DEPARTEMENT INFORMATIQUE - IUT 2 GRENOBLE (Majuscules 12pt)

Année Universitaire 2023-2024

MEMOIRE DE STAGE

REMISE A NIVEAU D'UN LOGICIEL D'ANALYSE D'ENREGISTREMENT ARINC

THALES AVS FRANCE

Présenté par

Axelle Broyer

Jury

IUT : M. Bardolph

IUT : M. Bleuse

Société : M. Grosdidier

Déclaration de respect des droits d'auteurs

Par la présente, je déclare être le seul auteur de ce rapport et assure qu'aucune autre ressource que celles indiquées n'ont été utilisées pour la réalisation de ce travail. Tout emprunt (citation ou référence) littéral ou non à des documents publiés ou inédits est référencé comme tel et tout usage à un outil doté d'IA a été mentionné et sera de ma responsabilité.

Je suis informé qu'en cas de flagrant délit de fraude, les sanctions prévues dans le règlement des études en cas de fraude aux examens par application du décret 92-657 du 13 juillet 1992 peuvent s'appliquer. Elles seront décidées par la commission disciplinaire de l'UGA.

A Valence Le 23/05/24 Signature

Remerciements

Je tiens à adresser mes sincères remerciements à Augustin Grosdidier pour son accompagnement tout au long de mon stage. Augustin a su m'intégrer à THALES et m'orienter quand j'en avais besoin. Il s'est montré compréhensif et m'a régulièrement encouragée, m'offrant ainsi un stage de qualité autant au regard de la vie en entreprise que du développement de mes compétences en autonomie.

Je remercie également Jennifer Garde sans qui ce stage n'aurait pas été possible. Jennifer m'a donné confiance avant le début du stage et s'est assurée que celui-ci se déroule dans les meilleures conditions, toujours à l'écoute et de bon conseil. Merci à Kelly Beaucoral, assistante de mon département, pour son suivi régulier, sa bienveillance, sa patience.

J'adresse aussi mes remerciements à Bernard Bellon qui m'a été d'une aide précieuse. Il s'est rendu disponible et a pris du temps pour comprendre mes problèmes ; je suis toujours ressortie de nos discussions avec des solutions.

Enfin je remercie Léa, Johnny, Jean, Maxime, Jean-Marc, Charles et toute l'équipe d'Augustin Grosdidier pour leur accueil au département Capteurs.

I. Table des matières

II. Table des figures

III. Résumé du stage

Du 15 avril au 5 juillet, j'ai effectué un stage de fin de BUT2 à Thales AVS* France. Thales AVS est spécialisé dans la fabrication, production et distribution d'équipements destinés à l'aéronautique. Installée au département Capteurs, j'ai travaillé sur un logiciel en lien avec l'IESI*, appareil de secours placé dans le cockpit comportant des capteurs. Le logiciel, appelé GRADE*, est utilisé pour vérifier la robustesse de l'IESI après avoir été soumis à des tests environnementaux. Il est séparé en deux parties, une partie graphique et une partie dépouillement de données : Graph* et Depouil*. Mon stage a pour but de faire migrer ce logiciel, le convertir de Python 2.7 à Python 3.9, et porter les bibliothèques. Les objectifs restants à ce jour sont de corriger des bugs identifiés au préalable et implémenter des améliorations afin de remettre à jour le logiciel global. De plus, GRADE est un atout pour Thales et un des objectifs finaux serait de le proposer à d'autres projets, pour d'autres équipes.

Le premier objectif était de retrouver un environnement stable sous Python 2 et de recompiler le logiciel pour reproduire les exécutables de GRADE. Cette démarche a dû être abandonnée car je ne disposais pas d'un logiciel essentiel pour la recompilation*, Microsoft Visual C++ Redistributable*. Je suis alors passée au portage du code. J'ai premièrement mis à niveau le langage en lui-même, Python. C'était une étape simple grâce au module* Python 2to3[1], développé justement lors de la sortie de Python 3. Je suis ensuite passée aux bibliothèques liées à l'IHM*. Il y en avait deux : Qwt* et PyQt*. Qwt est une bibliothèque écrite en C++ et son wrapper avait été déprécié. J'ai, après des recherches, découvert la bibliothèque équivalente, PythonQwt*, laquelle intègre directement une des versions les plus récentes de Qwt. L'implémentation s'est ensuite faite très rapidement. Enfin, j'ai dû migrer PyQt. Liant la bibliothèque applicative Qt* et Python, ce *wrapper** est omniprésent dans le code. Le code était écrit en PyQt4 et il a d'abord fallu passer en PyQt5. Il s'agissait surtout de syntaxe et notamment autour des signaux*. C'est un objet que j'ai appris à définir et utiliser correctement. Une fois à l'aise avec PyQt, la version la plus récente étant PyQt6, j'ai remis à jour cette bibliothèque. Les changements concernaient de nouveau la syntaxe mais étaient beaucoup plus longs à implémenter[2]. Ainsi, j'ai cherché à automatiser cette tâche grâce à un script Python distribué sur le site officiel PyPi*. Il reste aujourd'hui la dernière étape, la compilation.

J'ai rencontré plusieurs difficultés durant cette phase de migration. D'une part, les ordinateurs de Thales sont très sécurisés ce qui a posé problème notamment au début. D'autre part, plusieurs erreurs sont apparues lors de la compilation. Elles sont en cours de correction. Porter Qwt a aussi été complexe et chronophage du fait du manque de documentation. Fière de mon travail, j'espère aller le plus loin possible : j'ai prévu de prendre du temps pour corriger les bugs identifiés par mon tuteur, de faire les tests de non-régression* et de rédiger la documentation technique.

Ce stage m'a beaucoup appris et enrichie. La vie en entreprise à Thales est agréable et le grand nombre de stagiaires m'a offert une insertion facile au début et de l'entraide tout au long du stage. L'autonomie de ce stage est une partie que j'ai appréciée et qui m'a été bénéfique pour développer mes connaissances et mes compétences.

[1] Voir sitographie [5]

[2] Voir sitographie [8]

IV. Introduction et mise en contexte

IV.1 *Présentation de l'entreprise*

THALES est un groupe français fondé par Denis Ranque et spécialisé dans les hautes technologies. Il provient de la fusion des sociétés Thomson-Brandt et de la Compagnie générale de la télégraphie Sans Fil, en 1968. Il a été renommé Thomson-CSF puis finalement Thales en l'an 2000, en référence au philosophe. Ses secteurs opérationnels sont notamment l'aéronautique et le spatial, la défense et la sécurité ainsi que la cybersécurité et l'identité numérique, et dans une moindre mesure les transports.

Le siège social de Thales se situe à Meudon en France et son PDG est Patrice Caine depuis 2014. Thales compte 77 000 collaborateurs et est présent dans 68 pays du monde. Aujourd'hui, Thales est subdivisé en six branches principales appelées « Global Business Units » ou GBU* selon les domaines métier évoquées ci-dessus. Mon stage était situé dans la GBU Avionics, renommé AVS.

La branche AVS développe des équipements avioniques, c'est-à-dire destinés à être embarqués dans les aéronefs. Elle est subdivisée en six sous-branches, « Business Lines » abrégé BL*, ayant chacun un domaine de compétence : par exemple, « Electrical Systems » ELS, « In Flyt Experience » IFE produisant des équipements multimédias pour les passagers, « Training and Simulation » T&S, ou encore « Flight Avionics », FLX*. Ses équipements ont été adoptés dans 67% des avions du monde et sont aussi conçus pour les hélicoptères et les drones. Aujourd'hui, Thales AVS France regroupe environ 7000 personnes spécialisées réparties sur 11 sites dont Valence, Vendôme, Châtellerault à Bordeaux et Toulouse.

