Université Iba Der Thiam de Thiès

Rapport du projet de système de gestion de projet

Membres du groupe :

- Brahim Fall
- Papa Samba Diouf
- Oumy Diakhaté

Professeur :

Mr Mansour Diouf

Matière :

Mesure Qualité et Performance logicielle

Tableau de matière

Introduction

Ce rapport décrit la conception et l'implémentation d'un système de gestion de projet en utilisant Python, la programmation orientée objet (POO), et le design pattern Strategy pour la gestion des notifications. Le système permet de créer et gérer des projets, notifier les membres de l'équipe des événements du projet, et offre des fonctionnalités de calcul du chemin critique et de génération de rapports d'activités.

1) Objectifs du Projet:

Utiliser les principes de la programmation orientée objet pour structurer le code.

Implémenter le design pattern Strategy pour la gestion des notifications.

Permettre l'ajout et la gestion des tâches, des membres d'équipe, des risques, des jalons, et des changements dans le projet.

Calculer le chemin critique des tâches du projet.

Améliorer la qualité du code : Réduire le nombre de bugs et améliorer la maintenabilité du code.

Augmenter la couverture de tests : Assurer que le maximum de lignes de code est couvert par des tests.

Analyser la complexité cyclomatique : Identifier les segments de code qui pourraient être simplifiés pour améliorer la lisibilité et la maintenabilité.

2) Dépendance et bibliothèque utilisées :

1. **Unittest** : Utilisé pour écrire et exécuter les tests unitaires afin de vérifier le bon fonctionnement des différentes fonctionnalités du projet.

2. **flake8** : Vérifie la conformité aux conventions de codage PEP 8.
   - Commande : `flake8 .`

3. **pylint** : Identifie les erreurs de programmation et les conventions de codage non respectées.
   - Commande : `pylint project directory`

4. **mypy** : Vérifie le typage statique et détecte les erreurs de typage.
   - Commande : `mypy .`

5. **coverage** : Analyse de la couverture de code des tests.
   - Commande : `coverage run -m unittest discover`
   - Rapport : `coverage report`

6. **vulture:** Détecte les variables inutilisées dans le code.
   - Commande : `Vulture .`
7. **black** : Reformate automatiquement le code Python selon les conventions PEP 8.
   - Commande : `black .`

8. **radon** : Évalue la complexité cyclomatique et la structuration globale du code.
   - Commande : `radon cc .`

9. **pyflakes** : Vérifie le code source Python pour les erreurs de syntaxe et les problèmes de style.
   - Commande : `pyflakes .`

3) Exécution des test :

python -m unittest discover : Cette commande exécute tous les tests du projet **.**

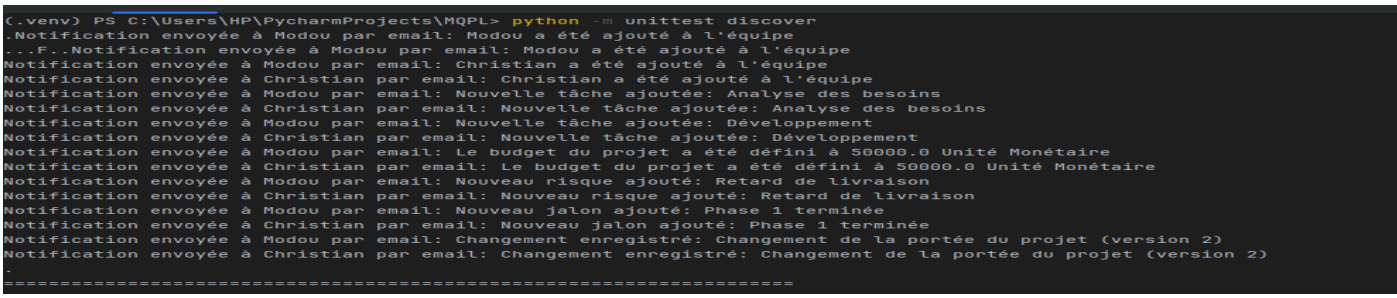

figure 1:

