POLYTECHNIQUE MONTRÉAL

Rapport de stage

Développement logiciel pour l'équipe Structures

Nom de l'entreprise : BETA Technologies

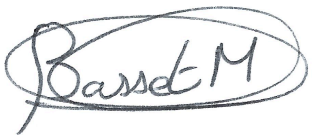
Trimestre du stage : Automne 2024

Lieu principal du stage

Ville : Montréal
Province : Québec
Pays : Canada

Nom du superviseur de stage : Alain Richer

Nom de l'étudiant : Matthieu BASSET
Matricule : 2225981



Présenté à : Lévis Thériault

Polytechnique Montréal
2025-01-31

Résumé

Ce stage est une réponse à un besoin de l'équipe de structures de l'entreprise BETA Technologies. Afin de gagner en productivité, ils désirent développer de nouveaux logiciels permettant aux ingénieurs de passer moins de temps sur des tâches répétitives. À partir des outils existants, il faut donc concevoir et développer des outils qu'ils puissent aisément prendre en main.

L'objectif est donc de proposer en quatre mois des livrables aidant déjà à soulager la charge de travail afin de pouvoir accélérer le processus de certification d'un avion.

Les logiciels développés ont effectivement permis *a posteriori* de supprimer certaines tâches répétitives.

Remerciements

Je tiens à remercier toutes les équipes des bureaux de Montréal de BETA Technologies qui ont été d'une grande bienveillance à mon égard et avec qui ce fut un plaisir de travailler et discuter. Plus particulièrement Alain Richer, mon superviseur, qui bien que me laissant une grande autonomie, a toujours été présent pour me guider dans mon travail en cas de besoin. Aussi Thomas Beeson, mon encadrant plus direct avec qui j'ai travaillé sur le *Skintool*, ses retours honnêtes, qui pointaient les améliorations possibles, mais soulignaient aussi le travail bien fait, m'ont motivé à me dépasser pour proposer un logiciel qui a su impressionner l'équipe.
Enfin un mot particulier pour mes collègues stagiaires Minh Anh Dao et Raphael Barral avec qui il aura été un plaisir d'échanger dans notre bureau au cours de ces quatre mois.

Table des matières

Liste des figures

Sources :
– Fig. 1 et Fig. 2 : https://www.beta.team/aircraft
– Fig. 4 : [1]
– Annexe A: [2]

Liste des annexes

Définitions, abréviations et sigles

Sigles

- VTOL: Vertical Takeoff and Landing (Décollage et atterrissage vertical)
- eVTOL: VTOL électrique
- CTOL: Conventional Takeoff and Landing (Décollage et atterrissage conventionnel)
- FAA : Federal Aviation Administration
- venv : Virtual Environment (Python)
- CI/CD : Continuous Integration/Continuous Deployment
- IDE : Integrated Development Environment
- IA : Intelligence Artificielle
- IRB : Inter-rivet buckling
- OSB : Open Span Buckling
- DCEL: Doubly Connected Edge List, défini plus précisément dans le rapport

Noms et concepts souvent utilisés

- *Mapping Tool* et *Skin Tool* : ce sont les deux outils sur lesquels j'ai travaillé, il s'agit de leur dénomination au sein de l'entreprise, j'ai donc conservé leur nom original, bien qu'étant anglais.
- Élément : Souvent un triangle ou quadrilatère dans un modèle utilisant les éléments finis. Détaillé dans le rapport.
- *GitLab* : Implémentation du système du contrôle de version de fichier *git* permettant d'héberger des instances privées.
- *Pipeline* : Ensemble des étapes de validations effectuées par le serveur *GitLab* quand demandé. Cet anglicisme est omniprésent et je ne lui ai pas trouvé d'équilvalent.
- IDE : Sigle défini ci-dessus. Il s'agit du logiciel principalement utilisé pour écrire le code (par exemple Microsoft Visual Studio Code, neovim, Intellij, etc.).
- *buckling* : Flambage, terme de mécanique des structures désignant un phénomène plus violent qu'un pli, c'est un état irréversible. Beaucoup d'études sont centrées sur ce phénomène, car il veut être évité
- *Inter-rivet buckling* : Flambage inter rivets. Phénomène se produisant quand un panneau entre deux rivets est compressé.
- *Open span buckling* : Flambage ouvert

I Introduction

Ce rapport détaille le développement de deux logiciels : le *Skin Tool* et le *Mapping Tool*. Ce sont des outils ayant pour vocation de soulager les ingénieurs de certaines tâches répétitives inhérentes à l'utilisation de certains logiciels de conception 3D utilisés pour concevoir, ici, un avion électrique.

Ce stage intevient dans un contexte dans lequel BETA Technologies s'apprête à effectuer un premier vol avec un avion de production. Les équipes de structures ont donc été fortement sollicitées en novembre.

J'ai décidé de consacrer une partie dédiée à chaque outil, puisque chacun permet de mettre en avant des difficultés différentes. Le *Mapping Tool* est un outil plus abouti avec une interface graphique et une version compilée disponible pour les employés en ayant besoin. C'est un outil de détection de frontières sur les maillages évitant de sélectionner des centaines de carrés à la main dans un logiciel pour ensuite les réordonner. Le *Skin Tool* est un prototype d'outil visant à unifier les méthodologies de toute l'équipe. C'est un outil dont le développement a nécessité de se plonger en profondeur dans la mécanique des matériaux afin de pouvoir proposer un logiciel robuste.

Ce rapport fait aussi état de mon ressenti du stage, et présente le contexte dans lequel l'entreprise était lorsque j'y ai effectué mon stage.

II Mise en Contexte

II.1 BETA Technologies

BETA Technologies est une société américaine basée dans le Vermont, plus précisément à South Burlington. Elle est spécialisée dans le domaine de l'aéronautique, notamment la conception et production d'avions électriques, mais également de bornes de recharge pour ces derniers. Son but est de proposer un avion électrique à décollage vertical (VTOL). Elle a été fondée en 2017 par Kyle Clark, un ingénieur et pilote, et Martine Rothblatt qui a grandement participé à son financement, étant PDG de United Therapeutics, premier client de BETA Technologies.

Le premier vol a eu lieu en 2018, les vols de test se sont multipliés jusqu'en novembre 2024, moment où le modèle CX300 (CTOL) a obtenu une autorisation de la FAA (agence américaine de réglementation de l'aviation civile) pour réaliser un vol.

L'entreprise a annoncé en mars 2023 avoir ouvert des bureaux à Montréal, c'est dans ces bureaux que j'ai eu l'occasion d'effectuer mon stage. Initialement situés en centre-ville, les locaux ont été déplacés au 9325 Avenue Ryan à Dorval, accolés aux pistes de l'aéroport Pierre-Elliott Trudeau de Montréal, afin d'avoir un accès à ces dernières pour accueillir les prototypes d'avions de la société. BETA Technologies propose plusieurs types de produits actuellement :

- ALIA CX300, le modèle CTOL c.-à-d. à décollage conventionnel, tel un avion de ligne classique, en prenant de la vitesse sur une longue piste de décollage. Un modèle de cette série a été construit et a effectué son premier vol durant mon stage le 13 novembre 2024. Voir Fig. 2.
- ALIA A250, le modèle VTOL de la boîte, mais également capable de CTOL. Le VTOL est un mode de décollage et atterrissage similaire à un hélicoptère, nécessitant moins d'espace pour décoller et atterrir. Voir Fig. 1.
- Les chargeurs, notamment le "Charge Cube" et le Mini Cube, sont capables de recharger des aéronefs électriques, mais aussi des véhicules terrestres. BETA Technologies est par ailleurs responsable de leur déploiement, qui s'étend peu à peu aux États-Unis, partant de la côte Est.

Fig. 1. – ALIA Modèle A250 (VTOL)

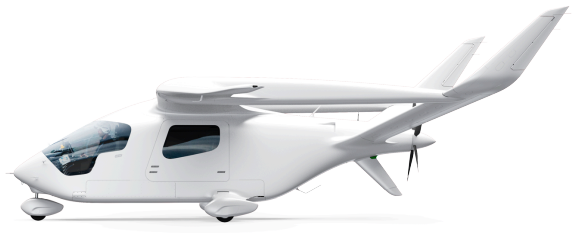


Fig. 2. – ALIA Modèle CX300 (CTOL)

L'entreprise a assurément su susciter l'intérêt, en octobre 2024, une troisième campagne de financement a amené le capital de l'entreprise à plus d'un milliard de dollars US.

II.2 Contexte du stage

Mon stage s'inscrit dans le désir des équipes de Montréal –notamment l'équipe de développement logiciel– de concevoir des outils permettant d'accélérer le travail des autres équipes, dans mon cas, il s'agit de développer des logiciels réalisant des tâches répétitives qu'il est pertinent d'automatiser, au moins dans une certaine mesure. Les équipes des bureaux de Montréal sont en grande partie spécialisées dans l'analyse de structures (au sens mécanique des structures, résistance des matériaux, etc.). Ces équipes ont souvent à effectuer des tâches sur des résultats de logiciels de modélisation 3D, que ce soit chercher des valeurs minimales/maximales dans des résultats d'analyses, sélectionner à la main des "éléments" sur un modèle.

