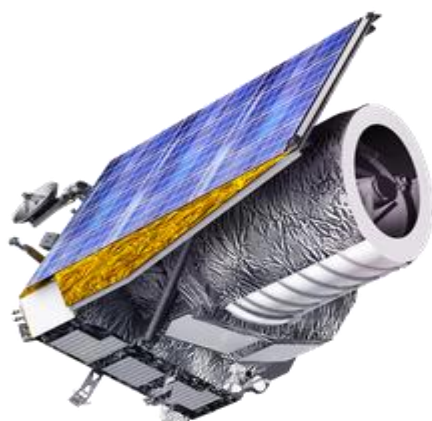




Université Gustave Eiffel

RAPPORT DE PROJET

**Projet : Outils automatique pour la qualité
dans Euclid**



**Master 1 Electronique, Énergie Électrique et Automatique,
2020/2021**

Étudiants :

LESZCZYNSKI Jimmy

ALI Moïse

ABDOULALIME Javeed

Tutrice de projet :

GAUTARD Valérie

Remerciements

Pour nous avoir accompagné tout au long de ce projet et avec des directives précises, nous tenons à dire un très grand merci à notre tutrice, Mme GAUTARD ! Elle a su nous guider pour mener à bien l'ensemble des tâches que nous avons réalisées lors de ce projet et nous a toujours encouragé.

Un remerciement également à Marc DEXET, que nous n'avons pu rencontrer mais qui est à l'origine du programme SARA et sans qui nous n'aurions pas pu réaliser ce projet.

Ensuite, nous souhaitons également remercier M. TAKHEDMIT, notre responsable de Master, pour nous avoir permis de choisir ce projet qui faisait partie de nos premiers choix et aussi pour ses retours lors de la soutenance à mi-parcours.

Nous remercions aussi M. CARRER, pour nous avoir permis durant la soutenance de mi projet de nous rendre compte des améliorations qu'on devait apporter dans nos explications pour en faciliter la compréhension.

Enfin, nous remercions Phuc-Toan NGUYEN pour ses retours sur le projet et ses questions qui nous ont permis d'améliorer nos présentations/explications et lui souhaitons bon courage dans la reprise de SARA.

Nous remercions aussi l'équipe d'Euclid France et les chercheurs du CEA qui ont porté attention à notre projet.

SOMMAIRE

1) Introduction (présentation du contexte de la mission Euclid)	4
2) SARA (Software Analysis and Report Automation – Marc Dexet)	6
a) Découverte du programme et prise en main des outils nécessaires au lancement de SARA	6
i) Git/Github	6
ii) Python	7
iii) Pycharm	8
iv) Installation	8
b) Compréhension des différents fichiers composant SARA	9
i) AsciiDoc (Template)	9
ii) Jinja2 (.yaml et .yml)	10
iii) Fonctionnement de SARA	10
iv) Exécution de SARA	12
3) Conversion pdf	12
a) Récupération du résultat du terminal dans un fichier asciidoc	12
i) Modification dans le fichier <i>cmd.py</i>	12
ii) Création de fichier sous Python	13
b) Génération du pdf	13
i) Ruby	14
ii) AsciiDoctor-pdf	14
iii) Os.system	15
iv) Génération du fichier pdf	15
4) Conversion WORD	18
a) Conversion vers un fichier docx (WORD)	18
i) Pandoc	18
ii) Conversion asciidoc vers docx	18
b) Problèmes liés à Pandoc pour la conversion vers Word	21
c) Mise en page du fichier généré	24
i) Affichage des styles et des tableaux	24
ii) Mise en page (<i>custom reference</i>)	25
5) Conversion EXCEL	28
a) Pandas	28
i) Fonctionnement	28
b) Génération des tableaux au format EXCEL	29
6) Présentations	32
7) Conclusion	32
8) Annexes	34
a) Lien Github	34
b) Tutoriel du lancement de SARA	34
c) Première page du pdf	35
d) Exemple de génération WORD avec plusieurs styles	36
e) Exemple de génération WORD avec rendus propre des tableaux imbriqués	37
Bibliographie/Sitographie	38
Tableaux des figures	39

1) Introduction (présentation du contexte de la mission Euclid)

Contexte du projet

Euclid est une mission organisée par l'Agence Spatiale Européenne (ESA) ayant pour but de lancer en 2022 un télescope spatial nommé Euclid. Ce télescope permettra de cartographier les propriétés de plus de deux milliards de galaxies, ce qui correspond à environ un tiers de la sphère céleste.

A ce jour, seulement 4% de la matière de l'univers (cf. figure 1) est connue, les 96% restants se partagent entre la matière noire (22%) et l'énergie sombre (74%) caractérisées par une pression négative similaire à une force gravitationnelle répulsive. L'objectif de cette mission est d'étudier l'énergie sombre et la matière noire.

L'expansion de l'univers devrait ralentir mais elle semble pourtant accélérer au cours du temps, l'Agence Spatiale Européenne (ESA) cherche à déterminer si cette accélération est due à la présence de cette « énergie sombre » ou non.

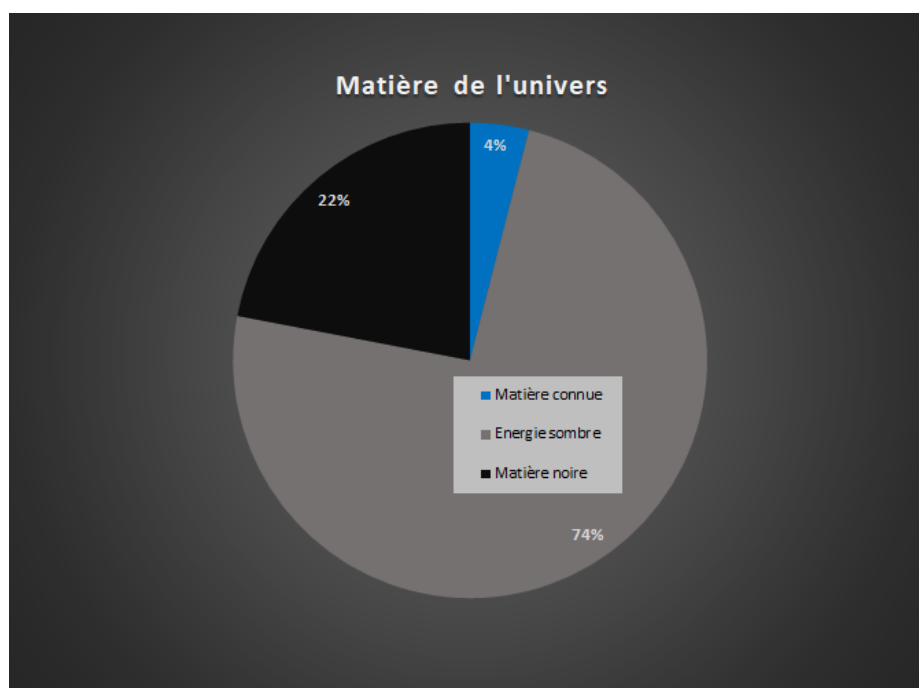


Figure 1 : Les trois types de matière composées par l'univers [1]

Membres de la mission Euclid

La mission Euclid est une mission majeure du programme Cosmic Vision de l'Agence Spatiale Européenne (ESA) qui regroupe de nombreux ingénieurs, techniciens et plus de 1500 chercheurs répartis en Europe et également aux Etats-Unis et au Canada. Une centaine de laboratoires appartiennent au consortium Euclid et une partie d'entre eux développent des logiciels afin d'analyser les données recueillies par la mission Euclid.

Enjeux du projet

La phase d'analyse est planifiée pour durer une dizaine d'années. Les logiciels correspondants ainsi que leur documentation, pendant leur développement, sont évalués une fois par an par des personnes chargées de faire des rapports de qualité formant une équipe dénommée PAQA (Product Assurance Quality Assurance). Cette évaluation permet d'attribuer des niveaux de maturité pour les logiciels élaborés ainsi que les documentations associées. Les évaluations sont chronophages en tâche répétitive laissant peu de temps pour les tâches à plus haute valeur ajoutée

Afin de limiter les interventions humaines pour mener cette charge redondante, nous allons automatiser la génération des rapports de qualité pour faciliter et optimiser les tâches de la mission Euclid.

Notre participation au projet sera encadrée par Mme GAUTARD Valérie du CEA (Commissariat à l'Energie Atomique et aux Énergies Alternatives).

Objectifs du projet

L'objectif de ce projet est de développer SARA. (Software Analysis and Report Automation) SARA est un programme écrit par Marc Dexet (Développeur de l'Institut d'Astrophysique Spatial (IAS)) dans le cadre de la mission Euclid. Le projet a débuté en 2020. L'objectif principal de SARA est de générer automatiquement des rapports de qualité après avoir analysé les logiciels et leur documentation. Le programme doit donc répondre à des besoins précis afin d'optimiser au maximum le temps.

Notre objectif au sein de ce projet sera de compléter le programme du logiciel SARA afin de générer automatiquement un rapport de qualité sous le format Word imposé par le consortium Euclid de l'Agence Spatiale Européenne (ESA). Par ailleurs, un des objectifs de ce projet pour la mission Euclid est de vérifier le respect des normes données pour la documentation ou le code d'un logiciel mais d'autres membres de l'équipe s'y consacreront.

Dans un premier temps nous allons analyser le logiciel SARA pour mieux aborder le projet. Ensuite, nous réaliserons une étape intermédiaire avec la conversion vers le format pdf pour mieux visualiser le rendu attendu. Puis, nous réaliserons l'automatisation de la génération des rapports de qualité sous format Word. Enfin, nous présenterons une solution pour la conversion automatique des tableaux issus des rapports de qualités en format Excel.

2) SARA (Software Analysis and Report Automation – Marc Dextet)

a) Découverte du programme et prise en main des outils nécessaires au lancement de SARA

Le lancement de SARA nécessitant l'utilisation de divers outils dont nous ignorions pour certains l'existence même, cela nous amène à nous documenter et à nous questionner sur diverses problématiques. Pour cela, nous avons procédé en plusieurs étapes en nous fixant des objectifs bien précis.

i) Git/Github

Dans l'optique de débiter le projet, nous nous sommes initiés à Git pour installer la version de SARA programmée par les personnes ayant travaillé sur le projet avant nous. GitHub est un site web et un service de cloud permettant aux développeurs d'héberger ainsi que d'assurer la gestion de leurs codes. L'intérêt de Github est de pouvoir partager son programme avec d'autres utilisateurs afin de collaborer sur un même projet. C'est un outil efficace permettant de contrôler et observer les différentes modifications apportées par chaque utilisateur au cours du temps.

Il existe de nombreuses commandes Git, il est possible de les taper directement depuis n'importe quel terminal ou bien de télécharger la version bureau du logiciel Github pour profiter des mêmes fonctionnalités. L'utilisation du logiciel est pratique et rapide et ne nécessite pas de connaître les commandes Git. On peut donc cloner (*git clone*) un répertoire depuis le serveur distant, créer des branches secondaires (*git branch*), effectuer des modifications et les envoyer (*git push*) sur Github ou encore récupérer (*git pull*) les changements apportés par d'autres collaborateurs en quelques clics. Pour cela, il suffit de créer un compte Github afin de commencer à collaborer sur un projet.

