnelles ayant une expérience, des connaissances et des compétences en matière de tests. Un centre de compétences en tests est généralement responsable des méthodologies et des processus de test. Il convient de noter qu'un testeur fait généralement partie d'une équipe DevOps au quotidien et qu'il participe donc aux tests continus dans le pipeline CI/CD, mais qu'il est également courant qu'un testeur appartienne à une organisation de test, telle qu'un centre de compétences en tests ou une guilda de test, qui se concentre sur la profession de testeur.

Une organisation de test nécessite un leadership dans les domaines liés aux tests et aux questions de qualité. Cependant, les tests doivent être considérés comme faisant partie intégrante du flux de valeur et les activités liées aux tests sont réparties entre les différents rôles de l'organisation DevOps. Afin d'établir les compétences en matière de tests dans les rôles nécessaires, une assistance qualité est fournie par un spécialiste des tests. L'option du centre de compétences en tests s'intègre bien dans le DevOps et garantit que les tests sont préservés en tant que discipline et traités avec sérieux. Une autre option dans le contexte DevOps consiste à donner à l'organisation de test la forme d'une guilda de test. Une guilda de test est généralement un groupe informel et autogéré de testeurs, qui sont membres de différentes équipes DevOps. Les activités importantes d'une guilda de test peuvent être le partage des connaissances, l'apprentissage en groupe, l'observation des tendances, etc. L'adhésion à une guilda peut être volontaire. Une guilda de test peut également se voir attribuer des tâches plus formelles, telles que l'établissement de parcours professionnels dans le domaine des tests, l'organisation de formations et la détermination, la planification et la mise en œuvre d'améliorations des processus de test. Pour qu'une guilda de test soit couronnée de succès, il est important que ses membres se voient attribuer du temps à consacrer aux différentes activités de la guilda.

3.1.2 SG2 Établir des fonctions de test pour les spécialistes de test

Le test est considéré comme une profession et une discipline à part entière. Une équipe DevOps a besoin de connaissances et de compétences en matière de tests. Le spécialiste des tests fournira ces connaissances et compétences et encadrera les autres membres de l'équipe tout en effectuant des activités de test. Les connaissances et compétences requises pour un testeur DevOps type sont différentes de celles requises dans une organisation traditionnelle. Le testeur aura bien sûr toujours besoin de connaissances et de compétences « traditionnelles » en matière de tests, mais il devra en outre généralement posséder des connaissances et des compétences en matière d'automatisation des tests et être capable d'effectuer des tâches en dehors des tests, par exemple la création de scripts ou l'ingénierie des exigences. Avec DevOps, le développement piloté par le comportement (BDD) et le développement piloté par les tests d'acceptation (ATDD) sont des approches populaires, qui doivent donc également faire partie des compétences des testeurs DevOps. En outre, la conception et la configuration de pipelines CI/CD, l'automatisation du déploiement, la gestion de la configuration et les stratégies de déploiement doivent également faire partie des compétences du testeur. Étant donné qu'un testeur DevOps travaille en équipe, les compétences relationnelles sont tout aussi importantes que l'amélioration des compétences techniques. Le testeur DevOps est un testeur dit « en T », qui possède de multiples domaines de connaissances solides, ce qui doit se refléter dans la description des fonctions de test. Certains testeurs peuvent être plus forts sur le plan technologique, tandis que d'autres peuvent se concentrer davantage sur la conception et la compréhension du client. Certaines organisations basées sur DevOps ont créé trois types de fonctions de test différents :

- Ingénieur développeur logiciel en test (SDET) – développement de solutions d'automatisation des tests et du déploiement, et de fonctionnalités de testabilité du produit, qui soutiennent les membres DevTest.
- DevTest – développeurs qui sont également responsables de tester le produit, capables d'analyser les exigences, de concevoir des tests, de les mettre en œuvre et de les exécuter (en utilisant généralement les outils d'automatisation des tests et les fonctionnalités de testabilité fournis par le SDET).
- Testeur (ou architecte de test) – supervise les activités de test entre les équipes DevOps, identifie et gère les risques, si nécessaire, et effectue également des tests spécialisés axés sur le produit, mais est principalement informé de l'effort global de test.

Quelle que soit sa forme, l'organisation des tests est composée de personnes qui ont les compétences et la motivation nécessaires pour être de bons testeurs dans un contexte DevOps. Elles sont affectées à une fonction de test spécifique (voir les exemples ci-dessus). Dans un contexte DevOps en particulier, étant donné que toute l'équipe effectuera des activités de test, il est très important d'élaborer d'autres descriptions de fonctions non spécifiques aux tests, avec des attentes concernant leurs activités de test, leurs rôles, leurs responsabilités et les connaissances et compétences requises en matière de test.

3.1.3 SG3 Établir des parcours de carrière en test

Des parcours de carrière sont établis pour permettre aux professionnels des tests d'améliorer leurs connaissances, leurs compétences, leur statut et leurs récompenses, et ainsi leur permettre de progresser dans leur carrière, par exemple en passant d'ingénieur de test DevOps à architecte de test DevOps, coach de test ou Product Owner, en s'appuyant sur leurs compétences en matière de compréhension des clients, ou en devenant Scrum Master en s'appuyant sur leurs compétences interpersonnelles. Le parcours de carrière ne doit pas nécessairement se limiter au domaine des tests, ce qui contribuera à diffuser à long terme la culture de la qualité dans l'ensemble de l'organisation. Sur la base des parcours de carrière définis, des plans de développement de carrière individuels sont élaborés et mis à jour pour chaque membre de l'organisation de

test. Cet objectif spécifique et les pratiques qui l'accompagnent sont également pleinement applicables dans un contexte DevOps. L'équipe DevOps a besoin de testeurs qui comprennent leur travail difficile, peuvent encadrer les autres membres de l'équipe et sont motivés. La plupart de ces aspects sont pris en compte dans les parcours professionnels liés aux tests.

Cependant, la définition réelle des parcours de carrière de test sera généralement quelque peu différente dans une organisation DevOps par rapport à celles qui suivent un cycle de vie séquentiel. Alors que dans une organisation traditionnelle, les parcours professionnels de test se concentrent souvent sur la croissance verticale (de testeur à responsable des tests), les parcours de carrière de test dans une organisation DevOps ont tendance à se concentrer sur la croissance horizontale (de testeur junior à testeur senior, puis à architecte de tests et peut-être à coach de tests), ou sur la croissance vers des rôles tels que Scrum Master ou Product Owner.

3.1.4 SG4 Déterminer, planifier et mettre en œuvre des améliorations du processus de test

Les améliorations des tests sont un élément fondamental de la mentalité d'amélioration continue et du cadre de gestion de la chaîne de valeur de l'organisation DevOps, qui se concentre sur les principes lean et la pensée systémique. Les idées principales s'articulent autour de l'amplification des flux, de l'élimination du gaspillage, de l'amplification du retour d'information et de l'exploitation des données recueillies à partir du pipeline DevOps, par exemple les indicateurs de flux ou les quatre indicateurs de performance DORA. Comme les autres améliorations, les améliorations des tests sont enregistrées, hiérarchisées et gérées dans le backlog de l'organisation.

Le processus d'amélioration tel que décrit par cet objectif TMMi spécifique et ses pratiques spécifiques est largement couvert par les rétrospectives organisées par une équipe DevOps, qui s'appuient sur les données et les faits générés par le pipeline. La rétrospective est l'occasion pour l'équipe DevOps de s'auto-évaluer et de définir des actions d'amélioration. Une rétrospective a généralement lieu après la revue d'itération et avant la planification de la prochaine itération. Dans le cadre du pipeline, des mesures de conception sont identifiées et la collecte de données est mise en place, afin de pouvoir les visualiser sur la carte de la chaîne de valeur et comprendre où se produisent les retards, les contraintes ou les problèmes de qualité des produits. Ces mesures permettent d'identifier les possibilités d'amélioration (des tests). Alors que les équipes DevOps se concentrent sur la mesure des activités qui leur sont plus proches, telles que les temps d'exécution des tests, les taux d'échec des tests de régression ou les temps de déploiement et de configuration de l'environnement de test, l'organisation dans son ensemble se concentre sur les mesures de flux et les quatre mesures de performance DORA.