II.3 Responsabilités et rôle en tant que stagiaire

Mon rôle, comme amorcé dans le contexte est de développer des outils pour soutenir l'équipe de structure des bureaux de Montréal, j'ai travaillé sur deux outils différents. Le *Mapping Tool* et le *Skin Tool*, le premier a pour vocation de créer des *mappings* c.-à-d. des tableaux représentant des éléments sur un panneau de l'avion, pouvant être ensuite exportés dans Microsoft Excel (sera appelé simplement Excel) afin de mener des analyses. Le second est un outil ayant pour objectif de centraliser les méthodologies d'analyse de structure au sein d'un unique logiciel pour harmoniser les analyses entre les différents ingénieurs en charge.

II.4 Déroulement du stage

Mes tâches variaient souvent d'une semaine à l'autre en fonction des besoins des équipes, je pouvais me concentrer plus sur un outil une semaine où les équipes avaient besoin d'avoir une version fonctionnelle du logiciel, ou alors retarder le début du travail sur un outil pour améliorer un outil plus important à perfectionner sur le moment.

La première semaine a principalement été dédiée à l'adaptation à l'environnement de travail (présentation des bureaux, des équipes que l'on rencontrera le plus souvent), et à la configuration de notre environnement de programmation (voir Section II.5). Dès cette semaine, j'ai déjà été amené à réfléchir à un algorithme pour détecter les frontières d'un ensemble de points en trois dimensions, qui se trouvera être la clef de voûte du *Mapping Tool*. J'ai continué à travailler dessus pendant le mois de septembre, tout en commençant à me familiariser avec les méthodologies que j'allais devoir implémenter dans le *Skin Tool*.

En octobre, j'ai commencé à réaliser une interface graphique autour de l'algorithme de détection de frontière, cela a été l'occasion de me familiariser avec *Qt*, un cadriciel conçu pour développer des applications graphiques, avec la possibilité de programmer en C++ ou en Python, ce dernier sera retenu, car c'est le langage principalement utilisé au sein de l'équipe logiciel.

Le mois de novembre est marqué, comme la fin du mois d'octobre par la date du premier vol (13 novembre) se rapprochant. Les équipes de structures étant très fortement sollicitées pour finir les analyses à temps, les interactions étaient moins possibles, mon travail était donc plus concentré sur des corrections de problèmes existants ou le perfectionnement de certains détails en attendant de pouvoir avoir de nouveaux retours concrets des utilisateurs. J'ai travaillé à ce moment à la conception de tests (unitaires et globaux) sur les méthodologies, ainsi qu'à leur intégration dans les outils. L'objectif de ces tests est de certifier que le logiciel donnera les mêmes résultats que l'ancienne méthode de calcul (souvent des grands classeurs Excel).

Finalement, en décembre, il a été temps de conclure mon travail, j'ai consolidé la documentation que j'avais écrite au fil du stage afin de permettre à mon encadrant Thomas de pouvoir continuer le travail sur mes outils. C'est aussi l'introduction d'une infrastructure plus formelle de gestion de projet avec des tickets Jira pour les retours et la mise en place de nouveaux outils pour mieux suivre l'avancement des équipes. J'ai aussi pu travailler à mettre en place un prototype d'interface pour le *Skin Tool*.

II.5 Environnement de travail

Le mot d'ordre de BETA Technologies sur l'environnement de travail est de toujours laisser le plus de choix aux employés et stagiaires pour que chacun travaille d'une manière qui lui convient (travail hybride, liberté sur les logiciels).

II.5.1 Locaux et cadre de travail

Comme dit précedemment, les locaux de l'entreprise sont situés au 9325 Avenue Ryan à Dorval. Ces locaux étaient loin de chez moi (environ 50 min de temps de trajet aller, 1 h 30 avec bouchons), ce qui m'a amené à souvent télétravailler afin d'économiser ce temps. Bien que mon stage était hybride (partiellement en présentiel et télétravail) sans contrainte spécifique de présence au bureau, j'ai essayé d'être en moyenne trois jours par semaine (mardi, mercredi, jeudi), car le cadre de travail était vraiment agréable.

Tout était pensé pour faciliter la vie des employés : bureaux montés sur verrins et chaises ergonomiques pour éviter le mal de dos, cuisine avec de nombreux snacks, des fours à micro-ondes et des réfrigérateurs pour se restaurer avec la nourriture présente ou amener son repas. L'entreprise proposait aussi un repas offert le mercredi, l'occasion pour plus de personnes de venir au bureau et sociabiliser.

Les locaux dans leur ensemble sont modernes et ce fut un plaisir d'y travailler. La proximité avec l'aéroport était aussi un avantage, pouvoir voir des avions atterrir toutes les cinq minutes est un luxe !

II.5.2 Logiciel

La configuration occupe une partie importante de la première semaine. Dans l'absolu le choix des outils est assez libre, par exemple le choix de l'IDE (bien que Microsoft Visual Studio Code était encouragé). Les ordinateurs étaient configurés pour utiliser Microsoft Windows 11, il n'était pas vraiment possible d'utiliser d'alternatives comme Linux en raison de la forte dépendance de mon travail avec la suite Microsoft Office et certains problèmes de portabilité de *Qt*.

Pour le développement, puisque tous les logiciels doivent être écrits en Python, sa configuration est importante. La versoin à utiliser n'était pas fondamentalement importante, j'ai utilisé Python 3.10, 3.11 et 3.12 en fonction des situations sans conséquences majeures. Python 3.13, sorti durant mon stage était trop récent pour une utilisation professionnelle.
Afin d'avoir une gestion de versions de paquets propre, l'outil PDM [3] a été très utile. Il s'agit d'un gestionnaire de paquets Python, à l'instar de *poetry* ou *pip*. Cet outil permet de créer un environnement virtuel (venv) dédié par projet, permettant d'utiliser des versions de librairies différentes par projet par opposition à *pip* par exemple. Aussi PDM permet de créer des scripts afin de réaliser automatiquement des tâches prédéfinies (compilation, formattage, exécution de tests, etc.).

Aussi, BETA Technnologies dispose d'un serveur *GitLab* propre dans le but de stocker le code des différents prrojets. Il propose aussi des outils de CI/CD comme des *pipelines* que j'ai eu l'occasion d'utiliser. La connexion au serveur se faisait nécessairement via une clé SSH.

Comme évoqué précedemment, l'entreprise a également mis en place un système de tickets sur une instance de la plateforme Jira (développée par Atlassian). Ce système permet aux utilisateurs des logiciels de rapporter tout problème avec un des logiciels développé par l'équipe. L'équipe logiciel peut ensuite attribuer à un membre la charge de corriger le problème. L'utilisation de la plateforme Alignd a aussi été encouragé afin de pouvoir suivre au mieux l'avancement de chaque tâche plus facilement.

Pour la communication, l'application Slack était privilégiée, étant pratique pour compartimenter les thèmes de dialogue dans des salons de discussion spécifiques. Pour les réunions, le logiciel Zoom était préféré, l'intégralité des salles de réunion des bureaux étant munies de tablettes permettant de créer/gérer les réunions.

III Retour d'expérience

III.1 Compétences professionnelles et techniques

Durant ce stage, mes compétences techniques ont été largement sollicitées et j'en suis satisfait. La réalisation des logiciels demandés exigeait une grande rigeur dans l'utilisation du langage Python. Mon expérience dans plusieurs langages de programmation m'a également permis d'avoir une vision plus transversale de la programmation et donc de concevoir des algorithmes indépendamment de leur implémentation concrète.

J'ai aussi pu mettre à contribution mes connaissances et compétences en mécanique des structures et en aéronautique que j'avais acquises lors de mon cursus d'ingénieur initial à l'ISAE-Supaero. Cela a été un atout pour plus me projeter plus facilement dans les problématiques des équipes, ainsi, je n'ai donc pas été submergé par l'abondance de termes techniques. Cela m'a aussi aidé à plus aisément apréhender les méthodologies que je devais implémenter en Python, le fait de comprendre les phénomènes physiques derrière les calculs de structure rendaient leur appréhension plus aisée, mais me permettait par ailleurs de détecter d'éventuelles erreurs ou d'adapter certaines formules, au-delà de simplement les transcrire.

