Maintenance applicative TD1 à 5

Objectifs :

- Rétroconception
- Evolution d'un code existant et de son architecture
- Refactoring et clean code
- Enrichissement des tests selon le TDD
- Enrichissement de la documentation

Projet 1 : contexte et cahier des charges

Dans une équipe de développement, on vous confie la maintenance et l'évolution d'un outil de gestion de tickets codé en Python. Dans toute la suite, un ticket est caractérisé par différents champs au format string :

- id : identifiant de tickets
- name : nom du client
- details : description du problème ou de la demande d'amélioration
- type : "PR" (Problem report) ou "IR" (Improvement request)
- date : date de création
- state : état du ticket parmi : "new","assigned","analysis","solved", "delivery","closed"
- owner: responsable du ticket parmi "L1", "L2", "L3"

Un stagiaire a produit une première version d'un outil qui gère un ticket, son état et ses informations. On vous transmet le projet en vous indiquant que le programme est complet mais vous vous rendez compte rapidement que :

1. des fonctionnalités sont manquantes
2. les tests unitaires sont absents
3. la gestion est locale (sans base de données)
4. la gestion est sous terminal (sans gui)
5. la documentation est inexistante

Vous devez comprendre le code de départ et adresser les 5 points précédents afin de produire un logiciel complet, testable, maintenable, en interaction avec une base de données et possédant une GUI. Les commentaires, le nom des fonctions et des variables sera produit en anglais.
Le travail à fournir est étalé sur 5 séances et il conduira à la création d'un programme en plusieurs fichiers que vous livrerez sur moodle pour évaluation.
Les étapes sont numérotées v1, v2, v3, v4 et v5 afin d'organiser vos séances.

v1 Maintenance corrective : fonctionnalités manquantes

Le fichier tms_v0.py est une base de départ. Elle regroupe 5 cas d'utilisation des tickets : création, mise à jour, fermeture, recherche de mot clés et affichage des informations.

Vous devez comprendre puis faire évoluer le code pour supporter les fonctionnalités manquantes :

1. A la création d'un ticket :
   L'utilisateur doit fournir le contenu des champs id, name, details et type
   Le champ type est toujours mis à PR alors que l'utilisateur devrait pouvoir saisir PR ou IR.
   Le format de l'attribut : id sera conservé même s'il n'est pas (encore) documenté. Les tests d'acceptation du format des données d'entrée sera conservé et enrichi pour que la valeur de type soit PR ou IR.
   Si le format est valide, le ticket sera créé dans le backlog sinon un message indiquant l'erreur de format sera affiché. Il n'est pas nécessaire de lever une exception.
   La création d'un ticket ne doit avoir lieu que si l'id du ticket n'existe pas encore dans le backlog sinon il faut retourner un message d'erreur.
2. A la mise à jour :
   L'utilisateur doit fournir le contenu des champs id, nouvel état, nouveau owner
   Dans le code de départ, il n'est pas possible de passer l'état d'un ticket à analysis, ni solved, ni in_delivery et de mettre à jour le champ owner. La fonction Assign a ticket devra évoluer en une mise à jour plus étendue.
   Des tests d'acceptation du format id, nouvel état et nouvel owner seront codés. Si le format est valide, le ticket sera cherché dans le backlog. S'il est trouvé, il sera modifié dans le backlog sinon un message indiquant l'erreur de format ou de ticket non trouvé sera affiché.
3. A la fermeture :
   La fermeture est possible pour tout responsable alors que le ticket ne peut être fermé que par le responsable L1. Un ticket fermé doit aller dans une autre liste qui sera appelée closed_backlog. Ces deux modifications seront codées.
4. Recherche de mots clés :
   La recherche peut être étendue sur tous les champs du ticket.

Les modifications demandées vous obligent à vous plonger dans un code écrit par un autre développeur (le fameux stagiaire) et à comprendre son organisation et ses fonctionnalités afin de les modifier. On parle de rétroconception.

v1 Maintenance préventive : TDD et tests unitaires

L'ajout ou la modification d'une fonctionnalité doit être couverte par un test dans l'esprit du TDD (méthode de codage adaptée aux projets de maintenance).
Il est attendu pour chacun de vos fichiers sources Python un fichier portant le même nom avec test_ devant. Celui-ci implémente les tests unittests ou pytest selon votre choix.

Dans l'approche avec unittest, vous devez créer une classe de test dérivée de unittest.TestCase avec des méthodes couvrant vos cas de tests. Vous pouvez également compléter une méthode créant des variables ou des objets indispensables aux tests en surchargeant la méthode setUp.
Exemple simple d'un fichier de test test_tms_v1.py

```
import unittest
import datetime
from tms_v1 import *

class TestTicket(unittest.TestCase):
    def setUp(self):
        self.backlog=[]
        self.template = {"id":"Case-001","name":"IUT","type":"PR","details":"texte","date":datetime.datetime.now()
    def test_creation1(self):
        self.assertEqual(create_ticket(self.template["id"],self.template["name"],self.template["details"],self.temp
    def test_creation2(self):
        with self.assertRaises(Exception):
            create_ticket("001",self.template["name"],self.template["details"],self.template["type"],self.backlog)


if __name__ == "__main__":
    unittest.main()
```