La BL dans laquelle je suis est FLX, elle s'occupe notamment des équipements d'assistance au vol. A Valence, FLX est divisé en plusieurs départements métiers : Packaging, Alimentation, Capteurs… Etant dans le département Capteurs, ma mission est en lien avec un produit Thales utilisant des capteurs.

IV.2 *Mise en contexte du stage*

En 1999, Thales sort un projet innovant, l'« Integrated Electronic Standby Instrument » abrégé IESI (*cf.* Figure 1). Aujourd'hui, plus de 40 000 aéronefs en sont équipés. La volonté de ce projet était de produire une solution compacte et intégrable facilement pour plus de sécurité dans l'aviation.

Figure 1 Image de la dernière version de l'IESI

Afin de respecter les objectifs de sécurité en cas de situations critiques, il faut pouvoir fournir au pilote les références minimales de pilotage qui sont la vitesse air, l'attitude et l'altitude (*cf.* Figure 2). Ainsi, lors d'un évènement grave *e.g.* un problème électrique mettant en panne les appareils, l'IESI est l'équipement de secours, dernier appareil fonctionnel et complètement indépendant des autres.

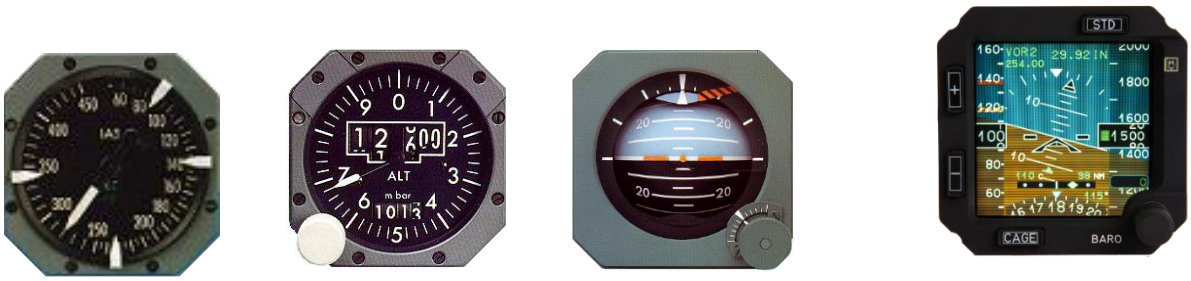

Figure 2 (De gauche à droite) Indicateurs de vitesse air, altitude et attitude réunis dans un seul écran, celui de l'IESI

Le projet IESI est multi-site : l'assemblage final est fait à Vendôme et le développement final à Valence. Plus de 100 employés Thales sont actuellement impliqués dans son développement. Valence est le site fournissant le plus de composants pour l'IESI : les deux capteurs de pressions (statique et totale), la carte d'alimentation, le logiciel.

IV.3 *Le logiciel GRADE*

GRADE est un logiciel codé en Python 2.7.2 de 2013 à 2017. Il est utilisé dans les phases d'Intégration, Validation et Vérification, IVV* et de qualifications environnementales de l'IESI (*cf.* Figure 3).

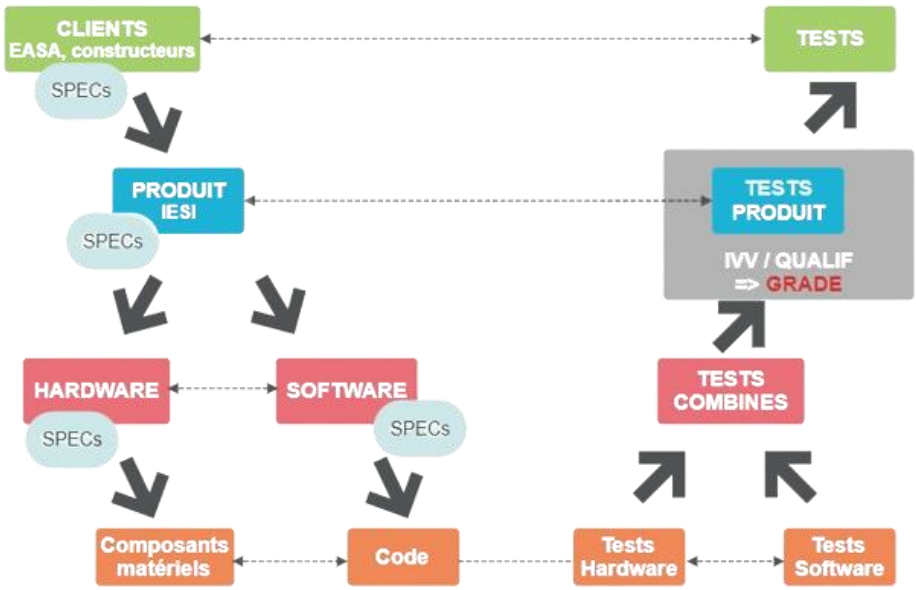

Figure 3 Modèle en V simplifié du cycle de développement IESI

Il permet donc de certifier les tests fonctionnels et les tests de tenue de performance dans le domaine environnemental attendu, lorsque par exemple on soumet l'IESI à une forte pression ou une température très basse.

GRADE fonctionne en s'appuyant sur le logiciel BAST[3]. Concrètement, on connecte l'IESI à un banc de test pour ARINC429 qui est lui connecté à un ordinateur muni d'un logiciel d'enregistrement (*cf.* Figure 4) qui va produire des fichiers .nar* et .lru*, .rec*. Les fichiers .nar, « *Network Application Requirements* » sont des fichiers de configuration pour ARINC429*, tandis que les fichiers .lru, « *Label Remplacement Unit* » sont les fichiers qui contiennent les informations sur les messages échangés avec ce protocole.

[3] BAST est l'acronyme de Bruit Acoustique et Structure Transitoire. C'est un logiciel embarqué développé par la société IXSEA pour l'analyse et le traitement des données. Il utilise le bus de données ARINC429 pour recevoir des données provenant de capteurs. Voir sitographie [1].

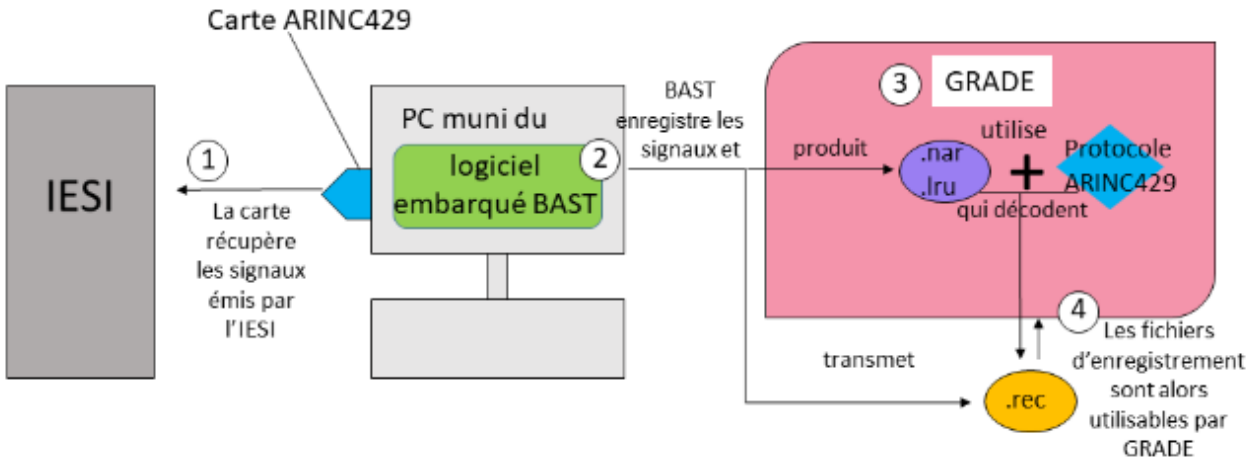

Figure 4 Schéma du fonctionnement de GRADE

GRADE Graph* analyse des données issues de fichiers, notamment au format ARINC[4]. Il permet d'appliquer des fonctions de traitements mathématiques et numériques dessus et d'afficher ces données graphiquement (*cf.* Figure 5).