III.2 Habiletés personnelles et relationnelles

Un des plus beaux moments de mon stage a été le premier retour sur le *Mapping Tool*. Après avoir déployé la première version de test du logiciel et l'avoir présenté rapidement dans une réunion, il a commencé à être utilisé. Le premier retour que j'ai reçu fut de la part d'un collègue me remerciant de lui avoir fait gagner plusieurs heures grâce à l'automatisation d'une tâche répétitive (sélection des éléments en bordure), qui prenait désormais seulement quelques minutes. Le cadre était complètement différent d'un environnement scolaire, mon logiciel était loin d'être parfait et avait encore des bogues dans certains cas, et manquait de fonctionnalité par rapport au produit final. Néanmoins, il permettait déjà d'éviter plusieurs heures de travail fastidieux.
C'est une des leçons que j'ai retenu, par opposition à un travail scolaire où on cherche une note maximale pour un travail parfait ou presque, en milieu professionnel, en tout cas dans ma situation pour un outil interne, on préfèrera avoir un outil fonctionnel au bout de deux mois, quitte à le perfectionner, plutôt que s'acharner six mois à couvrir des cas rarissimes que les utilisateurs vérifiraient déjà manuellement.
Aussi, il est important d'avoir des échanges fréquents pendant le développement des outils, afin de s'assurer que la direction convienne aux utilisateurs finaux. En effet, si une fonctionnalité n'est finalement pas désirée, il est préférable de le savoir au plus tôt, dans le but de s'éviter du temps de travail dessus.

J'ai aussi pu constater qu'il faut parfois tempérer ses objectifs, les utilisateurs n'ont pas toujours conscience de la complexité d'implémentation d'une tâche, ou de ce qui les aiderait au mieux. À plusieurs reprises, j'ai discuté avec mon encadrant afin de définir les exigeances pour les prochaines versions après en avoir discuté avec les équipes utilisant le logiciel, et souvent il en ressortait des lignes directrices bien différentes de ce qui avait été exprimé. En somme, les utilisateurs nous donnaient avant tout des pistes. Au-delà de l'implémentation des fonctionnalités correspondantes, nous évaluions la faisabilité, mais également la pertinence des propositions, pour décider de leur priorité, et occasionnellement les adapter pour les rendre réalisables dans un temps raisonnable.

III.3 Lien avec les apprentissages du programme d'études

Parmi les cours que j'ai suivi à Polytechnique Montréal, je pense que tous m'ont été utiles durant ce stage.
Tout d'abord, de manière très directe, le cours LOG8430 : « Architecture logicielle et conception avancée ». En effet, ayant développé des logiciels, les notions de ce cours se sont appliqués on ne peut plus directement. Que ce soient les différents patrons d'architecture logicielle, le contrôle de la qualité, la détection de mauvaises habitudes de code (permettant de revoir l'architecture logicielle afin de la rendre plus modulaire). Ces notions m'ont permis d'avoir un regard plus éclairé sur l'architecture de mes logiciels, au niveau macroscopique comme à l'échelle d'une fonction.
Je pense aussi au cours INF8775 : « Analyse et conception d'algorithmes » qui a consolidé mes connaissances sur la conception d'algorithmes d'un point de vue complexité (temporelle et spatiale). Une de mes premières tâches ayant été de concevoir un algorithme de détection de frontières d'un ensemble de points avec la contrainte explicite qu'il devait être efficace.

Au-delà des cours m'ayant servi de manière très directe, ayant suivi beaucoup de cours dont les applications sont à réaliser en Python (notamment beaucoup de cours sur l'IA), j'ai affûté, au fil de mon cursus à Polytechnique Montréal, mes connaissances en Python. Cela m'a permis d'appréhender le langage sous différents angles au fil de ma scolarité, et donc à terme, durant mon stage, d'être capable de choisir quelle approche je désirais afin de réaliser une tâche donnée. J'ai aussi eu l'occasion de travailler avec beaucoup de librairies différentes, être en équipe avec des étudiants qui avaient leur propre manière de coder, ou en consultant les sujets/corrigés de TP et ainsi me constituer mes propres habitudes, en gardant ce que je trouvais pertinent. Par exemple, j'ai découverts certains modules de la librairie standard comme `itertools` ou des librairies comme `pandas`.

Et outre les cours dans lesquels j'utilisais Python, mes cours en génie informatique et logiciel m'ont servi dans leur ensemble. En effet, il était même –selon moi– préférable d'avoir des cours demandant l'utilisation de langages différents. Devoir réaliser des travaux dans de nouveaux langages, en utilisant de nouveaux cadriciels, tout cela contribue à développer une capacité à voir

la programmation au-delà du langage de programmation. Ainsi, on réfléchit à un algorithme, à une structure de données, à une architecture et non leur implémentation dans un langage spécifique. Cela permet aussi d'amener des pratiques permettant une meilleure lisibilité de code en utilisant des patrons de conception qui ne sont pas nécessairement courants dans un langage. Par exemple l'utilisation d'`enums` en Python n'est pas très courant, mais c'est omniprésent en Rust, c'est une pratique qui permet d'assigner un type à une valeur qui aurait été autrement une chaîne de charactères, et donc permet un contrôle plus fin des données.

IV *Skin Tool*

IV.1 Mise en contexte

Le *Skin Tool*, ou *Skintool* est à l'origine le projet principal de mon stage. Il l'a techniquement été, puisque le *Mapping Tool* en sera un composant. Cependant, j'ai donc plus travaillé sur un composant de ce logiciel que sur le logiciel lui-même.
Le but du *Skin Tool* est de regrouper au sein d'un même logiciel les méthodologies, parfois différentes, utilisées par les analystes pour mener leurs calculs de structures sur les panneaux de l'avion. Un panneau étant une surface, souvent en matériau composite, qui compose la plupart de l'extérieur de l'avion (la partie blanche), par opposition aux poutres, souvent en métal et à l'intérieur de la structure.

Plusieurs analyses sont menées sur ces panneaux : *inter-rivet buckling* (IRB), *open span buckling* (OSB), *crippling*, etc. Les outils actuels pour ces méthodologies sont des classeurs Excel. Il a donc été décidé d'adapter ces outils en Python. Cela permettra à terme de gagner en performance, portabilité, modularité et homogénéité. Les différences actuelles entre méthodologies viennent principalement du fait qu'elles sont appliquées sur des parties différentes de l'avion (ailes, fuselage, empenage, etc.) avec des contraintes particulières différentes. Les équipes en charge de chaque partie de l'avion ont donc adapté des outils à leur situation particulière. Le défi du *Skintool* est donc de réussir à traiter tous ces cas avec un unique outil, tout en permettant à chaque équipe de pouvoir l'utiliser dans sa situation précise.

IV.2 Objectifs poursuivis

Deux grands objectifs furent suivis durant mon stage pour cet outil.
Le premier était de « traduire » des documents Excel en Python, c'est-à-dire regarder les formules présentes, identifier les entrées, les sorties, pour reproduire le comportement en Python.
Le second était d'écrire des tests pour les modules que j'avais écrit, mais aussi pour un module exitant. Ces tests avaient pour but de rassurer les analystes en garantissant que l'outil Python donnerait bien les mêmes valeurs que l'outil Excel. Les tests ont tout d'abord porté sur l'IRB,

car plus simple à mettre en place, effectivement, il n'y a besoin d'implémenter que deux formules, là où l'OSB comporte une dizaine d'étapes, beaucoup se basant sur des interpolations de courbes trouvées dans la littérature.

Un objectif n'ayant pas été complètement atteint était d'avoir une version fonctionnelle de l'outil, même avec seulement deux modules (IRB, OSB). L'idée est d'avoir un serveur pouvant recevoir des requêtes. Ces requêtes contiennent soit la définition d'un ou plusieurs panneaux, soit une indication pour commencer une ou plusieurs analyses. La structure se veut très modulaire, le serveur ne dépend pas de la provenance des données, seul leur format compte. Aussi le serveur peut être local tout comme hébergé, il n'y a pas de différence pour le client qui envoie une requête. L'architecture présentée en Fig. 3 illustre la structure voulue pour le serveur gérant les requêtes. Il s'agit d'un exécutable Python ayant un récepteur pour les requêtes HTTP.

