LOG3430 – Méthodes de test et de validation du logiciel

Automne 2021

Laboratoire 1

Groupe 3

1929407 – Nerese, Carlens

2164768 – Thomas, Julien

25 septembre 2021

Partie 1

Couverture sans tests customs.

```
----------------------------------------------------------------------
Ran 34 tests in 0.071s

OK
Name                    Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
crud.py                   195     46     96     11    71%
email_analyzer.py          46      0     14      0   100%
vocabulary_creator.py      69      0     26      0   100%
-------------------------------------------------------------
TOTAL                     310     46    136     11    81%
```

Couverture après l'ajout de tests

```
$ sh run.sh
.....................................................
----------------------------------------------------------------------
Ran 56 tests in 0.059s

OK
Creating vocabulary
Email 2/2

Name                    Stmts   Miss Branch BrPart  Cover
-------------------------------------------------------------
crud.py                   195      0     96      2    99%
email_analyzer.py          46      0     14      0   100%
vocabulary_creator.py      69      0     26      0   100%
-------------------------------------------------------------
TOTAL                     310      0    136      2    99%
```

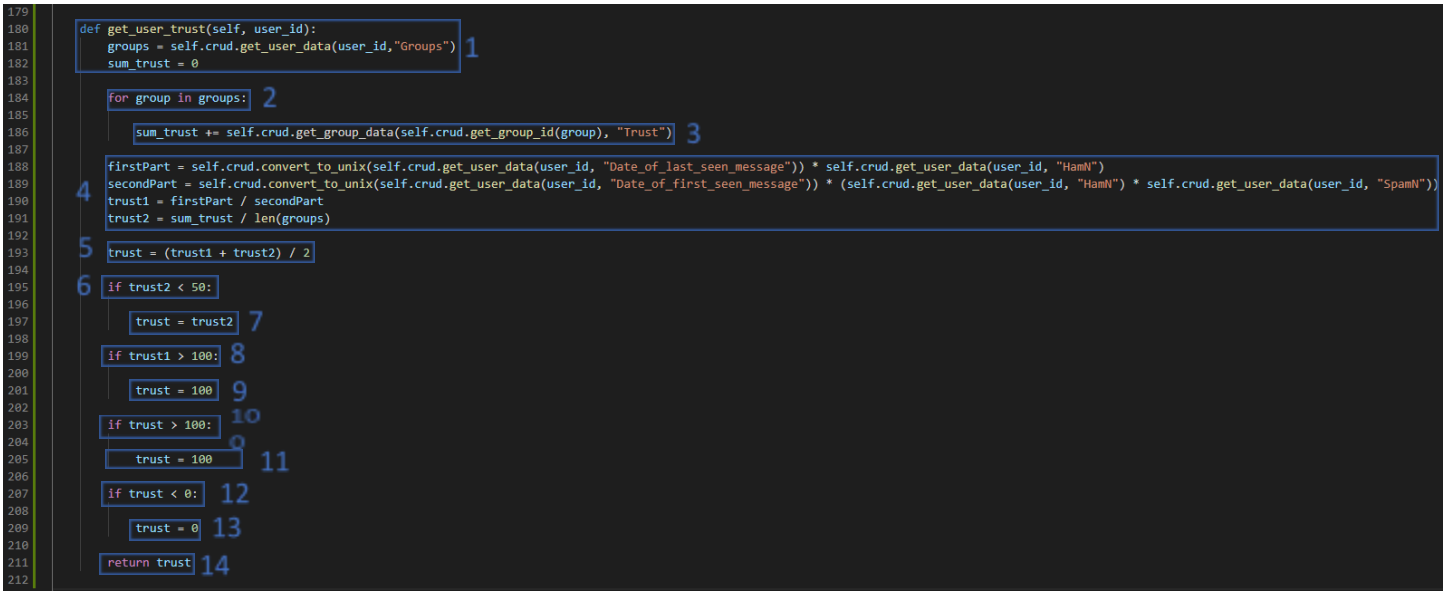
Les principaux tests effectués sont des tests de crud, avec notamment la modification d'utilisateur ou de groupe en modifiant chaque propriété du modèle group et user, et en lui donnant une bonne valeure et une mauvaise en payload. Cela permet d'obtenir une couverture proche de 100% De plus, les fonctions mockées, qui ont pour simple but de lire un fichier ou de dumper les données dans un fichier, n'ont pas été prises en compte dans le calcul de la couverture de code.