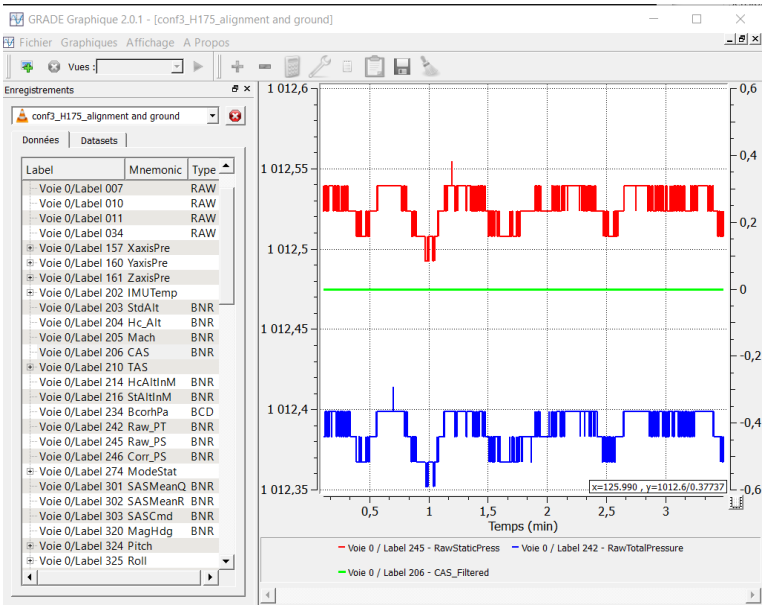

Figure 5 Exemple d'une utilisation de GRADE Graph

[4]ARINC est une norme pour l'aéronautique. Ce terme peut désigner à la fois le protocole utilisé et l'interface électrique. Une carte ARINC permet donc de collecter et décoder des signaux numériques provenant d'un périphérique ARINC429, mais aussi de convertir et transmettre ces données à l'ordinateur. Voir sitographie [2].

GRADE Depouil* est quant à lui un logiciel de dépouillement par lot[5]. Il teste les fichiers de données par rapport à un ensemble de critères listés dans un fichier .xml*, analyse les différences entre les fichiers avant/après *e.g.* avant d'être exposé à une forte contrainte environnementale et après (*cf.* Figure 6). Il permet aussi de produire un rapport standardisé sous Word.

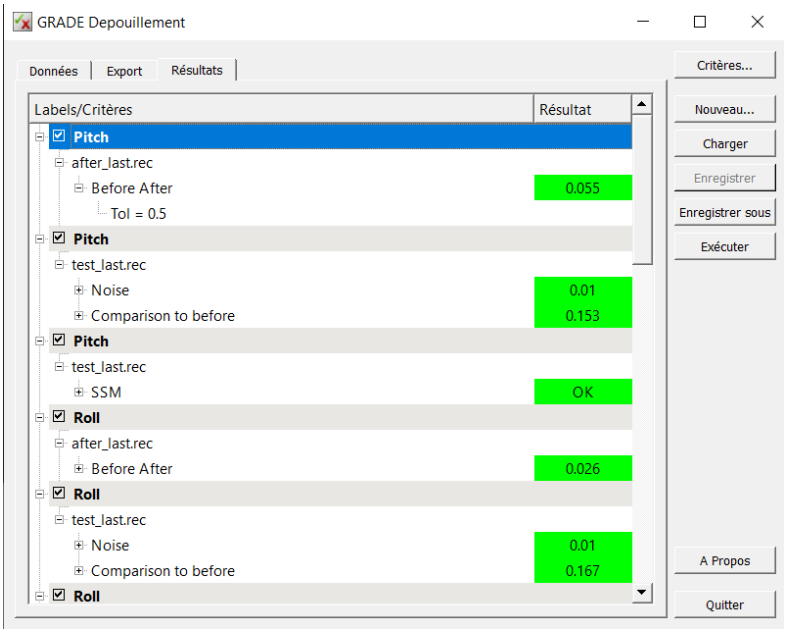

Figure 6 Exemple d'une utilisation de GRADE Depouil

IV.4 *Ma mission au sein de l'équipe IESI*

M.Grosdidier et son équipe sont appelés Integrated Product/Project Team, IPT*. M.Grosdidier a plusieurs rôles dans cette équipe : il écrit des spécifications produit, donc des exigences à satisfaire, pour le côté logiciel. Il est également manager dans la phase IVV/qualification et peut faire des propositions de design pour le code.

Dans l'aéronautique, il est essentiel que tout soit robuste. La durée de vie d'un projet visée est en général de 20 à 25 ans. Ainsi, pour continuer sur une lancée d'amélioration continue du projet, GRADE doit être remis à jour. En effet, non seulement Python 3 est maintenant déployé depuis plusieurs années, mais les bibliothèques associées ont bien changé vis-à-vis du code en Python 2. De plus, des bugs dans le code doivent être corrigés pour assurer une stabilité du logiciel et des améliorations doivent être implémentées pour élargir le champ d'action de GRADE. Une fois les deux .exe* produits (celui de Graph et celui de Depouil), il nous faudra faire passer les tests de non-régression. Il sera sûrement nécessaire de rajouter de nouveaux tests et la documentation devra être remise à jour.

[5] Un logiciel de dépouillement par lot est un logiciel permettant d'automatiser des tâches. Ici, GRADE Depouil permet de tester un grand nombre d'enregistrements par rapport à une bibliothèque de critères.

V. Mise à jour et amélioration de GRADE

V.1 *Etat du projet avant ma mission*

GRADE n'a plus que des versions obsolètes de bibliothèques. Certaines nécessitent simplement de mettre à jour la version (comme NumPy* par exemple), les suivantes nécessitent plus de travail :

- Python en lui-même
- PyQt
- Qwt

Le premier objectif était de pouvoir recompiler et exécuter le code source depuis mon nouveau poste de travail. M.Grosdidier m'avait conseillé de retrouver un environnement stable sous Python 2. Cela me permettrait de commencer la migration dans de bonnes conditions. Nous nous sommes donné deux semaines pour essayer.

V.2 *Environnement de développement et restauration Python 2*

A mon arrivée, j'ai reçu un PC sous Windows 10 à débloquer avec une carte à puce. J'ai commencé mon travail avec Visual Studio Code*, qui bien que moins réputé que PyCharm*, s'est avéré pratique pour gérer ce problème d'installation. En effet, PyCharm supporte, débugue et interprète moins bien les codes plus vieux contrairement à VS Code*. Grâce aux .vsix*, l'installation s'est fait facilement.