Au cours de la rétrospective, l'équipe discute (en partie sur la base de ses indicateurs) de ce qui a bien fonctionné dans l'itération, de ce qui pourrait être amélioré et de ce qui sera engagé pour améliorer la prochaine itération. Les compétences en matière de résolution de problèmes sont également utiles, par exemple l'analyse des causes racines avec Ishikawa ou les techniques des 5 pourquoi pour analyser les problèmes. Une culture d'apprentissage continu encourage les individus à acquérir en permanence des connaissances et à échanger leurs expériences et leurs idées pour DevOps. Au cours de chaque rétrospective, l'équipe DevOps identifie et planifie des moyens d'améliorer la qualité des produits, par exemple en améliorant les processus de travail ou en adaptant les critères de la DoR ou de la DoD au niveau de l'équipe ou du flux de valeur. Les rétrospectives peuvent déboucher sur des décisions d'amélioration liées aux tests, axées sur l'efficacité, la productivité et la qualité des tests. Elles peuvent également porter sur la testabilité des applications, des user stories, des features ou des interfaces système. Tous les membres de l'équipe,

testeurs ou non, peuvent apporter leur contribution aux activités liées ou non aux tests. À la fin de la rétrospective, l'équipe DevOps doit avoir identifié les améliorations (en matière de tests) qu'elle mettra en œuvre lors de la prochaine itération. La mise en œuvre de ces améliorations lors de la prochaine itération relève de la responsabilité de l'équipe DevOps elle-même, de sorte que les améliorations sont gérées comme des éléments du backlog. Les testeurs s'approprient généralement les améliorations en matière de tests afin de s'assurer qu'elles sont mises en œuvre au sein de l'équipe. La rétrospective est un mécanisme important qui permet à une équipe d'évoluer et de s'améliorer continuellement tout au long de la durée d'un projet.

Comme la portée est souvent limitée à l'itération, des améliorations modestes mais fréquentes sont apportées, qui visent principalement à résoudre des problèmes spécifiques à l'équipe. Ces améliorations ne sont souvent pas axées sur l'apprentissage inter-flux de valeur et l'institutionnalisation des améliorations. Si l'on examine la manière dont l'amélioration des processus (de test) est organisée et gérée, on constate qu'elle est moins axée sur un groupe de processus (de test) au niveau organisationnel et davantage sur l'autogestion des équipes. Bien que cela ne soit pas une mauvaise chose, il est important de créer également un mécanisme permettant de partager les améliorations locales réussies en matière de tests avec le reste de l'organisation, et de traiter les domaines d'amélioration qui dépassent les compétences de l'équipe DevOps. Une solution possible pourrait consister à demander à un représentant de l'organisation de test (centre de compétence ou guilde de test) d'assister aux réunions rétrospectives locales afin de garantir que les améliorations soient partagées et institutionnalisées dans toute l'organisation lorsque cela est approprié.

Les grandes organisations organisent souvent des événements rétrospectifs dédiés au niveau de la chaîne de valeur, afin d'identifier les problèmes potentiels au niveau du système. Le responsable de l'amélioration des processus de test peut également relever des problèmes fondamentaux plus larges ou des initiatives ayant une incidence sur les tests dans l'ensemble de l'organisation, et les ajouter au backlog de la chaîne de valeur.

3.1.5 SG5 Déployer le processus de test organisationnel et intégrer des retours d'expérience

De plus, pour un projet DevOps, il est important d'avoir accès et de réutiliser le processus de test standard de l'organisation et les actifs du processus de test autant que nécessaire et d'avoir une valeur ajoutée. Ils doivent bien sûr être alignés sur la méthode de travail DevOps, en mettant l'accent sur l'assurance qualité et le test des éléments du backlog à tous les niveaux. L'organisation de test veillera à ce qu'ils soient déployés dans toutes les équipes. (Pour plus d'informations sur le processus de test standard de l'organisation et les actifs du processus de test, ainsi que sur leur lien avec les projets DevOps, reportez-vous à la section Cycle de vie et intégration du test.) La mise en œuvre du processus de test standard de l'organisation et l'utilisation des actifs du processus de test dans les projets DevOps peuvent être surveillées par les équipes auto-organisées elles-mêmes, par exemple en abordant cette question lors de rétrospectives. Une excellente façon d'aller de l'avant consiste à intégrer les processus et les modèles dans les outils et à rendre les pratiques obligatoires dans les pipelines, par exemple le rejet automatique de toute demande de « pull » si les objectifs de couverture ne sont pas atteints ou si les normes de codage convenues ne sont pas respectées. Un représentant de l'organisation de test, par exemple un responsable de l'amélioration des processus de test, peut assister à ces réunions afin de suivre l'état d'avancement de la mise en œuvre et de traiter les problèmes inter-projets. Cette personne peut également transmettre à l'organisation de test les enseignements précieux tirés et les améliorations apportées par l'équipe DevOps. Les enseignements tirés et les améliorations apportées aux tests peuvent ensuite être soumis sous forme de propositions d'amélioration des processus de test et gérés en conséquence.

3.2 Domaine de processus 3.2 Programme de formation en test

L'objectif du domaine de processus Programme de formation aux tests est de développer un programme de formation qui facilite le développement des connaissances et des compétences des personnes afin que les tâches et les rôles de test puissent être exécutés de manière efficace et efficiente.

L'objectif principal du programme de formation aux tests est de fournir la formation (aux tests) nécessaire aux testeurs et aux autres personnes impliquées dans les tests, tant à l'intérieur qu'à l'extérieur de l'équipe. Un programme de formation efficace garantit que les personnes impliquées dans les tests continuent à améliorer leurs connaissances et leurs compétences en matière de tests et acquièrent les connaissances nécessaires dans le domaine ainsi que d'autres connaissances et compétences liées aux tests. La qualité et les tests étant une responsabilité de l'équipe, il est prévu que la formation liée aux tests ne soit pas seulement dispensée aux professionnels des tests, mais à l'ensemble de l'équipe DevOps.

3.2.1 SG1 Etablir une capacité organisationnelle de formation en test

Le processus commence par l'identification des besoins stratégiques de l'organisation en matière de formation en test. Ces besoins stratégiques consistent à doter les équipes des compétences et des connaissances nécessaires pour mettre en œuvre efficacement les pratiques de test dans les pipelines d'intégration continue et de livraison continue, tout en veillant à ce que les considérations de sécurité soient prises en compte. Outre les besoins de formation traditionnels et Agile des testeurs, par exemple sur les techniques de conception de tests, les tests exploratoires et les mesures de couverture, il est désormais nécessaire d'acquérir des connaissances et des compétences spécifiques aux tests DevOps. On peut citer par exemple les outils et les frameworks d'automatisation des tests, les tests de sécurité (par exemple, les tests statiques de sécurité des applications (SAST), les tests d'analyse dynamique de la sécurité (DAST) et les tests de pénétration), l'infrastructure en tant que code, les pratiques de test « shift-left », le développement piloté par les tests (TDD), le développement piloté par le comportement (BDD) et le développement piloté par les tests d'acceptation (ATDD). L'identification des besoins stratégiques en matière de formation aux tests ne se limite pas aux testeurs ; il est également primordial de préciser les besoins de formation liés aux tests pour les autres membres des équipes DevOps.

Comme dans toute autre organisation, les besoins en formation aux tests seront traduits en un plan de formation (allégé), qui pourra également répondre à certains besoins spécifiques en matière de formation aux tests pour des projets particuliers. Le plan de formation doit être revu et des engagements doivent être pris. Il convient de noter que, dans le domaine DevOps en particulier, certaines compétences sont transmises de manière efficace et efficiente par des moyens informels (par exemple, formation sur le terrain, sessions de formation pendant le déjeuner, coaching et mentorat), tandis que d'autres compétences nécessitent une formation formelle. De plus, les pratiques DevOps telles que « Embed Knowledge » (Intégrer les connaissances) et « Continually Experiment » (Expérimenter en permanence) détermineront l'approche qui sera adoptée pour développer et dispenser les programmes de formation.

