Département de génie logiciel et des TI

Rapport de laboratoire

Étudiant	Furtado, Eduardo
Cours	MGL802, Principes & appl.de la conception de logiciels
Session	Été 2020
Code permanent	FURE28019100
Travail pratique	#1
Chargé de cours	Prof. Yvan Ross
Date	2020-06-14

Table des matières

1. Introduction

Ce rapport documente le travail pratique qui a pour objectif de mieux comprendre la conceptualisation des logiciels avec un problème réaliste. L'idée est générer un modèle de conception partiel à partir d'un projet réel qui s'appelle « Snu Photo Manager ».

Snu Photo Manager s'agit d'un gestionnaire et éditeur de photos riche en fonctionnalités, écrit en python et utilisant la bibliothèque Kivy. Une fork du projet a été fournie par le responsable du cours.

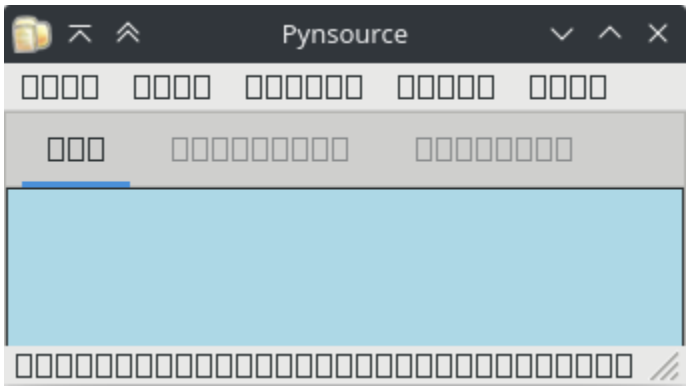
Le travail comprend l'analyse et la modélisation du projet avec des représentations UML, en utilisant des outils de développement et d'ingénierie inverse.

Une liste d'outils a également été fournie. Certains d'entre eux ont été immédiatement rejetés parce qu'ils n'étaient pas conçus pour des projets utilisant python (CppDepend, Ndepend, Sonargraph-Explorer). Ensuite, les outils de la liste ont été vérifiés, ce qui signifie que leur documentation a été inspectée. Une recherche sur DuckDuckGo a également permis de trouver quelques outils qui ont été testés [1]. Plus de détails sur les outils utilisés se trouvent dans les sections suivantes.

2. Diagramme de classes

Différents outils d'ingénierie inverse du projet en un diagramme de classe UML ont été testés :

- Umbrello - N'a pas du tout fonctionné lorsqu'il a essayé d'importer le code source du projet dans le programme.
- Pyreverse - Il a montré un diagramme avec des classes, mais sans aucune relation entre elles ou, en d'autres termes, sans aucune ligne reliant les classes.
- Epydoc - N'a pas fonctionné sur le projet. Il semble qu'il ne fonctionne qu'avec des fichiers python 2, mais aucune autre investigation n'a été faite.
- PyNSource[3] - Initialement, les menus n'ont pas été rendus correctement, comme le montre l'image ci-dessous :

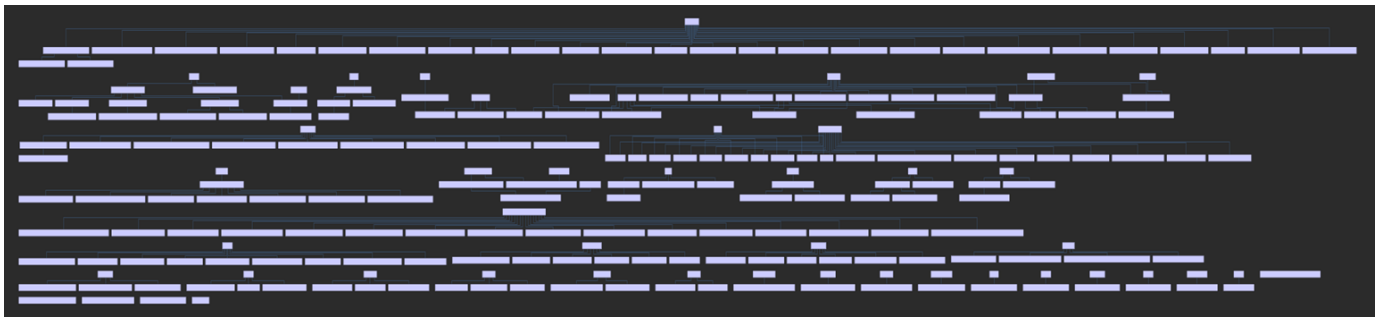


Aucune solution rapide n'a été trouvée lors de la recherche sur le web, et l'installation de polices système supplémentaires n'a pas non plus fonctionné.

Sous Windows 10 Enterprise LTSC, version 1809, le programme ne s'est pas ouvert du tout après l'installation, même après un redémarrage du système.

L'outil PyNSource a finalement fonctionné sans problème après avoir été compilé à partir du code source.