J'ai pu avancer dans ma tâche jusqu'à me heurter à un problème majeur : les .dll et le besoin du logiciel Microsoft Visual C++ Redistributable[6]. Les « Dynamic Link Libraries », DLL* sont des fichiers binaires précompilés et sont empaquetés dans des .pyd*, ce qui est essentiellement la même format mais adapté à Python[7]. PyQt les utilise pour faire ce lien entre Qt* et Python. Cependant j'ai rencontré deux erreurs liées à ces fichiers. Certains DLL semblaient manquer au vu de l'erreur « DLL Load Failed[8] », due au téléchargement sur des sites non officiels de versions obsolètes. De plus, les DLL sont spécifiques à un environnement d'exécution. Ainsi, je devais recompiler les fichiers DLL du code source. Or, Thales a une politique de sécurité très restrictive pour minimiser les risques de virus et malveillances. Bien que bénéfique, ces restrictions ont rendu l'installation du logiciel très compliquée. Le logiciel n'est jamais apparu dans mon poste de travail et je n'ai pas pu aller plus loin.

---

[6] Microsoft Visual C++ Redistributable est un ensemble de bibliothèques logicielles Microsoft permettant l'exécution d'applications écrites en C/C++. Ce logiciel est souvent essentiel surtout sous Windows pour assurer non seulement le bon fonctionnement mais aussi le déploiement d'une application.

[7] Voir sitographie [3]

[8] Voir sitographie [4]

Au bout d'environ deux semaines, je suis donc passée à la migration du code. Un dépôt Git* sous Bitbucket* a été créé pour le projet et une branche « migration » m'a été dédiée. Ainsi, j'ai pu gérer et enregistrer mes modifications régulièrement et en autonomie. J'ai aussi utilisé Beyond Compare* qui permet de comparer deux fichiers ou deux dossiers entiers, et Git Extension*, qui permet d'utiliser les outils de Git de manière plus visuelle.

Enfin, je suis passée de l'IDE VS Code à l'IDE PyCharm, plus performant.

V.3 *Plan d'action*

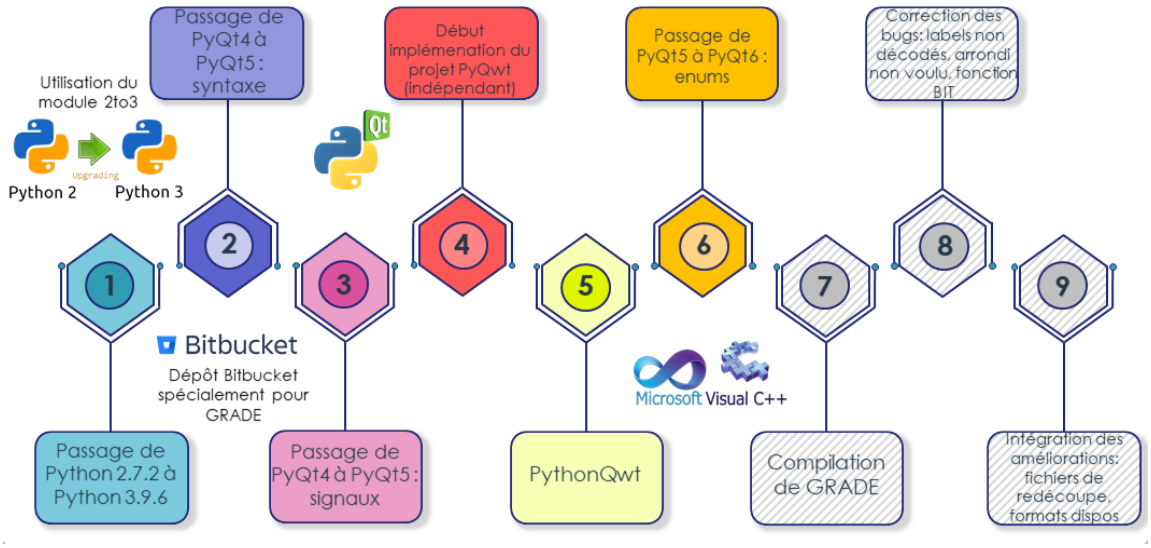

Figure 7 Graphe de l'état de l'art à mi-parcours

Le graphe présenté ci-dessus (*cf.* Figure 7) définit le plan d'action que j'ai mis en place pour le projet. Les étapes grisées sont les étapes en cours ou non commencées.

Durant mon stage, je me suis fixé un objectif quotidien (*e.g.* « résoudre tel problème »), un objectif hebdomadaire (*e.g.* « finir la mise à jour de cette bibliothèque ») et un objectif sur une plus longue période (*e.g.* « finir la migration d'ici mi-juin »).

V.4 *Passage à Python 3*

En premier lieu, il me fallait passer à Python 3. La version installée par défaut chez Thales est la 3.9.6, une des plus récentes. J'ai d'abord pris le temps d'analyser le code et de me documenter sur les changements de Python 2 à Python 3. Majoritairement de la syntaxe, j'ai automatisé cette tâche en utilisant le module 2to3*. Ci-dessous un exemple de trace quand on utilise ce script sur un module.

```
RefactoringTool: Skipping optional fixer: buffer

RefactoringTool: Skipping optional fixer: idioms

RefactoringTool: Skipping optional fixer: set_literal

RefactoringTool: Skipping optional fixer: ws_comma
```

```
RefactoringTool: Refactored hello.py
--- hello.py     (original)
+++ hello.py     (refactored)
@@ -1 +1 @@
-print 'Hello World'
+print('Hello World')
RefactoringTool: Files that were modified:
RefactoringTool: hello.py
```

N'ayant pas rencontré de difficulté particulière sur ce premier point, j'ai pu passer à la suite : les mises à jour des bibliothèques.

V.5 *Migration de PyQt*

Le premier défi est alors apparu : comprendre et maîtriser la bibliothèque PyQt*, associée à Qt*. PyQt est un module qui permet de lier la bibliothèque Qt écrite en C++ au langage Python. Il existe aussi PySide*. PySide et PyQt sont des boîtes à outils permettant de faire de l'interface humain-machine, IHM*, de manière simplifiée.

Il y avait notamment l'apparition de « signaux », objets que je ne connaissais pas et que j'ai appris à utiliser. Toutes les classes issues de la bibliothèque Qt implémentent des signaux prédéfinis, cohérents avec le but de chaque classe et objet. Cela correspond à une architecture Modèle-Vue-Contrôleur MVC* : ces signaux permettent de faire le lien entre ce que fait l'utilisateur avec la vue (par exemple, cliquer sur un bouton) et les contrôleurs* dans le code. Ils déclenchent des évènements (par exemple, l'ouverture d'un pop-up), appelés slots*. Mais l'on peut aussi définir ses propres signaux. Ensuite, il suffit de les connecter à un objet capable d'en recevoir, ou tout simplement un objet « QApplication » qui désigne une application entière (*cf.* Figure 8).