3.2.2 SG2 Fournir les formations en tests

D'une manière générale, les pratiques spécifiques pour SG2 « Fournir les formations en tests » sont les mêmes dans une organisation traditionnelle et dans une organisation DevOps. Sur la base du plan de formation de l'organisation, les membres de l'équipe qui ont besoin de suivre une formation afin de pouvoir remplir efficacement leur rôle de testeur sont identifiés. La formation est ensuite programmée, y compris les ressources nécessaires, et dispensée.

Comme indiqué ci-dessus, les moyens de formation informels sont souvent utilisés dans une organisation DevOps. Par exemple, en plus de la formation formelle, le mentorat et le coaching sur le terrain sont généralement encouragés et utilisés en continu. En fait, le mentorat est la responsabilité de chacun et est attendu à tous les niveaux d'une organisation. Le travail en binôme est un autre moyen couramment utilisé par les équipes pour améliorer les compétences et transférer les connaissances.

Les formations en tests dispensées, qu'elles soient informelles ou formelles, sont consignées et leur efficacité est évaluée. Une formation (en tests) suivie peut également être évaluée dans le cadre d'une réunion rétrospective au cours de laquelle il est discuté si les connaissances et les compétences des participants sont désormais plus adéquates pour accomplir leurs tâches de test.

3.3 Domaine de processus 3.3 Cycle de vie et intégration du test

L'objectif du cycle de vie des tests et de l'intégration est d'établir et de maintenir un ensemble utilisable d'actifs de processus de test organisationnels et de normes d'environnement de travail, et d'intégrer et de synchroniser le cycle de vie des tests avec le cycle de vie du développement. Le cycle de vie intégré garantit une implication précoce des tests tout au long de la chaîne de valeur. Le cycle de vie et l'intégration des tests ont également pour objectif de définir une approche cohérente des tests à plusieurs niveaux (dans le contexte DevOps, représentés par des étapes dans le pipeline CI/CD), sur la base des risques identifiés et de la stratégie de test définie, et de fournir un plan de test global, basé sur le cycle de vie des tests défini.

3.3.1 SG1 Établir les actifs du processus de test organisationnel

Une responsabilité importante de l'organisation des tests consiste à définir, documenter et maintenir un processus de test standard, conformément à la politique et aux objectifs de l'organisation en matière de tests. Il contient notamment une description des différents niveaux et types de tests à exécuter. Comme le TMMi n'impose pas de processus de test documentés en détail, les modèles, élaborés à partir d'exemples, constituent souvent une excellente approche pour établir une méthode de travail commune [Galen]. Les quadrants de test agile, dans lesquels les équipes s'accordent sur les niveaux et les types de tests qu'elles appliquent dans un projet, constituent un moyen simple et général de représenter le processus et la stratégie de test convenus. Les niveaux de test obligatoires sont ensuite consignés dans la Definition of Done sous forme de critères. Une vue d'ensemble des étapes de test dans le pipeline CI/CD constitue également un bon document commun. Un modèle identifie les activités requises pour recueillir des informations et documenter les résultats implicites dans sa structure. Un processus ne doit pas nécessairement être une séquence stricte d'activités. Lorsque des dépendances existent, elles peuvent être consignées dans des notes dans le modèle. Les modèles ont une valeur pratique car ils transmettent l'intention réelle d'un processus. Les modèles évitent également les ambiguïtés que l'on trouve couramment dans les documents de processus « verbeux ». En tant qu'organisation DevOps, si vous disposez de processus efficaces, il vous suffit peut-être de les documenter de manière succincte et d'y ajouter quelques éléments. D'une manière générale, les organisations DevOps s'appuient sur une automatisation poussée, et les processus et règles sont souvent consignés et imposés par des outils. Ainsi, même si les processus de test n'existent pas toujours sous forme de documents distincts, ils n'en restent pas moins un élément essentiel du processus de développement et de déploiement. Bien sûr, il est important que, même avec cette méthode de travail, toutes les personnes impliquées aient une compréhension commune des processus appliqués. Voici quelques lignes directrices clés souvent utilisées dans les organisations DevOps lors de la mise en place de processus :

- Les « obligations » du processus sont présentées séparément des lignes directrices.
- Un processus ne dépasse généralement pas deux pages.

- Les processus ne contiennent pas d'informations pratiques ni d'informations sur les outils, sauf si l'on a décidé de le rendre obligatoire pour tous les projets, quelle que soit leur taille et leur ampleur.
- Des directives distinctes contiennent des options de personnalisation/planification et des informations pratiques. Notez qu'avec les informations pratiques, on peut également décider de se référer simplement à un livre ou à un document existant.

Les grandes organisations peuvent utiliser des pratiques d'ingénierie de plate-forme, où les workflows standard des éléments en attente ou les workflows des pipelines CI/CD sont obligatoires. Cela signifie que les processus définis sont mis en œuvre dans des outils spéciaux qui doivent être utilisés par les équipes. Si l'adaptation est possible, l'étape du cycle de vie est facultative.

Il n'est pas rare dans de nombreuses grandes organisations de voir quatre niveaux d'actifs de processus tels que les politiques, les processus/pratiques, les instructions/procédures de travail et les facilitateurs/modèles. Cela n'est pas dû à une exigence de TMMi, mais à la manière dont de nombreuses grandes organisations ont choisi de mettre en œuvre leurs actifs de processus. Bien que le choix des actifs de processus appartienne à chaque organisation, de nombreuses organisations DevOps constatent aujourd'hui que deux niveaux d'actifs de processus sont suffisants. Pour ce faire, elles regroupent une déclaration de politique avec la description du processus associé qui résume « ce qui doit être fait » pour mettre en œuvre le processus. Le deuxième niveau contient des directives « pratiques » pour mettre en œuvre le processus et l'adapter. Ce niveau peut être considéré comme une aide

à l'adaptation du processus, donnant à l'équipe l'autonomie nécessaire pour décider comment travailler exactement dans le cadre défini, et comprend généralement des modèles de soutien. Dans la plupart des organisations DevOps, les procédures étape par étape sont remplacées par des guides d'utilisation des outils et des formations/mentorats. Comme indiqué ci-dessus, un modèle, tel qu'un modèle de charte de test, un modèle de feuille de session de test ou un modèle de plan de test, peut servir de processus avec les activités de processus requises implicites dans le modèle. Il s'agit d'une technique couramment observée pour développer des descriptions de processus Agile efficaces. [McMahon]

L'ensemble des processus de test standard de l'organisation peut être adapté par les projets ou les équipes afin de créer leurs propres processus spécifiques. Dans le contexte de la section SP 1.3 Établir des critères et des lignes directrices d'adaptation, de nombreuses organisations utilisent une approche dite « d'adaptation vers le bas ». Cette approche exige que les personnes expliquent pourquoi quelque chose n'est pas nécessaire. Lorsque vous adaptez vers le bas, jusqu'où pouvez-vous aller ? Y a-t-il un minimum défini ? Cependant, si vous commencez par un ensemble minimum que tout le monde doit respecter, vous éliminez le risque d'adapter un « incontournable ». Une approche « d'adaptation vers le haut » est beaucoup plus cohérente avec les approches Agile et DevOps, et est entièrement conforme à TMMi. Un ensemble de critères simples pourrait aider à poser les bonnes questions lors de l'adaptation vers le haut. Sans critères à l'appui, certains ont tendance à trop adapter vers le haut.