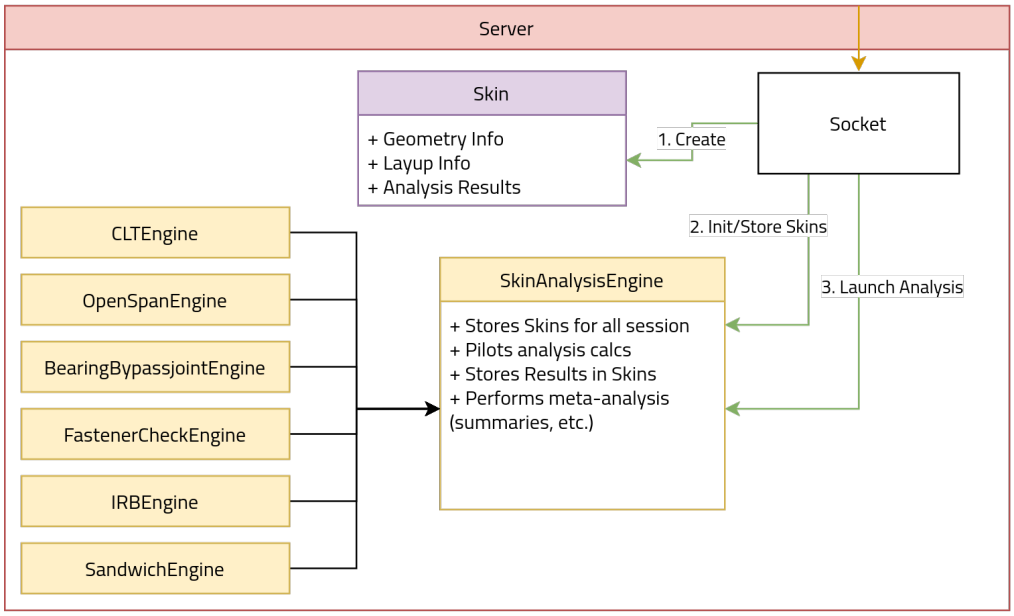


Fig. 3. – Architecture proposée pour le *Skintool* (UML informel présenté)

Le client était, lors des tests mi-décembre avant la fin de mon stage, un classeur Excel envoyant des requêtes via des macros.

IV.3 Revue documentaire et présentation des analyses

Afin d'adapter les méthodologies, au-delà des classeurs Excel, j'ai également consulté les ouvrages de référence pour comprendre les formules utilisées, en profiter pour vérifier l'exactitude de la procédure, parfois ajouter des tests lorsque le livre proposait une hypothèse à vérifier pour appliquer une formule. J'ai principalement revu les méthodologies de l'*inter-rivet buckling* via [1] et j'ai bien étudié la méthodologie de l'*open span bukling* via encore [1], mais aussi [4] et [2].

IV.3.1 *Inter-rivet buckling*

La méthodologie pour l'IRB est relativement simple, il s'agit d'une seule fomule.

$$\sigma_{\text{ir}} = \frac{c\pi^2 D_{11}}{ts^2}$$

Avec c un coeficient valant 1 ou 3 en fonction de la configuration des rivets, D_{11} étant un coeficient dépendant du matériau et t l'épaisseur du panneau. La représentation du problème est aussi assez claire comme on peut le constater sur la Fig. 4.

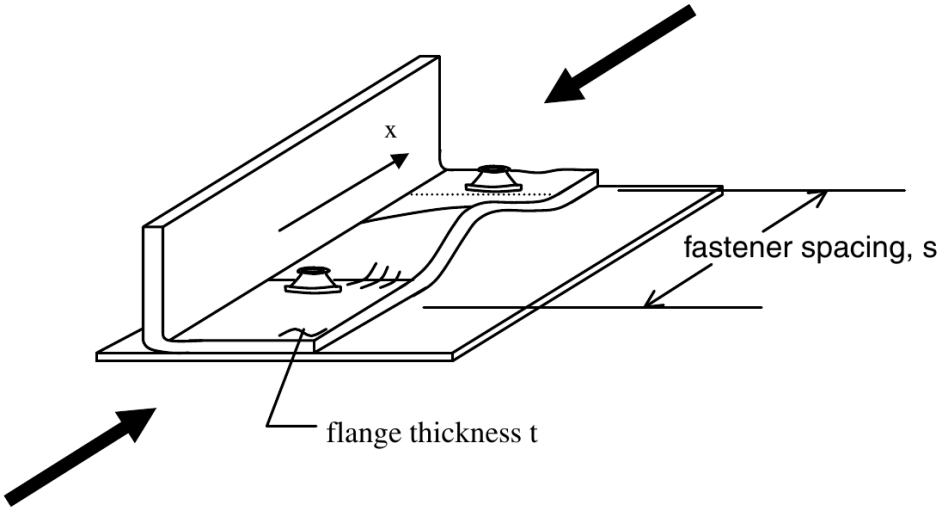


Fig. 4. – Définition du contexte de l'*inter-rivet buckling*

IV.3.2 *Open Span Buckling*

L'*Open Span Buckling* est bien plus complexe, comme en témoigne la Fig. 5 pour la géométrie, on a en effet la possibilité d'avoir un cœur plus épais, un trou dans le panneau, une rampe, une courbure du panneau. Chacun de ces paramètres pourra avoir une influence, il faudra vérifier s'ils sont présents et si oui, s'ils ont une influence sur les contraintes admissibles. Et les contraintes appliquées sont aussi plus complexes comme le témoigne la Fig. 6. De plus, il y a aussi des conditions aux limites sur l'état de chaque arête. Aussi, on aura besoin d'interpoler des valeurs sur des courbes empiriques (cf. Annexe A).

Sans rentrer dans le détail de chaque étape de cette analyse, on peut constater sa complexité bien plus importante. Cela en fait un cas pertinent pour l'implémentation de tests, en effet avec le grand nombre de cas particuliers à gérer, il est nécessaire de passer rigoureusement en revue la méthodologie afin d'en capturer toutes les subtilités.

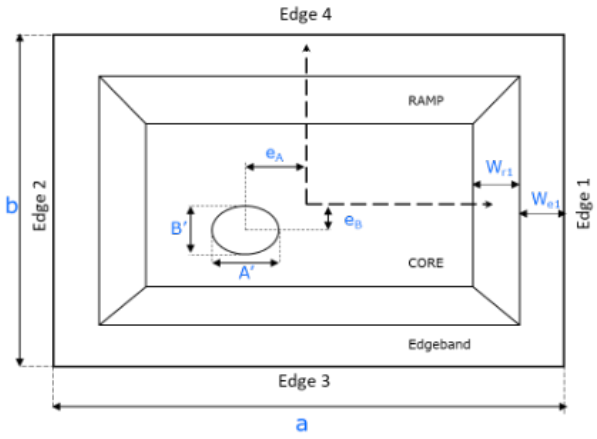


Fig. 5. – Définition de la géométrie de l'OSB

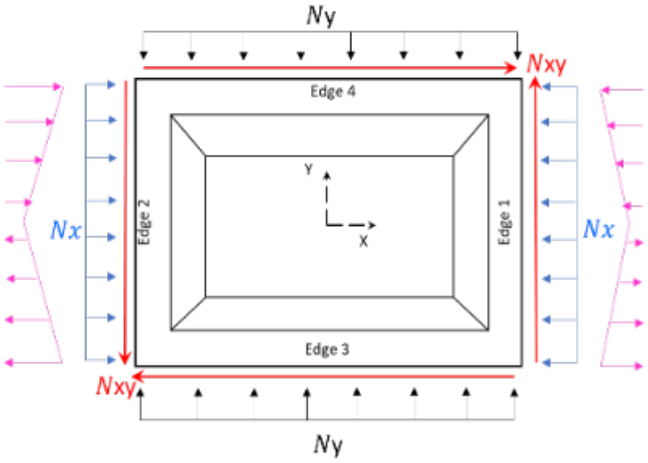


Fig. 6. – Définition des constraintes sur l'OSB

IV.4 Approche

La traduction des méthodologies et l'écriture des tests n'est pas fondamentalement complexe. Il s'agit simplement d'un processus demandant de la rigueur dans le but de s'assurer de l'exactitude des résultats. La revue détaillée de la méthodologie de l'*open span buckling* m'aura d'ailleurs permis de découvrir deux bogues présents dans l'implémentation existante.

Une difficulté a été de trouver des exemples pour tester mon implémentation. La plupart des cas de tests sont des fichiers Excel sur le *Google Drive* de l'entreprise, il a été un peu fastidieux de trouver des cas permettant de couvrir suffisamment de scénarios, permettant de s'assurer une couverture raisonnable de la méthode. Et au-delà de les trouver, il a également fallu trouver un moyen d'extraire les données des classeurs. Finalement, j'ai conçu un script Python allant lire toutes les valeurs du classeur afin de les compiler dans un fichier CSV (Comma Separated Value, format de stockage de fichier utilisé pour le stockage de données).

L'approche dans la couverture des tests a aussi été différente pour chaque analyse. Le module de test sur l'IRB est bien plus exhaustif. En effet, grâce à sa simplicité, il a été utilisé afin de concevoir un modèle pour les modules de test. Ainsi, j'ai écrit des tests unitaires pour chaque fonction et vérifié tous les cas. À l'inverse, pour l'OSB, j'ai pu écrire quelques tests sur les premières étapes, car elles sont simples, mais en avançant dans l'analyse, j'ai décidé d'écrire en priorité un grand test macroscopique testant simplement les valeurs des paramètres à la fin de chaque étape.

IV.5 Résultats

Le module de test a été validé début décembre, je me suis également occupé de configurer *GitLab* (la *pipeline* avait besoin d'une version spécifique de PDM pour fonctionner), une fois mis en place, les tests, passant en local (comme on peut le voir sur la Fig. 7), fonctionnait aussi sur le serveur

(voir Fig. 14), et si l'on change du code en rapport avec le calcul, les tests ne passent évidemment plus, bloquant le déploiement (par exemple sur Fig. 15).