De plus, les fixtures utilisées sont telles qu'on obtient chaque cas particuliers pour les classes EmailAnalyzer (un email spam, un email non spam, des mots répétés…).

Tout cela est renseigné dans le code.

Partie 2

Figure 1 : Capture d'écran de la méthode demandée



```
def get_user_trust(self, user_id):
    groups = self.crud.get_user_data(user_id,"Groups")
    sum_trust = 0

    for group in groups:

        sum_trust += self.crud.get_group_data(self.crud.get_group_id(group), "Trust")

    firstPart = self.crud.convert_to_unix(self.crud.get_user_data(user_id, "Date_of_last_seen_message")) * self.crud.get_user_data(user_id, "HamN")
    secondPart = self.crud.convert_to_unix(self.crud.get_user_data(user_id, "Date_of_first_seen_message")) * (self.crud.get_user_data(user_id, "HamN") * self.crud.get_user_data(user_id, "SpamN"))
    trust1 = firstPart / secondPart
    trust2 = sum_trust / len(groups)

    trust = (trust1 + trust2) / 2

    if trust2 < 50:

        trust = trust2

    if trust1 > 100:

        trust = 100

    if trust > 100:

        trust = 100

    if trust < 0:

        trust = 0

    return trust
```

Note :

Dans le bloc 4, à la ligne de secondPart, on a HamN + SpamN et non HamN * SpamN. Cette erreur n'est pas présente dans le code.