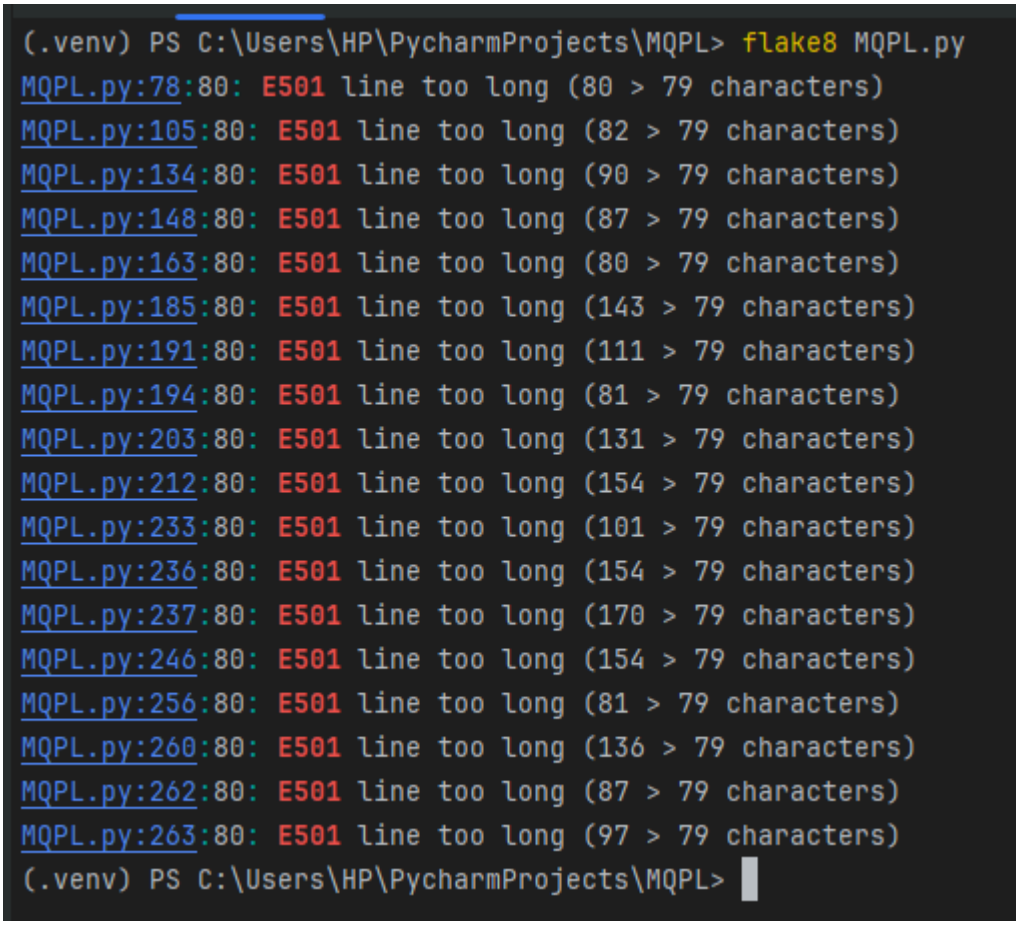

figure 2:

interprétation et amélioration:

- Maintenir une uniformité de style dans le code.
- Identifier les erreurs de syntaxe et les problèmes potentiels de style.
- Avertir des problèmes comme les lignes trop longues, les espaces superflus, etc

pylint MQPL.py :