La traduction des modules d'IRB et OSB ont bien amorcé le travail de l'équipe logiciel sur le *Skintool*, le module de test a été jugé très pertinent et servira désormais de modèle pour l'écriture de tests pour les autres outils. Enfin, la démonstration deux jours avant la fin de mon stage d'une implémentation de client via Excel et communiquant avec le serveur Python a ouvert beaucoup de perspectives pour la suite du développement de l'outil.

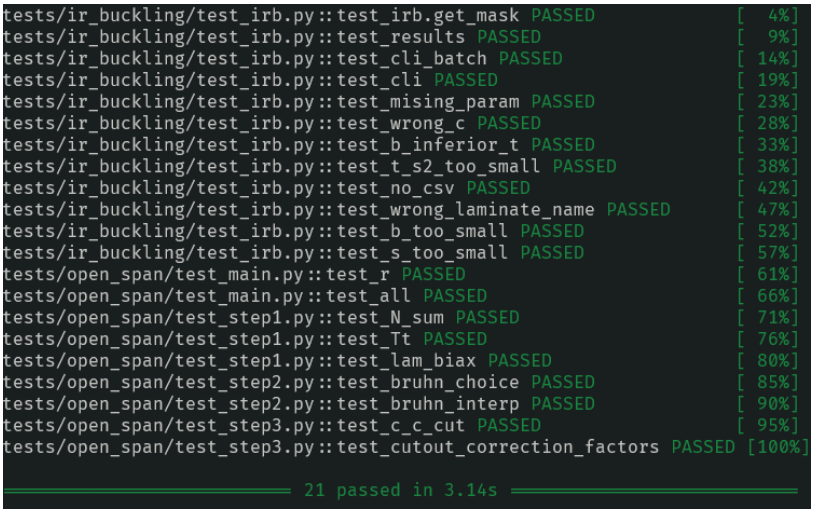


Fig. 7. – Validation locale des tests sur la méthodologie

V *Mapping Tool*

V.1 Mise en contexte

Le *Mapping Tool* est chronologiquement le premier outil sur lequel j'ai travaillé. À la fin de ma première semaine, en attendant de finir d'obtenir mes accès aux différents services (*GitLab*, etc.), on m'a donné une tâche à réaliser : trouver un algorithme permettant de trouver tous les éléments en bordure d'un panneau, plus précisément, un maillage d'éléments.
On peut visualiser une structure similaire à celle présente sur la Fig. 10, j'ai accès à une liste d'éléments (des polygones fermés, en pratique des triangles ou des quadrilatères quelconques), ces éléments ont des sommets qui ont des coordonnées dans l'espace. Et mon but est donc de trouver quels éléments sont sur la bordure de cette structure, sachant qu'elle peut également avoir un trou au milieu.

Une fois ce travail réalisé, il pourra s'intégrer dans un outil permettant de construire des *mappings*, ce sont des représentations en deux dimensions de la surface, avec les identifiants des

éléments agencés dans un tableau, de manière à reconstituer leur agencement sur la surface. Un exemple de *mappings* est donné en Fig. 11.

Ce logiciel s'inscrit dans les outils utilisés pour les calculs sur modèles à éléments finis (modèles approximant la géométrie réelle avec des « éléments », facilitant les calculs). En effet, une fois le *mapping* obtenu, des fichiers Excel sont disponibles pour mener des analyses sur le panneau en question, le *mapping* permet d'avoir ainsi une représentation sur un plan de la répartition des contraintes une fois les résultats obtenus. Et puisque réaliser un *mapping* manuellement est très fastidieux, automatiser ce processus était la bienvenue.

Enfin, afin d'obtenir les coordonnées des sommets à partir du modèle de l'avion et pouvoir travailler avec des données réelles, le logiciel s'est appuyé sur le grand nombre de fichiers « h5 » disponibles. Ce sont des fichiers permettant de compresser des données en un fichier binaire (cf. [5]), et dans ces fichiers de résultats de simulations se trouve une table de tous les éléments, et de tous les sommets avec leurs coordonnées.

V.2 Objectifs poursuivis

Mes tâches sur le *Mapping Tool* ont été très variées au fil de mon stage. J'ai d'abord dû concevoir l'algorithme trouvant les bordures, puis isoler quatres arêtes, ensuite construire un *mapping*, puis pouvoir détecter des éléments 1D (éléments composés de deux sommets, agissant comme une poutre, ils ont une utilité mathématique pour les simulations) qui se trouvent souvent sur les bords des panneaux et sont également importants pour les calculs.
J'ai par ailleurs dû créer une interface graphique afin de rendre le logiciel utilisable par le plus grand nombre, avec pour objectif d'avoir une interface claire, non surchargée et fonctionnelle.

V.3 Approche, travail réalisé

Lorsque j'ai commencé à travailler sur l'algorithme de détection de frontières, je me suis tourné vers une idée d'enveloppe convexe pour pouvoir trouver le bord du nuage de points, mais l'idée s'est retrouvée rapidement limitante lorsque la géométrie de l'aile était plus complexe qu'un plan bombé. J'ai alors opté pour une approche plus géométrique, basée sur la configuration des éléments. Une première piste était de compter le nombre d'élements auxquel appartient un nœud. Cette méthode fontionne lorsque le maillage n'est constitué que de carrés, mais est inutile dans les autres cas. J'ai alors cherché si des papiers de recherche avaient abordé ce problème en essayant de trouver des situations équilvalentes à la mienne. Finalement, en cherchant comment reboucher un maillage, j'ai lu l'article [6], qui propose une méthode basée sur des « *halfedge* » pour détecter la zone correspondant au trou. Je me suis alors renseigné sur cette méthode.

V.3.1 DCEL

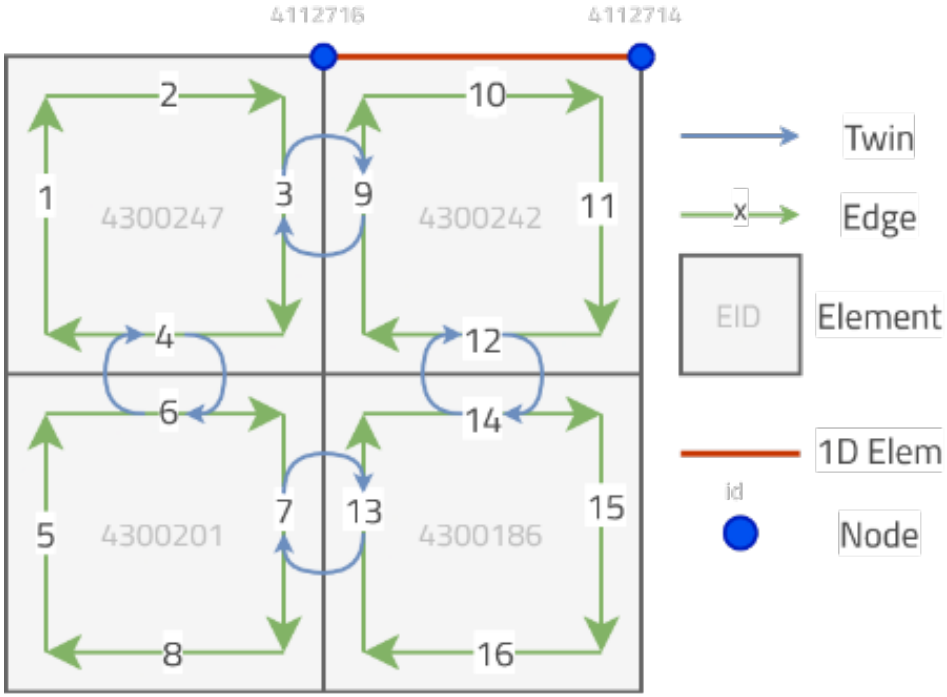


Fig. 8. – Représentation d'un modèle DCEL dans un cas simple

Grâce à la piste du modèle « *halfedge* » j'ai découvert l'existence de la structure de donnée « DCEL », (litt. liste d'arêtes doublement connectée) qui est une structure permettant justement de trouver les frontières d'un maillage aisément. Cette structure comporte :
– Des nœuds
– Des faces
– Des « *halfedge* », des arêtes orientées.

Une arête est rattachée à une face, a un sommet de départ, elle est liée à une arête la précédent et une suivante, ces deux dernières nécessairement attachées à la même face. Elle possède aussi possiblement une arête jumelle, une arête attachée à une face adjacente et qui est orientée dans le sens contraire.
Dans la Fig. 8, l'arête 3 a pour face l'élément 4300247, son arête précédente est 2 et la suivante 4, son nœud de départ est 4112716, son arête jumelle est la 9. L'arête 2 n'a pas de jumelle.