Lors de la prise en main du logiciel, nous avons effectué divers tests pour nous familiariser avec les notions de branches principales et secondaires. L'idée étant de pouvoir faire des modifications sans affecter la branche principale de notre projet. Lorsqu'une branche secondaire répond à nos attentes, il est alors possible de la "fusionner" (*git merge*) avec la branche principale.

On peut voir ci-dessous sur la figure 2 un schéma de fonctionnement des commandes Git :

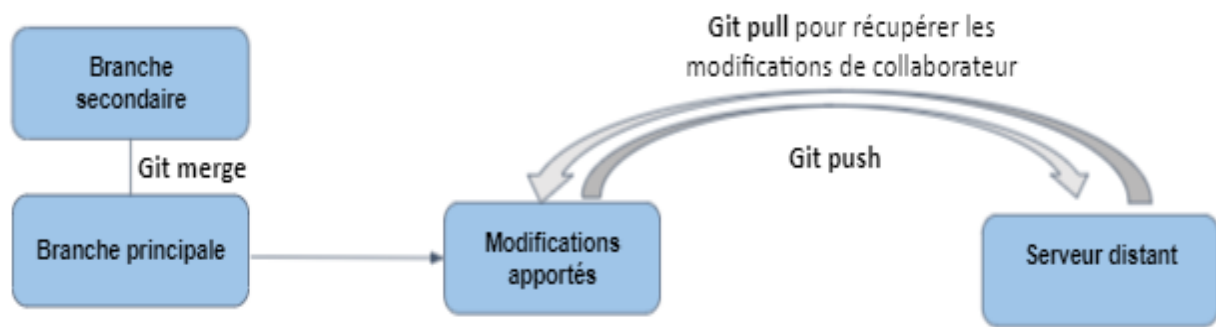


Figure 2 : Schéma explicatif des commandes Git utilisées

ii) Python

SARA est un programme écrit en Python, c'est la raison pour laquelle, il a fallu dans un premier temps apprendre les bases de ce langage pour pouvoir comprendre le programme dans son ensemble.

Python est un langage de programmation créé par Guido Van Rossum en 1991. Il s'agit d'un langage de haut niveau largement utilisé et facile à mettre en œuvre avec une syntaxe particulièrement simple à comprendre. Étant un langage interprété, il ne nécessite pas de compilation intermédiaire pour être exécuté contrairement à d'autres langages tels que le C.

De nombreuses applications sont offertes à travers Python telles que le calcul numérique, la création de jeu, l'intelligence artificielle ou l'automatisation. En effet, il y a des bibliothèques adaptées pour chacune de ces applications. Nous utiliserons ainsi divers modules permettant l'automatisation dans le cadre de notre projet. Malgré le fait que Python ne soit pas l'outil le plus puissant (contrairement à d'autres langages tel que le C++), il remplit les conditions nécessaires pour l'objectif qu'on s'est fixé, à savoir, l'automatisation de la génération des rapports qualité.

De plus, un autre avantage de Python est qu'il est multiplateforme, c'est-à-dire qu'il est disponible sur la plupart des plateformes (Windows, Mac OS, Linux, etc.) et donc on peut l'utiliser librement quels que soient les outils dont on dispose.

Enfin, l'installation de Python est gratuite car il s'agit d'une licence libre. En effet, il y a une grande communauté de développeurs Python qui participent régulièrement à des projets visant à améliorer l'expérience de chacun dans l'utilisation de Python dans le cadre de projets divers.

iii) Pycharm

Il est possible de programmer en Python depuis n'importe quel outil de traitement de texte après avoir installé Python. Il suffit pour cela de créer un nouveau fichier avec l'extension `.py` et de l'exécuter. Cependant, pour faciliter la création de projet, il est parfois nécessaire d'utiliser des outils plus adaptés que de simples éditeurs de texte.

En effet, lors d'un projet, on est systématiquement amené à travailler sur plusieurs fichiers et de régulièrement tester notre programme pour vérifier si nos modifications fonctionnent. Il devient alors très difficile de s'y retrouver et on perd énormément de temps. Pour pallier ce problème, on utilise un environnement de développement (IDE, *Integrated Development Environment*). Il s'agit d'un logiciel qui regroupe tous les outils nécessaires à la création de projet.

L'environnement de développement regroupe généralement un éditeur de texte, un terminal et un gestionnaire de fichier permettant de naviguer dans l'arborescence des dossiers et des fichiers que comporte un projet.

De plus, selon l'environnement de développement que l'on installe, il est possible de télécharger les différents modules que l'on souhaite utiliser dans notre programme directement depuis l'IDE. Cela nous évite d'aller chercher à chaque fois sur internet les modules que l'on veut installer.

Il existe de nombreux environnements de développement et il nous a fallu en choisir un parmi ceux qui existent. Nous nous sommes donc tournés vers Pycharm. La raison de ce choix est que cet IDE est particulièrement adapté pour le langage Python puisqu'il offre la possibilité d'installer et d'utiliser un très grand nombre de modules en quelques clics. Etant donné que le programme SARA utilise une grande variété de bibliothèques disponibles à travers Pycharm, on gagne énormément de temps dans l'installation des différents modules.

iv) Installation

Maintenant que nous avons compris l'intérêt de ces divers outils pour l'utilisation de SARA, on peut procéder à leur installation. Pour cela, nous avons suivi étape par étape les indications données sur le *README* du lien Github sur lequel nous avons récupéré le programme (voir annexe).

Le programme est composé de plusieurs dossiers et fichiers. Ainsi, dans un premier temps, sans en avoir la compréhension, nous avons cherché à le lancer correctement pour obtenir un résultat. Nous avons donc lancé les différentes commandes indiquées sur le *README*.

Parmi les étapes préliminaires, il nous a fallu installer les différents modules nécessaires au fonctionnement du programme. Nous avons donc installé une liste de modules qui se trouvent dans un fichier qui s'appelle *"Requirements.txt"*. Pour cela, on utilise la commande suivante : `"pip install -r Requirements.txt"`

Pip est un gestionnaire de paquets qui nous permet d'installer les modules que contenait le fichier texte. Cette commande permet donc d'installer de façon récursive (avec l'option "-r") chaque module spécifié dans le fichier "*Requirements.txt*".

On a ainsi pu lancer la commande pour exécuter le programme. Cependant, une erreur indiquait que certains modules étaient manquants et nous devions les installer pour obtenir un résultat. En regardant attentivement, les modules manquants étaient ceux qu'on venait d'installer avec la commande décrite précédemment. Il suffisait en fait de redémarrer Pycharm pour qu'il prenne en compte les installations.

Enfin, on a relancé la commande d'exécution du programme et cette fois, tout fonctionne correctement. Nous avons lancé SARA pour la première fois et obtenu un résultat sous forme de texte sur le terminal de Pycharm.

b) Compréhension des différents fichiers composant SARA

Après nous être familiarisés avec les outils permettant d'utiliser SARA et suite à un premier lancement du programme, ce dernier a généré un texte semblable à un rapport dans le terminal. Nous avons donc été amenés à rechercher la nature des fichiers qui composent SARA et à comprendre comment est-ce que le programme fonctionne.

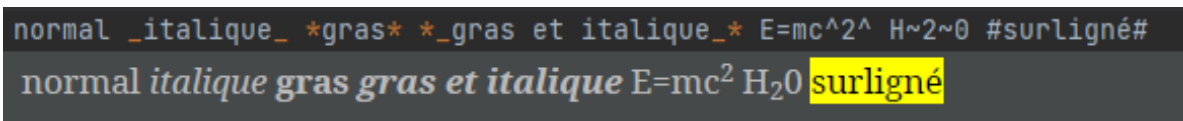
i) AsciiDoc (Template)

L'AsciiDoc est un langage de balisage léger qui est principalement utilisé pour écrire des documentations ou encore des articles [1]. Il offre de nombreuses options de traitement de texte et de mise en page. Aussi, il se présente sous forme de programme mettant en avant une syntaxe extrêmement simple, une personne non formée à son utilisation pourrait rapidement comprendre le fonctionnement d'un programme AsciiDoc.

Le langage AsciiDoc permet entre autres de manipuler des blocs de texte, il est par exemple possible de mettre un mot ou une phrase en gras en l'entourant de la balise « * ». On peut aussi les surligner en les entourant des balises « # ». Les balises sont pour la majorité personnalisables, on pourrait donc changer la balise utilisée pour mettre un mot en gras sous réserve de respecter certaines conditions décrites dans la documentation.

De plus, il est possible d'appliquer des styles à tout type d'élément, de citer d'autres fichiers AsciiDoc, de manipuler des listes, des tableaux, etc. Pour ces derniers, on a la possibilité de modifier la largeur des colonnes et des lignes, de fusionner et dupliquer des cellules, d'imbriquer des tableaux, de gérer l'affichage des bordures, etc.

Au sein de ce projet, nous utilisons les fichiers AsciiDoc comme des templates. Un template est un modèle qui associe une partie fixe et une partie variable. Voici un exemple de code asciidoc ci-dessous (cf. figure 3) avec son rendu illustrant quelques styles appliqués à du texte suivi d'un échantillon de notre template :



```
normal _italique_ *_gras* *_gras et italique_* E=mc^2^ H~2~0 #surligné#
```

normal italique gras gras et italique E=mc² H₂O surligné

Figure 3 : Exemple de code AsciiDoc avec son rendu

Nous allons maintenant nous intéresser au module qui permet de remplir la partie variable d'un template : Jinja2.

ii) Jinja2 (.yaml et .yml)

Jinja2 est un module du langage de programmation Python dont l'utilité principale est de manipuler des templates [2]. On reconnaît rapidement la syntaxe de Jinja2 qui se présente principalement sous ces deux formes : `{{commande}}` et `{%commande%}`.

Il offre la possibilité de créer des templates, de charger un template dans une variable ou encore de combiner plusieurs templates. Cependant, c'est principalement pour effectuer un rendu que Jinja2 est efficace. En effet, un rendu consiste à remplacer la partie variable d'un template par des informations qui sont contenues dans d'autres fichiers comme on peut le voir sur la figure 4. C'est précisément pour effectuer cette tâche que SARA utilise Jinja2.

```

47  === Information on the project and product analysed
48  //=== titre de section de niveau 3 (car il y a 3 =)
49
50  ==== Context of the analysis: periodic software quality analysis
51  //==== titre de section de niveau 4 (ce titre est un sous-titre par rapport au précédent)
52  This analysis has been done within the frame of periodic software quality analyses,
53  at least one per year, or one per minor version number (m in number version M.m.p).
54
55  ==== Development team and stakeholders
56  //==== titre de section de niveau 4
57  The table below lists the main stakeholders of the product analysed:
58
59  [width="100%"]
60  //Option permettant de choisir la largeur du tableau par rapport à la page
61  |=====
62  //ouverture du tableau. | marque une nouvelle colonne, un retour à la ligne marque une nouvelle ligne du tableau.
63  | Product lead                | {{project.leaders | join(', ', attribute='name')}}
64  | Product implementation lead | {{project.implementation_leaders | join(', ', attribute='name')}}
65  | Product validation lead     | {{project.validation_leaders | join(', ', attribute='name')}}
66  |=====
67  //fermeture du tableau

```

Partie variable du template

Figure 4 : Partie variable du template qui sera remplacée par des informations issues d'autres fichiers

iii) Fonctionnement de SARA

Après des heures passées à rechercher les liens qu'ont les divers fichiers de SARA entre eux ainsi que le fonctionnement du code, nous sommes arrivés à une compréhension globale du fonctionnement de SARA.