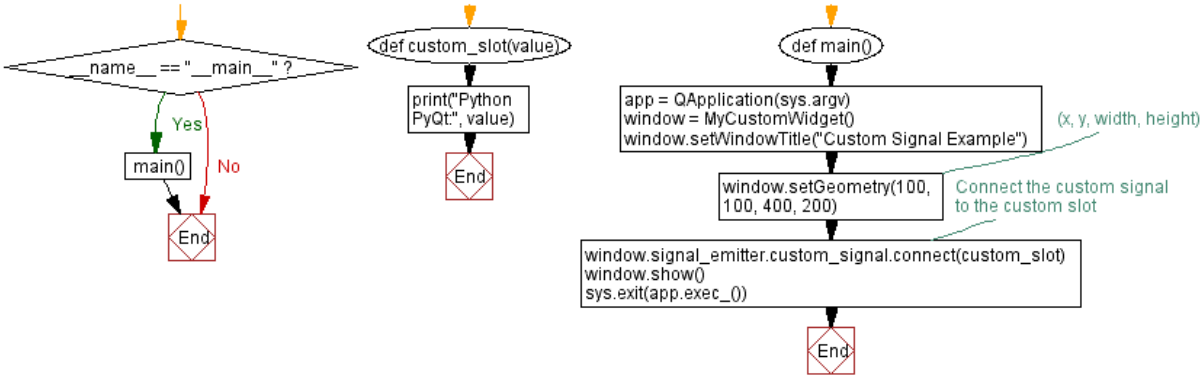

Figure 8 Fonctionnement et flux des connexions signal-slot pour un signal customisé

En PyQt4, version de la bibliothèque au moment de la reprise de GRADE, les signaux customisés n'avaient pas besoin d'être définis : il suffisait d'utiliser le macro « SIGNAL ». Il y

avait donc des changements de syntaxe à faire ainsi que la définition des signaux à ajouter, au bon endroit dans le code.

Mon problème principal résidait dans le besoin d'un nom unique pour chaque signal. Donner un nom unique à chaque nouvel objet est une bonne pratique et évite aussi de potentiels conflits. Cependant, certains signaux avaient le même nom, il s'agissait du « même » signal. Avec tous les imports dans le code, cela n'aurait pas posé de problème et aurait même simplifié la définition, mais les signaux de mêmes noms n'avaient pas tous la même signature[9]. Ci-après une figure représentant concrètement mon problème.

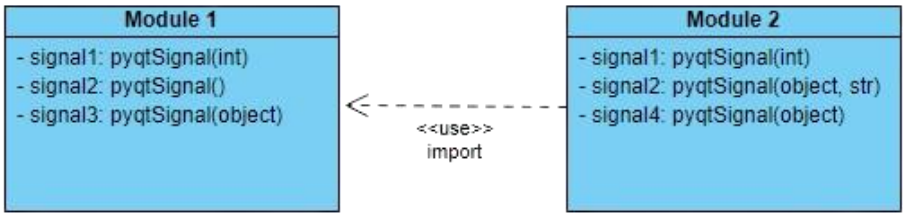

Figure 9 Diagramme UML de modules Python utilisant des signaux

Comme on peut le voir sur ce diagramme, certains signaux ont le même nom, mais le signal appelé signal2 n'a pas la même signature entre les deux modules.

Après avoir identifié les problèmes, j'ai commencé des essais pour corriger cela. Pour distinguer les signaux déjà existants des signaux customisés, j'ai cherché dans la documentation. J'ai d'abord pensé à créer un module « signals.py » contenant une classe par module qui utilise des signaux customisés. Il y avait alors beaucoup de redondances de signaux. Faute d'informations correspondant à mon problème, j'ai demandé de l'aide à une personne qui avait travaillé autour de GRADE. Elle a l'habitude de travailler avec PyQt et m'a expliqué que les signaux pouvaient être seulement des attributs de classe*. J'ai cherché les signaux customisés de chaque module, dans quelles classes exactement ils étaient utilisés, et je les ai définis dans ces classes. De cette manière, la redondance était moindre et il n'y avait aucun risque de conflit.

Une fois la syntaxe et la définition des signaux faites, je suis passée à la deuxième mise à jour, de PyQt5 à PyQt6. Je m'étais documentée et les changements étaient mineurs (il s'agissait encore de syntaxe et de quelques fonctions), mais chronophages. Il s'agissait des types « enum ». Ce qui s'écrit

```
widget.setCheckState(Qt.Checked)
```

en PyQt5 devient

```
widget.setCheckState(Qt.CheckState.Checked)
```
[10]

en PyQt6. On voit donc bien que du fait de la quantité importante de widgets* et autres outils IHM utilisés, cela allait être fastidieux.

[9] Voir sitographie [10]

[10] Extrait issu du site pythonguis.com. Voir sitographie [8]

Le point positif de cette étape était la facilité du portage. En effet, étant donné la comptabilité « à l'envers » de PyQt, c'est-à-dire de la version 6 vers la 5, le code peut être modifié pas à pas sans créer d'erreurs. En modifiant seulement la syntaxe sans changer le package* importé, c'est-à-dire en laissant la mention « import PyQt5 » tout en passant à PyQt6 la syntaxe, il n'apparaît pas d'erreur. J'ai donc pu travailler facilement sur la migration.

Pour commencer, j'ai cherché à automatiser la partie purement syntaxique, comme pour Python 3. Il n'existe pas de module officiel pour le passage de PyQt5 à PyQt6, à la différence de Python, mais des scripts ont été développés et publiés par des développeurs indépendants. J'ai choisi d'utiliser le projet PyQtEnumConverter* sur une copie du projet[11]. J'ai changé l'import pour importer PyQt6 et j'ai testé le script sur quelques modules.

Dans le code actuel, toutes les erreurs, notées en rouge dans PyCharm, ont été résolues. Les problèmes restants n'étaient alors plus des erreurs mais bien des avertissements (en jaune, ce sont des « warnings »). Certains sont en jaune clair, et d'autres en jaune foncé : ce sont les plus importants. Les avertissements peuvent être compliqués à comprendre car certains viennent d'une mauvaise interprétation du code par l'IDE. Par exemple, en PyQt6, PyCharm met un avertissement à chaque ligne utilisant un signal customisé. Il reconnaît le signal mais semble ne pas réussir à appliquer les fonctions de connexion, déconnexion et émission qui vont avec les signaux. Il peut alors sembler important à résoudre et pourtant, il ne s'agit que d'une mauvaise interprétation de l'IDE qui n'a pas d'influence sur la bonne exécution du code.

D'autres sont plus problématiques. Deux avertissements notamment, qui sont apparus à de nombreuses reprises. Un relevait un conflit sur les fonctions sort() et sorted(), qui ont changé de comportement et donc de signature[12]. L'autre venait du fait que lors d'une surcharge de méthode existante, si l'on veut changer la signature pour ajouter des paramètres, cela crée un avertissement. Ainsi, sans être des erreurs, ces situations sont suffisamment conflictuelles pour y accorder du temps.

Pour la première, il s'agit du fait que la fonction cmp() est obsolète, et c'étaient des arguments provenant de cette fonction qui étaient utilisés dans les fonctions sort() et sorted(). La fonction cmp() était très utilisée en Python 2 : elle servait à comparer deux objets et était donc pratique car elle s'utilisait avec n'importe quel type (des entiers, des listes…). Elle avait des arguments tels que cmpKey ou cmpFct qui permettaient de préciser le tri. Depuis, elle est dépréciée et Python 3 a implémenté à la place plusieurs fonctions de tri plus flexibles mais plus précises. Il suffit ainsi d'enlever les arguments. Pour la deuxième, deux solutions sont possibles : utiliser la mention super mais ce n'est pas une solution très stable, ou utiliser le décorateur « @override[13] » qui par contre n'est pas nativement pris en charge en charge par l'IDE PyCharm.

De manière générale, mon but était de réduire au minimum les avertissements plus graves. Beaucoup de fonctions spécifiques et moins utilisées en Python 2 ont été dépréciées mais peu de documentation en parle. Il faut alors chercher la fonction dans la documentation

[11] Voir sitographie [9]

[12] Voir sitographie [7]

[13] Voir sitographie [6]

de la version initiale et dans celle que l'on veut utiliser, de trouver le nouveau nom de cette fonction ou un équivalent.

