Récapitulatif

Notions introduites :

- Itération :
  - ✓ L'application est divisée en plusieurs applications plus petites (lotissement), divisant de ce fait le traitement de la complexité de l'application globale.
  - ✓ Le développement itératif permet la validation régulière par les utilisateurs.
  - ✓ Le contenu d'une itération n+1 comporte non seulement de nouvelles fonctionnalités mais aussi la prise en compte des améliorations de l'itération n.
  - ✓ Il est ainsi possible de rectifier au plus tôt les erreurs et de prendre en compte l'évolution des besoins.
- Prototypage :
  - ✓ Livraison d'un exécutable permettant une validation concrète

Méthodes Agiles

Introduction

- Pour gérer un cycle de vie, il est nécessaire d'utiliser :
  - ✓ Des méthodes
  - ✓ Des outils
- La méthode : règles et pratiques mis en œuvre par le chef de projet pour conduire son projet
- Longtemps dominant, Merise, avec son approche systémique (par la structure) et ses validations en cascade, est aujourd'hui critiquée pour sa rigidité, son manque d'adaptation et son effet tunnel
- On a vu alors apparaître une tendance à rapprocher les utilisateurs des développeurs, avec des cycles d'itérations plus courts…
- C'est l'émergence des méthodes agiles.

Motivations

- Les risques engendrés par les méthodes classiques:
  - ✓Changement des besoins fonctionnels
  - ✓Dépassements des délais et des budgets
  - ✓Bugs et Abandon du projet
  - ✓Au final, seuls ¼ des projets sont considérés comme réussis
- Nécessité de modèles plus légers
  - ✓moins de documentation
  - ✓moins de contrôle sur les procédés.
  - ✓Impliquer au maximum le client
  - ✓modèle qui s'ajustent aux changements des spécifications toute en garantissant des livraisons fréquentes
  - ✓Modèles Destinés à des projets de moyenne ou petite taille avec une équipe réduite

Modèles agiles

Méthodes agiles

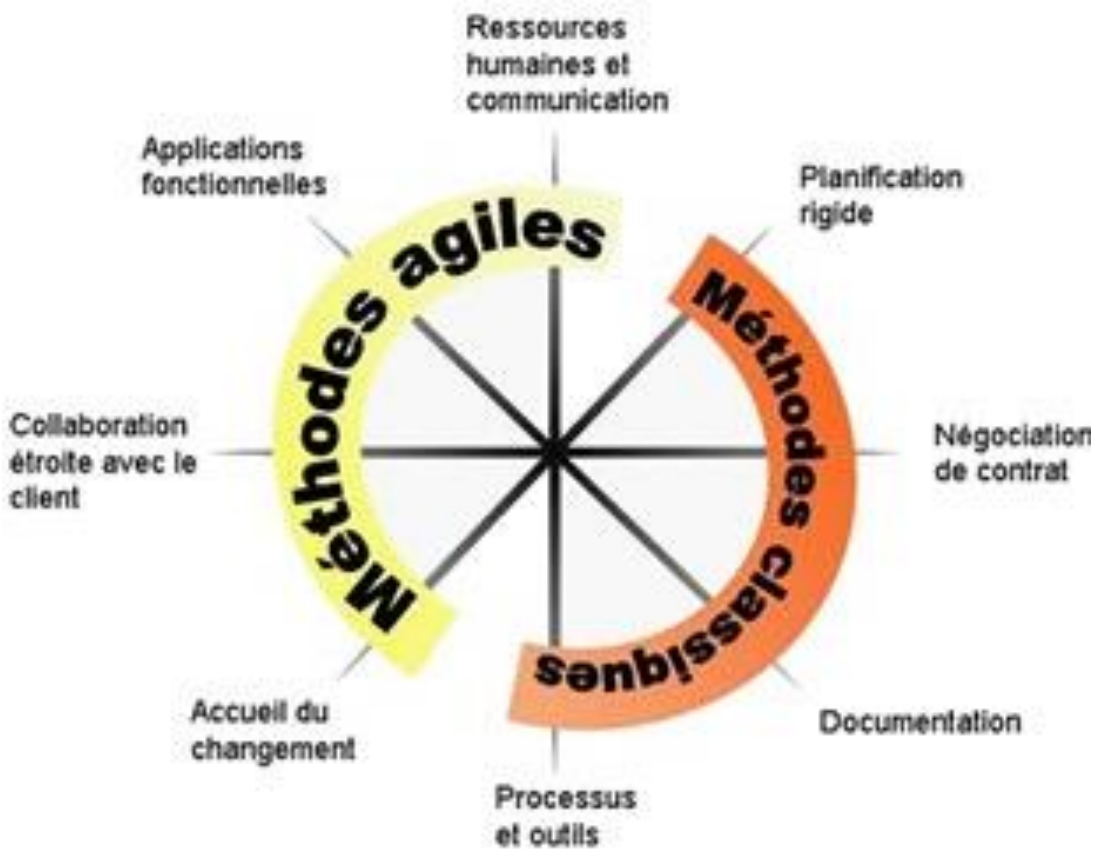
- Objectifs des méthodes agiles :
  - ✓Augmenter le niveau de satisfaction client
  - ✓Faciliter le travail de développement
  - ✓Performance
  - ✓Qualité
- Méthodes adaptatives versus prédictives
- Les pratiques des méthodes agiles :
  - ✓Souvent anciennes, admises et testées → "bon sens"
  - ✓Mais en général oubliées → "pas de mise en pratique"
  - ✓Un exemple significatif : les pratiques relatives aux tests