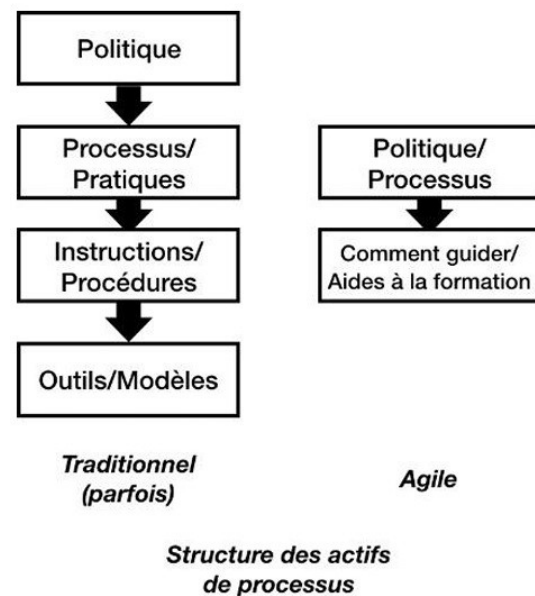


Figure 5 : Niveaux des actifs de processus

Pour soutenir le déploiement du processus de test standard, une bibliothèque d'actifs de processus de test sera créée et maintenue, souvent sur un wiki, dans laquelle les testeurs pourront trouver des modèles, des bonnes pratiques et des listes de contrôle qui les aideront dans leur travail quotidien. Généralement, ces modèles font partie des implémentations d'outils, par exemple sous la forme d'un plan de pipeline CI/CD, de champs obligatoires ou de modèles dans un outil de gestion des éléments de backlog. La création d'une bibliothèque de ressources pour le processus de tests est également pertinente dans un contexte DevOps, à condition que les équipes tirent une valeur ajoutée de l'utilisation des ressources fournies lors de leurs activités de test. Le même raisonnement s'applique plus ou moins à une base de données sur les processus de test dans laquelle les données de test, par exemple les estimations, la couverture et les mesures de qualité, sont collectées et mises à la disposition des équipes. Assurez-vous que tout ce qui est collecté et mis à disposition permet l'apprentissage et le partage d'expériences entre les équipes et apporte une valeur ajoutée à ces dernières.

Bien sûr, dans un contexte DevOps également, des normes relatives à l'environnement de travail sont utilisées pour guider la création des environnements de travail des projets.

3.3.2 SG2 Intégrer les modèles de cycle de vie de test aux modèles de développement

Le modèle standard du cycle de vie des tests définit les principales activités et les principaux livrables pour les différents niveaux de test. Le modèle standard du cycle de vie des tests est aligné sur le modèle du cycle de vie du développement, afin d'intégrer les activités de test en termes de phases, d'étapes, de livrables et d'activités. L'intégration du cycle de vie est réalisée de manière à garantir une implication précoce des tests dans les projets pour chaque élément du backlog. Bien sûr, avec DevOps, le développement et les tests doivent être totalement intégrés dans le cycle de vie, et les tests doivent avoir lieu le plus tôt possible. Comme les testeurs font partie de l'équipe et participent à la planification des itérations, les risques sont identifiés à l'avance et les activités de test sont convenues, les estimations et les engagements sont affectés, ce qui garantit l'intégration des tests dans le cycle de vie.

Cet objectif spécifique s'applique donc également à une organisation qui développe des logiciels DevOps. Dans le contexte DevOps, le développement et les tests doivent bien sûr déjà être pleinement intégrés au niveau de l'équipe, et il doit y avoir une compréhension commune de la manière dont cela est fait. Dans ce cas, aucune mesure supplémentaire n'est nécessaire. Dans d'autres circonstances, lorsque l'intégration et l'implication précoce ne sont pas encore pleinement mises en œuvre, cet objectif spécifique constitue un domaine d'amélioration très important et devrait conduire à un meilleur alignement des activités de développement et de test effectuées au cours d'une itération.

3.3.3 SG3 Établir un plan de test maître

Au niveau TMMi 3, les tests concernent la planification des tests principaux, qui traite de la coordination des tâches de test, des responsabilités et de l'approche de test à tous les niveaux de test. Cela permet d'éviter les redondances ou les omissions inutiles entre les différents niveaux de test et peut augmenter considérablement l'efficacité et la qualité du processus de test global. Le plan de test maître décrit l'application de la stratégie de test pour un projet particulier, y compris les niveaux particuliers à réaliser et la relation entre ces niveaux. Au niveau 2 de TMMi, la planification (des tests) est déjà abordée à la fois au niveau de la version et de l'itération dans le cadre du domaine de processus planification du test. Une équipe DevOps effectue généralement plusieurs niveaux de tests qui doivent tous être traités de manière appropriée dans le cadre de la planification des versions et des itérations.

Cependant, certains projets qui fonctionnent selon un modèle de cycle de vie DevOps ont adopté une approche quelque peu hybride, en ce sens qu'en plus des tests effectués au sein d'une équipe DevOps, certains tests sont organisés en dehors des équipes. Dans de telles circonstances, un plan de test maître définira et gèrera la relation entre les équipes DevOps et les niveaux ou types de tests effectués en dehors de leur champ d'application, par exemple les tests non fonctionnels externalisés (sécurité, performances) ou les tests d'intégration système avec des parties externes. Cela est généralement représenté comme une étape supplémentaire sur le tableau des tâches des éléments du backlog. Toutes ces pratiques spécifiques liées à des objectifs spécifiques seront alors généralement également applicables.

Si tous les tests sont effectués avec les équipes DevOps, un plan de test maître spécifique n'apporte aucune valeur ajoutée, et cet objectif TMMi n'est donc probablement pas pertinent dans ce cas. La planification au niveau de la version et de l'itération devrait couvrir tous les aspects nécessaires des tests. Cependant, il est utile de vérifier si l'équipe dispose d'une approche générique pour les tests qui fait l'objet d'un accord et est améliorée en permanence. En examinant la Definition of Done pour voir les niveaux de test spécifiques mentionnés, les pipelines/workflows CI/CD ou un simple document des quadrants de test Agile, ceux-ci peuvent également être considérés comme un plan de test maître allégé.

3.4 Domaine de processus 3.4 Test non fonctionnel

L'objectif du domaine de processus test non fonctionnel est d'améliorer les capacités du processus de test afin d'inclure les tests non fonctionnels lors de la planification, de la conception et de l'exécution des tests.

L'importance des tests non fonctionnels est indépendante du cycle de vie appliqué. Le produit en cours de développement déterminera les aspects non fonctionnels pertinents pour les tests. Cela vaut bien sûr également pour le développement DevOps. Les tests non fonctionnels sont donc également applicables au développement de logiciels DevOps.

Toutefois, en fonction de la nature des aspects non fonctionnels et de l'approche de test utilisée, certaines caractéristiques non fonctionnelles, telles que les performances et la fiabilité, ne peuvent pas être testées dans une itération courte en utilisant les méthodes traditionnelles. Une approche différente est donc souvent nécessaire pour tester les attributs non fonctionnels. C'est l'un des changements fondamentaux apportés par DevOps, dans la mesure où les attributs de qualité sont testés dès le début et ne sont pas laissés pour la fin. Tous les composants/fonctions ne seront pas disponibles dès la première itération pour effectuer un test non fonctionnel complet, mais les résultats de ces tests peuvent être utilisés pour donner des indications précoces sur les problèmes de qualité.

Les user stories doivent aborder à la fois les éléments fonctionnels et non fonctionnels afin de garantir que le produit développé est adapté aux utilisateurs et aux clients. Les caractéristiques de qualité de la norme ISO 25010 peuvent aider à structurer les exigences, et les testeurs doivent tenir compte de ces éléments non fonctionnels [ISO25010]. Avec DevOps, en plus des caractéristiques de qualité standard de la norme ISO 25101, l'observabilité (prise en charge par la télémétrie¹) est introduite comme caractéristique de qualité.

¹ L'observabilité désigne la capacité à comprendre l'état d'un composant ou d'un système en analysant ses sorties externes afin de permettre la surveillance, le dépannage et l'optimisation. L'observabilité utilise la journalisation, le traçage et les métriques pour fournir des informations et créer une vue cohérente de l'état du logiciel. Les informations sont automatiquement recueillies par télémétrie. La télémétrie recueille des mesures pour les caractéristiques de qualité fonctionnelles et non fonctionnelles. Parmi les exemples de télémétrie, on peut citer la télémétrie en direct du site pour mesurer le fonctionnement du logiciel, par exemple l'utilisation de la mémoire ou la charge du processeur, et la télémétrie utilisateur pour mesurer l'interaction de l'utilisateur, par exemple le suivi des clics pour l'utilisation des fonctionnalités ou le suivi du taux de remplissage des formulaires.

