Travaux Pratiques 2
Analyse des Métriques Logiciel

Objectifs

- Calculer et analyser les métriques de qualité logicielle pour des programmes en C, Java et Python.
- Utiliser des échelles de mesure pour interpréter les métriques de manière cohérente.
- Apprendre à utiliser des outils spécifiques de mesure de métriques pour chaque langage et en interpréter les résultats.
- Comprendre et construire le graphe de flot de contrôle pour évaluer la complexité cyclomatique.

Outils de Mesure

- **C** : Cppcheck pour l'analyse statique et Lizard pour la complexité cyclomatique.
- **Java** : SonarQube pour une analyse complète incluant les métriques orientées objet.
- **Python** : Radon pour la complexité cyclomatique et les métriques de Halstead.

Étape 1 : Installation et Prise en Main des Outils

1. **Installation de Cppcheck et Lizard pour C** Installez **Cppcheck** pour obtenir des mesures de code, et **Lizard** pour calculer la complexité cyclomatique.
2. **Installation de SonarQube pour Java** Installez et configurez **SonarQube** pour analyser un projet Java. Cet outil permet d'obtenir les métriques de qualité et d'évaluer la complexité orientée objet.
3. **Installation de Radon pour Python** Installez **Radon** pour analyser la complexité cyclomatique et les métriques de Halstead dans le code Python.

Étape 2 : Programmes à Analyser

Programme en C

```
#include <stdio.h>
int somme(int a, int b) {
   return a + b;
}
int produit(int a, int b) {
   return a * b;
}
int main() {
   int x = 5, y = 10;
   printf("Somme : %d\n", somme(x, y));
   printf("Produit : %d\n", produit(x, y));
   return 0;
}
```

Programme en Java

```
public class Calculatrice {
   public int addition(int a, int b) {
```

```
        return a + b;
    }
    public int multiplication(int a, int b) {
        return a * b;
    }
    public static void main(String[] args) {
        Calculatrice calc = new Calculatrice();
        System.out.println("Addition : " + calc.addition(5, 10));
        System.out.println("Multiplication : " + calc.multiplication(5, 10));
    }
}
```

Programme en Python

```
def addition(a, b):
    return a + b
def multiplication(a, b):
    return a * b
if __name__ == "__main__":
    x, y = 5, 10
    print("Addition :", addition(x, y))
    print("Multiplication :", multiplication(x, y))
```

Étape 3 : Calcul des Métriques et Application des Échelles de Mesure

1. **Exécution des Outils de Mesure**
   - Exécutez **Cppcheck** et **Lizard** sur le programme C pour obtenir les métriques de complexité et les erreurs potentielles.
   - Exécutez **SonarQube** sur le programme Java pour obtenir des métriques orientées objet (WMC, DIT, CBO) et la complexité cyclomatique.
   - Exécutez **Radon** sur le programme Python pour les métriques de Halstead (taille, volume, difficulté) et la complexité cyclomatique.
2. **Interprétation des Échelles de Mesure**
   - **Nominale** : Utilisée pour classer qualitativement les erreurs trouvées (ex. erreurs syntaxiques, erreurs de logique).
   - **Ordinale** : Utilisée pour des mesures d'ordre, comme la criticité des erreurs (faible, modérée, élevée) et les niveaux de maintenabilité.
   - **Intervalle** : Appliquée aux métriques permettant d'identifier des écarts relatifs, comme la fréquence d'exécution d'une fonction.
   - **Ratio** : Appliquée aux mesures quantitatives telles que le nombre de lignes de code, la complexité cyclomatique, et le volume des métriques de Halstead.
3. **Interprétation des Résultats par Langage**
   - **C** : Analyser les résultats de Cppcheck et Lizard pour identifier la complexité cyclomatique. Utiliser une échelle de ratio pour interpréter le nombre de chemins d'exécution indépendants.
   - **Java** : Examiner les métriques de SonarQube (WMC, DIT, CBO) et interpréter leur impact sur la complexité orientée objet. Par exemple, un DIT élevé indique une hiérarchie profonde, ce qui peut complexifier la compréhension et la maintenance du code (échelle ordinale).

- **Python** : Interpréter les métriques de Halstead pour évaluer la difficulté de lecture et de modification du code. Le volume de Halstead peut être interprété avec une échelle de ratio pour quantifier l'effort requis.

Étape 4 : Transformation en Graphe de Flot de Contrôle et Calcul de la Complexité Cyclomatique

1. **Construction des Graphes de Flot de Contrôle**
   - Pour chaque programme, dessinez le graphe de flot de contrôle (CFG) en identifiant les points de décision (ex. instructions if, for, while).
   - **Questions de réflexion** :
     - Combien de nœuds et d'arcs comporte chaque graphe ?
     - Comment le graphe aide-t-il à visualiser le flux d'exécution et les chemins indépendants du programme ?
2. **Calcul de la Complexité Cyclomatique**
   - Calculer la complexité cyclomatique de McCabe de trois façons.
   - Comparez la complexité des trois programmes et discutez des implications pour la maintenabilité.

Étape 5 : Rapport d'Analyse et Recommandations

1. **Présentation des Résultats**
   - Organisez les métriques obtenues pour chaque langage, en y intégrant les valeurs de complexité cyclomatique obtenues à partir des graphes de flot de contrôle.
2. **Questions d'Analyse et de Réflexion**
   - Comment les métriques orientées objet (WMC, DIT, CBO) permettent-elles de repérer les classes complexes ou fortement couplées dans le programme Java ?
   - Quelle est l'influence d'une complexité cyclomatique élevée sur la maintenabilité et les tests ?
   - Comment le volume et la difficulté des métriques de Halstead en Python peuvent-ils indiquer si le code est trop complexe ?
3. **Recommandations**
   - Faites des suggestions d'amélioration en fonction des résultats obtenus.
   - Par exemple, proposez des refactorisations pour réduire la complexité cyclomatique ou le couplage excessif, pour améliorer la maintenabilité et la testabilité du code.