Graphe 1 : Graphe CFG

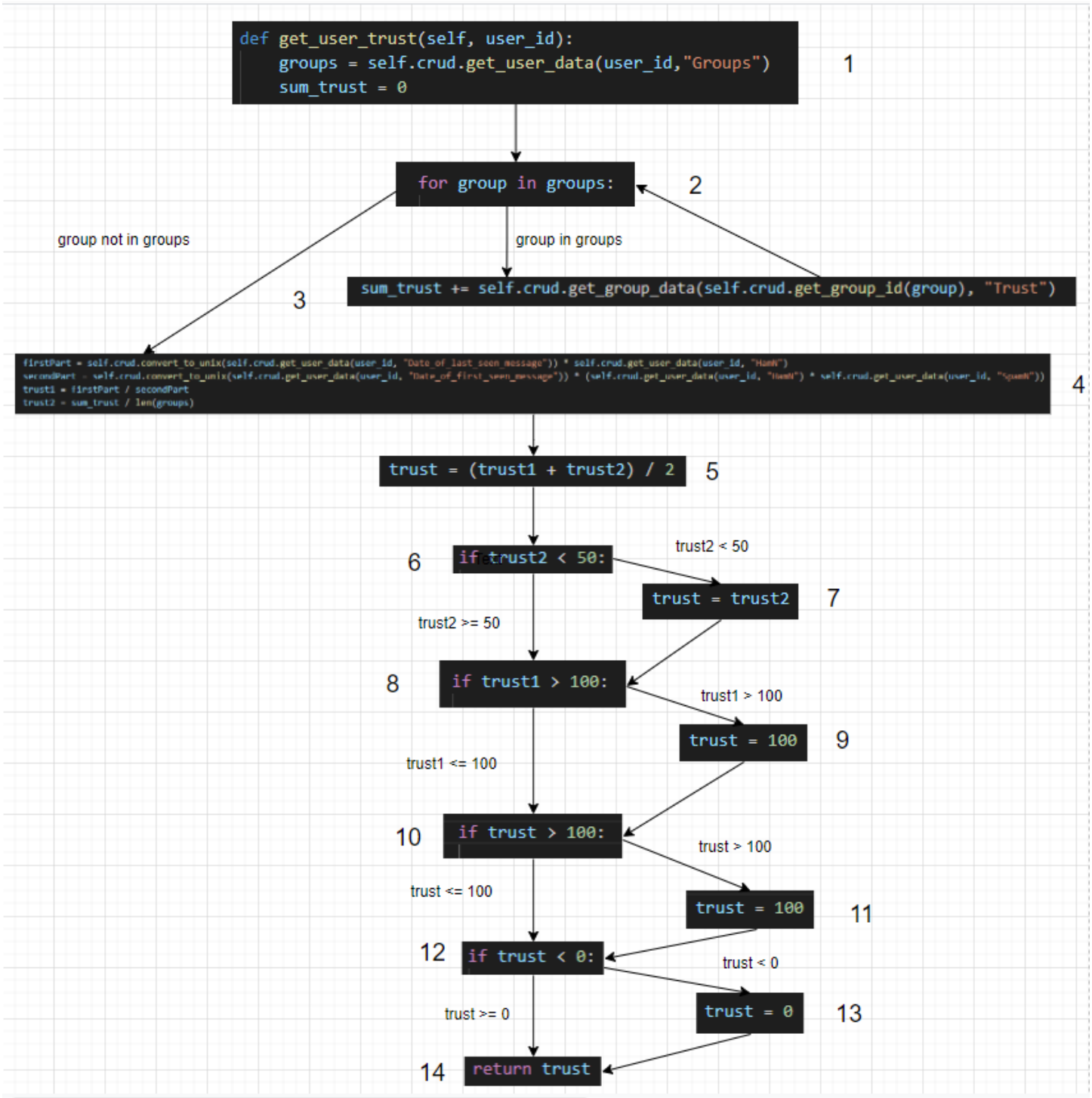


Tableau 1

Nœud	DEF	C-USE	P-USE
user_id	1	1, 4	
groups	1	4	2
sum_trust	1, 3	3	
group	2	3	2
firstPart	4	4	
secondPart	4	4	
trust1	4	5	8
trust2	4	5	6
trust	5, 7, 9,11, 13	14	10, 12

All-Definition coverage : au moins un def-clear path pour chaque nœud de définition

DC-PATH (user_id, 1, 1) = {1}

DC-PATH (groups, 1, 2) = {1, 2}

DC-PATH (sum_trust, 1, 3) = {1,2,3}

DC-PATH (group, 2, 3) = {2, 3}

DC-PATH (group, 2, 2) = {2}

DC-PATH (first_part, 4,4) = {4}

DC-PATH (second_part, 4,4) = {4}

DC-PATH (trust1, 4,5) = {4,5}

DC-PATH (trust1, 4,8) = {4, 5, 6, 8}

DC-PATH (trust2, 4,5) = {4,5}

DC-PATH (trust2, 4,8) = {4, 5, 6}

DC-PATH (trust, 5,14) = {5, 6, 8, 10, 12, 14}

DC-PATH (trust, 5,10) = {5, 6, 8, 10}

DC-PATH (trust, 5,12) = {5, 6, 8, 10, 12}

Jeu de test:

PathA = {1, 2, 3, 2, 4, 5, 6, 8, 10, 12, 14} couvre tous les DC-PATHS.

d1 = <user_id, 1>, {return = 65}, on peut mock les fonctions pour retourner les valeurs que l'on retrouve dans l'image ci-dessous :

Ce cas de test nous permet de passer par tous les noeuds du PathA.

All C-USE coverage : au moins un chemin def-clear à partir de chaque nœud de définition et vers chaque nœud de C-Utilisation.

DC-PATH (user_id, 1,1) = {1}

DC-PATH (groups, 1,4) = {1, 2, 4}

DC-PATH (sum_trust, 1,3) = {1,2,3}

DC-PATH (sum_trust, 3, 3) = {3}

DC-PATH (group, 2,3) = {2,3}

DC-PATH (first_part, 4, 4) = {4}

DC-PATH (second_part, 4, 4) = {4}

DC-PATH (trust1, 4, 5) = {4, 5}

DC-PATH (trust2, 4, 5) = {4, 5}

DC-PATH (trust, 5, 14) = {5, 6,8,10,12,14}

DC-PATH (trust, 7, 14) = {7,8,10,12,14}

DC-PATH (trust, 9, 14) = {9,10,12,14}

DC-PATH (trust, 11, 14) = {11,12,14}

DC-PATH (trust, 13, 14) = {13,14}

Jeu de test:

PathA = {1, 2, 3, 2, 4, 5, 6, 7, 8, 9, 10, 12, 14} trust2 < 50, trust1 > 100

PathB = {1, 2, 3, 2, 4, 5, 6, 8, 10,11, 12, 14} trust > 100

PathC = {1, 2, 3, 2, 4, 5, 6, 8, 10, 12,13, 14} trust < 0

d1 = <user_id = 1>, {return = 95} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 95,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 300,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 40,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

d2 = <user_id = 1>, {return = 100} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 125,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 200,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 150,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

d3 = <user_id = 1>, {return = 0} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": -25,
        "SpamN": 3,
        "HamN": -1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 400,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 150,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

Ces trois tests nous permettent de parcourir les chemins A, B et C.