Historique

- Barry W. Boehm a introduit en 1986 un nouveau modèle de développement itératif et incrémental, précurseur des méthodes Extreme programming (XP), Scrum ou Crystal clear…
- En 2001, un manifeste écrit par 17 experts introduit 4 valeurs fondamentales déclinées en 12 principes permettant de définir une nouvelle façon de développer des logiciels.

Les 4 valeurs fondamentales

- Les personnes et les interactions plutôt que les processus et les outils.
- Un logiciel opérationnel plutôt qu'une documentation exhaustive.
- La collaboration avec le client plutôt que la négociation du contrat.
- Réagir au changement plutôt que le suivi d'un plan



Les 4 valeurs fondamentales

- Les personnes et les interactions:
  - ✓Les collaborateurs sont les clés de la réussite.
  - ✓Les « seniors » échoueront s'ils ne collaborent pas en tant qu'équipe.
  - ✓Un bon collaborateur n'est pas forcement un bon programmeur.
  - ✓Une surabondance massive d'outils et aussi mauvaise que leur manque.
  - ✓Démarrer petit et investir peu au démarrage.
  - ✓Construire l'équipe est plus important que construire l'environnement.

Les 4 valeurs fondamentales

- logiciel opérationnel plutôt qu'une documentation exhaustive:
  - ✓Un code sans documentation est un désastre
  - ✓Trop de documentation est encore pire
  - ✓Difficulté à produire et à synchroniser la documentation avec le code
  - ✓Souvent les documents sont des « mensonges » formels
  - ✓Le code ne ment jamais sur lui-même.
  - ✓Produire toujours des documents aussi courts que possible

Les 4 valeurs fondamentales

- La collaboration avec le client plutôt que la négociation du contrat :
  - ✓Très difficile de décrire la totalité du logiciel depuis le début
  - ✓Les projets réussis impliquent les clients de façon régulière et fréquente.
  - ✓Le client doit avoir un contact direct avec l'équipe de développement.

Les 4 valeurs fondamentales

- Réagir au changement plutôt que le suivi d'un plan :
  - ✓Tout change: technologies, environnements et surtout les besoins
  - ✓Les chefs de projets classiques fonctionnent sur la base de méthodes de répartition des tâches (PERT, GANTT)
  - ✓Avec le temps les diagrammes se dégradent à causes de nouvelles tâches à ajouter et des tâches inutiles à supprimer.
  - ✓Une meilleure stratégie est de planifier très court (1 semaine à un mois)
  - ✓Planning détaillé pour la semaine à venir, rigoureux pour les trois mois
  - ✓et très vagues au delà.