Construire une telle structure de données est une opération initialement coûteuse, mais une fois obtenue, beaucoup d'opérations sont simplifiées. De plus, la taille des données en entrée ne dépassant jamais 1000 éléments, la complexité s'est avérée un problème moins important que prévu. Afin de savoir si un élément est sur une frontière, il suffit de regarder si l'une de ses arêtes n'a pas de jumelle. Pour savoir si un maillage a un trou, il suffit de regarder s'il existe plus d'une boucle fermée.

V.3.2 Arêtes, éléments 1D, *mapping*

Maintenant se pose un nouveau problème : découper la boucle extérieure en quatres arêtes. J'ai finalement opté pour un calucl de produit scalaire le long de la boucle. Et les classe par valeur croissante, ainsi les premiers éléments correspondent aux virages « brutaux », puisque le produit scalaire approche 0 lorsque deux arêtes sont orthogonales ou presque.
Parmi les trois cas rencontrés dans les modèles, le troisième (Fig. 9) est problématique puisque le coin est moins marqué, et pourrait être confondu avec une courbure forte localement. Le cas ne s'est jamais présenté en pratique, mais la vérification graphique dans le logiciel est présente en partie pour ces cas difficiles à juger.

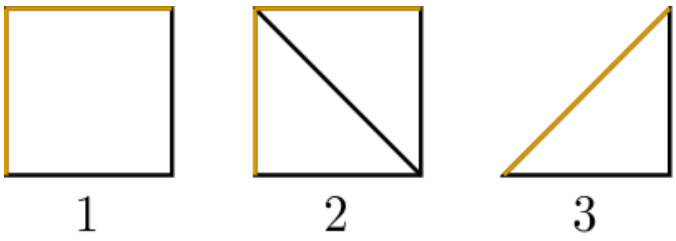


Fig. 9. – Trois cas rencontrés lors de la détection de coins

La détection d'éléments 1D est faite grâce à des requêtes dans la base de données du modèle. On cherche si un élément 1D partage deux sommets avec les éléments sur la bordure, on peut les observer sur la Fig. 10, ce sont les traits oranges sur les bordures.

Un autre défi de la séparation en quatre arêtes a été leur labélisation. Si le panneau n'est pas parfaitement aligné avec les axes du repère, il devient très difficile de trouver avec précision quelle arête est à gauche, à droite, en haut, etc. Cette idée a donc été abandonnée à la fin de mon stage au profit de la possiblité de renommer les arêtes manuellement dans l'interface graphique, il s'agit du « 1. » dans la Fig. 18.

Enfin, le *mapping*, le cœur du logiciel. Après avoir construit tous ces éléments ce n'est plus si difficile (dans les cas faciles). Pour le réaliser, on prend une des quatres arêtes comme point de départ, puis on parcourt la structure avec des bandes en prenant à chaque fois les voisins de la bande précédente. Un résultat est l'image de droite de la Fig. 10. Les cas comportant des triangles sont les cas non triviaux. Lorsque la structure en possède, le parcours à base de voisins se décale au fil des bandes. Le même problème apparaît lorsque l'on a un trou dans la structure. Il est possible de pallier partiellement ce problème en effectuant deux *mappings* en partant d'arêtes opposées, puis le reconstituer à partir de la sortie du programme. Cette solution convenait à l'équipe, c'est un compromis, chercher à améliorer l'algorithme de manière à couvrir toutes les géométries serait chronophage.

V.3.3 Interface graphique

Initialement la visualisation utilisait un navigateur, mais cette solution était relativement peu réactive, ainsi, j'ai opté pour une application de bureau en *Qt*. J'ai pris cette décision basé sur le fait que d'autres applications *Qt* ont déjà été développées par l'équipe logiciel de Montréal. J'ai donc pu partir d'une base pour construire ma propre application.

V.4 Résultats

Le logiciel est évidemment le résultat principal, c'est le livrable le plus important de mon stage. Comme dit dans la Section III.2, cet outil permet de faire gagner plusieurs heures à des ingénieurs voulant réaliser des *mappings*.

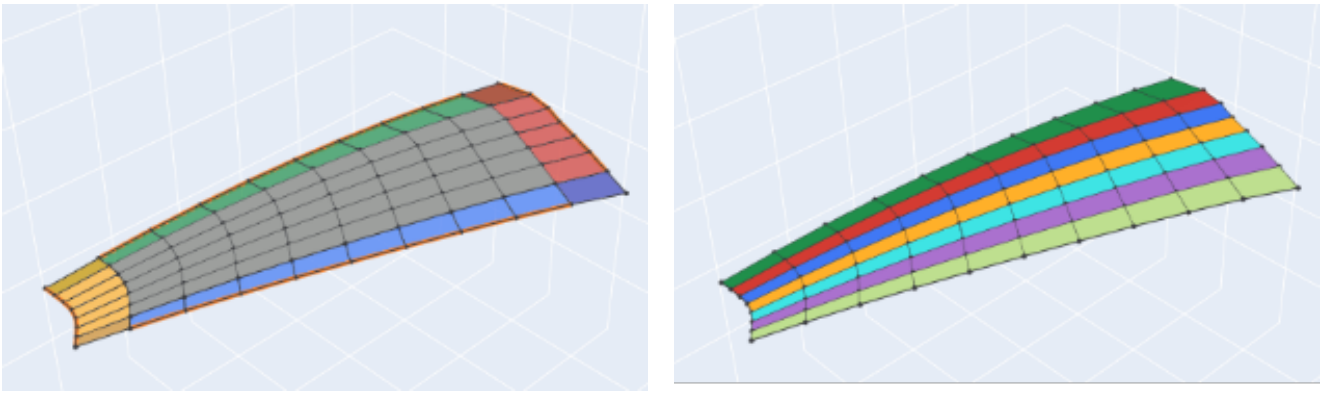


Fig. 10. – Exemple de rendus proposé par le logiciel

L'interface graphique et sa visualisation n'est pas la seule sortie du logiciel. Il génère aussi un fichier CSV et un fichier Excel (même contenu, juste mieux formaté, voir Fig. 11). Le *mapping* est exporté de manière à garder les paramètres de rotation de l'utilisateur (l'interface permet de faire tourner le *mapping* de manière à bien l'aligner).

L	M	N	O	P	Q	R	S	T	U	V	W
					Mapping						
	4203292	4203274	4203254	4203233	4203213	4203193	4203181	4203173			
4300823	4106272	4106189	4106115	4106074	4106024	4105976	4105933	4105892	4105857	4105824	4300264
4300825	4106274	4106202	4106137	4106086	4106030	4105980	4105939	4105911	4105884	4105851	4300276
4300826	4106286	4106216	4106154	4106094	4106039	4106004	4105969	4105930	4105895	4105866	4300289
4300836	4106294	4106224	4106162	4106106	4106069	4106027	4105983	4105951	4105927	4105890	4300302
4300846	4106299	4106238	4106179	4106139	4106091	4106040	4106017	4105978	4105936	4105923	4300319
4300856	4106302	4106267	4106215	4106159	4106108	4106084	4106036	4106000	4105972	4105935	4300334
4300870	4106320	4106291	4106232	4106177	4106148	4106098	4106064	4106033	4105994	4105974	
	4203313	4203305	4203296	4203284	4203271	4203260	4203246	4203235			

Fig. 11. – Exemple de *mapping*, en orange les éléments 1D

VI Conclusion

J'ai eu l'occasion pendant ce stage de travailler sur des sujets passionnants et très concrets : la conception et l'implémentation d'outils ayant pour portée principale de simplifier le travail des autres employés.

Ce stage aura été une réelle opportunité pour moi de mettre mes compétences en programmation au profit directe d'une équipe professionnelle, c'est une expérience très rafraichissante après deux ans d'études. J'ai aussi pu constater que programmer à plein temps permet d'apprendre à utiliser de nouveaux outils extrêmement rapidement. J'ai pu me perfectionner en Python, apprendre à utiliser *Qt* que j'utiliserai peut-être à nouveau pour des projets personnels.

Par ailleurs, ce fut également l'occasion de mettre en pratique mes connaissances en aéronautique et mécanique des matériaux acquises lors de mon cursus à l'ISAE-Supaero, faisant de ce stage la parfaite conclusion de mon double diplôme.

Références

[1] C. Kassapoglou, *Design and Analysis of composite structures*. John Wiley & Sons, 2013.

[2] Bruhn E.F., *Analysis and design of flight vehicle structures*. Tri-State Offset Company, 1973.