SARA est un programme dont un des objectifs est de générer un rapport de qualité. Son fonctionnement est assez simple. En combinant l'utilisation de Jinja2 et de fichiers AsciiDoc, SARA charge un template AsciiDoc dans une variable. Puis il remplit la partie variable du template (modèle) à l'aide d'informations précises qui sont contenues dans deux fichiers : un fichier du nom de `"project.yml"` et un fichier du nom de `"document.yml"`. On obtient alors un template rempli qui correspond à un rapport.

On retiendra pour la suite que le template rempli est le fichier AsciiDoc généré lors de l'utilisation de SARA et qui s'apparente à un rapport de qualité.

On peut voir sur la figure 5 et 6, un schéma logique et une illustration du fonctionnement de SARA :

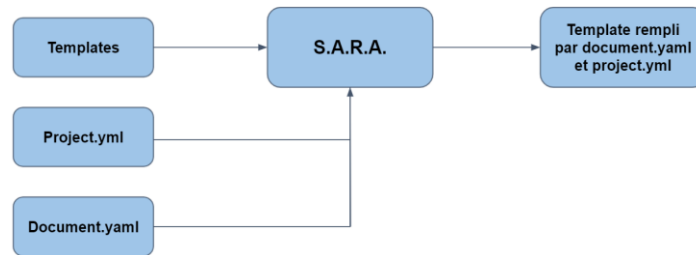


Figure 5 : Schéma explicatif représentant le fonctionnement de SARA



Figure 6 : Schéma illustrant le fonctionnement de SARA avec les différents fichiers

iv) Exécution de SARA

Le fichier principal de SARA est le fichier Python nommé *cmd.py*. C'est lui qui contient toute la partie programmation de SARA (outre les templates). On l'exécute à l'aide de la commande suivante :

```
py sara/cmd.py render --location=sara/templates --project=demo/project.yml --  
document=demo/document.yml --template=software_review_master.adoc
```

Le programme commence par récupérer les cinq *arguments* qui sont surlignés. Le premier *argument* peut prendre deux valeurs : *render* ou *sample*. La valeur *sample* permet d'obtenir un échantillon, un exemple de rapport. *Render*, l'autre valeur possible, nous intéresse particulièrement puisqu'elle nous permet d'indiquer au programme que l'on va générer un rapport. Le deuxième *argument* récupéré par le programme contient l'emplacement des différents templates (modèles) qu'il chargera par la suite. Le troisième et le quatrième *argument* donnent respectivement l'emplacement du fichier *project.yml* et du fichier *document.yml*. Ce sont les deux fichiers qui contiennent les informations permettant de remplir la partie variable du template. Enfin, le dernier *argument* indique au programme le nom du template qu'il doit charger et dont il doit remplacer la partie variable.

On peut noter que dans notre cas, le template "*software_review_master.adoc*" est constitué de plusieurs autres templates mis les uns à la suite des autres, c'est l'emplacement de tous ces modèles qui est renseigné par le deuxième *argument*. De plus, grâce à l'ingéniosité du programme initié par M. Dexet, les fichiers sont modulables. C'est-à-dire qu'il est possible d'appeler d'autres fichiers que les fichiers *project.yml* et *document.yml* pour remplir le template, sous réserve qu'ils aient le bon format. Aussi, il est possible de choisir un autre template à remplir. Nous avons réussi à remplir notre template, il nous faut alors obtenir un rendu du programme AsciiDoc. C'est pourquoi nous cherchons maintenant à convertir ce fichier AsciiDoc au format pdf.

3) Conversion pdf

Le premier travail qui nous a été assigné lors de ce projet a été de modifier le programme SARA dans le but de générer automatiquement un rapport sous format pdf. Pour mener cette tâche à bien, nous avons dû modifier le programme afin d'enregistrer le template rempli dans un fichier AsciiDoc. Puis, nous nous sommes heurtés à divers problèmes lors de la conversion du fichier AsciiDoc vers le format pdf.

a) Récupération du résultat du terminal dans un fichier asciidoc

Lorsque nous avons pris le projet en main, le template rempli était directement *généré* dans le *terminal* nous ne pouvions alors pas faire la conversion à l'aide de ce qui était *généré* dans le *terminal*.

i) Modification dans le fichier *cmd.py*

Dans le fichier (*cmd.py*) principal du logiciel SARA, nous avons réussi après analyse des différents fichiers, à trouver la partie du programme (cf. figure 7) où la partie variable du template est modifiée puis imprimée dans une variable.

```
result = self._template.render(args)
print(result)
```

Figure 7 : Code pour le remplissage de la partie variable et son affichage

Ainsi, grâce à cette variable, nous avons décidé d'*imprimer* dans un fichier asciidoc l'intégralité du contenu de cette variable.

ii) Création de fichier sous Python

Pour pouvoir créer ce fichier nous avons dû réaliser un petit code où dans un premier temps on utilise une fonction propre à python qui est *open()* comme on peut le voir sur la figure 8 :

```
w = open("temporaryrender.adoc", "w")
w.write(result)
w.close()
```

Figure 8 : Code pour l'ouverture et la création du fichier asciidoc

Cette fonction permet d'ouvrir un fichier. Elle prend en premier *argument* (où on a écrit "*temporaryrender.adoc*") le nom du fichier avec l'extension que l'on désire ouvrir. Le second *argument* permet de choisir entre deux types d'ouvertures.

Nous pouvons soit ouvrir le fichier en *lecture* avec le caractère "*r*" ou en écriture avec "*w*". Notons que si le fichier n'existe pas et que l'on a choisi de l'ouvrir en *écriture*, Python le crée alors automatiquement.

Dans notre cas, comme on a besoin de remplir un fichier, nous ouvrons donc en écriture à l'aide de "*w*". On stocke ainsi le fichier ouvert dans une variable que l'on a ici nommée "*w*" pour pouvoir lui appliquer des instructions. Enfin, il suffit de remplir le fichier vide "*temporaryrender.adoc*" avec l'ensemble du template rempli, sauvegardé dans la variable "*result*". On utilise alors la fonction *write()* afin d'écrire dans le fichier stocké dans la variable "*w*".

b) Génération du pdf

Lorsque nous nous sommes lancés dans la tâche qui consiste à convertir notre rapport au format pdf, nous nous sommes aperçus qu'à ce jour, le seul moyen de convertir un fichier Asciidoc vers un autre format est d'utiliser le module Asciidoctor. Asciidoctor est le module de traitement principal du langage AsciiDoc, il sert à convertir un fichier de format AsciiDoc en format pdf, HTML, EPUB3 ou encore DocBook [3]. Cependant, ce module n'existe pas sous Python, il nous était alors impossible de procéder à la conversion du fichier Asciidoc vers le format pdf. C'est pourquoi nous avons cherché un autre moyen que Python pour utiliser asciidoctor.

i) Ruby

Pour pouvoir utiliser le module AsciiDoctor et réaliser notre conversion vers le format pdf nous avons, après recherches, trouvé Ruby.

Ruby est un langage de programmation libre, interprété, orienté objet ainsi que multiparadigmes (manière de programmer sur plusieurs principes et théories) [4].

C'est un langage disposant d'une syntaxe facile à manipuler aussi bien pour l'écriture que la lecture et qui est utilisé pour la programmation d'applications web [5]. Ruby ressemble beaucoup au langage Python, c'est un langage de haut niveau écrit en C/C++, donc très orienté objet.

Dans notre cas, nous n'allons pas programmer à l'aide du langage Ruby mais plutôt l'utiliser pour pouvoir installer des modules grâce à ses bibliothèques appelées "gems".

Les gems sont conçus pour améliorer les capacités d'un langage pour répondre aux exigences des développeurs. Elles peuvent simplifier les fonctionnalités existantes comme par exemple s'il est nécessaire de manipuler des fichiers ou d'ajouter d'autres fonctionnalités comme transformer un texte en ASCII art (création d'image à l'aide de lettres et caractères spéciaux du code ASCII).

Les gems sont donc en quelque sorte des "zips" qui sont fournis par la communauté de Ruby. Ces dernières contiendraient un module à installer [6]. C'est grâce à cela que l'on a pu télécharger le module asciidoctor sur le terminal de Pycharm, à l'aide des deux commandes suivantes :

→ *gem install asciidoctor* (permet l'installation d'asciidoctor)

→ *gem install asciidoctor-pdf* (permet l'installation du module de conversion vers le pdf d'asciidoctor)

ii) AsciiDoctor-pdf

Une fois le module asciidoctor installé, nous avons cherché la commande permettant de réaliser la conversion d'un fichier asciidoc vers le format pdf.

Nous avons donc utilisé la commande suivante en indiquant le fichier "*temporaryrender.adoc*" que nous avons créé lors de la modification du fichier principal (*cmd.py*) de Python :

→ *asciidoctor-pdf temporaryrender.adoc*

Cette commande permet de convertir le fichier "*temporaryrender.adoc*" qui contient le template rempli / rapport, du format asciidoc vers le format pdf.

iii) `os.system`

L'utilisation de commandes sur le terminal est une tâche que nous voulons éviter à tout prix.

En effet, nous cherchons à automatiser la génération des rapports ce qui explique qu'il faut éviter le plus possible les interventions manuelles quelles que soient les tâches liées aux rapports de qualité donc pour la conversion et la génération des rapports dans notre cas.

Or, pour convertir notre template rempli vers le format pdf, nous devons utiliser l'invite de commande pour lancer AsciiDoctor à travers Ruby.

Ainsi, nous avons poursuivi nos recherches afin d'automatiser cette conversion et nous avons trouvé un autre module appelé "`OS`" qui est un module fourni par Python et qui permet d'interagir avec le système d'exploitation [7].

Ce module permet de gérer de nombreuses informations concernant par exemple l'arborescence des fichiers, ou obtenir des informations concernant le système d'exploitation, créer un répertoire, vérifier si un répertoire existe ou non, afficher tous les fichiers d'une extension spécifique et bien plus encore.

Dans notre projet nous utilisons le module `OS` pour pouvoir, directement dans notre programme Python, lancer les commandes que nous écrivons dans le terminal afin d'utiliser AsciiDoctor [8].

Pour pouvoir utiliser ce module il nous suffit de l'importer dans notre fichier python à l'aide de "`import os`" et nous fournissons le nom de la commande au système d'exploitation sous la forme de chaîne de caractères :

→ `os.system("commande que l'on veut écrire sur le terminal de la machine")`

iv) Génération du fichier pdf

Après avoir créé un fichier asciidoc "`temporaryrender.adoc`" contenant notre template modifié, puis avoir trouvé un moyen d'installer le module asciidoctor à travers Ruby, il ne nous reste plus qu'à réaliser la conversion du fichier AsciiDoc vers le format pdf.