Le portage de cette bibliothèque a donc été très formateur. Cette phase de migration m'a appris l'importance du respect des bonnes pratiques : cela permet un code durable et portable. On ne sait pas comment les prochaines versions de langages et de bibliothèques vont évoluer. Certaines fonctions seront de nouveau dépréciées, des nouvelles fonctionnalités seront ajoutées, des syntaxes ne seront plus permises… Ci-après un exemple d'avertissement plus léger mais pouvant peut-être mener à des conflits ; et la solution associée.

```
if  mode1 :

        if data is None :

                status = …

        if data is not None :

                status = …
```

```
Warning : Local variable "status" might be referenced before assignment
```

La solution est donc ici d'initialiser la variable « status » à « None » (soit à rien, c'est une variable vide de sens pour l'instant) avant de rentrer dans la première condition « if mode1 ». On peut imaginer que dans les futures versions de Python, ne pas faire cela au préalable ne sera plus du tout accepté.

V.6 *Migration de PyQwt*

Les problèmes avec PyQt étant corrigés, je me suis penchée sur la migration de la troisième bibliothèque importante : Qwt et PyQwt. Qwt est une bibliothèque pour l'IHM écrite en C++ qui a donc été empaquetée par PyQwt, de la même manière que Qt et PyQt. Qwt est utilisée dans ce code pour créer la fenêtre graphique, les graphes et les éléments associés comme les boîtes de dialogue. La version initiale était Qwt5. Contrairement à PyQt, la plus récente n'est que la 6.3.

Je n'ai trouvé que très peu de documentation sur PyQwt, mais j'ai compris que le wrapper avait été déprécié. En effet, Qwt sert surtout à faire des graphiques comme des courbes, des nuages de points ou plus récemment des histogrammes. Des modules très similaires et plus utilisés tels que Matplotlib* ou PyQtGraph* étant bien maintenus pour Python, PyQwt n'avait plus beaucoup d'intérêt. PyQwt s'est donc arrêté à PyQt4, dû aux changements importants dans l'API Qt entre les deux versions.

J'ai d'abord pensé à porter le code vers Matplotlib mais il s'agissait d'un niveau de compétences et de connaissances en Python que je n'avais pas. J'ai ensuite trouvé un projet indépendant sur Github recréant PyQwt[14] mais il était impossible à implémenter : j'avais de nouveau besoin du logiciel MVCR*. J'ai pu finalement être aidée : il existe un nouvel

[14] Voir sitographie [11]

équivalent, appelé PythonQwt[15]. Développé en février 2024, il n'y avait quasiment aucune documentation dessus. J'ai installé la wheel* avec pip*, il suffisait alors d'écrire « import qwt ». Après avoir testé rapidement dans un module temporaire de test la performance de PythonQwt, la migration de cette bibliothèque était terminée.

Qwt est maintenant directement intégré à PythonQwt, et non pas empaqueté. Les développeurs ont imposé la version de Qwt utilisée. Il s'agit de la 6.1.5, l'une des plus récentes. Alors, on peut considérer la migration finie.

```
C:\windows\system32>python
Python 3.9.6 (tags/v3.9.6:db3ff76, Jun 28 2021, 15:26:21) [MSC v.1929 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import qwt
>>> print(qwt.__version__)
0.12.1
>>> print(qwt.QWT_VERSION_STR)
6.1.5
>>>
```

Figure 10 Affichage dans la console Python sous Windows des versions de Qwt

Mon but était maintenant de passer en développement mené par les tests, *test-driven development* en anglais. J'allais donc lancer la compilation des modules et à chaque erreur, je m'arrête, je corrige l'erreur et je recommence. N'ayant plus d'erreurs apparentes détectées par le code, il m'aurait été impossible de trouver moi-même les erreurs. Cette méthodologie me paraissait donc beaucoup moins chronophage, et la meilleure à adopter.

V.7 *Premières compilations*

Structure du projet

La structure du projet est la suivante :

```
GRADE/
|-- controllers/
|   |-- A429
|   |-- TXT
|   |-- __init__.py
|   |...
|-- UI/
|   |-- UI_Graph_Main.py
|   |-- UI_Analyzer_Main.py
|   |...
|-- ...
|-- GRADE_Graph.pyw
|-- GRADE_Depouil.py
|-- make.py
|-- __init__.py
```

On retrouve donc différents packages Python organisés selon leur fonction : le package « controllers » regroupe les fonctions essentielles pour le traitement de données selon

[15] Voir sitographie [12]

le format (ARINC, texte…). Il peut s'agir de petites bibliothèques ou de fonctions utilitaires par exemple. Il y a un package spécialisé pour traiter la gestion de la bibliothèque de critères utilisée par Depouil, un package pour l'IHM, un pour la documentation, et d'autres encore.

En Python 2, il est essentiel que les dossiers contiennent un fichier spécifique toujours appelé init.py pour que le dossier soit reconnu comme un package. Ce fichier pouvait être vide ou rempli comme un module* Python classique. Cela est nécessaire pour éviter les erreurs lors d'appels de fonctions d'autres modules, pour les imports relatifs, et donc pour que tous les modules fonctionnent ensemble. Depuis Python 3, cela est révoqué. C'est pourquoi le package pour l'IHM n'en a pas, il n'y a pas de besoin. Le module « make.py » (qu'on abrégera « make » par la suite) sert quant à lui à fabriquer les exécutables, donc produire les logiciels Graph et Depouil. Il regroupe le contrôleur principal contenant le « main* » pour chaque logiciel.

Enfin, ce projet contient des fichiers de configuration « config.ini ». Celui du package « controllers » est rempli avec des fonctions et des variables pour traiter les méta-données* des contrôleurs. Il existe aussi un fichier de configuration du même nom à la racine du projet, contenant le nom d'un chemin appelé « AppliDir », qui sert à identifier un répertoire de stockage général à Thales, utile à l'application. Cela montre le manque de portabilité du code : on devra peut-être retravailler ce fichier de configuration pour rendre le code plus modulaire et flexible.

Exécution et difficultés rencontrées

Le « make » se lance suivi de l'argument « build ». J'obtiens à tous les lancements la même erreur :

```
error: Multiple top-level packages discovered in a flat-layout:
['controllers', 'UI', ...].

To avoid accidental inclusion of unwanted files or directories,
setuptools will not proceed with this build.

If you are trying to create a single distribution with multiple packages
on purpose, you should not rely on automatic discovery.
Instead, consider the following options:

1. set up custom discovery (`find` directive with `include` or `exclude`)
2. use a `src-layout`
3. explicitly set `py_modules` or `packages` with a list of names
```

Il s'agit d'une erreur classique depuis les versions récentes de Python. En effet, comme on a pu le constater avec les fichiers d'initialisation de packages, il existe de nombreux fichiers spécifiques en Python.

Les « Python Enhancement Proposals appelés » PEP* sont des recommandations qui sont régulièrement proposées par des développeurs ou même utilisateurs dans le but d'améliorer Python. Dans le PEP 518 le fichier « pyproject.toml* » est mentionné, et il est repris plus tard dans les PEP 517, 621 et 660. Ce fichier est fait pour contenir les exigences du système pour pouvoir construire le projet et le mener à bout.

Il peut contenir de nombreux champs donnant plus ou moins d'informations sur le projet. Le champ noté « [build-system] », qui précise l'outil utilisé pour construire le projet (parmi différents packages Python), est le plus important.