Les principes des méthodes Agile

- Livraison ASAP de versions fonctionnelles (⟸ feedback)
- Le changement comme avantage concurrentiel
- Livraisons intermédiaires aussi souvent que possible
- Coopération quotidienne entre utilisateurs et développeurs
- Construction d'une équipe motivée
- Favoriser l'échange oral sur l'écrit
- 1er indicateur d'avancement du projet : le fonct. de l'application
- Rythme soutenable pour les utilisateurs et les développeurs
- Attention continue à l'excellence technique et à la conception
- Toujours favoriser la simplicité
- Responsabilité confiée à l'équipe entière et volontariat
- Auto-ajustement de l'équipe pour améliorer son efficacité

Les principes des méthodes Agile

- Les méthodes Agiles responsabilise l'équipe :
  - ✓ l'équipe connaît les besoins et les priorités,
  - ✓ elle fait les estimations,
  - ✓ elle décide de son organisation,
  - ✓ elle produit un travail de qualité,
  - ✓ elle remonte les problèmes

Méthodes agiles principales

- **RAD** (Rapid Application Development)
- **UP** (Unified Process), **RUP** (Rational Unified Process), **2TUP** (Two Tracks Unified Process)
- **ASD** (Adaptative Software Development)
- **DSDM** (Dynamic Software Development Method)
- **SCRUM** (mêlée au rugby)

sont des approches par la structure et par les besoins

- Des approches plus radicales telles que :

- **XP** (eXtreme Programming)
- **Crystal Clear**

sont pilotées par les besoins et se veulent totalement incrémentielles et itératives

Comparaison avec les méthodes classiques

Méthodes classiques	Méthodes agiles
Phases séquentielles	Itératif et incrémental
Planification prédictive	Planification adaptative
Documentation produite en masse	Documentation réduite au strict minimum
Une équipe avec des ressources spécialisées, dirigée par un chef de projet et	Une équipe responsabilisée où l'initiative la communication sont privilégiées et soutenues par le chef d' équipe
Contrôle qualité à la fin du cycle de développement	Un contrôle qualité précoce et permanent (produit et processus)
Le client découvre le produit fini	Le client visualise les résultats tôt et fréquemment
Résistance au changement	Accueil favorable des changements
Suivi minutieux de l'avancement	Le nombre de fonctionnalités ajouté et le seul indicateur d'avancement
Mesure de succès : respect des engagement initiaux	Mesure de succès : satisfaction du client

Limites et risques

- Refactorisation de code(remanier le code, afin de le rendre plus clair, structuré et simple)
  - ✓ réaliser par une seule personne. Les autres membres de l'équipe ne s'y trouvent plus dans le code
- Documentation:
  - ✓ difficile de définir le niveau de documentation à fournir, et quelle documentation
  - ✓ Lorsque deux équipes, une agile, l'autre non, participent à un projet, la documentation est source de conflit
- Auto-organisation de l'équipe
  - ✓ Il n'est pas toujours facile de remanier une équipe
  - ✓ la mise en place des méthodes agiles peut être plus difficile que prévu.

Limites et risques

- Relation avec le client
  - ✓ Les clients peuvent avoir du mal à s'impliquer dans un projet (le client ne comprend pas le principe même des méthodes agiles, il va refuser de s'investir, n'y voyant qu'une perte de temps)
- Négociation de contrat
  - ✓ Dans la majorité des cas le client va désirer un contrat fixe, ce qui n'est pas compatible avec les méthodes agiles.
  - ✓ Il va exiger un prix fixe, une date de livraison, et un périmètre bien défini

Quelques MA

Méthode UP

- **Processus unifié :**
  - ✓ Développé à l'origine par Philippe Kruchten et Ivar Jacobson sous la coupe de la société Rational
  - ✓ Le processus unifié est un ensemble structuré de "bonnes pratique" issues de l'expérience
  - ✓ Le terme Unifié fait d'ailleurs référence à cet aspect fusion entre les pratiques issues de méthodes antérieures.