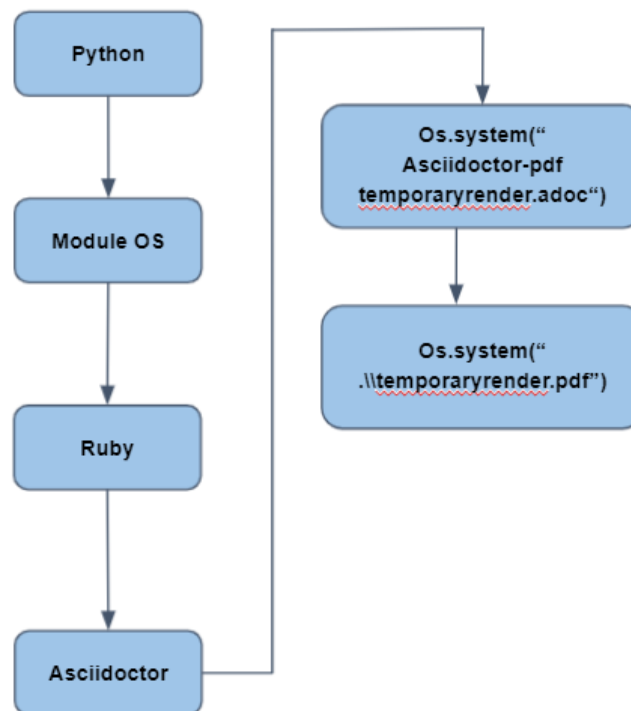
Nous utilisons donc le module `OS` présenté précédemment dans notre programme python comme on peut voir ci-dessous :

Enfin on génère le fichier pdf grâce à une seconde commande (cf. figure 9) qui permet d'ouvrir le fichier pdf par le lecteur pdf défini par défaut de notre ordinateur :

```
os.system(".\temporaryrender.pdf")
```

Figure 9 : Code pour l'ouverture du fichier pdf

Nous avons illustré (cf. figure 10) le cheminement de la conversion et la génération de notre fichier pdf de la façon suivante :



:

Figure 10 : Schéma explicatif pour la conversion en format pdf

Ainsi, on peut voir ci-dessous sur la figure 11 le remplissage de la partie variable expliqué précédemment à l'aide de Jinja et des fichiers *project.yml* et *document.yml* ci-dessous


```

137
138 === Information on the project and product analysed
139 //=== titre de section de niveau 3 (car il y a 3 =)
140
141 ===== Context of the analysis: periodic software quality analysis
142 //===== titre de section de niveau 4 (ce titre est un sous-titre par rapport au précédent)
143 This analysis has been done within the frame of periodic software quality analyses,
144 at least one per year, or one per minor version number (m in number version M.m.p).
145
146 ===== Development team and stakeholders
147 //===== titre de section de niveau 4
148 The table below lists the main stakeholders of the product analysed:
149
150 [width="100%"]
151 //Option permettant de choisir la largeur du tableau par rapport à la page
152 |=====
153 //ouverture du tableau. | marque une nouvelle colonne, un retour à la ligne marque une nouvelle ligne du tableau.
154 | Product lead | John Doe, Jeanne Doe
155 | Product implementation lead | Missing information
156 | Product validation lead | Missing information
157 |=====
158 //fermeture du tableau
159

```

La partie variable du template à été remplie par les informations du fichier project.yml

Conversion au format PDF

Information on the project and product analysed

Context of the analysis: periodic software quality analysis

This analysis has been done within the frame of periodic software quality analyses, at least one per year, or one per minor version number (m in number version M.m.p).

Development team and stakeholders

The table below lists the main stakeholders of the product analysed:

Product lead	Jeanne Doe, John Doe
Product implementation lead	Missing information
Product validation lead	Missing information

Figure 11 : Extrait de la conversion et génération en format pdf

On peut voir le rendu de la première page du rapport pour la conversion du fichier asciidoc vers le format pdf en annexe c)

4) Conversion WORD

Après avoir généré un rapport sous format pdf, nous avons été amenés à générer ce même rapport sous un format différent : le format docx (WORD). Ce format est demandé par le consortium Euclid de l'Agence Spatiale Européenne (ESA) pour les rapports de qualité car c'est un moyen rapide et efficace de permettre la modification à la main de certaines données qui ont été omises ou pour compléter des informations. C'est pourquoi cette partie est une étape cruciale dans notre projet. Il est difficile de réaliser parfaitement cette conversion, c'est pourquoi nous verrons dans cette partie de quelle manière nous avons géré la conversion d'un fichier Asciidoc vers le format docx. Puis, nous explorerons les divers problèmes que nous avons rencontrés et quelles solutions nous leur avons trouvées.

a) Conversion vers un fichier docx (WORD)

Ainsi, nous avons dû nous renseigner sur les outils permettant d'assurer cette conversion. Puis nous avons modifié le programme SARA en vue de convertir un rapport vers le format utilisé par le logiciel WORD : le format docx.

i) Pandoc

Pandoc est un outil assurant la conversion à partir de fichiers de 37 formats différents vers 56 autres formats [9]. Il se lance en écrivant une ligne de commande sur le terminal de la machine et offre de nombreuses options. Il est notamment possible de générer automatiquement une table des matières, de retirer les commentaires présents dans le texte ou encore d'ajouter du texte avant ou après le corps de texte, dans l'en-tête du fichier, etc. [10] D'autres options plus avancées existent aussi selon le format du fichier de sortie. Il est par exemple possible d'utiliser un fichier de référence lorsqu'on convertit un fichier vers un fichier docx. Pandoc l'utilisera lors de la conversion pour affecter un style aux différents titres de paragraphe, aux tableaux, etc.

Nous nous servons de Pandoc afin de convertir notre fichier (template rempli) vers le format docx. Le format docx est le format attendu par l'Agence Spatiale Européenne (ESA) pour les rapports de qualité, c'est donc pour cette raison que nous devons assurer cette conversion.

ii) Conversion asciidoc vers docx

Malgré la grande flexibilité qu'offre Pandoc en matière de conversion, il n'offre pas la possibilité de convertir un fichier Asciidoc ou bien un fichier pdf vers d'autres formats. Ainsi, il est impossible d'effectuer la conversion en fichier docx sans passer d'abord par un format intermédiaire.

Suite à nos recherches, nous avons remarqué qu'il était possible de convertir un fichier Asciidoc vers le format HTML en utilisant le module de conversion Asciidoctor.

De plus, il est aussi possible de convertir un fichier HTML vers le format docx en passant par le biais de Pandoc.

Nous avons donc pris la décision d'utiliser d'abord AsciiDoctor afin de convertir notre fichier AsciiDoc vers le format HTML puis de convertir ce fichier HTML en docx par le biais de Pandoc. Pour ce faire, nous avons d'abord dû ajouter dans le fichier python principal "*cmd.py*" les lignes de code permettant de générer le fichier HTML. Elles sont semblables à celles utilisées pour la génération du fichier pdf. Ainsi la commande à utiliser pour la conversion vers l'HTML est :

→ *asciidoctor temporaryrender.adoc* (Ici pour la conversion en HTML nous n'avons pas besoin de préciser l'extension « .html » car AsciiDoctor a le format HTML comme format par défaut pour une conversion)

Ensuite il suffit de réaliser la génération du fichier HTML à l'aide la commande suivante pour qu'on puisse l'ouvrir dans notre lecteur par défaut de l'HTML et également sauvegarder le fichier HTML automatiquement dans notre logiciel SARA :

→ *.\temporaryrender.html*

Enfin, nous avons utilisé Pandoc afin de réaliser la conversion d'un fichier HTML vers le format docx. Nous avons ajouté au fichier python "*cmd.py*" les lignes de codes suivantes :

→ *pandoc temporaryrender.html -o temporaryrender.docx*

Nos toutes premières recherches nous ont menés vers une conversion qui n'était vraiment pas correcte (cf. figure 12). D'une part, nous avons beaucoup de textes qui n'étaient pas convertis, nous récupérions seulement quelques lignes de la première page du fichier asciidoc (qui doit contenir 4 pages au total). De plus nous n'avions aucune mise en forme, on obtenait seulement les textes du tableau etc.

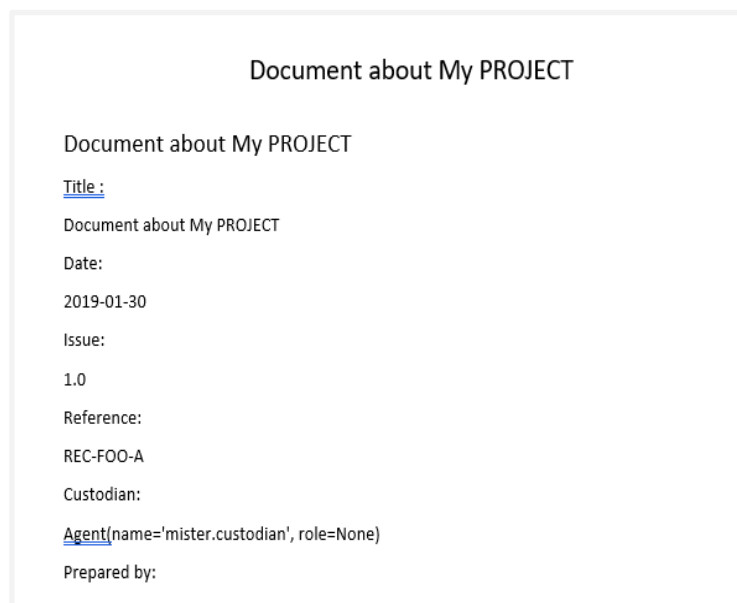


Figure 12 : Première conversion en format Word non fonctionnelle

Cela était dû au fait que la version de Pandoc utilisée n'était pas adéquate pour la conversion vers le format Word. En effet, il y existe de nombreuses mises à jour concernant Pandoc et toutes les versions ne permettent pas d'avoir le même rendu.

Lorsque nous avons installé Pandoc via la commande *pip* à l'aide du terminal de Pycharm, nous nous sommes rendu compte après avoir tenté de résoudre ce problème que la version installée (en vérifiant dans les paramètres du projet sur Pycharm) était très ancienne et mise par défaut pour cette version. C'est pourquoi le fichier word qu'on avait généré avait de nombreux soucis.

Pour résoudre ce problème nous avons tenté de mettre une version plus récente de pandoc proposé sur Pycharm sauf qu'elle n'était pas fonctionnelle car le module Pandoc n'était plus reconnu lors de l'appel dans le terminal de Pycharm. Finalement, plutôt que d'utiliser Pandoc sur Pycharm nous l'avons installé directement en le téléchargeant sur internet ce qui nous a permis d'avoir une version récente et fonctionnelle dans notre projet.

Nous avons grâce à cette solution et aux lignes de code décrites précédemment, réussis à générer un fichier word (cf. figure 13) contenant l'ensemble des informations du template rempli comme on peut le voir ci-dessous sur la première page du fichier asciidoc. Cependant cette méthode nous mène à deux autres problèmes détaillés dans la sous-partie suivante.