3.4.1 SG1 Effectuer une évaluation des risques produit non fonctionnels

Les sessions et techniques relatives aux risques produit pour les projets DevOps, par exemple le poker des risques, le ROAM et la modélisation des menaces, telles qu'identifiées et décrites dans l'objectif spécifique SG1 Effectuer une évaluation des risques produit dans le cadre du domaine de processus Planification du test, seront désormais explicitement étendues pour inclure également les aspects et risques non fonctionnels. Avec DevOps, l'accent est souvent mis sur l'observabilité, la sécurité et la conformité. À mesure que les organisations deviennent plus numériques et basées sur le cloud, les systèmes informatiques sont confrontés à des risques et des vulnérabilités accrus. La modélisation des menaces est un processus effectué au début du cycle de vie des fonctionnalités, pendant l'analyse/le raffinement, où une évaluation des risques est menée pour identifier les menaces et les atténuer à l'aide d'un modèle structuré. Le résultat se traduit souvent par des critères d'acceptation non fonctionnels au niveau des fonctionnalités, pouvant entraîner la création de user stories supplémentaires ou des décisions architecturales spécifiques. Au niveau de l'itération, l'évaluation des risques est effectuée au niveau des user stories pendant le raffinement des stories, qui définit les exigences non fonctionnelles comme une contribution majeure et aboutit à une mise à jour des critères d'acceptation des stories. Il est préférable que tous les membres de l'équipe, y compris le Product Owner, et éventuellement d'autres parties prenantes, participent aux sessions sur les risques produit. Pour certains domaines non fonctionnels, l'aide de spécialistes peut être nécessaire. Les sessions sur les risques produit aboutissent généralement à une liste documentée des risques non fonctionnels prioritaires, qui permet de mettre à jour la documentation sur les fonctionnalités et les stories. Cette liste de risques liés au produit ne sert pas seulement à orienter la conception du pipeline CI/CD et l'approche des tests non fonctionnels, mais elle façonne également activement l'architecture du produit et la technologie choisie pour le développement, par exemple en utilisant la conteneurisation pour la scalabilité et la fiabilité. Comme indiqué dans la section consacrée à la planification des tests, dans les projets DevOps, le processus utilisé et la documentation qui en résulte seront beaucoup plus légers que dans un projet traditionnel suivant un modèle de cycle de vie séquentiel.

3.4.2 SG2 Établir une approche de test non fonctionnel

Une approche de test pour les caractéristiques de qualité non fonctionnelles pertinentes est définie afin d'atténuer les risques produit non fonctionnels identifiés et prioritaires. Le pipeline CI/CD doit être conçu de manière à couvrir également les tests non fonctionnels. Pour une itération spécifique, les aspects non fonctionnels supplémentaires à tester sont identifiés lors de la planification de l'itération, tandis que les tests non fonctionnels du pipeline CI/CD sont exécutés pour chaque modification. Cependant, les caractéristiques de qualité non fonctionnelles sont également mesurées après le déploiement en production, à l'aide de la télémétrie qui s'appuie sur l'observabilité. Les indicateurs de niveau de service (SLI) et les objectifs de niveau de service (SLO) sont un moyen classique de gérer cela dans DevOps. La liste hiérarchisée des aspects non fonctionnels à tester est généralement liée aux user stories développées et testées au cours de l'itération. De nouveaux risques produit non fonctionnels peuvent apparaître au cours de l'itération, nécessitant des tests supplémentaires. Les questions telles que les nouveaux risques produit non fonctionnels nécessitant des tests supplémentaires sont généralement discutées lors des réunions quotidiennes.

L'approche de test non fonctionnel définie au niveau de l'itération pour atténuer les risques non fonctionnels couvre généralement l'identification des méthodes et techniques de test non fonctionnel appropriées en fonction du niveau et du type de risques non fonctionnels. Elle aborde également l'utilisation d'outils d'aide et l'approche de l'automatisation des tests pour les tests non fonctionnels. L'approche de test pour les tests non fonctionnels au niveau de la version sera d'un niveau beaucoup plus élevé et devra être basée sur le

pipeline CI/CD défini. L'approche de test non fonctionnel au niveau de la version et de l'itération fait partie de l'approche de test globale et sera conservée ou affichée sur le wiki de l'équipe/du projet. Les résultats des tests non fonctionnels doivent être présentés sur les tableaux de bord des produits.

Les critères de sortie non fonctionnels font partie de la « Definition of Done » (DoD). Il est important que la DoD comporte des critères spécifiques liés aux tests non fonctionnels, par exemple les temps de réponse des API ou les attentes en matière de sécurité, comme « les pages web frontales ont été testées pour la liste des 10 principaux risques OWASP ». Les SLI et SLO mesurés après le déploiement dans le cadre du pipeline CI/CD servent également de critères de sortie, l'objectif étant généralement d'éviter la dégradation des SLI ou SLO. L'itération doit aboutir à la mise en œuvre de l'ensemble convenu de critères d'acceptation non fonctionnels et éventuellement des user stories, et répondre aux critères de sortie non fonctionnels (tests) tels que définis dans la DoD. Non seulement il existe une DoD pour les éléments non fonctionnels au niveau de l'itération, mais il existe souvent également une DoD au niveau des features ou des versions, couvrant plusieurs itérations. De plus, dans le contexte DevOps, l'accent est davantage mis sur les approches Shift Right, qui mesurent les caractéristiques non fonctionnelles en production grâce à une surveillance continue, en utilisant les SLI et les SLO après le déploiement.

3.4.3 SG3 Effectuer l'analyse et la conception des tests non fonctionnels

Cet objectif spécifique suit en grande partie les mêmes pratiques, mais désormais d'un point de vue non fonctionnel, comme pour l'objectif spécifique SG1 Réaliser l'analyse et la conception des tests à l'aide des techniques de conception des tests du domaine de processus Conception et exécution des tests. Au cours de l'analyse et de la conception des tests, l'approche de test pour les tests non fonctionnels est traduite en conditions de test et en tests tangibles. Dans DevOps, l'analyse et la conception des tests, ainsi que l'exécution des tests, sont des activités qui se soutiennent mutuellement et qui s'exécutent généralement en parallèle tout au long d'une itération. Cela vaut également pour la plupart des tests non fonctionnels. L'analyse des tests non fonctionnels n'est donc pas une activité distincte explicite, mais plutôt une activité implicite que les testeurs (ingénieurs qualité DevOps) effectuent dans le cadre de leur rôle au sein du développement et du raffinement collaboratifs des user stories.

Sur la base de l'analyse des critères d'acceptation non fonctionnels et des user stories, les conditions de test non fonctionnelles sont identifiées. Les conditions de test consistent essentiellement à identifier les « éléments » qui doivent être testés/couverts. Si les critères d'acceptation définis sont suffisamment détaillés et clairs, ils remplacent généralement les conditions de test traditionnelles. Les critères d'acceptation sont ensuite traduits en tests non fonctionnels. Dans le cadre des tests non fonctionnels, il est souvent avantageux d'effectuer une analyse des tests à un niveau supérieur plutôt que de se limiter aux user stories. Par exemple, analyser une feature, une epic ou un ensemble de stories afin d'identifier des conditions de test non fonctionnelles plus abstraites que celles au niveau des user stories et couvrant plusieurs user stories. Avec le principe « test first » appliqué avec DevOps, les tests non fonctionnels qui couvrent l'ensemble des conditions de test non fonctionnelles seront identifiés (et éventuellement automatisés) avant, ou au moins parallèlement au développement du code.