Un autre outil qui a fonctionné était un plugin appelé UML de l'IDE pycharm de JetBrains. Lorsqu'on l'utilise le plugin pour dessiner un diagramme de classes à partir de l'ensemble du projet, il donne le résultat suivant :



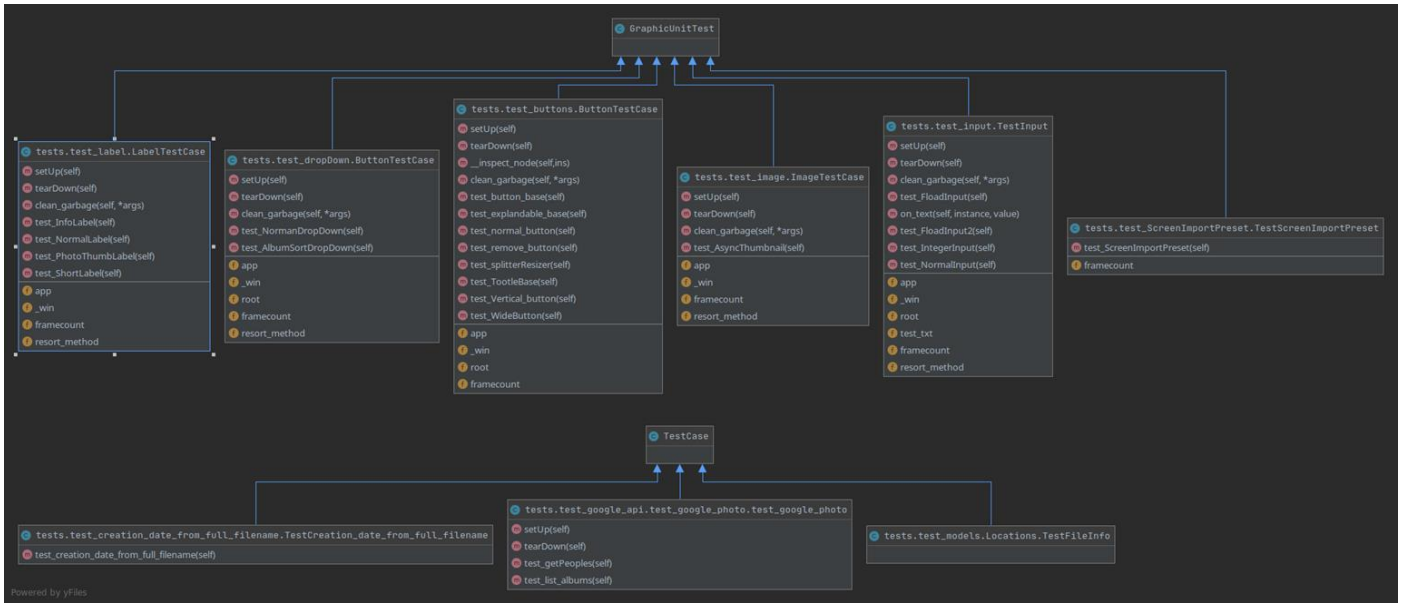
L'image montre la vue statique de l'ensemble du projet. Il est impossible de dire ce qui se trouve dans le diagramme, et la seule information que l'on peut en tirer est qu'il y a de nombreuses classes. C'est pourquoi il faut choisir un thème, qui puisse réellement rendre un diagramme représentant un aspect du logiciel étudié.

Lors de la sélection d'un thème, l'intention était de choisir celui qui était pertinent pour le logiciel. La documentation officielle a été consultée, essentiellement la liste des fonctionnalités présente dans le *Readme* du projet [2].