Document about My PROJECT

Document about My PROJECT

Title :
Document about My PROJECT

Date:
2019-01-30

Issue:
1.0

Reference:
REC-FOO-A

Custodian:
Agent(name='mister.custodian', role=None)

Prepared by:
-

Date:
Signature
John Doe
2019-01-01
-

Contributors:
-

Date:
-
John Doe
2019-01-01
-

John Doe
2019-01-01
-

John Doe
2019-01-01
-

Approved:
-

Date:
Signature
John Doe
2019-01-01
-

Issue	Date	Page	Description of Change	Comment
0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.
0.2	2018-11-01	2	Typo very long	baibaibaibaiba
0.3	2018-11-01	3.2	Typo	datetime field formatted using formatter.
0.4	2018-11-01	2	Typo	baibaibaibaiba
0.5	2018-11-01	2	Typo	Adding your own field types is fairly straightforward

Purpose and scope

The purpose of this document is to describe the results obtained in the software quality analysis and code inspection of the (LONG_DESCRIPTION) software product.

First objective is to HELP the development team.
Please contribute to improve this report.
Any comments, ideas are welcome !

Figure 13 : Conversion en format Word contenant l'ensemble des informations du fichier asciidoc

b) Problèmes liés à Pandoc pour la conversion vers Word

Cependant, nous avons dû faire face à deux nouveaux problèmes qui sont dus à Pandoc. Pandoc permet de faire des conversions entre de nombreux formats mais ces dernières ne sont parfois pas entièrement fonctionnelles.

Le premier problème était un souci au niveau de deux tableaux AsciiDoc car ils n'avaient plus la forme de tableaux mais plutôt de lignes mises les unes après les autres bien que les autres tableaux aient été parfaitement convertis.

Nous avons donc essayé de trouver la cause de ce problème afin de trouver une solution. Après avoir analysé les fichiers asciidoc, nous avons remarqué que les deux tableaux (*doc_identification_cartouche* et *doc_life_cycle_cartouche*) qui posaient problème utilisaient la fusion de cellule. Nous sommes donc arrivés à la conclusion que la fusion de cellule n'a probablement pas été reconnue par Pandoc lors de la conversion et que, par conséquent, le tableau entier a été en quelque sorte reconfiguré sous la forme de lignes de texte.

Après réflexions, nous avons choisi comme première solution de remplacer les fusions de cellules par des cellules classiques séparées.

Pour ce faire, nous avons modifié la partie du code qui posait problème sur le fichier asciidoc. Nous avons retiré la syntaxe qui était propre à la fusion de cellule et ajouté des colonnes vides pour pouvoir générer des cellules classiques comme on peut le voir ci-dessous sur la figure 14 (ici ce n'est pas le fichier word généré mais le fichier pdf pour visualiser ce qu'on est censé obtenir sur le fichier word) :


Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Figure 14 : Fichier pdf représentant une visualisation de ce qu'on doit obtenir en format Word

Nous avons ensuite re-généré le fichier docx et le problème des cellules a été corrigé, nous avons obtenu un tableau visible dans le docx généré. Notre hypothèse a donc été vérifiée, Pandoc ne reconnaît pas la fusion de cellule lors de la conversion.

Le problème avec cette solution est qu'elle n'est pas optimale (cf. figure 15) car le rendu n'est pas très propre et ne conviendrait pas aux membres de la mission EUCLID.



Ici nous n'avons pas de bordures sur le tableau lors de la génération du word pour cette version de fichier, elle a seulement été ajoutée manuellement pour visualiser le premier problème qui est la fusion de cellule

Document about My PROJECT

Le rendu n'est pas propre car on n'a pas une fusion de cellule

Document about My PROJECT

Title:	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Issue	Date	Page	Description of Change	Comment
0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.
0.2	2018-11-01	2	Typo very long	balbalbalbalb
0.3	2018-11-01	2.3	Typo	datetime field formatted using formatter.
0.4	2018-11-01	2	Typo	balbalbalbalb
0.5	2018-	2	Typo	Adding your own field types is fairly

Figure 15 : Conversion en format Word qui ne contient pas de fusion de cellule

La deuxième solution qu'on a trouvée est d'imbriquer un tableau dans un tableau pour faire office de fusion de cellules. Pour cela, on s'est documenté sur la réalisation de l'imbrication du tableau en asciidoc puis on a une fois de plus modifié les fichiers asciidoc. La réalisation d'un tableau imbriqué dans un autre tableau (cf. figure 16) se fait comme dans le programme qui suit (voir les commentaires expliquant la programmation du langage asciidoc) :

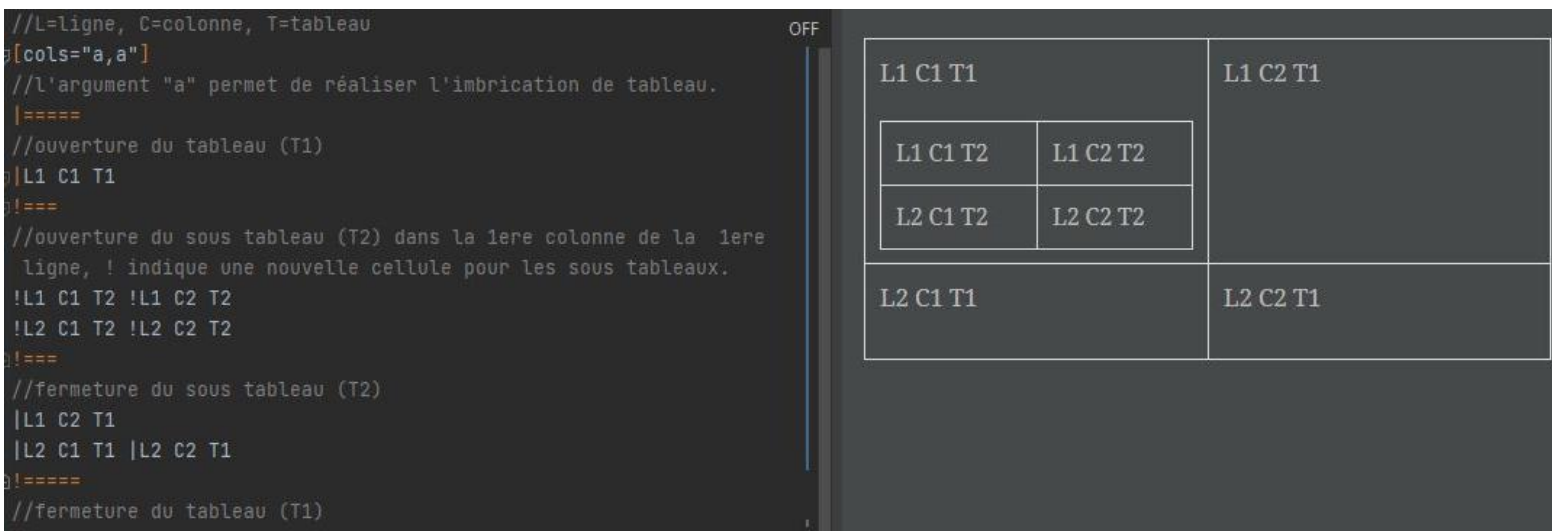


Figure 16 : Exemple de réalisation de tableau imbriqué dans un autre tableau

Une fois l'imbrication de tableau réalisée, nous avons re-généré le fichier comme on peut le voir ci-dessous sur la figure 17 :

Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Figure 17 : Fichier pdf représentant une visualisation de ce qu'on est censé obtenir en format Word

Ainsi, le rendu est meilleur que celui de la première solution où on avait des cellules classiques plutôt que des fusions de cellules.

Sur la photo ci-dessus, nous n'avons pas tout de suite généré le fichier word avec la correction des deux solutions mais plutôt le fichier pdf pour vous montrer encore une fois le rendu qu'on est censé obtenir car la correction d'un second problème (non-affichage des bordures de tableaux) nécessite l'introduction d'une autre fonctionnalité.

c) Mise en page du fichier généré

Le second problème est que les lignes, les colonnes et les bordures des tableaux ne sont pas visibles. Dans ce cas, c'est la mise en forme des tableaux qui pose problème lors de la génération du word comme on le remarque ci-dessous sur la figure 18 :

Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		
Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Figure 18 : Conversion et génération en format .docx où les lignes, colonnes, et bordures du tableau ne sont pas visibles

i) Affichage des styles et des tableaux

Pour pouvoir résoudre ce deuxième problème qui concerne l'affichage des bordures de nos tableaux, nous avons cherché diverses solutions. Cela a été difficile car il y a très peu de moyens permettant de modifier la mise en forme propre à un format sans accéder au code permettant la génération et la conversion du format voulu.

De plus, on ne peut pas régler ce problème à travers les fichiers asciidoc ou html car ce problème n'apparaît qu'après la conversion en docx avec Pandoc. On a donc cherché à contourner ce problème grâce aux fichiers de style que propose Pandoc à partir de la version 1.19.

Pandoc possède par défaut un fichier de style qu'il utilise pour pouvoir faire une conversion vers un format docx. Ce fichier contient toute la mise en forme que Pandoc applique au fichier converti. En cherchant un peu dans les documentations Pandoc, on s'est rendu compte qu'il y avait un moyen d'accéder à ces styles prédéfinis pour pouvoir les modifier et les appliquer à chaque fichier que nous décidons de générer. C'est ce qui va nous permettre de résoudre le problème concernant les bordures des tableaux. Nous observerons le fonctionnement du fichier de style plus en détail dans la partie qui suit.

ii) Mise en page (*custom reference*)

Les styles par défaut qui sont appliqués à chaque conversion sont contenus dans le fichier “*custom-reference*” qui présente les styles pour les titres, sous-titres, le texte, les tableaux, etc. comme on peut le voir ci-dessous sur la figure 19.

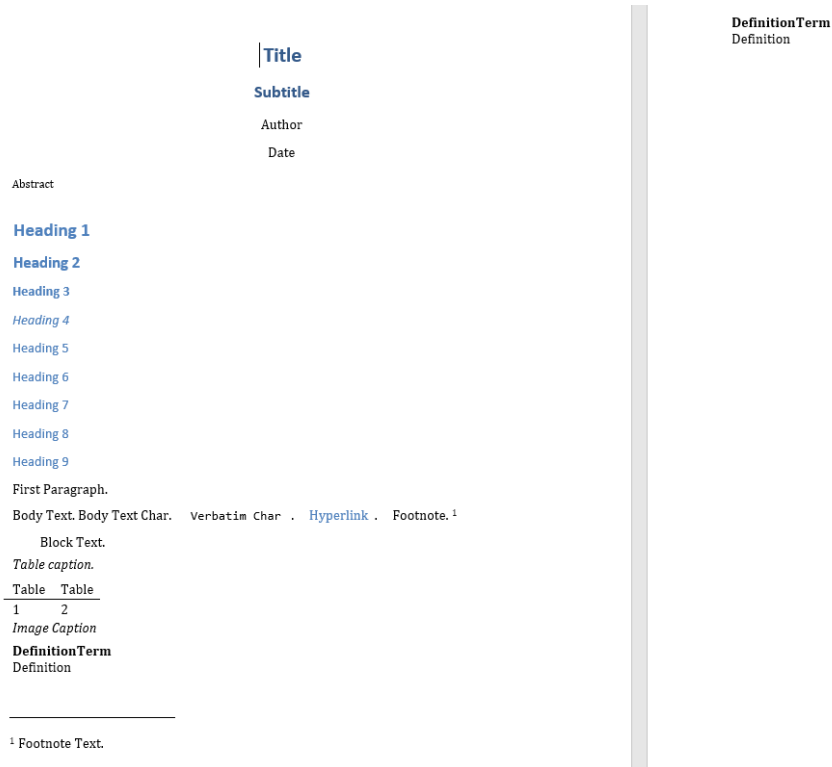


Figure 19 : Fichier *custom-reference* de Pandoc contenant les styles par défaut appliqués à la conversion Word

La méthode pour appliquer de nouveaux styles consiste à utiliser la commande suivante dans le terminal de Pycharm :

→ `pandoc -o custom-reference.docx --print-default-data-file reference.docx`

Cette commande permet de créer un fichier word de style qui sera placé automatiquement dans les fichiers du projet SARA sur Pycharm. Il est appelé “*custom-reference*” (le nom reste bien évidemment modifiable) et contient tous les styles précédents que Pandoc applique pendant la conversion vers un fichier docx. Cette commande n’est pas implémentée dans le programme Python via le module OS car on crée seulement une seule fois le fichier *custom-reference* qu’on modifiera en fonction de nos besoins en matière de style.