Une exécution synthétique : python nom_du_fichier_test.py

```
(base) C:\Users\thier\anacondacode\R606\v1>python -m unittest test_tms_v1.py
.........
----------------------------------------------------------------------
Ran 9 tests in 0.002s
```

Une exécution détaillée :

```
(base) C:\Users\thier\anacondacode\R606\v1>python -m unittest -v test_tms_v1.py
test_creation1 (test_tms_v1.TestTicket) ... ok
test_creation2 (test_tms_v1.TestTicket) ... ok
test_creation3 (test_tms_v1.TestTicket) ... ok
test_creation4 (test_tms_v1.TestTicket) ... ok
test_creation5 (test_tms_v1.TestTicket) ... ok
test_creation6 (test_tms_v1.TestTicket) ... ok
test_creation7 (test_tms_v1.TestTicket) ... ok
test_creation8 (test_tms_v1.TestTicket) ... ok
test_creation9 (test_tms_v1.TestTicket) ... ok

----------------------------------------------------------------------
Ran 9 tests in 0.003s

OK
```

Les noms des tests peuvent être plus explicites selon vos besoins.

v2 Maintenance corrective : architecture du code et organisation des fichiers

Le code est très linéaire et il manque de factorisation : les champs ne sont pas centralisés (ni les attributs d'un ticket ni les états possibles), le découpage fonctionnel est pauvre et de nombreuses tâches différentes sont mélangées au sein de chaque fonction dans la version 1 (version de départ corrigée et enrichie d'un fichier test).

Dans votre entreprise, on vous demande d'enrichir l'outil de gestion des cas afin de répondre aux points 3 et 4 du cahier des charges. La version 2 prépare donc le programme à supporter une interface console ou GUI ainsi que l'utilisation de données en ram ou en DB.

Le code fourni est procédural et vous avez le choix (ou non) de passer en POO.

Pour la version 2, le code de la v1 sera découpé en différentes fonctions et intégré dans le « bon » module. On distingue 3 briques principales : les données (en ram ou en DB), la logique du programme et l'interface (console ou gui) correspondant à 5 modules (procéduraux ou POO) et de les nommer :

- db : le backlog est représenté par des données en DB. Ce code gère les requêtes CRUD de la DB.
- backlog : le backlog est représenté par des listes en RAM et ce module gère les deux listes backlog et closed_backlog.
- tms : ce module gère les tickets, il fait l'intermédiaire entre l'interface utilisateur et le backlog
- interface : l'interface utilisateur est la console pour les commandes et les affichages, c'est à dire l'approche initiale de la v0 et de la v1
- gui : l'interface utilisée est une gui permettant à l'utilisateur d'indiquer ses commandes, leurs paramètres et d'afficher les données attendues

Vérifier que la v2 offre le même niveau de fonctionnalité que la v1. Adaptez les tests à la nouvelle structure de fichiers.

v3 Maintenance corrective : GUI

Créer une interface graphique permettant de supporter les fonctionnalités de la v2 et celle du cahier des charges.
La gui sera créée avec le module pysimplegui, qui est un véritable framework.
Une gui est un objet de type Window. Pour sa création, l'objet a besoin du placement des éléments dans la fenêtre (boutons, zone de texte, menus …) appelé layout.
Une fois créée, la gui est gérée par les événements. La méthode read de l'objet Window permet de remonter les événements utilisateurs et de réagir. Par défaut, read est bloquant mais un timeout peut être précisé si nécessaire.

Exemple de code :

Procédural

```
import PySimpleGUI as sg
def gui_function():
    layout = [ [sg.Text('My layout')],
       [sg.Input(key='-IN-')],
     [sg.Button('Go'), sg.Button('Exit')] ]

     window = sg.Window('My new window',
layout)

    while True:      # Event Loop
      event, values = window.read()
      if event in (sg.WIN_CLOSED, 'Exit'):
         break

      if event == 'Go':
       sg.popup('Go button clicked', 'Input
value:', values['-IN-'])

    window.close()

gui_function()
```

Objet

```
import PySimpleGUI as sg
class SampleGUI():
    def __init__(self):
   self.layout = [ [sg.Text('My layout')],
   [sg.Input(key='-IN-')],
   [sg.Button('Go'), sg.Button('Exit')] ]
      self.window = sg.Window('My new
window', self.layout)

    def run(self):
       while True:  # Event Loop
          self.event,self.values
               = self.window.read()
        if self.event in (sg.WIN_CLOSED,
'Exit'):
          break

        if self.event == 'Go':
           self.button_go()
           self.window.close()

    def button_go(self):
        sg.popup('Go button clicked', 'Input
value:', self.values['-IN-'])

# Create the class
my_gui = SampleGUI()
# run the event loop
my_gui.run()
```

En savoir plus

v3 Maintenance corrective : DB

La DB choisie est PostgreSQL qui fait partie des 5 DB les plus utilisées. PostgreSQL supporte la majorité des requêtes SQL du jeu standard. Un serveur PostgreSQL tourne sur la machine linserv-info-01 du réseau de l'IUT.

