Gestion des Risques et des Rendus

Equipe GL 13

Gestion des Risques

Identification des risques

- **Risque technique** : Peu d'implémentation
- **Risque organisationnel** : Retards dans la planification et mauvaise répartition des tâches au sein de l'équipe.
- **Risque de qualité** : Bonne couverture de tests pour l'instant
- **Risque lié aux outils** : Problèmes avec Maven, Git encore chez certains mais ça sera réglé sous peu.

Évaluation des risques

Chaque risque sera évalué selon deux critères :

- **Probabilité** : Faible, moyenne ou élevée.
- **Impact** : Faible (peu de conséquences), moyen (ralentit le projet), élevé (compromet l'objectif final).

Stratégies pour la suite

- **Risque technique** :
  - Écrire des tests unitaires dès la phase de développement.
  - Utiliser une intégration continue pour détecter les régressions.
- **Risque organisationnel** :
  - Suivre le planning prévisionnel régulièrement et ajuster en cas de retard.
  - Utiliser l'outil Notion pour suivre l'avancement des tâches.
- **Risque de qualité** :
  - Maintenir une base de tests complète et organisée (Valide, Invalide, etc.).
  - Utiliser Jacoco pour mesurer la couverture de tests.
- **Risque lié aux outils** :
  - Documenter les processus pour utiliser Maven, Git et les scripts.
  - Prévoir des alternatives si un outil pose problème.

Plan d'urgence

En cas de concrétisation d'un risque :

- Réattribuer des tâches en cas de retard.
- Réduire l'ambition des extensions pour garantir la qualité du compilateur de base.
- Garantir une version fonctionnelle du compilateur à tout moment.

Gestion des Rendus

Organisation des livrables

- **Rendu intermédiaire** : Compiler les éléments du compilateur sans-objet (analyse syntaxique, vérifications contextuelles, génération de code).
- **Rendu final** : Compilateur objet et extension, les documentations, et les scripts de validation.
- **Documentation** : Fournir un manuel utilisateur clair et complet pour le compilateur.

Procédures de tests avant les rendus

- Bons scripts de tests, automatisés et conformes à toutes les attentes du compilateur.
- Exécuter le script `common-tests.sh` pour s'assurer que le compilateur répond aux spécifications minimales.
- Compléter avec une base de tests personnelle pour couvrir les cas d'usage complexes.
- Organiser les tests dans les répertoires spécifiés (Valide, Invalide, etc.) pour faciliter les évaluations.
- rajouter une pipeline Git qui lance tous les tests à chaque merge sur la branche `main`, pour s'assurer de la qualité constante de la version présentable du compilateur.

Contrôles de qualité

- Vérifier la conformité du compilateur avec les spécifications en ligne de commande (`decac -p`, `decac -v`).
- Vérifier l'interface et les messages d'erreur pour qu'ils soient compréhensibles et informatifs.

Gestion des versions

- Maintenir une branche principale (`develop`) propre, avec des commits clairs en anglaus et significatifs.
- Utiliser GitFlow, avec les branches `feature/*` pour chaque incrément.
- Versionnement sur la branche `main`, sur laquelle `develop` est merge à chaque fois qu'il y a une bonne évolution du compilateur.

Processus d'intégration continue

- Surveiller et résoudre immédiatement les régressions.

Anticipation des pénalités

- Prévoir une marge de correction dans le planning pour éviter de manquer les tests obligatoires.
- Un compilateur qui échoue sur les tests obligatoires entraînera une pénalité de 3 points sur la note finale.