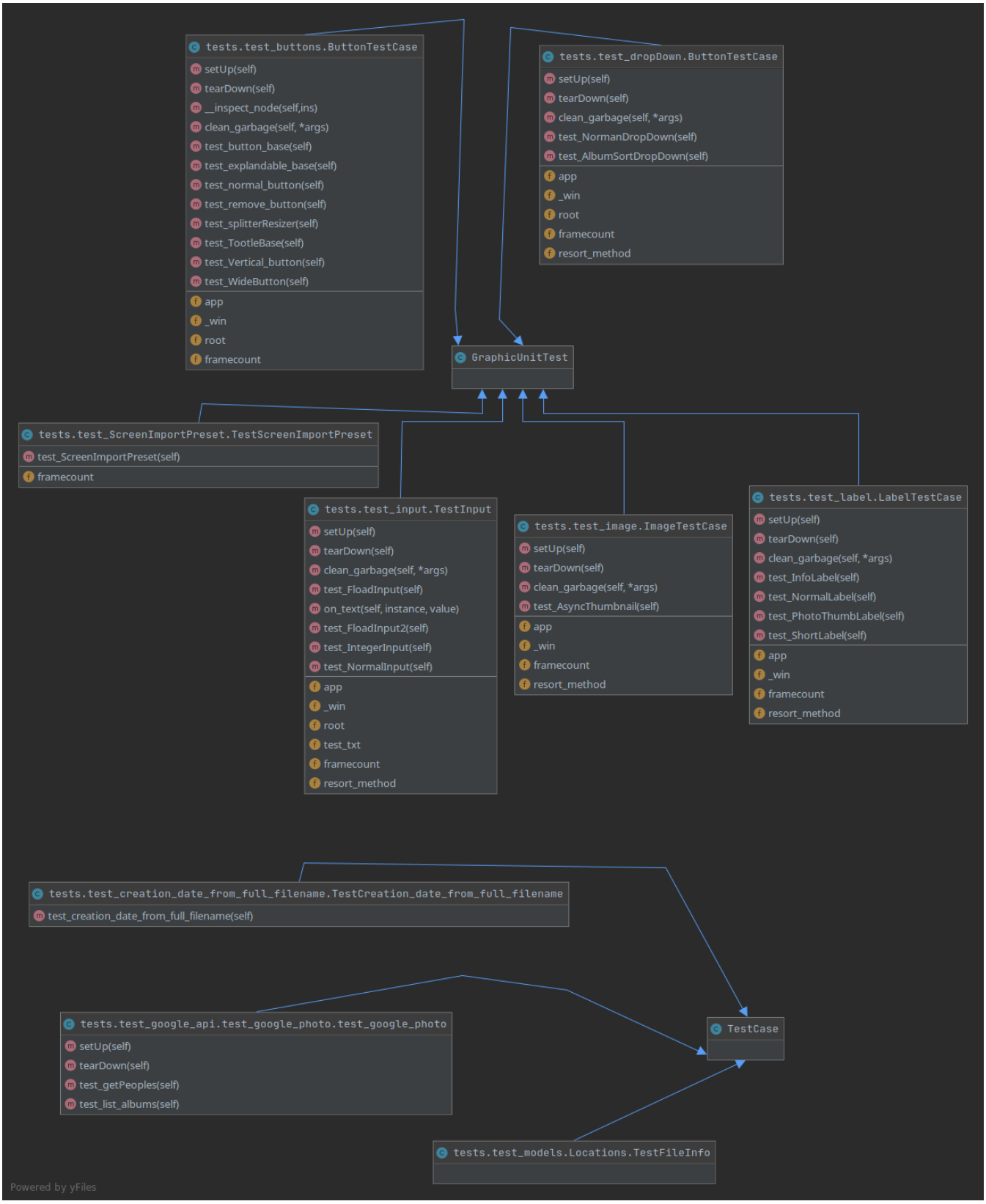
Mais les fonctionnalités peuvent ne pas être liées aux classes, ou même des modules, ainsi, en inspectant la structure des dossiers du projet, il a été remarqué qu'elle contenait un dossier pour les fichiers de test.

Les tests sont en fait du code, et ils font partie de tout logiciel de qualité. C'est donc le thème des tests a été choisi.

Lorsque le plugin est exécuté dans le dossier "tests/", le diagramme suivant s'affiche :



Il est difficile de voir le diagramme dans ce rapport, car il occupe trop d'espace horizontal. Il a donc été réorganisé pour occuper plus d'espace vertical :


tests.test_buttons.ButtonTestCase
setUp(self)
tearDown(self)
__inspect_node(self,ins)
clean_garbage(self, *args)
test_button_base(self)
test_explandable_base(self)
test_normal_button(self)
test_remove_button(self)
test_splitterResizer(self)
test_TootleBase(self)
test_Vertical_button(self)
test_WideButton(self)
app
_win
root
framecount
tests.test_dropDown.ButtonTestCase
setUp(self)
tearDown(self)
clean_garbage(self, *args)
test_NormanDropDown(self)
test_AlbumSortDropDown(self)
app
_win
root
framecount
resort_method
GraphicUnitTest
tests.test_ScreenImportPreset.TestScreenImportPreset
test_ScreenImportPreset(self)
framecount
tests.test_input.TestInput
setUp(self)
tearDown(self)
clean_garbage(self, *args)
test_FloadInput(self)
on_text(self, instance, value)
test_FloadInput2(self)
test_IntegerInput(self)
test_NormalInput(self)
app
_win
root
test_txt
framecount
resort_method
tests.test_image.ImageTestCase
setUp(self)
tearDown(self)
clean_garbage(self, *args)
test_AsyncThumbnail(self)
app
_win
framecount
resort_method
tests.test_label.LabelTestCase
setUp(self)
tearDown(self)
clean_garbage(self, *args)
test_InfoLabel(self)
test_NormalLabel(self)
test_PhotoThumbLabel(self)
test_ShortLabel(self)
app
_win
framecount
resort_method
tests.test_creation_date_from_full_filename.TestCreation_date_from_full_filename
test_creation_date_from_full_filename(self)
tests.test_google_api.test_google_photo.test_google_photo
setUp(self)
tearDown(self)
test_getPeoples(self)
test_list_albums(self)
TestCase
tests.test_models.Locations.TestFileInfo
Powered by yFiles

L'utilisation de cet outil pour réorganiser le diagramme était très frustrante, comme cela peut être déduit des lignes de l'image. Il semble que l'intention du plugin soit de permettre des représentations visuelles rapides à partir du code source sur lequel le travail est effectué dans l'IDE, et non pas vraiment d'être un outil d'édition complète pour les diagrammes.