En Python, le module psycopg2 permet de se connecter et d'effectuer des requêtes CRUD sur la DB. Pour se connecter et créer la base de travail pour le module R606, merci de suivre la procédure indiquée dans le document « R606 bases PostgreSQL ».

Une fois la base de travail créée, vous devrez développer les fonctions du module db.py permettant de se connecter et d'échanger avec la base données. Pour cela, les objets connection et cursor sont incontournables. Ils sont expliqués dans la documentation en ligne du module psycopg. Les requêtes et réponses passent toutes par ces deux objets. En savoir plus

Exemple de code Python permettant de créer une table dans PostgreSQL :

```
import psycopg2

conn = psycopg2.connect(host="linserv-info-01",dbname="xxxxxxxxxxxx@campus.unice.fr",\
                        user="xxxxxxxxxxxx@campus.unice.fr",password="xxxxxxxxxxxx")
cur = conn.cursor()

requete = """CREATE TABLE backlog (id VARCHAR(32) PRIMARY KEY,name VARCHAR(32)); """
cur.execute(requete)
conn.commit()

cur.close()
conn.close()
```

Vous vérifierez par des tests que les objets attendus sont bien créés dans la base. Pour cela, vous pouvez utiliser un client postgreSQL (document « R606 bases PostgreSQL ») afin d'interroger la DB en parallèle de votre développement.

```
tvert_r606=> \d
              Liste des relations
 Schéma |      Nom       | Type  | Propriétaire
--------+----------------+-------+--------------
 public | backlog        | table | tvert
 public | closed_backlog | table | tvert
(2 lignes)

tvert_r606=> select * from backlog;
    id    | name | details | type |        date         | state  | owner
----------+------+---------+------+---------------------+--------+-------
 Case-001 | a    | b       | IR   | 2024-01-24 09:00:06 | solved | L3
(1 ligne)

tvert_r606=>
```

v4 Maintenance préventive : qualité et documentation

Le projet a bien avancé et les fonctionnalités attendues sont développées. Il est temps de revenir sur la qualité logicielle en ajoutant les fichiers de tests correspondant aux différents fichiers source.

Dans votre code, les noms des fonctions et variables, attributs et méthodes seront explicites. Vous ajouterez également des commentaires et des docstrings pour fournir un code documenté en anglais.

Les docstrings seront enrichis avec les mots clés du cours afin de générer le manuel technique sous forme de page web. Un outil de génération de documentation, Sphinx, est utilisé. Vous utiliserez la procédure de démarrage rapide ci-dessous. Vous pouvez enrichir cette procédure notamment à l'aide de css pour les pages web. En savoir plus.

Procédure de démarrage rapide de sphinx :
Dans le terminal de votre interpréteur Python :

- installer sphinx avec pip install sphinx
- créer un répertoire docs dans le répertoire de travail
- aller dans ce répertoire et taper la commande : sphinx-quickstart
  renseigner les informations demander
- ouvrir le fichier conf.py puis
  modifier la ligne extensions :

```
extensions = ["sphinx.ext.autodoc","sphinx.ext.viewcode","sphinx.ext.napoleon"]
```

  configurer le chemin vers les fichiers source en ajoutant par exemple :

```
# -- Path setup --------------------------
import sys
import os
sys.path.insert(0,os.path.abspath("../src"))
```

- revenir dans le répertoire de travail et taper : sphinx-apidoc -o docs src ce qui génère les fichiers rst
- modifier index.rst en ajoutant modules dans son contenu :

```
.. toctree::
   :maxdepth: 2
   :caption: Contents:

   modules
```

- retourner dans le répertoire docs puis taper : make html

v5 Maintenance corrective : les métriques

La version 4 du logiciel offre un bon niveau de qualité logicielle : les 5 points du cahier des charges sont remplies. Le logiciel est complet, documenté et testé.

Il reste une dernière fonctionnalité à ajouter dans le cadre de la maintenance corrective. Les équipes du support logiciel sont pilotées par des métriques, exposées en cours : TAR-3, TAR-10 et TAR-20.

Un bouton métrique sera ajouté à l'interface gui ou dans le menu de la console console.
Il permettra de parcourir le backlog des tickets et d'afficher le nombre et le numéro des tickets critiques selon 3 catégories :

- les tickets en état new de plus de 3 jours
- les tickets en état assigned ou analysis de plus de 10 jours
- les tickets dans n'importe quel état de plus de 20 jours

Il permettra de parcourir le closed_backlog des tickets et d'afficher le pourcentage de tickets fermés sous 3 jours, entre 3 et 10 jours et plus de 10 jours. Un ticket fermé sera compté une seule fois dans la meilleure catégorie [0,3jours[ou [3 jours , 10 jours [ou plus de 10 jours.

Les informations calculées seront affichées mais ne seront pas stockées.

Vous n'oublierez pas de mettre à jour vos tests et de vérifier qu'aucune erreur n'apparaît sur le code existant. Vous mettrez également à jour la documentation générée par Sphinx.