[3] « PDM python packet manager ». Consulté le: 30 janvier 2025. [En ligne]. Disponible sur: https://pdm-project.org/

[4] Jones R.M., *Buckling of Bars, Plates, and Shells,*. Bull Ridge Publishing. [En ligne]. Disponible sur: https://books.google.com/books?id=UzVBr8b_jS8C,

[5] « The HDF5® Library & File Format ». Consulté le: 31 janvier 2025. [En ligne]. Disponible sur: https://www.hdfgroup.org/solutions/hdf5/

[6] C. Feng, J. Liang, M. Ren, G. Qiao, W. Lu, et S. Liu, « A Fast Hole-Filling Method for Triangular Mesh in Additive Repair », 2020. [En ligne]. Disponible sur: https://www.mdpi.com/2076-3417/10/3/969

Annexes

A. Courbes OSB

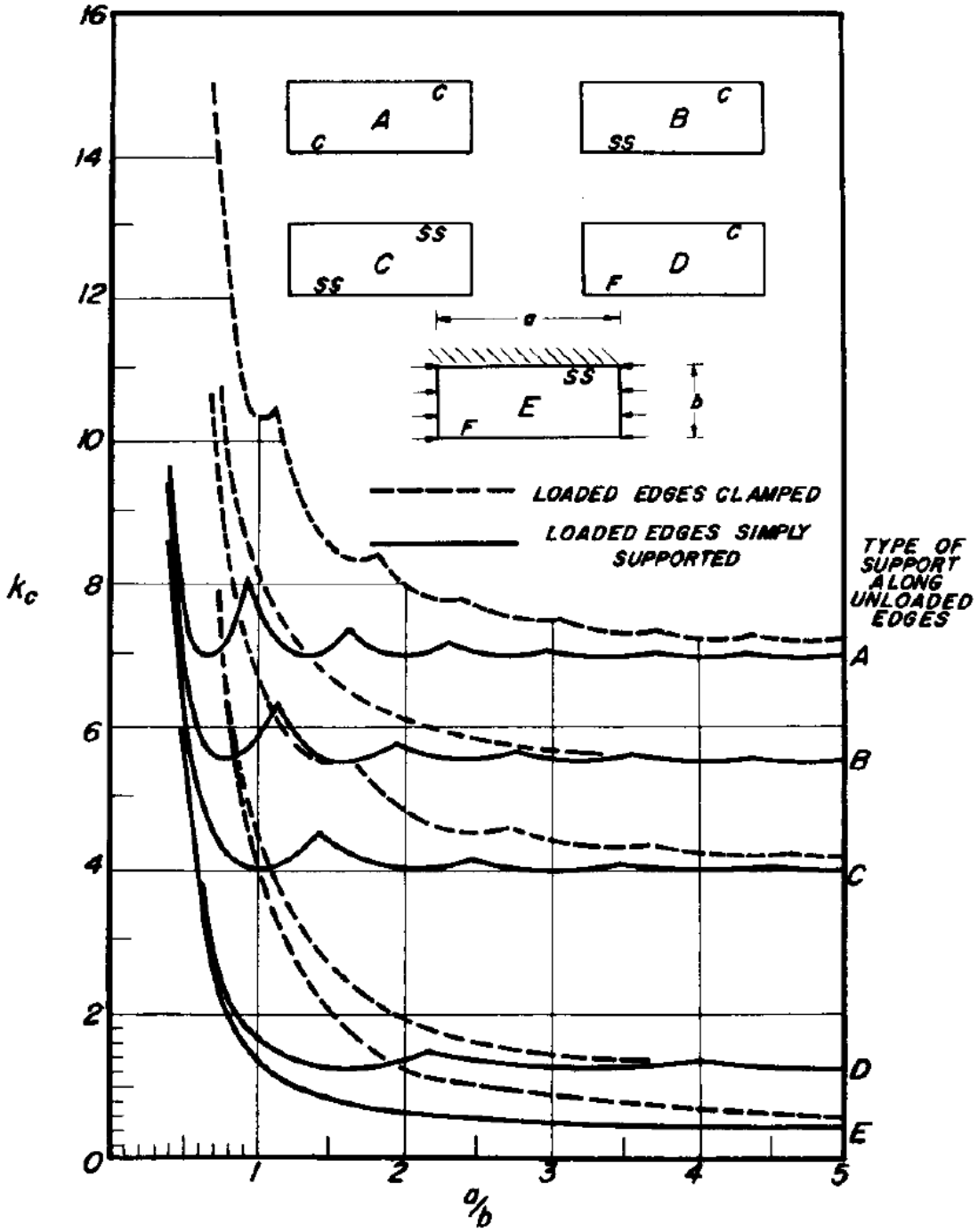


Fig. 12. – Exemple de courbe empirique

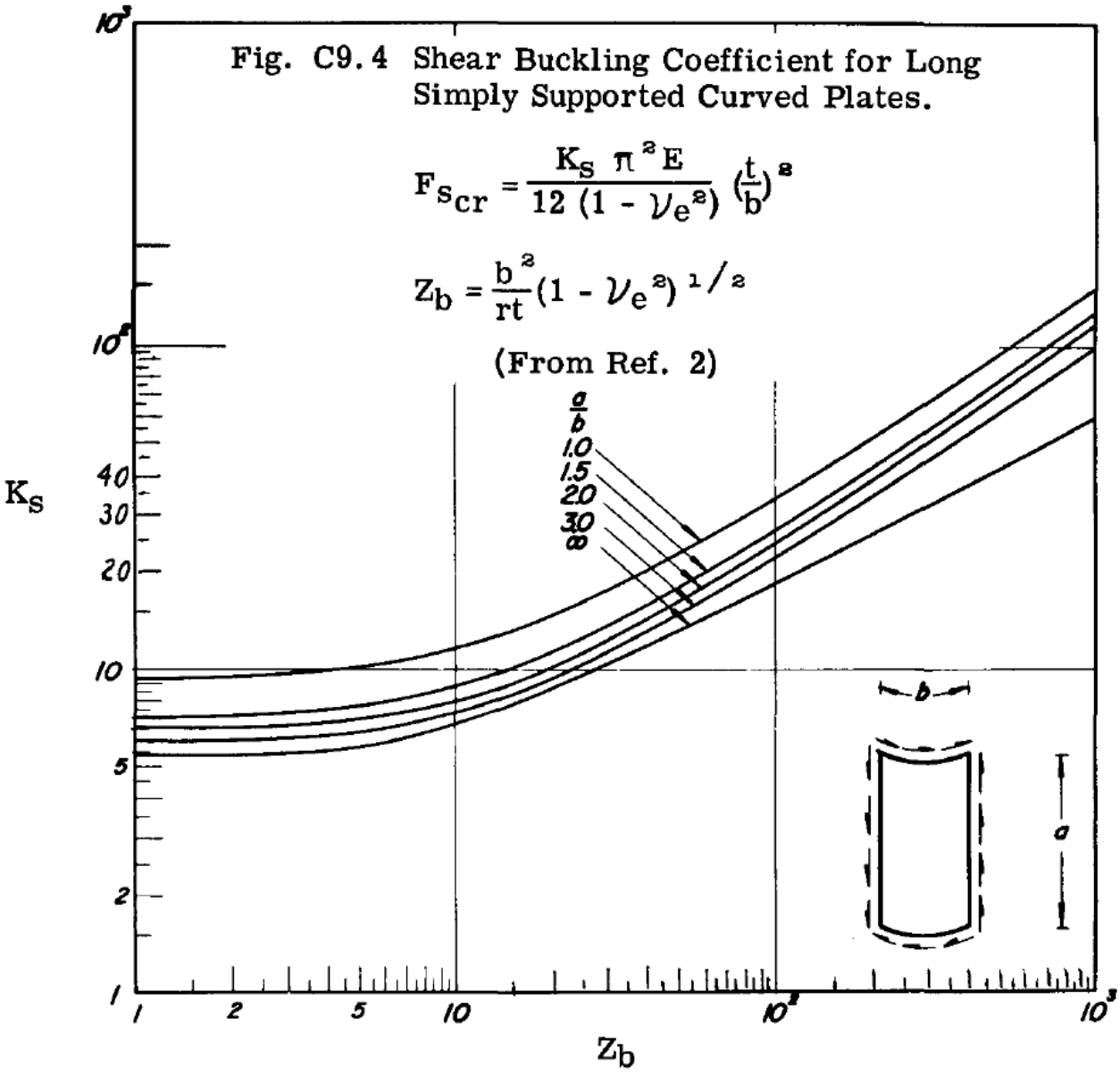


Fig. 13. – Autre exemple de courbe

B. Exemples de *pipelines*

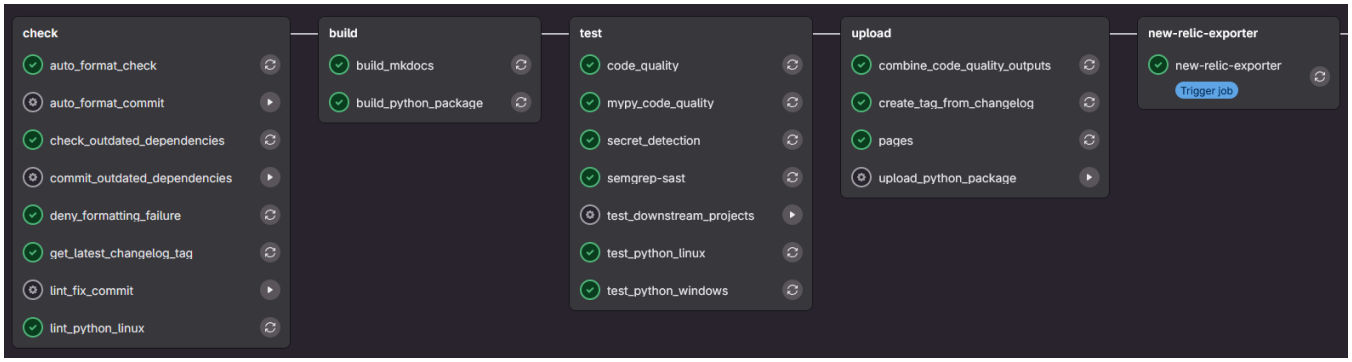


Fig. 14. – Exemple de *pipeline* fonctionnelle