À partir du diagramme, nous pouvons facilement voir la hiérarchie des classes dans le module de tests, comme il est précisé par des lignes de connexion se terminant par un triangle (généralisation).

La classe GraphicUnitTest est la classe de base des sous-classes :

- TestScreenImportPreset
- ImageTestCase
- LabelTestCase
- TestInput
- test_buttons.ButtonTestCase
- test_dropDown.ButtonTestCase

Et la classe TestCase est la classe de base des sous-classes :

- test_google_photo
- TestFileInfo
- TestCreation_date_from_full_filename

Non seulement la hiérarchie des classes est visible, mais les méthodes et les attributs des classes sont également présents. Par exemple, la classe TestInput possède les méthodes suivantes :

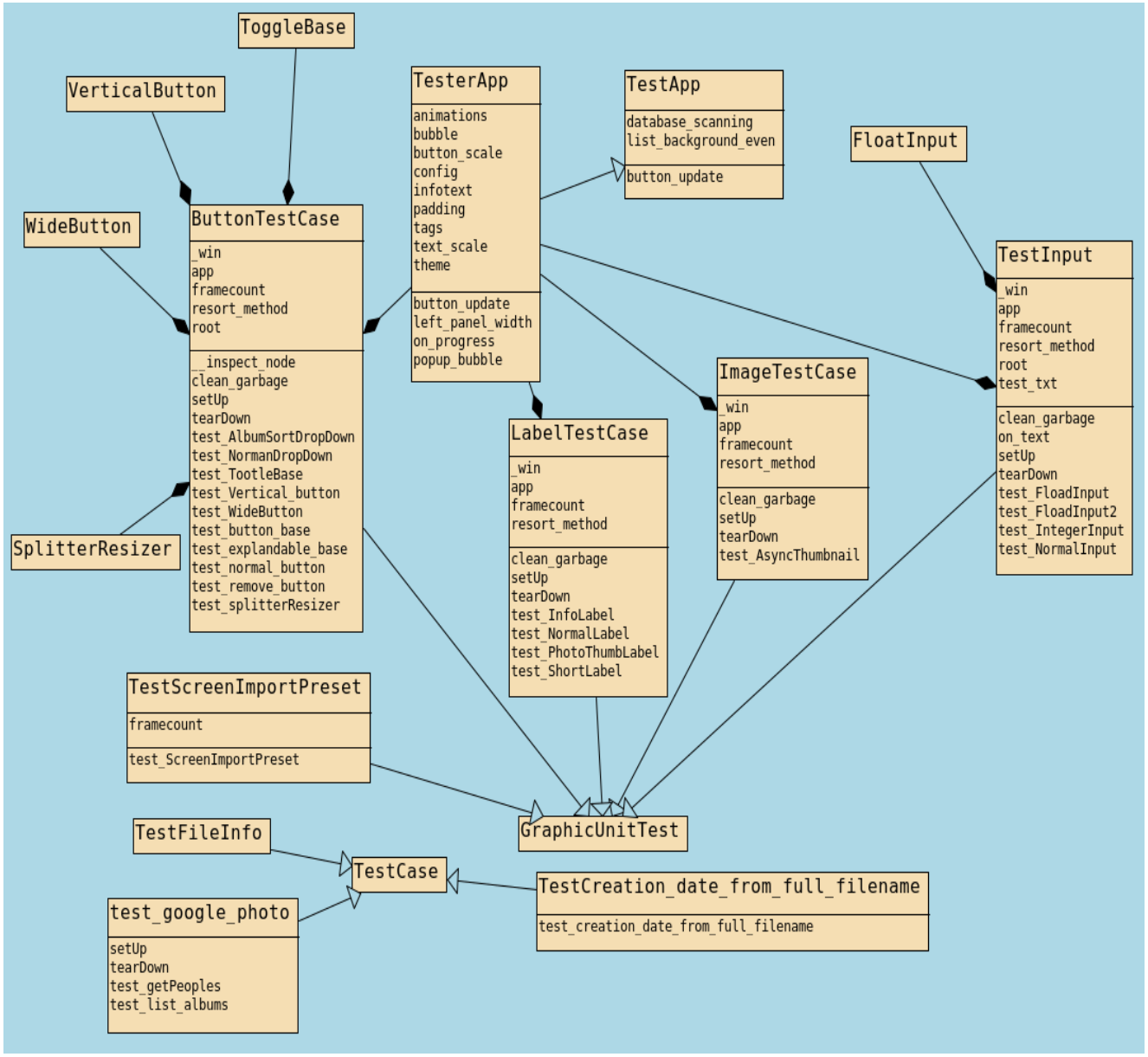
- clean_garbage
- on_text
- setUp
- tearDown
- test_FloadInput
- test_FloadInput2

- test_IntegerInput
- test_NormalInput

Et la classe TestInput possède également les attributs :

_win; app; framecount; resort_method; root; test_txt.