Pour la plupart des tests non fonctionnels manuels, les tests seront identifiés/affinés au fur et à mesure que l'équipe progressera dans l'exécution des tests non fonctionnels. Les tests ne sont généralement pas documentés de manière aussi détaillée que dans les projets traditionnels, mais plutôt sous la forme d'idées de tests lorsque l'on utilise des tests exploratoires. Pour les domaines non fonctionnels complexes et critiques, une approche plus traditionnelle de la conception des tests et du développement des cas de test à l'aide de techniques formelles de conception de tests non fonctionnels peut être le meilleur moyen de couvrir les

risques. Cependant, même avec cette approche, la quantité de documentation sera limitée par rapport aux tests non fonctionnels dans un environnement de cycle de vie séquentiel traditionnel. La hiérarchisation des tests non fonctionnels suit généralement la hiérarchisation des user stories qu'ils couvrent. Cependant, la hiérarchisation peut également être déterminée par le temps nécessaire à la préparation et à l'exécution d'un certain test non fonctionnel.

L'identification des exigences et des stories en matière d'observabilité et de télémétrie pourrait également être le résultat d'une analyse et d'une conception non fonctionnelles, afin d'aider à mesurer les éléments non fonctionnels en production.

Les données de test spécifiques nécessaires pour soutenir l'exécution des tests non fonctionnels sont identifiées. Dans DevOps, les données de test nécessaires ne sont généralement pas d'abord spécifiées dans leur intégralité dans un document de spécification de test, mais elles sont souvent enregistrées dans le cadre de la user story, par exemple sous la forme d'un persona. Toutefois, à condition que les outils et/ou fonctionnalités nécessaires soient disponibles, les données de test nécessaires pour prendre en charge les tests non fonctionnels sont le plus souvent créées instantanément afin de permettre un démarrage rapide de l'exécution des tests manuels non fonctionnels. En revanche, pour les tests non fonctionnels automatisés, les données doivent généralement être spécifiées à l'avance. De plus, comme certaines des exigences non fonctionnelles sont testées en production, les données de test peuvent également être des données de production. Bien sûr, dans ce cas, la confidentialité et le RGPD sont essentiels.

La traçabilité entre les exigences non fonctionnelles, les conditions de test et les tests doit être établie et maintenue. Les équipes doivent clairement indiquer qu'elles ont couvert les différentes user stories non fonctionnelles et les critères d'acceptation dans le cadre de leurs tests. Dans la plupart des organisations DevOps, les user stories servent de base à l'élaboration d'un ensemble de critères d'acceptation connexes, puis de tests. Cette approche permet d'assurer la traçabilité horizontale entre les exigences et les tests.

3.4.4 SG4 Implémenter les tests non fonctionnels

L'implémentation des tests consiste à mettre en place tout ce qui est nécessaire pour commencer l'exécution des tests. En général, l'élaboration de la documentation relative aux tests, par exemple les procédures de test, pour soutenir l'exécution des tests est réduite au minimum. On privilégie plutôt l'élaboration et la hiérarchisation de scripts de test automatisés (régression). Avec DevOps, tout ce qui peut être automatisé doit l'être.

La mise en œuvre des tests non fonctionnels suivra bon nombre des pratiques déjà décrites dans l'objectif spécifique SG2 Implémenter les tests dans le cadre du processus conception et exécution des tests. Ce qui doit être fait précisément et la manière dont l'implémentation des tests non fonctionnels est effectuée dépendent en grande partie de l'approche définie, des techniques utilisées et des caractéristiques non fonctionnelles qui doivent être testées et à quel niveau. Par exemple, les tests exploratoires, qui nécessitent moins de préparation, peuvent convenir aux tests d'utilisabilité, mais sont souvent moins adaptés aux tests approfondis de fiabilité et de performance.

Pour certaines caractéristiques de qualité non fonctionnelles, la disponibilité des données de test est essentielle et celles-ci doivent être créées pendant l'implémentation des tests. Cela correspond en grande partie à ce qui se passe dans les projets traditionnels, à l'exception du fait qu'elles peuvent être documentées de manière succincte, ce qui permet de les réutiliser pour les tests de régression. Comme indiqué précédemment, certaines exigences non fonctionnelles étant testées en production, les données de test peuvent également être des données de production.

Bien entendu, l'implémentation et la préparation des tests commenceront dès que possible, parallèlement aux autres activités (de test). Il ne s'agit pas d'une phase distincte, mais plutôt d'un ensemble d'activités (répertoriées sur le tableau des tâches) qui doivent être réalisées pour permettre une exécution efficace et efficiente des tests.

3.4.5 SG4 Exécuter les tests non fonctionnels

Comme pour l'objectif précédent dans ce domaine, nous nous référerons largement à l'objectif spécifique SG 3 Exécuter les tests, qui fait partie du domaine Conception et exécution des tests abordé plus haut dans ce document. Les pratiques relatives à l'exécution des tests non fonctionnels, au signalement des défauts de test et à la rédaction des journaux de test dans un environnement DevOps sont fondamentalement les mêmes que pour les tests fonctionnels dans un environnement DevOps. Elles nécessitent généralement beaucoup moins de documentation que dans un projet traditionnel. Souvent, aucune procédure de test détaillée ni aucun journal de test n'est produit. Notez que les éléments non fonctionnels sont également testés/mesurés après le déploiement, les indicateurs de niveau de service (SLI) et les objectifs de niveau de service (SLO) étant un moyen courant de gérer cela.

L'exécution des tests non fonctionnels fait partie intégrante du pipeline CIO/CD et est effectuée conformément aux priorités définies lors de la planification des itérations. Certains tests peuvent être exécutés à l'aide d'une procédure de test documentée, mais généralement, de nombreux tests non fonctionnels sont également exécutés à l'aide de tests exploratoires et basés sur des sessions. Avec le développement itératif, il est de plus en plus nécessaire d'organiser et de structurer les tests de régression. Cela se fait à l'aide de tests de régression automatisés et d'outils de soutien. Les tests de régression s'appliquent bien sûr également aux aspects non fonctionnels du système qui ont été identifiés comme importants à tester.

Les défauts non fonctionnels détectés lors des tests peuvent être consignés et signalés par l'équipe. Les projets DevOps font généralement l'objet d'une discussion pour déterminer si tous les défauts détectés doivent effectivement être consignés. Certaines équipes ne consignent que les défauts qui échappent aux itérations, d'autres les consignent s'ils ne peuvent pas être corrigés immédiatement, d'autres encore ne consignent que les défauts hautement prioritaires. Si tous les défauts détectés ne sont pas consignés, des critères doivent être disponibles pour déterminer quels défauts doivent être consignés et lesquels ne doivent pas l'être. Parfois, les défauts sont consignés sur le tableau des tâches, soit sous forme d'autocollant sur une tâche, soit sous forme de tâche distincte, afin de visualiser qu'ils empêchent la réalisation d'une story et de ses tâches. Certaines équipes choisissent d'enregistrer et de documenter les défauts détectés à l'aide d'un outil, par exemple un outil de gestion ou de suivi des défauts. Un tel outil doit alors être utilisé de manière aussi rationnelle que possible et ne pas imposer la soumission de rapports de défauts de tests non fonctionnels qui n'ont aucune ou trop peu de valeur ajoutée. Si les exigences non fonctionnelles sont mesurées en production, un indicateur de dégradation (SLI ne remplissant pas les SLO) déclenchera un retour automatique à une version précédente, maintenant ainsi les performances globales du système.

Il est également considéré comme une bonne pratique dans les équipes DevOps d'enregistrer les données pendant l'exécution des tests non fonctionnels afin de déterminer si les éléments testés répondent aux critères d'acceptation définis et peuvent effectivement être considérés comme « terminés ». Les informations enregistrées doivent être saisies et/ou résumées dans un outil de gestion des statuts (par exemple, outils de gestion des tests, outils de gestion des tâches, tableau des tâches), de manière à permettre à l'équipe et aux parties prenantes de comprendre facilement l'état actuel de tous les tests effectués.

3.5 Domaine de processus 3.5 Revues par les pairs

L'objectif du domaine de processus « Revues par les pairs » est de vérifier que les produits d'activité répondent aux exigences spécifiées et d'éliminer rapidement et efficacement les défauts des produits d'activité sélectionnés. Un effet corollaire important est de mieux comprendre les produits d'activité et les défauts qui pourraient être évités.