L’erreur qui apparaît ici est donc liée à la construction du projet. Lorsque le « make » est lancé, le projet essaie donc de se construire pour produire les exécutables, mais sans fichier « pyproject.toml », cela est compliqué.

Enfin, lors de la compilation des contrôleurs de Graph et Depouil, les fenêtres s’ouvrent. C’est une vraie réussite. Cependant, les fenêtres crashent et se ferment toutes seules à l’ouverture ou l’importation fichier.

L’erreur affichée dans le terminal de PyCharm est la suivante :

```
Process finished with exit code -1073740791 (0xC0000409)
```

L’erreur affichée dans l’invite de commandes Windows, c’est-à-dire le terminal, en utilisant la commande Python, est :

```
Unhandled Python exception
```

C’est une erreur assez référencée dans la littérature. Officiellement, cela traduit une surcharge dans la mémoire qui ferait se terminer le processus (de ce fait, la fenêtre se ferme brutalement). Dans les faits, cela peut revenir d’une erreur

V.8 *Prochains objectifs*

Résumé des difficultés rencontrées

A l’heure actuelle, 90% de la migration sont achevés. On rappelle ici les principales difficultés.

Tout d’abord, la bibliothèque Qwt et son wrapper PyQwt qui, ne proposant que très peu de documentation, a pris du temps à migrer. La compréhension des signaux, du fait du grand nombre de références indiquant plusieurs syntaxes différentes, m’a aussi pris du temps. Enfin, certaines erreurs moins habituelles telles que celle mentionnée ci-dessus demandent un temps considérable à débuguer.

Aussi, les demandes d’installation de logiciels et les restrictions imposées par Thales par souci de sécurité ont pu être des obstacles au bon déroulement de la phase de migration.

Objectifs sur le court terme

Mes objectifs sur le court terme sont au nombre de trois.

Le point prioritaire est de pouvoir garder les fenêtres ouvertes. Cela implique donc de régler absolument cette erreur de surcharge de la mémoire. Il faut donc débuguer pas à pas les fichiers avec l’aide du debugger intégré dans PyCharm. Ensuite, il faudra implémenter le fichier « pyproject.toml » et faire fonctionner le « make ». Enfin, toutes les icônes n’apparaissent pas : lorsqu’une erreur s’affiche, avant que la fenêtre de Graph ou Depouil ne se ferment, des pop-ups apparaissent avec des espaces vides au lieu d’icônes. Ce problème devrait disparaître avec le fichier .toml*, puisqu’à ce moment-là, les dossiers seront tous reconnus pendant la construction du logiciel.

VI. Conclusion

VI.1 *Mes avancées*

Pour conclure sur mes avancées, il est certain que la phase de migration m'a pris plus de temps que prévu. Il faut néanmoins prendre en compte les restrictions techniques et technologiques, l'installation du poste de travail, les erreurs inattendues...

Cependant, le projet a bien avancé : les fenêtres apparaissent lorsque l'on compile les modules des contrôleurs principaux de Graph et Depouil. Les signaux semblent aussi fonctionner correctement car en cliquant sur un bouton tel que « ouvrir » ou « charger », les fichiers du PC apparaissent bien.

De plus, les bugs identifiés peuvent se corriger en parallèle du débogage de la compilation. J'ai déjà plusieurs idées quant à la provenance de ces bugs.

Enfin, j'ai beaucoup appris en peu de temps. Grâce au grand nombre d'erreurs rencontré, je vais maintenant beaucoup plus vite dans la correction des problèmes.

VI.2 *Perspectives d'évolution*

Afin de mener à bien ma mission, j'aimerais ne m'accorder qu'une semaine supplémentaire sur le portage du code. Je prévois en parallèle de commencer à m'intéresser aux bugs à corriger et à la manière d'implémenter les améliorations souhaitées. En effet, les deux sont compatibles, et je prévois de créer des modules à part pour régler les bugs et améliorer le code actuel de manière isolée. Ainsi, les erreurs de compilation n'entraveront pas mes tests.

Je me réserve ma dernière semaine de stage, donc à compter du 1r juillet, pour rédiger de la documentation, mes notes de version et les nouveaux tests.

VI.3 *Ressenti personnel*

Jusqu'ici, je suis très satisfaite de mon stage. Je me sens à ma place dans la vie en entreprise, ce qui confirme mon choix de faire une alternance en BUT3. L'ambiance de travail est agréable et bienveillante. L'entraide est de mise et il arrive souvent que l'on prenne du temps entre stagiaires pour réfléchir à des problèmes que d'autres rencontrent, ce qui est aussi très enrichissant.

J'ai de plus appris énormément de choses. Bien que je n'aie pas prévu de continuer dans le développement, Python est un langage très utilisé avec de nombreuses applications autre que le développement applicatif et les IHM. Ce langage n'ayant pas été beaucoup abordé en cours, ce stage est presque comme une formation en plus d'un moyen de mettre en application mes compétences.

Enfin, l'autonomie particulière demandée dans cette mission est exactement ce que je souhaitais. Le fait que mon tuteur ait confiance en moi et m'ait laissé autant de liberté quant aux technologies m'aide à m'épanouir. Ce stage aura été une formidable façon d'apprendre et une belle expérience.

VII. Glossaire

.

.exe : Raccourci de « executable », une application, un fichier que l'on peut lancer
.lru : Label Replacement Unit, fichiers de données sur les messages ARINC429
.nar : Network Application Requirements, fichiers de configuration pour ARINC429
.pyd : Fichiers DLL empaquetés dans Python
.rec : Raccourci de « record », fichier contenant un enregistrement
.toml : « Tom's Obvious Minimal Language », format pour les fichiers de configuration
.vsix : « Visual Studio Extension Installer », packages pour extensions VS Code
.xml : « Extensible Markup Language », langage à balisage pour structurer des données

2

2to3 : Script Python pour automatiser la migration de Python 2 à Python 3

A

ARINC429 : Voir page 10
Attribut de classe : Variable définie et utilisée dans une classe
AVS : Anciennement « Avionics », GBU aéronautique de Thales

B

BAST : Voir page 9
Beyond Compare : Outil de comparaison de fichiers ou dossiers
Bitbucket : Outil basé sur Git optimisé pour fonctionner avec Jira
BL : « Business Line », sous-branche d'une GBU

C

Compilation : Traduction d'un code en langage interprétable par la machine
Contrôleur (*controller*) : Intermédiaire entre l'utilisateur et l'application

D

DLL : « Dynamic Link Librairies », fichiers précompilés pour programmes Windows

F

FLX : « Flight Avioncis », BL de AVS produisant des équipements pour aéronefs

G

GBU : « Global Business Unit », domaine métier majeur de Thales, il y en a 6
Git : Outil de *versioning*, d'historisation pour enregistrer les modifications d'un code
Git Extension : Outil visuel facilitant l'utilisation de Git
GRADE : Voir page 9
GRADE Depouil : Voir page 11
GRADE Graph : Voir page 10

I

IDE : « Integrated Development Environment », outil pour programmer
IESI : « Integrated Electronic Standby Instrument », voir page 7
IHM : « Interface Humain-Machine », dispositif permettant d'interagir avec un système
IPT : « Integrated Product Team », nom général d'une équipe à Thales
IVV : « Intégration, Validation, Vérification », phase de tests d'un produit