Mais le processus d'ingénierie inverse n'est pas parfait, et il semble varier d'un outil à l'autre. Dans le cas du plugin UML de l'IDE pycharm, un type de relation entre les classes, appelé Composition, établi entre les classes TestInput et FloatInput, qui est une entité utilisée par la classe TestInput. Cela devient plus clair quand on compare avec le diagramme réalisé avec l'outil Pynsource :



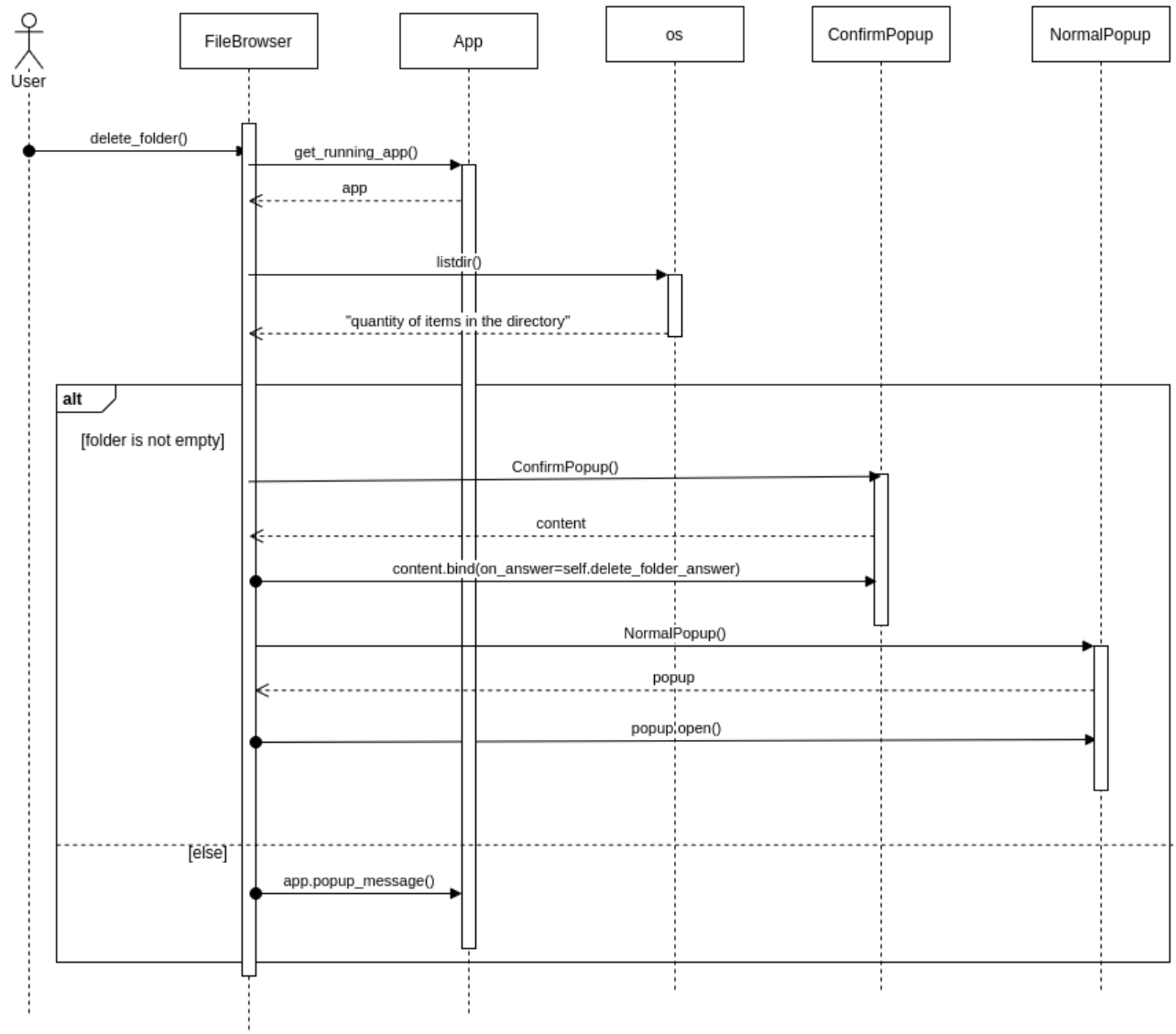
Dans le diagramme ci-dessus généré par l'outil Pynsource, les relations de composition sont indiquées par les lignes se terminant par un diamant noir, mais ce n'est pas non plus une représentation parfaite du code par ingénierie inverse. Par exemple, dans le diagramme ci-dessus, il y a deux classes avec le même nom, ButtonTestCase (des modules test_buttons et test_dropDown) qui n'ont pas pu être affichée séparément , et l'outil semble en avoir choisi une seule à rendre.