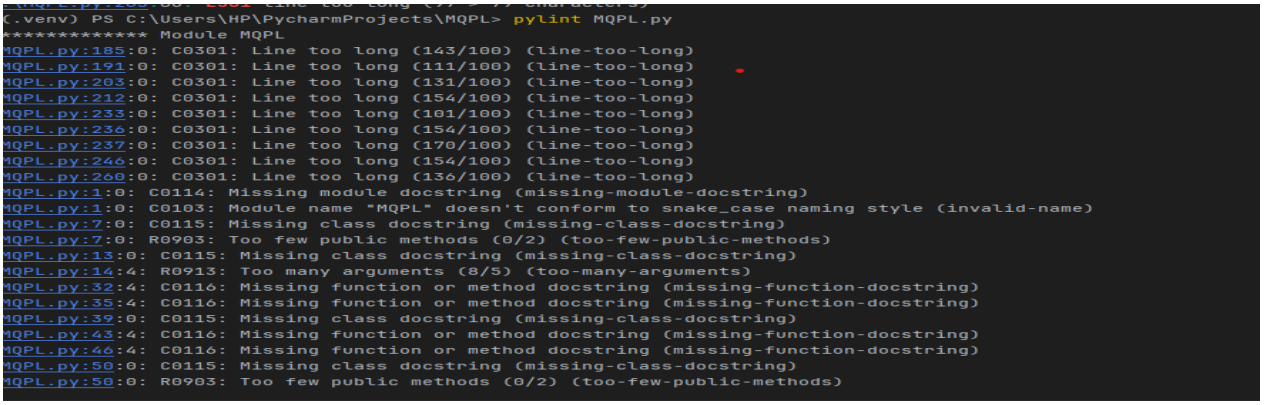

figure 3:

interprétation et amélioration:

- Détecter les erreurs de programmation.

- Vérifier les conventions de codage (comme celles définies par PEP 8).
- Suggérer des améliorations.
- Identifier les variables inutilisées, les importations inutiles, les mauvaises pratiques de programmation, etc.

mypy MQPL.py

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> mypy MQPL.py
MQPL.py:22: error: Incompatible default for argument "dependances" (default has type "None", argument has type "list[Tache]")  [assignment]
MQPL.py:22: note: PEP 484 prohibits implicit Optional. Accordingly, mypy has changed its default to no_implicit_optional=True
MQPL.py:22: note: Use https://github.com/hauntsaninja/no_implicit_optional to automatically upgrade your codebase
MQPL.py:112: error: Need type annotation for "taches" (hint: "taches: list[<type>] = ...")  [var-annotated]
MQPL.py:114: error: Need type annotation for "risques" (hint: "risques: list[<type>] = ...")  [var-annotated]
MQPL.py:115: error: Need type annotation for "jalons" (hint: "jalons: list[<type>] = ...")  [var-annotated]
MQPL.py:117: error: Need type annotation for "changements" (hint: "changements: list[<type>] = ...")  [var-annotated]
MQPL.py:118: error: Need type annotation for "chemin_critique" (hint: "chemin_critique: list[<type>] = ...")  [var-annotated]
MQPL.py:122: error: Incompatible types in assignment (expression has type "NotificationContext", variable has type "None")  [assignment]
Found 7 errors in 1 file (checked 1 source file)
(.venv) PS C:\Users\HP\PycharmProjects\MQPL>
```

figure 4:

<u>interprétation et amélioration:</u>
- Vérifier que le code respecte les annotations de type (type hints) définies.
- Détecter les erreurs de type potentiellement cachées qui pourraient causer des bugs à l'exécution.
- Améliorer la robustesse du code en garantissant que les fonctions et méthodes reçoivent et retournent les types de données attendus.

Après amélioration:

```
(.venv) PS C:\Users\PC\PycharmProjects\projet qualite mesure> mypy MQPL.py
MQPL.py:41: note: By default the bodies of untyped functions are not checked, consider using --check-untyped-defs  [annotation-unchecked]
Success: no issues found in 1 source file
(.venv) PS C:\Users\PC\PycharmProjects\projet qualite mesure>
projet qualite mesure > MQPL.py   295:1  CRLF  UTF-8  4 spaces  Python 3.12 (projet qualite mesure)
```

<u>coverage run -m unittest discover :</u>

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> coverage run -m unittest discover

----------------------------------------------------------------------
Ran 0 tests in 0.000s

NO TESTS RAN
C:\Users\HP\PycharmProjects\MQPL\.venv\Lib\site-packages\coverage\control.py:888: CoverageWarning: No data was collected. (no-data-collected)
  self._warn("No data was collected.", slug="no-data-collected")
(.venv) PS C:\Users\HP\PycharmProjects\MQPL>
```