Une fois ce fichier créé dans le projet, on peut modifier le fichier *custom-reference* et donc certains aspects de la mise en forme et du style qui seront appliqués à notre rapport/template rempli lors de la conversion en docx.

Pour faire cela, il faut tout d’abord ouvrir le fichier *custom-reference* créé, ensuite pour appliquer un style voulu on doit toujours repérer un moyen d’avoir une case à cocher indiquant que l’on souhaite appliquer ces styles à tous les nouveaux fichiers.

Si cette option n'est pas proposée alors cela signifie que l'on ne pourra pas l'appliquer à la conversion de l'html vers le docx. Les modifications du fichier *custom-reference* doivent être effectuées soigneusement sur word, en suivant un protocole précis pour éviter des bugs lors de la conversion.

Un exemple pour la modification des styles : les bordures de tableaux

Pour modifier les bordures de tableau, il faut sélectionner le tableau puis double cliquer et étirer les styles pour choisir « modifier les styles ». Ensuite après avoir choisi les modifications à effectuer dans “format”, on coche « Nouveaux documents basés sur ce modèle » puis on sauvegarde comme on peut le remarquer ci-dessous sur la figure 20.

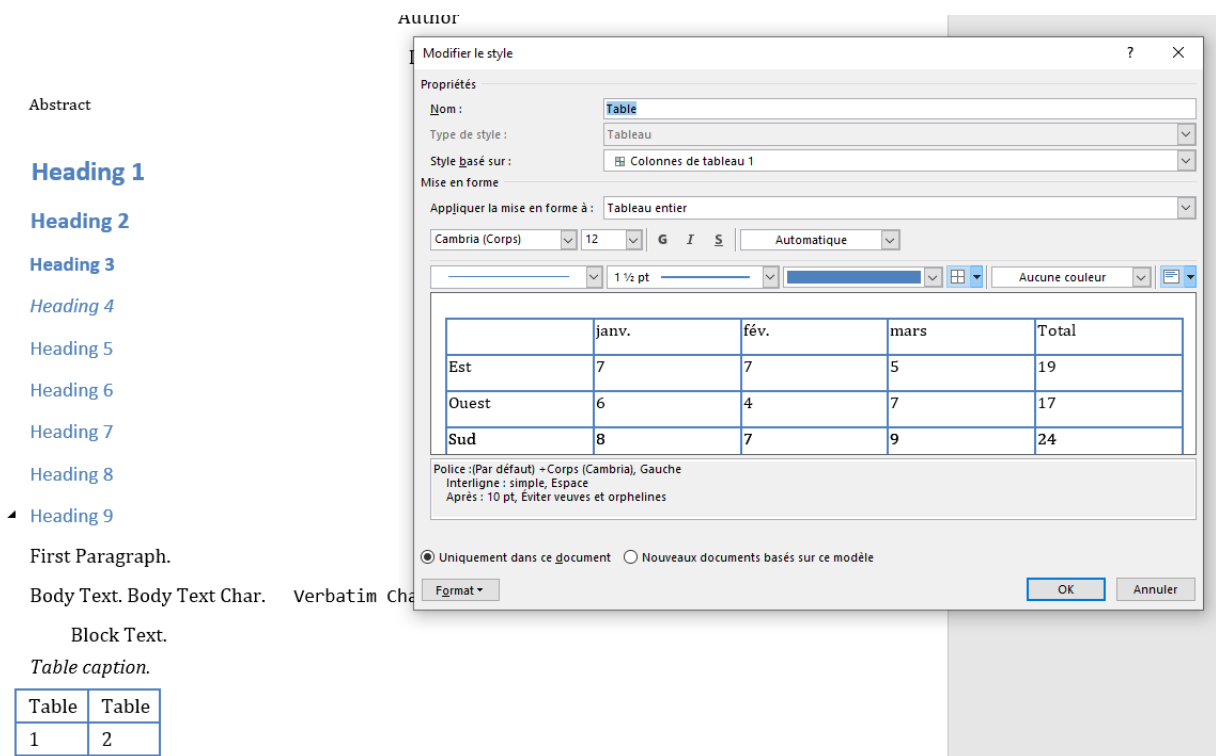


Figure 20 : Modification des bordures de tableau sur le fichier *custom-reference*

Une fois cela fait, nous avons deux commandes à utiliser pour appliquer les styles contenus dans notre fichier *custom-reference.docx* pendant la conversion du html vers le docx :

→ `pandoc --reference-doc custom-reference.docx temporaryrender.html -o temporaryrender.docx`

Cette commande permet de réaliser la conversion du fichier asciidoc vers le format docx en appliquant les styles du fichier *custom-reference* au fichier docx généré.

Enfin il suffit d'ouvrir le fichier *temporaryrender.docx* avec le lecteur .docx par défaut qui est Word (mais on peut l'ouvrir avec Libreoffice, d'après nos tests il n'y a pas de soucis concernant le choix du logiciel) à l'aide de la commande suivante :

→ `.\temporaryrendertest.docx`

Ces deux commandes (cf. figure 21) sont encore une fois implémentées directement dans le programme principal (*cmd.py*) de SARA à l'aide du module `OS` et de la fonction `os.system()` pour qu'on puisse automatiser la génération des rapports :

```
os.system("pandoc --reference-doc customreftest.docx temporaryrender.html -o temporaryrendertest.docx")
os.system(".\\temporaryrendertest.docx")
```

Figure 21 : Code permettant d'automatiser la génération des rapports

Nous pouvons alors voir le fichier *temporaryrender* (cf. figure 23) qui est converti avec la mise en forme, les bordures des tableaux et une sorte de fusion de cellules. Le rendu est plus propre, les deux problèmes sont résolus. Nous avons également mis le fichier *custom-reference* (cf. figure 22) pour vous montrer les styles que nous avons appliqués :

Title

Subtitle

Author

Date |

Abstract

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Heading 7

Heading 8

Heading 9

First Paragraph.

Body Text. Body Text Char. Verbatim Char . [Hyperlink](#) . Footnote. ¹

Block Text.

Table caption.

Table	Table
1	2

Image Caption

DefinitionTerm
Definition

DefinitionTerm
Definition

¹ Footnote Text.

1

Figure 22 : Fichier *custom-reference* avec les nouveaux styles à appliquer au fichier à convertir en Word

Document about My PROJECT

Document about My PROJECT

Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by: - **Date:** Signature

John Doe 2019-01-01 -

Contributors: - **Date:** -

John Doe 2019-01-01 .

John Doe 2019-01-01 .

John Doe 2019-01-01 .

Approved: - **Date:** Signature

John Doe 2019-01-01 -

Issue **Date** **Page** **Description of Change** **Comment**

0.1 2018-11-01 2 Typo file to view the source code until proper documentation is generated.

0.2 2018-11-01 2 Typo very long balbalbalbalb

0.3 2018-11-01 2.3 Typo datetime field formatted using formatter.

0.4 2018-11-01 2 Typo balbalbalbalb

0.5 2018-11-01 2 Typo Adding your own field types is fairly straightforward

1

2

Purpose and scope

The purpose of this document is to describe the results obtained in the software quality analysis and code inspection of the {LONG_DESCRIPTION} software product.

First objective is to HELP the development team.
Please contribute to improve this report.
Any comments, ideas are welcome !

Other objectives are:

- Deliver a Quality status on the code;
- Communicate it to the code authors, the whole development team and managers;
- Possibly set-up action plan for improvement.

For each of the measurements, we cover the following items:

- What is measured and why;
- The measurement tool(s) used;
- The measurement results;
- An analysis of the results and, potentially, actions to be carried out.

The conclusions are derived from good practices and should be taken as a guide instead of a prescription.

This analysis has been done without knowledge (science, SW implementation...) on this project. Please do not hesitate to mention any error or misunderstanding.

Important	Feedback from the development team would be very appreciated.
------------------	---

Summary: strengths and weaknesses of the product

Here we push the instructions and explanations :

Please create
'strengths_and_weaknesses_of_the_product.adoc'
document with the two subsections :

- Top priority
- Other recommendations

Top priority

- Use a *linter* or *pylint* to resolve the undefined variable problems (see *Code issues* section)

Figure 23 : Fichier représentant le template asciidoc converti en format Word avec l'ensemble des corrections

5) Conversion EXCEL

a) Pandas

i) Fonctionnement

Pour convertir un tableau AsciiDoc, nous avons dans un premier temps installé le module Pandas à l'aide de la commande pip de python. Pandas est une librairie open source pour le langage de programmation Python qui nous permet de manipuler et gérer des données. En effet, elle permet de manipuler des tableaux de données, de les lire et de les écrire à l'aide ou vers un fichier tabulé.

Pandas est notamment utilisé dans de nombreux domaines comme en économie, pour la finance, pour les statistiques, les mesures ou encore dans le commerce.

L'équipe EUCLID de l'agence spatiale européenne (ESA) a proposé de réaliser une conversion de tableau asciidoc vers un tableau de format .xlsx (Excel) pour faciliter la gestion des tableaux. Microsoft Excel est un outil puissant pour gérer d'énormes quantités de données issues de tableaux. C'est un outil qui peut être très utile dans de nombreuses applications comme le tri, l'analyse, effectuer des calculs complexes ou visualiser des données. C'est la raison pour laquelle nous avons décidé de réaliser l'extraction et la conversion des tableaux issus d'un fichier Asciidoc vers un fichier Excel en utilisant les fonctionnalités de Pandas [11].

Pour cela nous devons utiliser des "Dataframes", c'est-à-dire un objet utilisé pour le traitement de données, plus précisément une matrice et qui correspond ici aux tableaux. En utilisant principalement la fonction `read_html()` de Pandas, nous pouvons récupérer des dataframes qui seront chaque tableau contenu dans notre template rempli. Cette commande permet d'extraire des tableaux présents sur des pages HTML et comme nous avons déjà réalisé une conversion vers l'HTML pour le word nous allons pouvoir réaliser des procédures similaires au Word.

b) Génération des tableaux au format EXCEL

Dans un premier temps, nous avons essayé de réaliser la conversion d'un unique tableau en format xlsx. Pour cela, nous avons fait des recherches et nous avons trouvé une solution qui a été d'utiliser une page web déjà existante pour la conversion.