Méthode UP

- Un processus de développement logiciel définit qui fait quoi, quand et comment pour atteindre un objectif donné
- U.P. (Unified Process - Processus Unifié) = processus de développement logiciel prenant en charge le cycle de vie d'un logiciel et de son développement
- Contrairement aux démarches antérieures :
  - ✓UP prend en compte l'ensemble des intervenants : client, utilisateur, gestionnaire, qualiticien, … d'où l'adjectif "unified",
  - ✓UP est générique pour les logiciels orientés objets utilisant UML
  - ✓comme langage de modélisation,
  - ✓UP est itérative et incrémentale.

Méthode UP

- Principes d'UP :
  1. Proximité avec les utilisateurs et pilotage par les cas d'utilisation
  2. Pratique de la modélisation graphique des exigences
  3. Centré sur l'architecture
  4. Fondé sur la production et l'assemblage de composants
  5. Développement itératif et incrémental du logiciel (chaque fin d'itération doit générer un prototype exécutable)
  6. Gestion des besoins et des exigences (traçabilité)
  7. Souci permanent de la qualité (recettes fréquentes de versions intermédiaires, automatisation des tests, revus par les pairs)
  8. Gestion des risques permanente
  9. Gestion des demandes de changement

Méthode UP

- Objectifs de UP:
  - ✓produire un logiciel de qualité en respectant des contraintes de délai, de
  - ✓coûts et de performance
  - ✓fournir des lignes directrices pour un développement efficace d'un logiciel de qualité, en réduisant les risques et en améliorant les prévisions
  - ✓Décrire les meilleures méthodes de travail pour apprendre des expériences précédente et l'amélioration du support de formation
  - ✓Établit une vision et une culture commune
- Caractéristiques majeures de UP:
  - ✓UP utilise UML
  - ✓UP est piloté par les cas d'utilisation
  - ✓UP est centré sur l'architecture
  - ✓UP est itératif et incrémental
  - ✓UP est à base de composants

Les 4 phases de UP

- **Phase de Démarrage:** définition de l'objectif du projet et identification de tous les acteurs et les cas d'utilisation essentiels (20% du modèle), élaboration d'un plan de gestion de projet déterminant les ressources nécessaires
- **Phase d'Elaboration:** permet d'avoir une bonne connaissance des besoins (90%) et d'établir une base de l'architecture
- **Phase de Construction:** développement du produit en plusieurs itérations (3 à 5 pour un projet complexe) pour une version bêta
- **Phase de Transition:** préparation du produit pour l'utilisateur final, formation, installation, support, …

Après chacune des 4 phases, les activités sont évaluées selon des critères spécifiques.

Jalons d'évaluation et itération dans UP

Après chacune des 4 phases on évalue les activités grâce à des critères spécifiques

- Chaque phase peut comporter de 0 à n jalons.
- Chaque fin de phase est ponctuée par un jalon principal et la fin d'une ou plusieurs itérations
- Entre 2 jalons, on parle d'itérations :

Une Itération est une séquence d'activités planifiées et pouvant être vérifiées grâce à un critère d'évaluation

- ✓ Elle a pour but de vérifier les activités au fur et à mesure
- ✓ Itération internes : au sein de l'équipe de développement
- ✓ Itération externes : avec le client et idéalement les utilisateurs finaux

- Chaque enchaînement d'activité dure une itération et s'inscrit dans un modèle incrémental