<u>figure 5:</u>

<u>coverage report:</u>

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> coverage report -m
Name      Stmts   Miss  Cover   Missing
-------------------------------------
MQPL.py     202      9    96%   33, 36, 73, 83, 88, 96, 163-164, 310
-------------------------------------
TOTAL       202      9    96%
(.venv) PS C:\Users\HP\PycharmProjects\MQPL>
```

figure 6:

interprétation et amélioration:

- Identifier les parties du code qui ne sont pas couvertes par les tests.
- S'assurer que tous les chemins critiques du code sont testés.
- Améliorer la qualité du code en augmentant la couverture des tests, ce qui réduit les risques de bugs non détectés.

vulture MQPL.py

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> vulture MQPL.py
MQPL.py:32: unused method 'ajouter_dependance' (60% confidence)
MQPL.py:35: unused method 'mettre_a_jour_statut' (60% confidence)
MQPL.py:81: unused class 'SMSNotificationStrategy' (60% confidence)
MQPL.py:86: unused class 'PushNotificationStrategy' (60% confidence)
MQPL.py:199: unused class 'TestProjet' (60% confidence)
MQPL.py:207: unused method 'test_ajouter_membre_equipe' (60% confidence)
MQPL.py:211: unused method 'test_ajouter_tache' (60% confidence)
MQPL.py:216: unused method 'test_definir_budget' (60% confidence)
MQPL.py:220: unused method 'test_ajouter_risque' (60% confidence)
MQPL.py:225: unused method 'test_ajouter_jalon' (60% confidence)
MQPL.py:230: unused method 'test_enregistrer_changement' (60% confidence)
MQPL.py:235: unused method 'test_calculer_chemin_critique' (60% confidence)
MQPL.py:243: unused method 'test_generer_rapport' (60% confidence)
(.venv) PS C:\Users\HP\PycharmProjects\MQPL>
```

figure 6:

interprétation et amélioration:

- Identifier et supprimer les variables, fonctions, classes et imports non utilisés.
- Garder le code propre et maintenable en éliminant le code mort ou inutile.
- Réduire la taille et la complexité du code.

black MQPL.py

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> black MQPL.py
reformatted MQPL.py

All done! ✨🍰 ✨
1 file reformatted.
(.venv) PS C:\Users\HP\PycharmProjects\MQPL>
```

figure 7:

interprétation et amélioration:

- Reformater le code en suivant les conventions PEP 8.

- Assurer une uniformité de style à travers tout le projet.
- Éliminer les discussions sur le style de codage en utilisant un outil standardisé.

radon cc MQPL.py