Voici un exemple (cf. figure 24) avec un site internet [12] qui contient de nombreux tableaux, nous *extrayons* grâce à l'url de ce site web le tableau (correspondant à l'*indice* choisi) qui est contenu sur cette page :

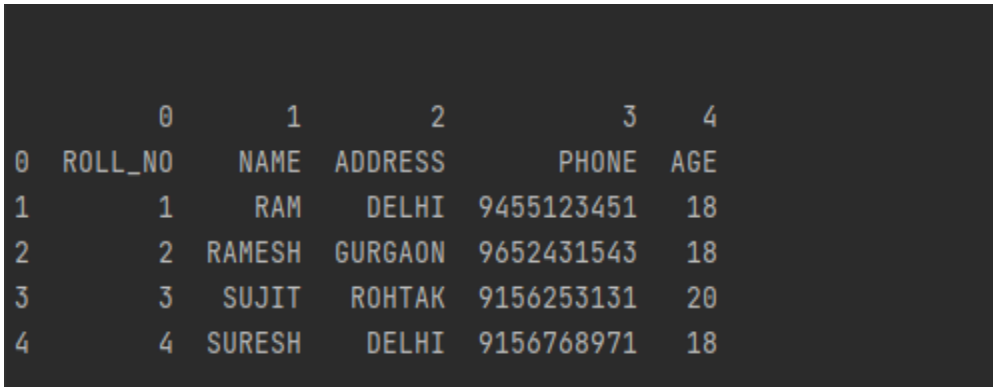
```
# The webpage URL whose table we want to extract
url = "https://www.geeksforgeeks.org/extended-operators-in-relational-algebra/"

# Assign the table data to a Pandas dataframe
table = pd.read_html(url)[0]

# Print the dataframe
print(table)
```

Figure 24 : Code utilisé pour extraire un tableau issu d'un site web

Puis en réalisant un "*print*" de la table on remarque dans le *terminal* Pycharm, visible sur la figure 25 ci-dessous, que nous obtenons bien un tableau correspondant au premier tableau affiché sur le site web. Cela a été récupéré grâce à l'*indice* 0 qu'on fixe lors de la lecture de la page web, c'est-à-dire grâce à "[0]" lors de l'appel de la fonction `read_html()` qu'on peut voir ci-dessus.



	0	1	2	3	4
0	ROLL_NO	NAME	ADDRESS	PHONE	AGE
1	1	RAM	DELHI	9455123451	18
2	2	RAMESH	GURGAON	9652431543	18
3	3	SUJIT	ROHTAK	9156253131	20
4	4	SURESH	DELHI	9156768971	18

Figure 25 : Affichage dans le terminal du tableau extrait de la page web

Après avoir compris le principe de la conversion pour un unique tableau, nous nous sommes concentrés sur l'*extraction* et la conversion de plusieurs tableaux d'un fichier HTML.

En réutilisant le fichier HTML que nous avons généré et utilisé pour la partie Word, nous *extrayons* cette fois les différents tableaux en utilisant un programme basé sur la *gestion de fichier*.

Pour lire notre fichier converti en HTML de la partie Word (cf. figure 26) on utilise le même principe que pour la partie "Récupération du résultat du terminal dans un fichier AsciiDoc" en ouvrant cette fois le fichier en *lecture* :

```
file_path = 'temporaryrender.html'
with open(file_path, 'r') as f:
```

Figure 26 : Code permettant d'ouvrir notre fichier html contenant le template rempli en lecture

Ensuite on applique la méthode expliquée précédemment avec l'exemple de la page Web, mais cette fois pour notre fichier HTML en prenant en *argument* la *lecture* du fichier HTML comme on peut le voir sur la figure 27 ci-dessous :

```
table = pd.read_html(f.read())
```

Figure 27 : Code permettant d'extraire les tableaux issus du fichier html

Cependant on s'est rendu compte que la *fonction* permettant la conversion vers l'excel ne permet pas d'être appliquée à des tableaux imbriqués les uns dans les autres sur chaque *indice*. C'est-à-dire qu'on est obligé de choisir un seul *indice* du tableau (ex : `table[2]`) car chaque case du tableau est un tableau.

Ainsi pour obtenir l'intégralité des tableaux, il nous a fallu savoir combien de tableaux ce fichier Html contient (à l'aide la fonction `len()`) afin de réaliser une *concaténation* des tableau dans l'ordre du fichier html en un seul et unique tableau grâce à la fonction `concat()` de Pandas.

Puis, on utilise la fonction `to_excel()` de Pandas pour réaliser la conversion et la création d'un fichier `.xlsx`. On peut voir sur la figure 28 les lignes de code à mettre en place.

```
with open(file_path, 'r') as f:
    table = pd.read_html(f.read())
    len_tab = len(table)
    print(f'Total tables: {len_tab}')
    df = pd.concat(table[0:len_tab])
    df.to_excel("data.xlsx")
```

Figure 28 : Code permettant de réaliser la conversion et création du fichier Excel

Enfin à l'aide de la bibliothèque `OS system` et sur le même principe que pour le fichier Word et le fichier pdf nous ouvrons le fichier excel sous l'extension `.xlsx` comme on peut voir ci-dessous sur la figure 29 :

```
os.system("python data.xlsx")
```

Figure 29 : Ligne de code permettant d'ouvrir le fichier Excel contenant les tableaux convertis

Le rendu final pour le fichier excel contenant l'ensemble des tableaux du fichier asciidoc est présenté sur la figure 30 ci-dessous :

	A	B	C	D	E	F	G	H	I	J	K	L
1	0		1	2	3	4	AD	Date	Document Reference	Issue	RD	Title
2	0	Title :	Document about My PROJECT									
3	1	Date:	2019-01-30 Issue: 1.0									
4	2	2019-01-30	Issue:	1								
5	3	Reference:	REC-FOO-A									
6	4	Custodian:	Agent(name='mister.custodian', role=None)									
7	0	2019-01-30	Issue:	1								
8	0	Prepared by: -	Date:	Signature								
9	1	Prepared by:	-									
10	2	John Doe	2019-01-01	-								
11	3	John Doe										
12	4	Contributors: -	Date:	-								
13	5	Contributors:	-									
14	6	John Doe	2019-01-01	-								
15	7	John Doe	2019-01-01	-								
16	8	John Doe	2019-01-01	-								
17	9	Approved: -	Date:	Signature								
18	10	Approved:	-									
19	11	John Doe	2019-01-01	-								
20	12	John Doe										
21	0	Prepared by:	-									
22	0	John Doe										
23	0	Contributors:	-									
24	0	Approved:	-									
25	0	John Doe										
26	0	Issue	Date	Page	Description of Change	Comment						
27	1	0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.						
28	2	0.2	2018-11-01	2	Typo very long	balbalbalbalbalb						
29	3	0.3	2018-11-01	23	Typo	datetime field formatted using formatter.						
30	4	0.4	2018-11-01	2	Typo	balbalbalbalbalb						
31	5	0.5	2018-11-01	2	Typo	Adding your own field types is fairly straightforward						
32	0	Important	Feedback from the development team would be very appreciated.									
33	0	Product lead	John Doe, Jeanne Doe									
34	1	Product implementation lead	Missing information									
35	2	Product validation lead	Missing information									
36	0						AD1	2019-07-19	EUCL-CNE-RD-8-001	1,1		EC SGS Coding Standards

Figure 30 : Fichier contenant l'ensemble des tableaux issus du template rempli converti en format Excel

6) Présentations

Nos travaux au sein de ce projet ont suscité l'intérêt des équipes d'Euclid France, de membres du CEA et de chercheurs italiens. C'est pourquoi nous avons été amenés à préparer plusieurs présentations en français et en anglais pour ces diverses équipes.

a) Euclid France

Nous avons préparé une présentation de SARA pour l'équipe d'Euclid France afin de faire une démonstration du programme à ses membres. Nous avons peaufiné cette démonstration, qui est prévue pour le 1er Juin 2021, lors de divers entretiens avec Mme GAUTARD. C'est une étape cruciale pour SARA puisque cette démonstration pourrait permettre de débloquent des fonds pour le développement du programme. Pour nous, cela représente une expérience unique et une grande opportunité qui témoigne à la fois de notre sérieux et aussi de notre investissement dans ce projet. Notons que nous avons traduit cette démonstration en anglais afin de la montrer à des chercheurs italiens, cependant cette présentation n'aura finalement pas lieu.

b) CEA

Des chercheurs du CEA qui ne sont pas membres de la mission Euclid se sont aussi intéressés à SARA. Bien que nous programions SARA pour la mission Euclid, grâce à l'ingéniosité du code de Marc DEXET le programme est modulable et suite à nos travaux il offre des résultats dans des formats divers. SARA pourrait donc facilement être réutilisé par d'autres organismes pour d'autres applications que celles que nous lui avons données. En effet, ce programme pourrait grandement faciliter la génération de documents avec un contenu variable qu'une entreprise, un laboratoire ou quelconque organisme pourrait réutiliser pour éviter des tâches redondantes et ainsi gagner du temps dans la gestion de ses informations.

7) Conclusion

Au sein de ce projet, nous avons pour objectif de faire progresser le programme SARA. L'objectif final de ce programme est d'être capable de vérifier que des logiciels ainsi que leurs documentations respectent les normes puis de générer automatiquement un rapport de qualité. Notre travail a été d'automatiser la génération de rapports sous divers formats. Nous avons réussi à tenir tous nos objectifs, c'est-à-dire générer un rapport sous format pdf, puis sous format docx (WORD) et même sous format Excel.

Grâce à l'aide de notre tutrice Mme Gautard, nous avons réussi à trouver des solutions à tous les problèmes que nous avons rencontrés. Mme Gautard nous a aussi aidé à organiser notre travail tout au long du projet ainsi que les présentations que nous avons été amenés à réaliser pour les membres de la mission Euclid et aussi pour ceux du CEA.

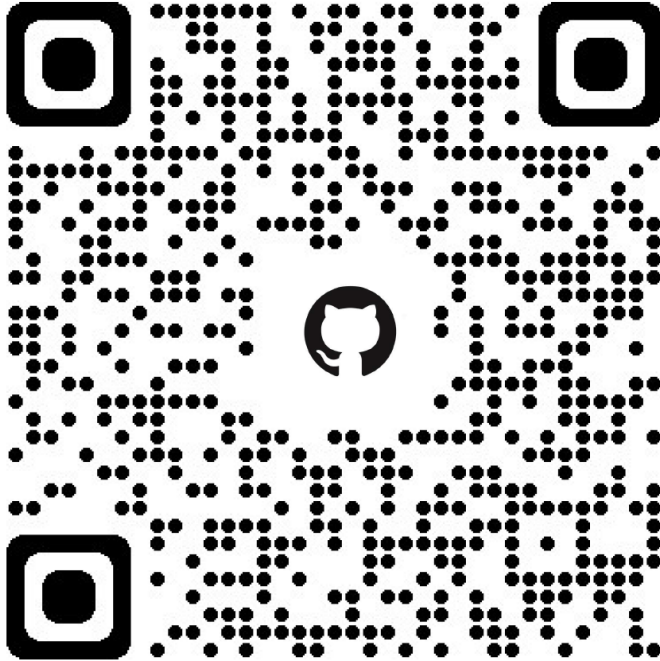
Ce projet fut fort intéressant, c'était la première fois que nous travaillions sur un projet ayant déjà débuté. Il a d'abord été difficile de le prendre en main, il nous a fallu bien saisir les enjeux du projet. Nous avons dû nous former à l'utilisation de nombreux outils informatiques, nous n'avions par exemple jamais codé en Python. Nous former à l'utilisation de tous les modules nous a pris des centaines d'heures de recherche et de lecture de documentations rédigées en anglais.