Les activités de UP

- Expression des besoins (modélisation métier) :
  - ✓ Modélisation des possibilités du système et besoins des utilisateurs
  - ✓ Expose les cas d'utilisation et leurs relations avec les utilisateurs
- Analyse (modélisation des besoins):
  - ✓ Modélisation du système et des besoins détaillés des utilisateurs
  - ✓ Détail des cas d'utilisation et première répartition du comportement du système sur divers objets
- Conception :
  - ✓ Définit la structure statique du système sous forme de sous système, classes et interfaces
  - ✓ Définit les cas d'utilisation réalisés sous forme de collaborations entre les sous systèmes, les classes et les interfaces
- Implémentation : Intègre les composants (code source) et la correspondance entre les classes et les composants
- Tests :
  - ✓ Concerne la vérification du système dans son ensemble
  - ✓ Décrit les cas de test vérifiant les cas d'utilisation
- Déploiement :
  - ✓ Concerne la livraison du système et la formation des utilisateurs.
  - ✓ Définit les nœuds physiques des ordinateurs et l'affectation de ces composants sur ces nœuds.

Méthode RUP

- RUP est l'une des plus célèbres implémentations de la méthode UP permettant de donner un cadre au développement logiciel.
- RUP utilise UML
- RUP est piloté par les cas d'utilisation
- RUP est centré sur l'architecture
- C'est une méthode générique, itérative et incrémentale.

Des cycles d'évolution en 4 phases

Cette méthode propose de se concentrer sur quatre phases d'évolution. Ils sont découpés en cycles.

- La création – évaluation des risques, architecture, planification,
- L'élaboration – spécification du besoin, validation, environnement projet,
- La construction – production du logiciel et de la documentation support, tests,
- La transition – test système et utilisateur, correction, déploiement.

À l'intérieur de ces phases, sont dispatchées toutes les activités du projet, en détail. Ainsi, la méthode RUP propose d'élaborer un tableau itératif regroupant tous les points mentionnés ci-dessus.

Les avantages de la méthode RUP

- Permet de s'assurer de toujours répondre aux besoins des clients en temps et en heure.
- Les livrables sont gérés et contrôlés tout au long du développement.
- Permet d'utiliser une architecture qui est basée sur les composants.
- Les modifications qui interviennent au cours du projet peuvent être prises en charge et intégrées directement :
  - ✓ Chaque intégration se fait petit à petit, au fur et à mesure du projet, et non tout à la fin quand les risques sont plus grands.
  - ✓ Les problèmes sont ainsi découverts en cours de projet et rectifiés.

Quand utiliser la méthode RUP ?

- Dans tous les projets complexes, la méthode RUP est indispensable :
  - ✓ les intégrations progressives permettent de gérer les problèmes sans drame.
  - ✓ Permet de garder un contrôle fort sur les projets compliqués.
- Dans le cas où l'exigence client est très forte:
  - ✓ elle assure un logiciel à haut niveau de qualité, avec des stades d'incrémentation plus sûrs.
- Dans le cas de budgets très serrés:
  - ✓ Les coûts sont bien maîtrisés et les retards de livraison, sont généralement très réduits.