3. Diagramme de séquences

Les diagrammes de séquence peuvent être très détaillés, dans le sens où ils représentent explicitement des actions presque réelles sur les données d'une application. Un diagramme de séquence pour l'ensemble de l'application serait énorme, il fallait donc choisir un thème.

Dans ce cas, l'intention était de représenter un cas d'utilisation d'une action concrète de l'utilisateur consistant à supprimer un dossier, ce qui est possible parce que l'application en question dispose d'un navigateur de fichiers inclus.

L'outil draw.io [4] a été utilisé pour dessiner le diagramme de séquences suivant, qui représente la logique contenue dans le fichier filebrowser.py du projet :



Le diagramme se lit de gauche à droite et de haut en bas. Il commence par l'acteur utilisateur, qui interagit avec le programme pour lancer le processus de suppression du dossier et créer une fenêtre de confirmation, avec la méthode delete_folder() de l'entité FileBrowser.

L'objet FileBrowser demande une instance de l'application en cours d'exécution à l'entité App et obtient une réponse avec une entité de l'entité App appelée "app".

L'objet FileBrowser demande à l'entité "os" la quantité de fichiers dans le dossier à supprimer et obtient une réponse.

Une section "alt" commence, ce qui signifie qu'il y a un conditionnel (if/else).

Dans le cas où le dossier n'est pas vide :

L'objet FileBrowser appelle le constructeur de l'entité ConfirmPopup et obtient une réponse avec une instance de celle-ci appelée "content".

L'objet FileBrowser appelle la méthode bind() de l'objet "content", qui lie la méthode FileBrowser.delete_folder_answer() au popup. Il est intéressant de noter que la méthode delete_folder() initialement activée par l'utilisateur ne supprime pas vraiment un dossier, elle ne fait que lancer le processus et créer ce dialogue popup qui est ce qui peut réellement supprimer un dossier. En d'autres termes, l'action est déléguée à l'autre méthode, qui n'est pas représentée dans le diagramme.

L'objet FileBrowser appelle le constructeur de l'entité NormalPopup et obtient une réponse avec une instance de celle-ci appelée "popup".

Enfin, sur cette branche du conditionnel, l'objet FileBrowser appelle la méthode open() de l'objet "popup".

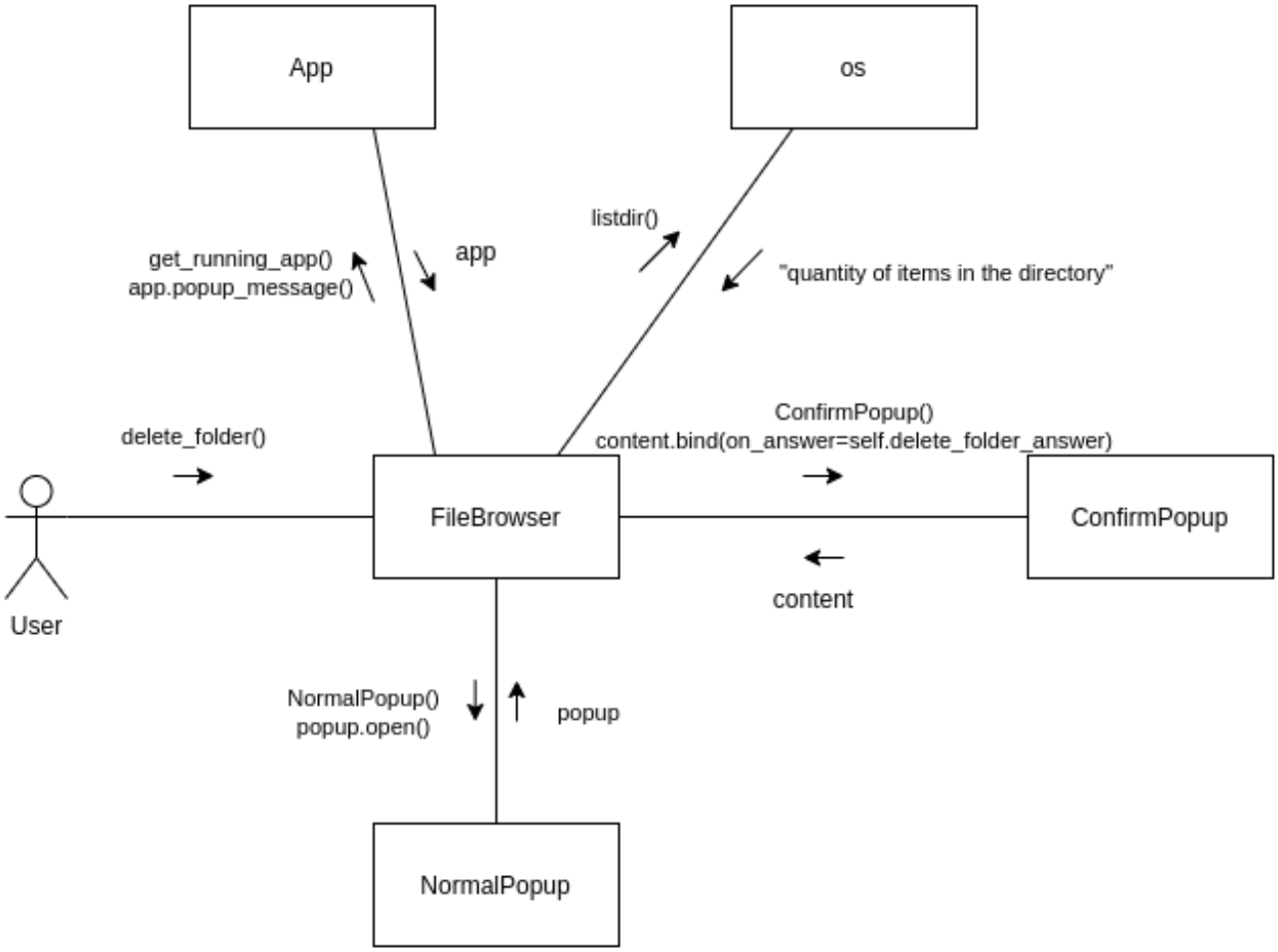
Dans le cas alternatif du conditionnel où le dossier est vide :