M

main : « principal », environnement d'exécution principal pour lancer l'application
Matplotlib : Bibliothèque Python pour visualiser des données
Méta-donnée(s) : Donne des informations sur une donnée comme son format
Module : Fichier contenant du code Python, des fonctions, des classes entières…
MVC : « Modèle-Vue-Contrôleur », modèle d'architecture logicielle classique
MVCR : « Microsoft Visual C++ Redistributable », ensemble de bibliothèques et d'outils permettant d'installer, compiler et exécuter des applications écrites en C++

N

(tests de) non-régression : Tests pour garantir que les modifications dans un code n'ont pas introduit de nouveaux bugs
NumPy : Bibliothèque d'aide aux calculs numériques et multidimensionnels pour Python

P

Package : Ensemble de modules, fonctionne comme un dossier / Extensions de Python
PEP : « Python Enhancement Proposals », ensemble de propositions de design du langage
pip : Gestionnaire pour les packages Python, permettant de les installer/désinstaller
PyCharm : IDE développé par JetBrains pour faciliter la programmation en Python
PyPi ; pypi.org : Index des packages Python
pyproject : Fichier de configuration minimal pour construire un projet Python
PyQtEnumConverter : Script Python automatisant la conversion des types « enum »
PyQt : Bibliothèque IHM pour Python empaquetant la bibliothèque Qt écrite en C++
PyQtGraph : Bibliothèque Python permettant des graphiques scientifiques et de l'IHM
PyQwt : Ancien wrapper de Qwt pour Python
PySide : Alternative à PyQt
PythonQwt : Nouveau wrapper pour la bibliothèque Qwt, recrée PyQwt

Q

Qt : API orientée objet développée en C++ permettant la création d'applications
Qwt : Bibliothèque contenant des widgets pour l'IHM et permettant des graphiques

S

Signaux : Objet Python permettant de déclencher un slot ou évènement
Slots : Evènement, action effectuée dans le code, visible ou non par l'utilisateur

V

VS Code : « Visual Studio Code », IDE général (contrairement à PyCharm)

W

Wheel : Fichier binaire précompilé pour distribuer des packages Python
Widgets : Elément graphique interactif, composant d'une application
Wrapper : Permet d'empaqueter un code écrit dans un langage en général plus bas niveau pour être utilisé dans un code haut niveau

VIII. Sitographie

Ci-dessous quelques références bibliographiques en ligne utilisées dans mes recherches, par ordre d'apparition.

Logiciel BAST et ARINC429

[1] TechSAT. Analyseurs ARINC 429 – NeoMore, les analyseurs de TechSAT [en ligne]. Documentation technique disponible en anglais sur : https://neomore.com/test-et-mesure/arinc-429-analyseur- en cliquant sur ARINC429-BAST (consulté le 08/05/2024)

[2] Wikipédia. ARINC 429 [en ligne] Disponible sur : ARINC 429 — Wikipédia (wikipedia.org) (consulté le 09/05/2024)

Fichiers DLL et .pyd

StackOverflow : stackoverflow.com

[3] What exactly are DLL files, and how do they work ? [en ligne] https://stackoverflow.com/questions/124549/what-exactly-are-dll-files-and-how-do-they-work (consulté le 18/04/2024)

[4] Error "ImportError : DLL Load Failed: %1 is not a valid Win32 application" [en ligne] https://stackoverflow.com/questions/14629818/error-importerror-dll-load-failed-1-is-not-a-valid-win32-application (consulté le 18/04/2024)

Python 3

Python documentation : docs.python.org.

[5] 2to3 – Automated Python 2 to 3 code translation – Python 3.12.4 documentation [en ligne] 2to3 — Automated Python 2 to 3 code translation — Python 3.12.4 documentation (consulté le 22/04/2024)

StackOverflow : stackoverflow.com

[6] How do I overwrite method in python? [en ligne] https://stackoverflow.com/questions/37625866/how-do-i-overwrite-method-in-python (consulté le 04/06/2024)

[7] Inheritance – Python Method overriding, does signature matter? [en ligne] https://stackoverflow.com/questions/6034662/python-method-overriding-does-signature-matter (consulté le 05/06/2024)

PyQt

[8] PythonGUIs : pythonguis.com. PyQt5 vs PyQt6: What are the differences, and is it time to upgrade? [en ligne]
https://www.pythonguis.com/faq/pyqt5-vs-pyqt6/#:~:text=The%20upgrade%20path%20from%20PyQt5,both%20PyQt%20and%20Qt%20itself. (consulté le 22/05/2024)

[9] PyPi : pypi.org. PyQtEnumConverter
https://pypi.org/project/PyQtEnumConverter/#description (consulté le 22/05/2024)

StackOverflow

[10] Python – How to connect pyqtSignal between classes in PyQT [en ligne] https://stackoverflow.com/questions/3891465/how-to-connect-pyqtsignal-between-classes-in-pyqt (consulté le 26/04/2024)

PyQwt

[11] Github : github.com. GauiStory/PyQt-Qwt : Python PyQt wrapper for Qwt6 [en ligne] https://github.com/GauiStori/PyQt-Qwt (consulté le 13/05/2024)

[12] PyPi : pypi.org. PythonQwt [en ligne] PythonQwt · PyPI (consulté le 23/05/2024)

IX. Résumé

GRADE est un logiciel de visualisation graphique et de dépouillement par lot d'enregistrements ARINC. L'objectif est de le remettre à niveau : il est écrit en Python 2 et utilise des bibliothèques C++ pour l'IHM, Qwt5 et PyQt4, très obsolètes. Je l'ai d'abord passé en Python 3.9 automatiquement grâce au module 2to3. Le wrapper de Qwt, PyQwt, avait été déprécié et non porté après PyQt4. J'ai donc utilisé le nouvel équivalent, PythonQwt, qui a très bien fonctionné. Il me fallait ensuite passer de PyQt5 à PyQt6. Passer à PyQt5 a impliqué un changement de syntaxe et dans le comportement des signaux. Je les ai donc écrits correctement, en les définissant au préalable et en changeant la syntaxe. Passer à PyQt6 impliquait également un travail sur la syntaxe, très laborieux. J'ai utilisé un script d'automatisation appelé PyQtEnumConverter. Ayant tout mis à jour, j'ai lancé la compilation. Il me reste maintenant à corriger les erreurs de compilation, les bugs divers et à passer les tests de non régression, tout en tenant à jour la documentation technique. Cette mission aura été très stimulante et enrichissante.

Mots clés :

Aéronautique, Qt, PyQt, Qwt, 2to3, logiciel graphique, dépouillement, migration, obsolète, amélioration, correction, bugs, Python, PythonQwt, ARINC429, IHM, wrapper, compilation

Abstract :

abstract: In the aeronautics field, equipments and technologies have to last. GRADE is a visualization and reduction software and its last updates is from 2017. In order to make it fit the always-growing needs of performance, we have to upgrade it. Here, we reflect on the migration of the code from Python 2 to Python 3 and its libraries, PythonQwt and PyQt. We first analyzed the code and used the 2to3 module to upgrade the language. Then, we moved onto the librairies. PyQwt was the official wrapper of the C++ UI library Qwt but knowing it has been deprecated, we used PythonQwt. We installed it with pip. We finished the migration phase by porting PyQt to 5 then 6 version. We mostly changed the syntax and defined the signals for the signals-slots functionality of PyQt. We finally started running the main file. We discuss how now how to fix the errors in the most efficient way, whether they are from the compilation or in the code in order to obtain the executables of GRADE.

Keywords :

Keywords: Aeronautics, visualization software, reduction software, Qt, PyQt, Qwt, 2to3, UI, deprecated, upgrading, migrating, patching, bugs, Python, PythonQwt, wrapper, compilation