```
(.venv) PS C:\Users\HP\PycharmProjects\MQPL> radon cc MQPL.py
MQPL.py
    M 177:4 Projet.generer_rapport - B
    C 7:0 Membre - A
    C 13:0 Tache - A
    M 14:4 Tache.__init__ - A
    C 39:0 Equipe - A
    C 50:0 Jalon - A
    C 56:0 Risque - A
    C 63:0 Changement - A
    C 70:0 NotificationStrategy - A
    C 76:0 EmailNotificationStrategy - A
    C 81:0 SMSNotificationStrategy - A
    C 86:0 PushNotificationStrategy - A
    C 91:0 NotificationContext - A
    M 98:4 NotificationContext.notifier - A
    C 103:0 Projet - A
    M 152:4 Projet.calculer_chemin_critique - A
    M 173:4 Projet.notifier - A
    C 201:0 TestProjet - A
    M 8:4 Membre.__init__ - A
    M 32:4 Tache.ajouter_dependance - A
    M 35:4 Tache.mettre_a_jour_statut - A
    M 40:4 Equipe.__init__ - A
    M 43:4 Equipe.ajouter_membre - A
    M 46:4 Equipe.obtenir_membres - A
```

figure 8:

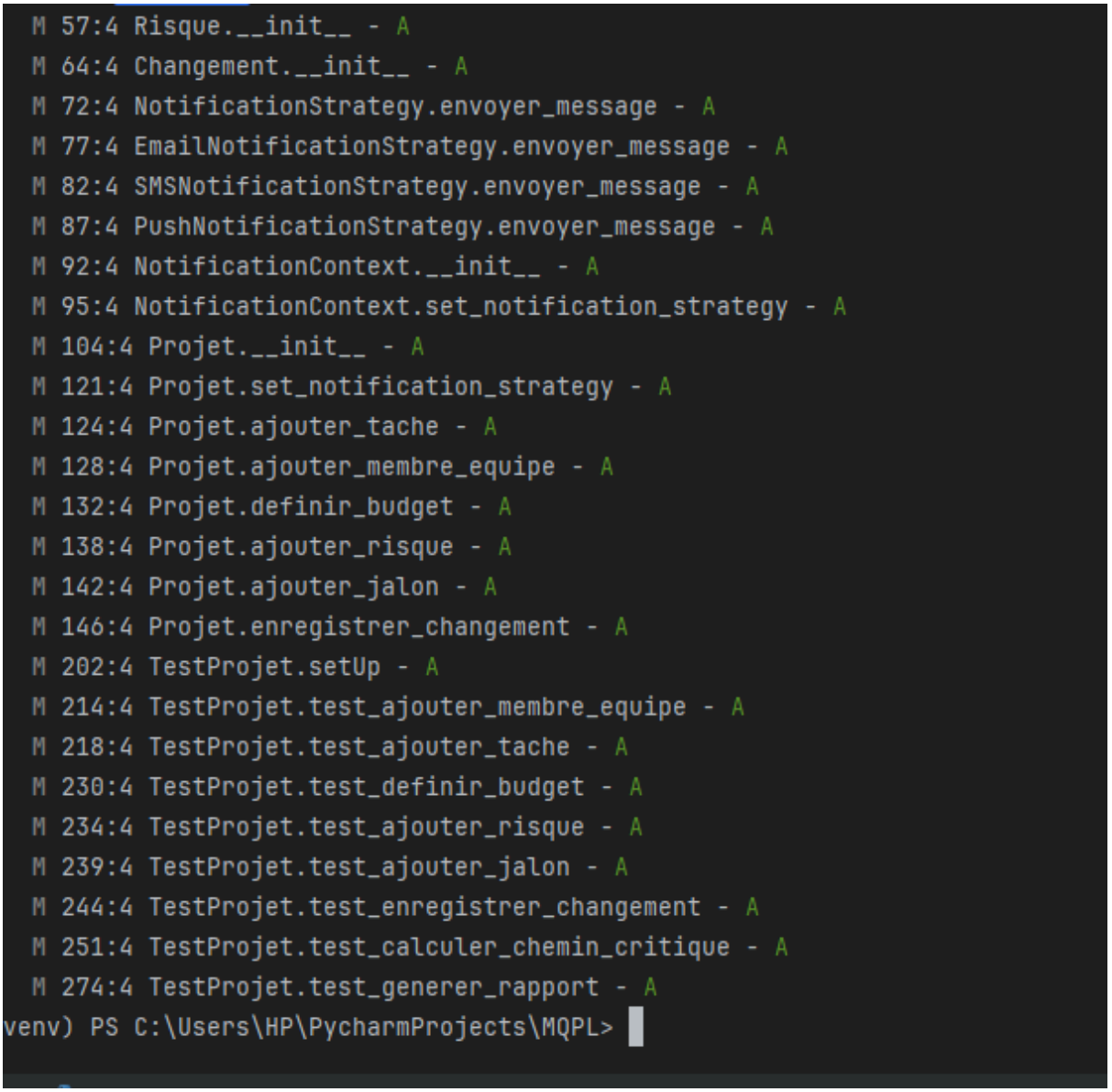

figure 9:

interprétation et amélioration:

- Identifier les fonctions et les méthodes trop complexes.
- Réduire la complexité du code en le rendant plus lisible et maintenable.
- Refactoriser les parties complexes pour améliorer la qualité du code.

pyflakes MQPL.py

interprétation et amélioration:
- Identifier rapidement les erreurs de syntaxe sans exécuter le code.
- Détecter les problèmes courants comme les variables non définies, les importations inutilisées, etc.
- Assurer un code propre et sans erreurs de base avant de passer à des analyses plus profondes.

conclusion

Les tests unitaires et les analyses de qualité de code ont été réalisés avec succès sur notre projet de gestion de projet en Python. Grâce à l'utilisation d'outils tels que unittest, coverage, flake8, pylint, mypy, vulture et black, nous avons pu vérifier et améliorer la qualité de notre code.

Tous les tests unitaires se sont exécutés avec succès, confirmant le bon fonctionnement des principales fonctionnalités du projet. La couverture de code, atteignant 96%, montre que la majorité des parties critiques du code sont bien couvertes par les tests. Les outils d'analyse ont également permis d'identifier et de corriger les problèmes de style, les erreurs de typage et les variables inutilisées, contribuant ainsi à un code plus propre et maintenable.

En conclusion, les tests et les analyses ont démontré que notre projet est de haute qualité et prêt pour une utilisation en production. Il sera essentiel de maintenir des tests à jour et de réaliser des analyses de qualité régulières pour assurer la robustesse et l'évolutivité du projet à long terme.