Enfin, sur cette branche du conditionnel, l'objet FileBrowser appelle la méthode popup_message() de l'objet "app".

Le processus de dessin du diagramme était manuel, ce qui signifie que le code était lu et interprété pour générer le diagramme sans outil permettant d'automatiser le processus de ingénierie inverse.

4. Diagramme de communication

L'outil draw.io [4] a été utilisé pour dessiner le diagramme de communication (ou diagramme de collaboration) suivant, qui représente la même logique que celle présente dans le diagramme de séquences présenté dans la section 3 :



Le diagramme de communication et le diagramme de séquence sont similaires. Ils sont sémantiquement équivalents, c'est-à-dire qu'ils présentent les mêmes informations, et il est possible de transformer un diagramme de communication en un diagramme de séquence et vice versa.

La principale distinction entre eux est que le diagramme de communication organise les éléments en fonction de l'espace et le diagramme de séquence en fonction du temps.

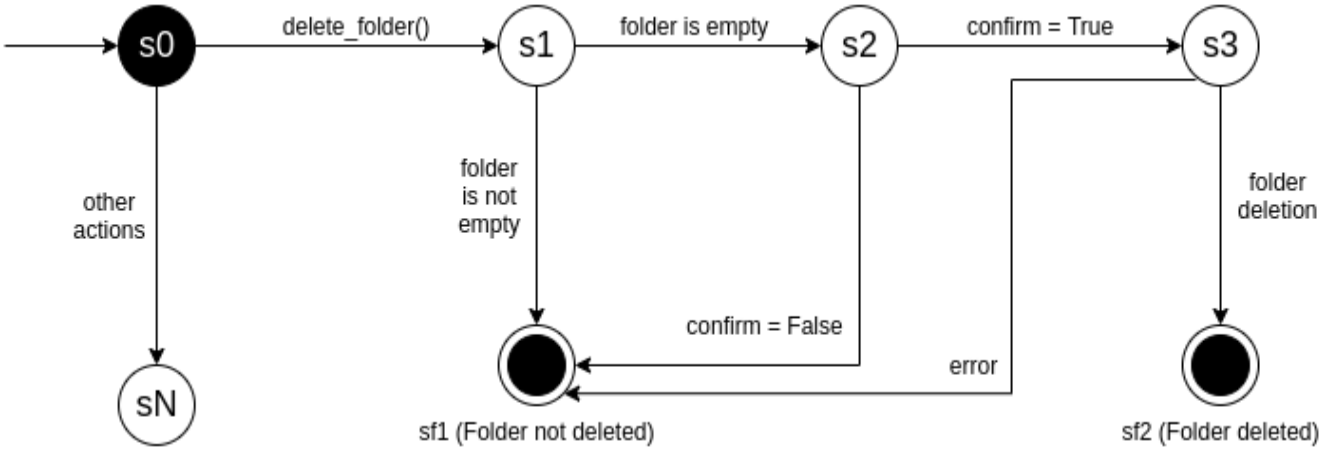
Cependant, la notion de temps pourrait encore être représentée dans le diagramme de communication, si les lignes de connexion sont numérotées selon l'ordre dans lequel elles se produisent.

De même, la notion d'espace est également présente dans le diagramme de séquence, mais elle n'est pas représentée aussi clairement que dans les diagrammes de communication.