3.5.1 SG1 Établir une approche de revue par les pairs

Les équipes DevOps ne procèdent généralement pas à des revues par les pairs formelles, dans le sens où elles ne disposent généralement pas d'un moment précis où les personnes se réunissent pour donner leur avis sur un produit. Cependant, elles atteignent l'objectif des revues par les pairs en effectuant des revues par les pairs moins formelles tout au long du développement. Il s'agit d'une pratique courante dans de nombreuses organisations DevOps. Comme DevOps se concentre encore plus sur le flux et met en œuvre des pratiques lean dans le développement de logiciels, toute pratique de revue soutient ce développement rapide, sans créer de goulot d'étranglement ou de retard dans le flux des activités. Il est courant d'examiner des produits d'activité plus petits, par exemple en effectuant des revues fréquentes de quelques lignes de code et en définissant de manière collaborative les critères d'acceptation des éléments en attente. Les pratiques de revue sont si importantes qu'elles sont rendues incontournables dans le processus, par exemple en imposant une revue à chaque demande de « pull » ou en imposant le respect de la Definition of Ready. Cet objectif TMMi couvre les pratiques visant à établir une approche de revue par les pairs au sein d'un projet. Une approche de revue définit comment, où et quand les activités de revue doivent avoir lieu et si ces activités sont formelles ou informelles. La mise en place d'une approche de revue par les pairs s'applique également aux projets DevOps. Dans la mesure du possible, les éléments de l'approche de revue sont établis et rendus incontournables en les implémentant dans les outils, par exemple entre les étapes du tableau des tâches gérant le flux de travail, dans les Pull Requests ou en tant que contrôles de qualité dans le pipeline CI/CD.

Exemples d'évaluations par les pairs généralement effectuées dans le cadre de projets DevOps :

- organiser régulièrement des sessions de raffinement du backlog sur les user stories avec l'équipe et les parties prenantes métier tout au long d'une itération
- organiser des réunions quotidiennes avec les autres membres de l'équipe pour discuter des produits d'activité, par exemple le code ou les tests, en cours de développement et fournir des commentaires
- la démonstration précoce et fréquente des produits aux clients, au moins à la fin d'une itération lors de la revue d'itération.
- examiner les produits d'activité tels que les user stories, les modifications de code via des pull requests, les scripts et les fichiers de configuration dans le cadre de l'Infrastructure-as-Code.
- travail en binôme sur toute activité pendant l'itération.

De mauvaises spécifications sont souvent une cause majeure d'échec des projets. Les problèmes de spécification peuvent résulter d'un manque de compréhension des besoins réels des utilisateurs, de l'absence d'une vision globale du système, de fonctionnalités redondantes ou contradictoires et d'autres malentendus. Dans le développement DevOps, les user stories sont rédigés afin de saisir les exigences du point de vue des représentants métier, des développeurs et des testeurs. Dans le développement séquentiel, cette vision commune d'une fonctionnalité est réalisée grâce à des revues formelles après la rédaction des exigences ; dans le développement DevOps, cette vision commune est réalisée grâce à des revues informelles fréquentes pendant l'établissement des exigences. Ces sessions de revue informelles sont souvent appelées sessions de raffinement du backlog. Au cours des réunions de raffinement, l'équipe et le représentant métier utilisent des

techniques de revue pour établir une compréhension commune, trouver le niveau de détail nécessaire à la mise en œuvre et clarifier les questions en suspens.

La revue quasi continue par l'équipe des produits d'activité en cours de développement fait partie de l'approche globale de l'équipe. Elle est effectuée dans le but d'identifier rapidement les défauts et de repérer les possibilités d'amélioration. Les méthodes et techniques appliquées avec DevOps, telles que XP ou le travail en binôme, incluent également les revues par les pairs comme pratique fondamentale pour créer des boucles de rétroaction pour les équipes. Bien sûr, en cas de complexité ou de risque élevé, l'équipe peut choisir d'appliquer une technique de revue semi-formelle ou formelle, par exemple une inspection. Dans ces cas, il existe une justification claire pour consacrer des efforts à une revue plus formelle et appliquer une méthode de travail plus rigoureuse. Dans le cadre de l'approche de revue, des critères peuvent être définis pour déterminer quand appliquer une technique de revue plus formelle.

Alors que l'ingénierie des exigences met l'accent sur la validation des exigences via des méthodes telles que les revues informelles, les relectures techniques, les inspections ou la lecture basée perspective, les méthodes Agile et DevOps s'efforcent plutôt de valider les exigences grâce à un retour d'information fréquent et précoce afin de mettre en œuvre rapidement des incréments de produit utiles. La nécessité d'une validation formelle précoce est réduite par la présentation de résultats rapides sous forme de prototypes ou d'incrément de produit intégrés déployés en production. Si l'amélioration ne répond pas entièrement aux exigences des parties prenantes, la différence est réintégrée dans le backlog du produit sous la forme de nouvelles exigences et classée par ordre de priorité avec tous les autres éléments du backlog. Une démonstration, lors de la revue d'itération, de ce qui a été réellement construit au cours d'une itération est tout simplement un moyen très efficace de dynamiser une conversation basée sur la validation autour de quelque chose de concret. Rien ne permet mieux de centrer la conversation que de pouvoir voir concrètement comment quelque chose fonctionne et de comprendre l'impact des fonctionnalités déployées sur le comportement des utilisateurs finaux. La démonstration est une activité qui est réalisée pendant la revue d'itération. Les fonctionnalités intégrées au système sont présentées aux parties prenantes et discutées avec elles, et les adaptations nécessaires sont apportées au backlog du produit ou au plan de lancement afin de refléter les nouveaux enseignements tirés de la discussion.

Les critères de revue par les pairs peuvent être définis de manière automatisée ou sous forme de liste de contrôle dans le flux de travail. Dans le cas des revues de code, des critères d'entrée tels que l'exécution obligatoire de tests unitaires, les mesures de complexité du code et la conformité au guide de style du code sont définis. Cela permet de s'assurer que seul le code à examiner est conforme aux critères d'entrée. Comme critère de sortie, le code doit avoir été approuvé par plusieurs membres de l'équipe. Si ce n'est pas le cas, la revue doit être répétée après la mise à jour du code. Cela signifie généralement que le code ne peut pas être mergé.

Avec les éléments du backlog, les critères d'entrée ne s'appliquent souvent qu'aux revues formelles et sont donc susceptibles de ne pas être pertinents pour les sessions de raffinement dans le cadre des projets DevOps. Cependant, l'utilisation de critères de sortie est applicable et apporte une valeur ajoutée. INVEST [Wake] est un exemple d'ensemble de critères de sortie couramment utilisés dans les projets DevOps lors de la revue et de la mise à jour des user stories. INVEST est un acronyme qui englobe les exigences suivantes, qui constituent une bonne user story :

- Indépendant
- Négociable
- Valorisable
- Estimable
- Suffisamment petit
- Testable.



Figure 6 : Critères INVEST pour les user stories

Bien sûr, il existe d'autres critères de sortie liés à la qualité qui peuvent également être utilisés de manière avantageuse.

Les critères de sortie de revue sont définis dans la Definition-of-Done pour les différents produits d'activité. Les éléments qui ne remplissent pas les critères sont renvoyés et ne peuvent pas passer à l'étape suivante.

3.5.2 SG2 Effectuer des Revues par les Pairs

Bien sûr, comme toute approche, elle ne doit pas seulement exister en tant qu'approche définie et convenue, mais elle doit également être respectée. L'équipe doit consacrer beaucoup de temps aux sessions de raffinement du backlog pendant une itération, en appliquant des revues informelles (et éventuellement formelles) dans le cadre de sa routine quotidienne, et organiser régulièrement des démonstrations pour les parties prenantes. Des démonstrations pour les parties prenantes/clients sont attendues au moins à la fin de chaque itération.