C'est aussi la première fois que nous transmettons un projet à d'autres personnes. Cela nous a poussé à organiser nos rapports et à présenter nos résultats d'une manière qui soit à la fois simple et précise. Ainsi, nous espérons permettre à nos successeurs de prendre rapidement en main le programme SARA et de continuer de faire avancer ce projet auquel nous avons participé.

Aujourd'hui le programme est entre les mains de Phuc-Toan Nguyen, étudiant à l'école centrale de Lyon et stagiaire au sein du CEA encadré par Mme Gautard. Nous lui avons fourni les renseignements nécessaires à la prise en main du projet et avons répondu à ses diverses questions. Nous espérons que le travail porté par toutes les personnes ayant participé à ce projet porte ses fruits et que SARA soit utile pour de nombreuses et diverses applications.

8) Annexes

a) Lien Github



URL : <https://bit.ly/3uizOno>

b) Tutoriel du lancement de SARA

Disponible sur le lien ci-dessus.

→ Ajout d'une solution pour MAC à venir après confirmation avec Mme Gautard.

c) Première page du pdf

Document about My PROJECT

Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Issue	Date	Page	Description of Change	Comment
0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.
0.2	2018-11-01	2	Typo very long	balbalbalbalbalb
0.3	2018-11-01	2,3	Typo	datetime field formatted using formatter.
0.4	2018-11-01	2	Typo	balbalbalbalbalb
0.5	2018-11-01	2	Typo	Adding your own field types is fairly straightforward

Purpose and scope

The purpose of this document is to describe the results obtained in the software quality analysis and code inspection of the {LONG_DESCRIPTION} software product.

First objective is to HELP the development team.

Please contribute to improve this report.

Any comments, ideas are welcome !

Other objectives are:

- Deliver a Quality status on the code;

d) Exemple de génération WORD avec plusieurs styles

Title

Subtitle

Author

Date

Abstract

Heading 1

Heading 2

Heading 3

Heading 4

Heading 5

Heading 6

Heading 7

Heading 8

Heading 9

First Paragraph.

Body Text. Body Text Char. Verbatim Char. Hyperlink. Footnote.¹

Block Text.

Table caption.

Table	Table
1	2

Image Caption

DefinitionTerm

Definition

DefinitionTerm

Definition

1

Footnote Text.

Document about My PROJECT

Document about My PROJECT

Title:	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Agent(name='mister.custodian', role=None)		

Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	-
John Doe		2019-01-01	-
John Doe		2019-01-01	-
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Issue	Date	Page	Description of Change	Comment
0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.
0.2	2018-11-01	2	Typo very long	balbalbalbalb
0.3	2018-11-01	3,2	Typo	datetime field formatted using formatter.
0.4	2018-11-01	2	Typo	balbalbalbalb
0.5	2018-11-01	2	Typo	Adding your own field types is fairly straightforward

Purpose and scope

The purpose of this document is to describe the results obtained in the software quality analysis and code inspection of the (LONG_DESCRIPTION) software product.

First objective is to HELP the development team.

Please contribute to improve this report.

Any comments, ideas are welcome !

Other objectives are:

- Deliver a Quality status on the code;
- Communicate it to the code authors, the whole development team and managers;
- Possibly set-up action plan for improvement.

For each of the measurements, we cover the following items:

- What is measured and why;
- The measurement tool(s) used;
- The measurement results;
- An analysis of the results and, potentially, actions to be carried out.

The conclusions are derived from good practices and should be taken as a guide instead of a prescription.

This analysis has been done without knowledge (science, SW implementation...) on this project. Please do not hesitate to mention any error or misunderstanding.

Important Feedback from the development team would be very appreciated.

Summary: strengths and weaknesses of the product

Here we push the instructions and explanations :

Please create 'strengths_and_weaknesses_of_the_product.docx' document with the two subsections :

- Top priority
- Other recommendations

Top priority

- Use a linter or pylint to resolve the undefined variable problems (see Code issues section)
- Refactor LE3_DET_CL_P2nav to separate concerns and make process readable.
- Improve coverage of critical components.

Other recommendations

- Remove commented out code.
- Use a accessible managed list of issue.
- Make the unit test more readable using the given,when,then pattern.

Information on the project and product analysed

Context of the analysis: periodic software quality analysis

This analysis has been done within the frame of periodic software quality analyses, at least one per year, or one per minor version number (n in number version M.m.p).

Development team and stakeholders

The table below lists the main stakeholders of the product analysed:

Product lead	John Doe, Jeanne Doe
Product implementation lead	Missing information
Product validation lead	Missing information

Applicable documents

AD	Title	Document Reference	Issue	Date
AD1	EC SGS Coding Standards	EUCL-CNE-RD-B-001	1.1	2018-07-19

Reference documents

RD	Title	Document Reference	Issue	Date
----	-------	--------------------	-------	------

Analysis tools and source code inputs

Environment

Code analysed

Code top-level structure

Product size and category

CODEEN compliance

Configuration management

Product documentation

Generation

Compliance with the Elements framework

Maintainability

Dependencies

Last updated 2021-05-26 16:18:33 +0200

Étudiants : Jimmy LESZCZYNSKI, Javeed ABDOULALIME
et Moïse ALI
Tutrice de projet : GAUTARD Valérie

36

28/05/2021

e) Exemple de génération WORD avec rendus propre des tableaux imbriqués

Document about My PROJECT**Document about My PROJECT**

Title :	Document about My PROJECT		
Date:	2019-01-30	Issue:	1.0
Reference:	REC-FOO-A		
Custodian:	Aqent(name='mister.custodian', role=None)		



Prepared by:	-	Date:	Signature
John Doe		2019-01-01	-
Contributors:	-	Date:	-
John Doe		2019-01-01	.
John Doe		2019-01-01	.
John Doe		2019-01-01	.
Approved:	-	Date:	Signature
John Doe		2019-01-01	-

Issue	Date	Page	Description of Change	Comment
0.1	2018-11-01	2	Typo	file to view the source code until proper documentation is generated.
0.2	2018-11-01	2	Typo very long	balbalbalbalbalb
0.3	2018-11-01	3,2	Typo	datetime field formatted using formatter.
0.4	2018-11-01	2	Typo	balbalbalbalbalb
0.5	2018-11-01	2	Typo	Adding your own field types is fairly straightforward

Purpose and scope

The purpose of this document is to describe the results obtained in the software quality analysis and code inspection of the {LONG_DESCRIPTION} software product.

First objective is to HELP the development team.

Please contribute to improve this report.

Any comments, ideas are welcome !

Other objectives are:

- Deliver a Quality status on the code;
- Communicate it to the code authors, the whole development team and managers;
- Possibly set-up action plan for improvement.

For each of the measurements, we cover the following items:

- What is measured and why;

Bibliographie/Sitographie

- [1] AsciiDoctor, *AsciiDoc Language Documentation*,
URL : <https://docs.asciidoctor.org/asciidoc/latest/>
- [2] Jinja, *Jinja documentation*,
URL : <https://jinja.palletsprojects.com/en/3.0.x/>
- [3] AsciiDoctor, *AsciiDoctor Documentation*,
URL : <https://docs.asciidoctor.org/asciidoctor/latest/>
- [4] Lunaweb, *Le langage Ruby*
URL : <https://www.lunaweb.fr/blog/le-langage-ruby/?cn-reloaded=1>
- [5] Esecad, *Programmer sous langage web Ruby*
URL : <https://www.esecad.com/formation-web/integrateur-developpeur/ruby.htm>
- [6] Linuxmemo, *Ruby Gem*
URL : <http://linuxmemo.free.fr/index.php?title=Ruby>
- [7] Trèsfacile, *Le module OS en Python*
URL : <https://www.tresfacile.net/le-module-os-en-python/>
- [8] Technochouette, *Qu'est-ce que le module OS de Python et comment l'utilisez-vous?*
URL : <https://technochouette.istocks.club>
- [9] Pandoc, *Pandoc's User Guide*,
URL : <https://pandoc.org/MANUAL.html>
- [10] Github, *Pandoc*,
URL : <https://github.com/jgm/pandoc>
- [11] Towards Data Science, *Scraping data from html tables*,
URL : <https://towardsdatascience.com/all-pandas-read-html-you-should-know->
- [12] Geeksforgeeks, *Convert an HTML table into excel*
URL : <https://www.geeksforgeeks.org/python-convert-an-html-table-into-excel/>

Source des images

- [Page de garde] Wikipedia, *Euclid (spacecraft)*
URL : [https://en.wikipedia.org/wiki/Euclid_\(spacecraft\)](https://en.wikipedia.org/wiki/Euclid_(spacecraft))
- [1] Futura-sciences, *Modèle cosmologique décrivant l'univers*
URL : <https://www.futura-sciences.com/sciences/dossiers>

Tableaux des figures

Figure 1 : Les trois types de matière composées par l'univers
Figure 2 : Schéma explicatif des commandes Git utilisées
Figure 3 : Exemple de code AsciiDoc avec son rendu
Figure 4 : Partie variable du template qui sera remplacée par des...
Figure 5 : Schéma explicatif représentant le fonctionnement de SARA
Figure 6 : Schéma illustrant le fonctionnement de SARA avec les différents...
Figure 7 : Code pour le remplissage de la partie variable et...
Figure 8 : Code pour l'ouverture et la création du fichier asciidoc...
Figure 9 : Code pour l'ouverture du fichier pdf
Figure 10 : Schéma explicatif pour la conversion en format pdf
Figure 11 : Extrait de la conversion et génération en format pdf
Figure 12 : Première conversion en format Word non fonctionnelle
Figure 13 : Conversion en format Word contenant l'ensemble des informations du...
Figure 14 : Fichier pdf représentant une visualisation de ce qu'on doit...
Figure 15 : Conversion en format Word qui ne contient pas de...
Figure 16 : Exemple de réalisation de tableau imbriqué dans un autre...
Figure 17 : Fichier pdf représentant une visualisation de ce qu'on est...
Figure 18 : Conversion et génération en format .docx où les lignes...
Figure 19 : Fichier *custom-reference* de Pandoc contenant les styles par défaut...
Figure 20 : Modification des bordures de tableau sur le fichier *custom-reference*
Figure 21 : Code permettant d'automatiser la génération des rapports
Figure 22 : Fichier *custom-reference* avec les nouveaux styles à appliquer au...
Figure 23 : Fichier représentant le template asciidoc converti en format Word...
Figure 24 : Code utilisé pour extraire un tableau issu d'un site...
Figure 25 : Affichage dans le terminal du tableau extrait de...
Figure 26 : Code permettant d'ouvrir notre fichier html contenant le template...
Figure 27 : Code permettant d'extraire les tableaux issus du fichier html
Figure 28 : Code permettant de réaliser la conversion et création du...
Figure 29 : Ligne de code permettant d'ouvrir le fichier Excel contenant...
Figure 30 : Fichier contenant l'ensemble des tableaux issus du template rempli...