Malgré leur équivalence sémantique, le diagramme de séquence semble être plus largement utilisé que le diagramme de communication, et la raison en est peut-être qu'il s'agit d'une convention, ou peut-être que l'expression du temps dans le diagramme de communication dépend de l'utilisation d'une notation, et non du modèle lui-même.

5. Diagramme d'état

L'outil draw.io [4] a été utilisé pour dessiner le diagramme d'état suivant, qui a le même thème que les sections 3 et 4, la suppression d'un dossier, mais cette fois, le diagramme ne représente pas jusqu'à l'affichage du dialogue de confirmation, mais jusqu'à l'achèvement de l'action:



Le cercle noir représente l'état initial, appelé "s0". À partir de l'état initial "s0", il y a deux transitions possibles :

- "other actions", pour représenter les transitions pour toutes les actions qui mènent à des états qui n'ont rien à voir avec le thème de la suppression d'un dossier.
- "delete_folder", qui mène à l'état "s1".

L'état sN représente tous les états qui n'ont rien à voir avec le thème de la suppression d'un dossier.

À partir de l'état "s1", il y a deux transitions :

- "folder is empty", ce qui conduit à l'état "s2".
- "folder is not empty", ce qui conduit à l'état final "sf1".

L'état final "sf1", représenté par un cercle noir encerclé, est l'état dans lequel le flux du programme se termine et le dossier n'a pas été supprimé.

À partir de l'état "s2", il y a deux transitions :

- "confirm = False", qui mène à l'état final "sf1".

- "confirm = True", qui mène à l'état "s3".

  À partir de l'état "s3", il y a deux transitions :

- "error", qui est une erreur externe, est produite et le dossier ne peut pas être supprimé, ce qui conduit à l'état final "sf1".
- "folder deletion", qui conduit à l'état final "sf2".

L'état final "sf2" est l'état dans lequel le flux du programme se termine et le dossier a été supprimé avec succès.

Le processus de dessin du diagramme était manuel, ce qui signifie que le code était lu et interprété pour générer le diagramme sans outil permettant d'automatiser le processus d'ingénierie inverse.

Lorsqu'un système plante et que nous voulons le faire fonctionner à nouveau, les redémarrages d'un système semblent fonctionner comme par magie, mais en fait ce n'est pas le cas. Ce qui se passe, c'est que nous pouvons considérer un système comme une machine à états (qui peut être représenté par un diagramme d'état), et lorsque le système est activé, il peut se retrouver dans un état qui crée des problèmes si le système ne sait pas comment se comporter en fonction de certains *inputs*. Ainsi, lorsque nous redémarrons un système, nous le faisons partir de l'état initial, où le comportement à partir de là est clair.

6. Conclusion

Toutes les tâches ont été accomplies et l'objectif de mieux comprendre la conceptualisation des logiciels avec un problème réaliste a été atteint.

L'ingénierie inverse ne donne pas de résultats magiques, car elle n'est pas capable de récupérer toutes les informations sur un modèle de conception à partir d'un code source.

La meilleure pratique consiste à documenter la conceptualisation d'un logiciel avant sa mise en ouvre, car il est difficile de faire de l'ingénierie inverse pour accéder à la documentation à partir du code source.

Après avoir exploré la liste des outils et en avoir trouvé d'autres sur le web [1], il est apparu que l'outil utilisé pour l'ingénierie inverse de certains codes est d'une importance vitale pour la réussite de la tâche.

Le fichier fourni avec une liste d'outils pouvant être utilisés pour les programmes d'ingénierie inverse manquait de documentation à propos des outils listés, car la plupart des éléments n'étaient que des liens vers différents outils. Beaucoup d'entre eux n'aidaient même pas à terminer les tâches de ce projet. Cependant, il était tout de même utile de vérifier les outils listés, car ils sont en effet des outils puissants pour aider les ingénieurs en logiciel.

C'était vraiment un projet amusant, car il est très gratifiant quand il est possible de comprendre un logiciel à partir du code source en faisant de l'ingénierie inverse et en essayant de penser comme l'architecte original du programme.

7. Références

La plupart des connaissances nécessaires pour mener à bien ce projet proviennent de la documentation présente sur le site moodle du cours : https://ena.etsmtl.ca/course/view.php?id=11784.

[1] « Curated list of UML tools – 2019 edition » - https://modeling-languages.com/uml-tools/#python - consulté le 14 juin 2020.

[2] Documentation présente sur le fichier Readme.md sur le *repository* du projet: 's20202-mgl802-lab1-eduardoxfurtado created by GitHub Classroom' https://github.com/cc-yvanross/s20202-mgl802-lab1-eduardoxfurtado - consulté le 14 juin 2020.

[3] Outil Pynsource, version 1.76 pour Linux - https://pynsource.com - consulté le 14 juin 2020.

[4] Outil draw.io 13.0.3 pour Linux - https://drawio-app.com - consulté le 14 juin 2020.