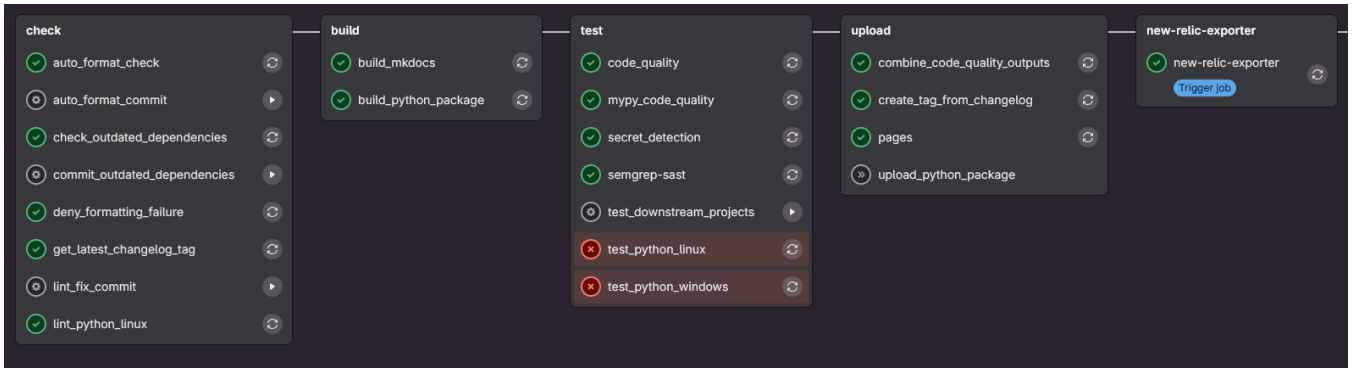


Fig. 15. – Exemple de *pipeline* échouant à passer les tests

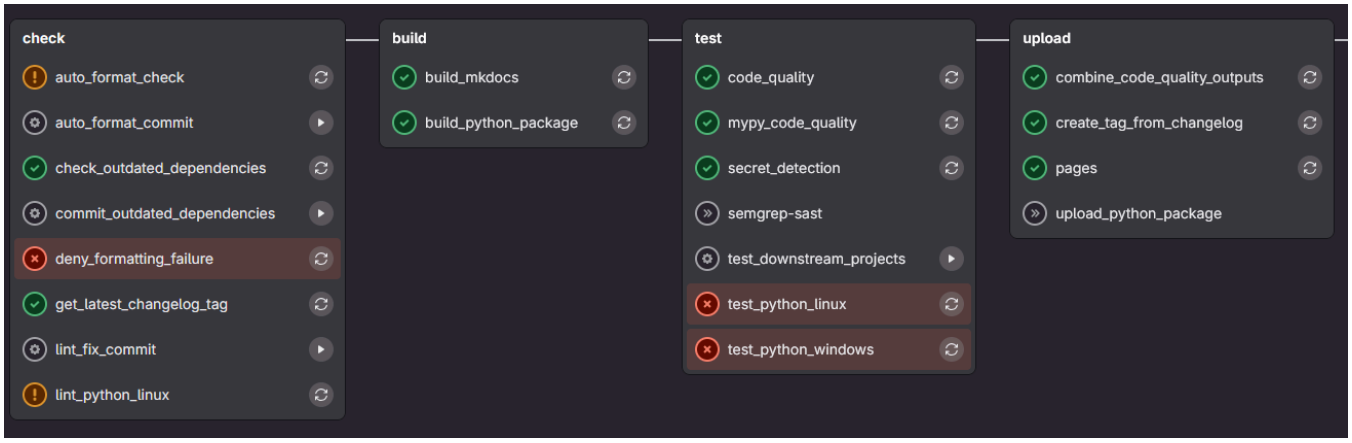


Fig. 16. – Exemple de *pipeline* totalement défectueuse

C. Guide d'utilisation du *Mapping Tool*

Extrait de ma présentation de fin de stage, permet de voir l'interface. Les commentaires indiquant les fonctionnalités sont également issus de la présentation (laissés en anglais par conséquent).

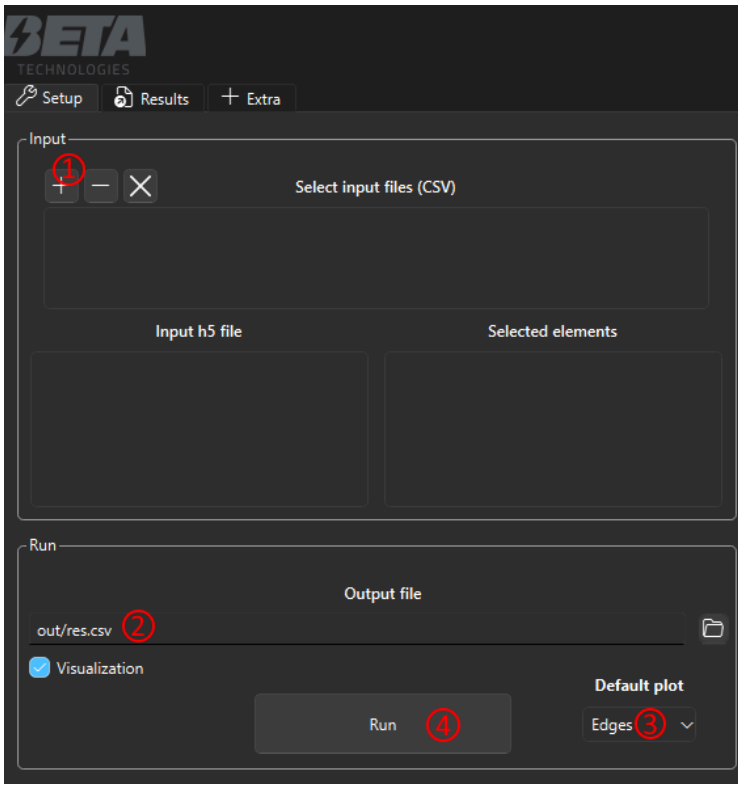


Fig. 17. – Guide *Mapping Tool* (1)

1. Add a compatible CSV file as input
2. Select the output
3. Select the type of plot
4. Run the algorithm

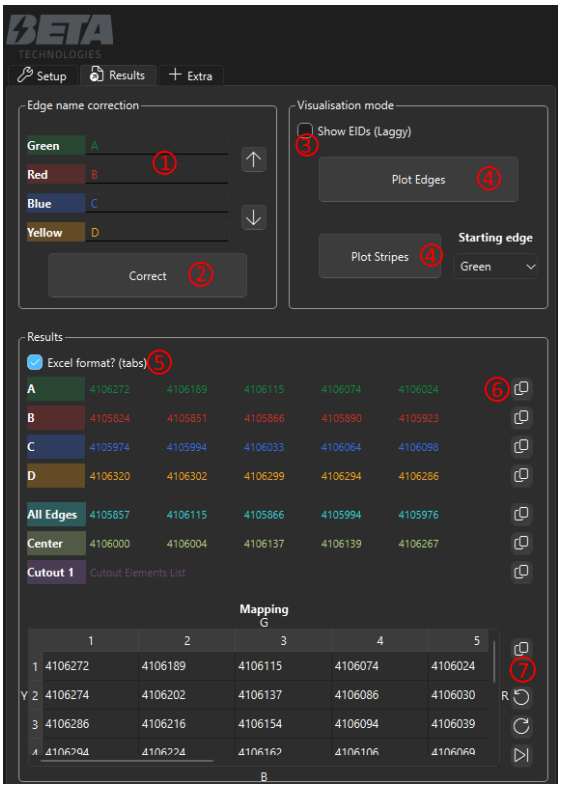


Fig. 18. – Guide *Mapping Tool* (2)

1. Edge renaming
2. Update names and result files/visuals
3. Enable EIDs on visual
4. Re-plot, change mapping start (resets mapping!)
5. Switch between tabs/spaces in lists
6. Copy elements
7. Mapping rotation, flip, copy