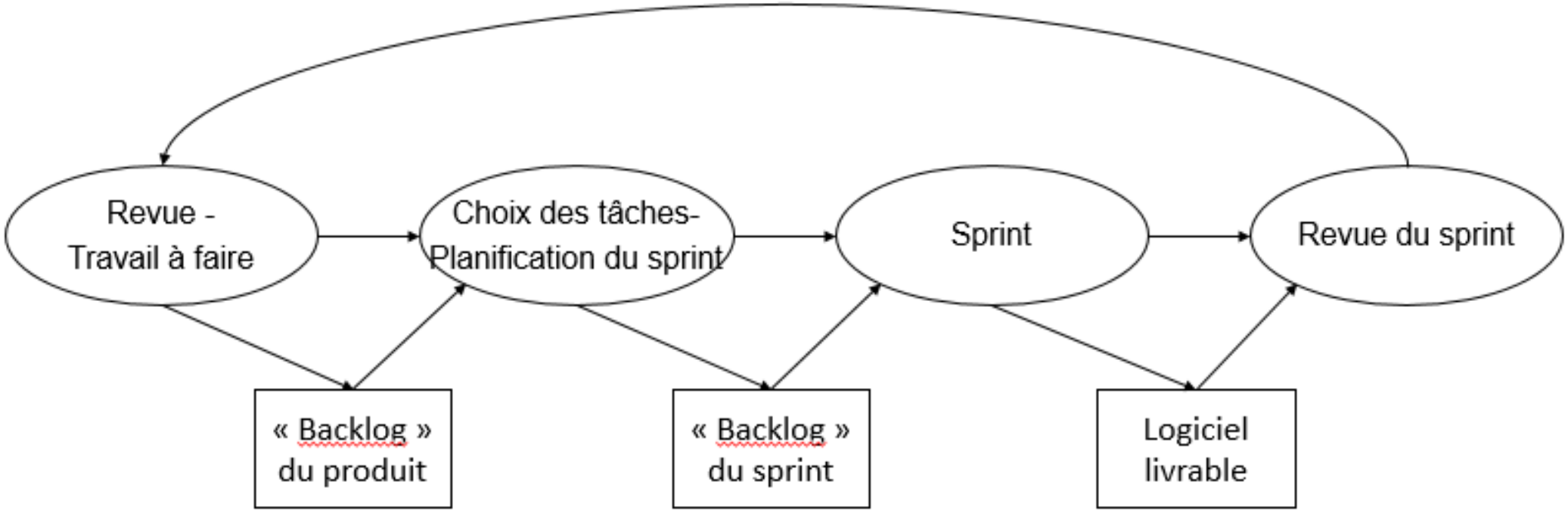
Méthode Scrum

- qui si
- n
- deux se

Méthode Scrum

- Scrum est une méthode agile qui se concentre sur le gestion itérative de projet en exploitant les propriétés créatives des membres de l'équipe de développement.
- On peut distinguer trois phases :
  - ✓ La phase initiale au cours de laquelle les fonctionnalités du système sont listés et une architecture logicielle générale est définie
  - ✓ Suit une série de "sprints", chaque sprint correspondant à un incrément du système
  - ✓ La phase de terminaison du projet développe les derniers artefacts (manuel d'utilisation …) et tire les leçons apprises durant le développement.

Cycle de vie d'un sprint SCRUM



Terminologie Scrum

- Release : Version d'un produit/projet, composée de 1 à N Sprint.
- Sprint Planning : Cérémonial de planification d'un Sprint.
- Sprint : Période de réalisation d'un ensemble de fonctionnalités (Story), sur une durée fixe (Itération).
- Daily Meeting : Réunion quotidienne d'équipe, à heure fixe et d'une durée fixe de 15 min maximum.
- Sprint Review : Cérémonial de démonstration des Stories réalisées durant un Sprint.
- Sprint Retrospective : Cérémonial de bilan d'un Sprint, avec pour objectif d'identifier les axes d'amélioration à mettre en œuvre sur les sprints suivants.

Terminologie Scrum

- Product Owner : Responsable d'une Release et du Product Backlog.
- Scrum Master: Facilitateur de l'équipe de développement et gardien du processus.
- StakeHolder : Tout acteur, externe à l'équipe scrum, partie prenante du projet
- Product Backlog : Liste des fonctionnalités, activités techniques et anomalies à traiter durant une Release, priorisée.
- Sprint Backlog : Ensemble des activités à réaliser durant un Sprint.
- Product Burndown Chart : Graphique de suivi de l'avancement d'une Release.
- Sprint Burndown Chart : Graphique de suivi de l'avancement d'un Sprint.
- Velocite : Mesure de la valeur réalisée durant un Sprint ou une Release.

Terminologie Scrum

- Définition of Ready : Check-list d'éligibilité d'une Story à être réalisée durant un Sprint.
- Definition of Done : Check-list de validation d'une Story, d'une fin de Sprint ou d'une fin de Release.
- Feature : Élément fonctionnel micro à réaliser durant un Sprint.
- Technical Story : Élément technique micro, non lié à une fonctionnalité, à réaliser durant un Sprint.
- Defect Story : Anomalie (dette technique) détectée sur une Story.