All P-USE coverage : au moins un chemin def-clear à partir de chaque nœud de définition et vers chaque nœud de P-Utilisation.

DC-PATH (groups, 1,2) = {1,2}

DC-PATH (group, 2,2) = {2}

DC-PATH (trust1, 4,8) = {4,5,6,8}

DC-PATH (trust2, 4,6) = {4,5,6}

DC-PATH (trust, 5,10) = {5,6,8,10}

DC-PATH (trust, 5,12) = {5,6,8,10, 12}

DC-PATH (trust, 7,10) = {7,8,10}

DC-PATH (trust, 7,12) = {7,8,10,12}

DC-PATH (trust, 9,10) = {9 ,10}

DC-PATH (trust, 9,12) = {9 ,10, 12}

DC-PATH (trust, 11,12) = {11 ,12}

Jeu de test:

PathA = {1,2,3,2,4,5,6,7,8,9,10,12, 14} trust2 < 50, trust1 > 100, trust = 0

PathB = {1,2,3,2,4,5,6,7,8,10,11,12, 14} trust2 = 0, trust1 < 100

d1 = <user_id = 1>, {return = 95} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 95,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 300,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 40,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

d2 = <user_id = 1>, {return = 100} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 125,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 200,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 150,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

Ces deux tests permettent de parcourir les chemins A et B.

All USE coverage : au moins un chemin def-clear à partir de chaque nœud de définition et vers chaque nœud d'utilisation.

DC-PATH (user_id, 1,1) = {1}

DC-PATH (user_id, 1,4) = {1,2,3,4}

DC-PATH (groups, 1,2) = {1,2}

DC-PATH (groups, 1,4) = {1,2,3,4}

DC-PATH (sum_trust, 1,3) = {1,2,3}

DC-PATH (sum_trust, 3,3) = {3}

DC-PATH (group, 2,2) = {2}

DC-PATH (group, 2,3) = {2,3}

DC-PATH (first_part, 4,4) = {4}

DC-PATH (second_part, 4,4) = {4}

DC-PATH (trust1, 4,5) = {4,5}

DC-PATH (trust1, 4,8) = {4,5,6,8}

DC-PATH (trust2, 4,5) = {4,5}

DC-PATH (trust2, 4,6) = {4,5,6}

DC-PATH (trust, 5, 14) = {5, 6,8,10,12,14}

DC-PATH (trust, 7, 14) = {7,8,10,12,14}

DC-PATH (trust, 9, 14) = {9,10,12,14}

DC-PATH (trust, 11, 14) = {11,12,14}

DC-PATH (trust, 13, 14) = {13,14}

DC-PATH (trust, 5,10) = {5,6,8,10}

DC-PATH (trust, 5,12) = {5,6,8,10, 12}

DC-PATH (trust, 7,10) = {7,8,10}

DC-PATH (trust, 7,12) = {7,8,10,12}

DC-PATH (trust, 9,10) = {9 ,10}

DC-PATH (trust, 9,12) = {9 ,10, 12}

DC-PATH (trust, 11,12) = {11 ,12}

Jeu de tests:

PathA = {1, 2, 3, 2, 4, 5, 6, 7, 8, 9, 10, 12, 14} trust2 < 50, trust1 > 100

PathB = {1, 2, 3, 2, 4, 5, 6, 8, 10,11, 12, 14} trust > 100

PathC = {1, 2, 3, 2, 4, 5, 6, 8, 10, 12,13, 14} trust < 0

d1 = <user_id = 1>, {return = 95} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 95,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 300,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 40,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

d2 = <user_id = 1>, {return = 100} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": 125,
        "SpamN": 1,
        "HamN": 1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 200,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 150,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

d3 = <user_id = 1>, {return = 0} lorsqu'on mock les méthodes pour qu'elles retournent les valeurs selon le code ci-dessous.

```
//test case all c-use
[
    {
        "name": "farmer@paris.com",
        "Trust": -25,
        "SpamN": 3,
        "HamN": -1,
        "date_of_first_seen_message": 1,
        "date_of_last_seen_message": 400,
        "Groups": [
            "default"
        ]
    },
    {
        "name": "default",
        "Trust": 150,
        "List_of_members": [
            "farmer@paris.com"
        ]
    }
]
```

Les critères ALL USE et ALL C-USE semblent être les plus strictes puisqu'ils parcourent tous les deux 3 chemins. Toutefois, puisque ALL USE comprend les P-USE ET LES C-USE, nous dirons que c'est le critère le plus strict.