Il est important que le testeur, qui fait partie de l'équipe DevOps, participe aux sessions de revue. Cela est particulièrement vrai pour les sessions au cours desquelles sont discutés et examinés les produits d'activité qui servent de base aux tests tout au long de l'itération, par exemple lors des sessions de raffinement du backlog et de conception collaborative. En général, le point de vue unique du testeur permet d'améliorer la user story en identifiant les détails manquants ou les exigences non fonctionnelles. Un testeur peut notamment aider à identifier et à définir les critères d'acceptation d'une certaine user story. Des techniques telles que le BDD et l'ATTD peuvent être utilisées pour définir les critères d'acceptation sous la forme de tests d'acceptation, permettant ainsi de déterminer si la user story répond au critère de testabilité. Le testeur apporte ainsi sa contribution en posant aux représentants métier des questions ouvertes de type « et si ? » sur la user story, en proposant des moyens de tester la user story et en confirmant les critères d'acceptation.

La pratique spécifique 2.3 Analyser les données des revues par les pairs est particulièrement pertinente dans le cas des évaluations formelles. Les évaluations formelles nécessitent beaucoup d'efforts. Afin de garantir que ces efforts sont déployés de manière efficace et efficiente, les données issues des revues par les pairs sont collectées et communiquées à l'équipe afin d'apprendre et d'ajuster le processus d'évaluation. Comme indiqué, les revues formelles sont moins courantes dans les projets DevOps. Alors que la collecte de données est une partie essentielle des revues formelles, elle est beaucoup moins courante dans les revues informelles.

Dans le contexte de la décision de collecter des données lors des revues par les pairs, posez les questions suivantes :

- Qui va utiliser ces données si elles sont collectées ?
- Quel est le lien entre ces données et nos objectifs métier ?

Si personne n'est susceptible d'utiliser ces données de manière significative, ne gaspillez pas de précieuses ressources pour les collecter. Cependant, dans le cadre des DevOps, des outils sont généralement utilisés pour collecter les résultats et les données des évaluations, et les rendre disponibles sous forme d'informations sur un tableau de bord, ce qui rend le processus beaucoup plus efficace. De plus, lors des rétrospectives, lorsque l'équipe analyse les données et les conclusions de l'évaluation afin de comprendre les causes racines et d'identifier les possibilités d'amélioration, celles-ci prennent tout leur sens.

Glossaire de Termes spécifiques à DevOps

Déploiement bleu-vert	Stratégie de déploiement dans laquelle deux environnements de production identiques, l'un actif et l'autre inactif, sont maintenus afin de tester une nouvelle version du logiciel dans l'environnement inactif avant de transférer tout le trafic depuis l'environnement actif une fois qu'elle a été vérifiée.
Test Canary	Approche de test dans laquelle les modifications sont publiées auprès d'un petit sous-ensemble d'utilisateurs, ce qui permet de les valider avant un déploiement incrémental supplémentaire.
Pourcentage d'échec des changements	Proportion de changements déployés qui entraînent des échecs.
Délai de mise en œuvre des changements	Le temps nécessaire à une validation pour atteindre la production.
Pipeline CI/CD	Série d'étapes requises pour livrer une nouvelle version d'un logiciel.
Conteneurisation	Forme légère de virtualisation qui regroupe une application et ses dépendances dans une seule unité exécutable appelée conteneur.
Livraison continue (CD)	Procédure automatisée de développement logiciel dans laquelle les modifications du code sont automatiquement compilées, testées et déployables en production.
Déploiement continu (CD)	Procédure automatisée de publication de logiciels dans laquelle les modifications de code qui passent tous les tests sont automatiquement déployées en production sans intervention manuelle.
Surveillance continue	Collecte automatisée d'indicateurs de performance permettant de comprendre le comportement des utilisateurs et du système en temps réel.
Équipe DevOps interfonctionnelle	Groupe de personnes ayant des connaissances, des compétences et des capacités différentes mais complémentaires, qui travaillent ensemble vers un objectif commun.
Lancement discret	Stratégie de déploiement consistant à lancer une nouvelle fonctionnalité ou un nouveau produit sans annonce publique, afin de permettre aux équipes de tester et de surveiller les performances en production.
Fréquence de déploiement	Fréquence de publication des logiciels en production.
DevOps	État d'esprit, culture et ensemble de pratiques techniques qui favorisent l'intégration, l'automatisation et la collaboration nécessaires pour développer et exploiter efficacement une solution.
Temps de récupération après un déploiement échoué	Durée nécessaire pour rétablir le fonctionnement normal après un déploiement ayant échoué.
Commutateur de fonctionnalité	Mécanisme permettant de modifier le comportement du système sans changer le code.

Retour d'information	Échange continu d'informations visant à améliorer les produits, les services et les processus.
Flux	Mouvement fluide, linéaire et rapide des produits d'activités d'une étape à l'autre dans un flux de valeur pertinent.
Infrastructure en tant que code (IaC)	Pratique consistant à gérer et à provisionner l'infrastructure informatique à l'aide de fichiers de configuration lisibles par machine et à y appliquer des pratiques de développement logiciel.
Observabilité	Capacité à comprendre l'état d'un composant ou d'un système en analysant ses sorties externes afin de permettre la surveillance, le dépannage et l'optimisation.
Pull Request (PR)	Mécanisme permettant d'informer les membres de l'équipe des modifications apportées au code source, en proposant que ces modifications soient examinées, discutées et fusionnées dans le référentiel de code principal.
Indicateur de niveau de service (SLI)	Mesure quantifiable de la fiabilité d'un service.
Objectif de niveau de service (SLO)	Objectif de fiabilité pour un indicateur de niveau de service.
Ingénierie de fiabilité des sites (SRE)	Approche qui applique les pratiques et principes de développement logiciel à l'infrastructure et aux opérations tout en équilibrant la fiabilité du service et le rythme de développement de nouvelles fonctionnalités à l'aide de boucles de rétroaction automatisées.
Flux de valeur	Séquence complète des activités nécessaires pour fournir un produit ou un service à un client, y compris le flux d'informations et de matériaux.
Virtualisation	Simulation du logiciel et/ou du matériel sur lequel d'autres logiciels fonctionnent.

Références

- [Baah] A. Baah (2017), Agile quality assurance, Bookbaby
- [Beyer] B. Beyer, C. Jones, J. Petoff et N.R. Murphy (2016). Site Reliability Engineering: How Google Runs Production Systems, 1st. O'Reilly Media, Inc.
- [Cohn] M. Cohn (2009), Succeeding with Agile: Software Development using Scrum, Addison-Wesley
- [DORA16] A. Brown, N. Forsgren, J. Humble, N. Kersten et G. Kim (2016), Rapport 2016 sur l'état des DevOps, Puppet Labs, DORA (DevOps Research & Assessment]
- [DORA24] DORA [2024], DORA's software delivery metrics: the four keys, <https://dora.dev/guides/dora-metrics-four-keys/>, DORA (DevOps Research & Assessment]
- [Galen] R. Galen (2015), Les trois piliers de la qualité et des tests agiles : obtenir des résultats équilibrés dans votre parcours vers la qualité agile, RGCG, LLC
- [ISO25010] ISO/IEC 25010 (2023), Ingénierie des systèmes et des logiciels – Exigences et évaluation de la qualité des systèmes et des logiciels (SQuaRE) – Modèle de qualité des produits, Organisation internationale de normalisation
- [McMahon] Paul. E. McMahon (2011), Intégrer CMMI au développement agile – Études de cas et techniques éprouvées pour une amélioration plus rapide des performances, Addison Wesley
- [Veenendaal12] E. van Veenendaal (2012), PRISMA : Évaluation des risques liés aux produits pour les projets agiles, dans : Testing Experience, numéro 04/12, décembre 2012
- [Veenendaal24] Erik van Veenendaal, Rex Black et Dorothy Graham (2024), Fondements des tests logiciels – Certification ISTQB (5e édition), Cengage
- [Wake] B. Wake (2003), INVEST in Good Stories, and SMART Tasks, xp123.com/articles/invest-in-good-stories-and-smart-tasks