Arts et Métiers Sciences et Technologies

Rapport PJT2A: FMEAgen

Mai 2022

Auteurs :
DANG Khoa-Minh
KANG Laëtitia
PLOTEAU Leïla
ZHANG Alex

Professeurs encadrants :
PETRONIJEVIC Jelena
ZHOURI Wahb

PJT2A – FMEAgen

Auteurs : DANG Khoa-Minh, KANG Laëtitia, PLOTEAU Leïla, ZHANG Alex
Professeurs encadrants : PETRONIJEVIC Jelena, ZHOURI Wahb

Table des matières

Table des figures

Table des tables

Introduction

Dans le cadre de la conception de produit, il est important d'être en mesure d'évaluer les risques liés au disfonctionnement du produit. Ces risques sont formalisés sous la forme de modes de défaillance. La gestion de risques a alors pour objectif d'identifier les causes, la probabilité d'occurrence ainsi que la gravité de ces modes afin de permettre au concepteur de modifier sa conception pour les limiter voire les supprimer. [1]

La plupart des méthodes actuelles de gestion des risques ont un principal défaut : elles sont subjectives. En effet, c'est à la personne chargée d'identifier et d'évaluer les risques de décider lequel est plus grave, lequel est le plus susceptible de se produire. La gestion des risques repose donc en grande partie sur l'expérience et les connaissances d'une ou plusieurs personnes. De plus, le processus apparaît comme étant principalement manuel. Les personnes impliquées dedans peuvent être amenées à commettre des erreurs ou encore à oublier certains modes de défaillance, voire des interactions implicites qui peuvent exister entre eux. [2]

Nous allons donc chercher à nous défaire de cette subjectivité en évaluant, par une méthode systématique et automatisée, tous les risques possibles, ainsi qu'en relevant les liens et interactions entre les différentes défaillances.

Pour cela, nous allons utiliser deux méthodes différentes pour créer la méthode systématique et automatisée que nous recherchons. La première méthode est la méthode AMDEC, qui permet d'identifier et évaluer les défaillances d'un produit. La deuxième est FBS Linkage, qui est initialement une méthode de représentation du produit sous forme de réseau et qui va nous permettre d'étudier la propagation des risques. Seules, ces deux méthodes ne permettent pas de réaliser un processus systématique et automatique mais en les combinant, nous allons tenter d'en créer une nouvelle qui remplira ces critères. [3][5]

Pour améliorer les méthodes AMDEC et FBS linkage, nous allons répondre aux deux questions suivantes :

- Comment extraire des modes défaillances d'un réseau FBS ?
- Comment améliorer la méthode AMDEC pour générer des modes de défaillance complets en retraçant les interactions entre les différents risques ?

Pour répondre à ces questions, nous nous sommes divisés en deux groupes. Le premier binôme a étudié en particulier les réseaux FBS et le deuxième s'est intéressé à la méthode AMDEC. Ensuite, pour le développement d'un premier outil, un des binômes s'est intéressé au mode de propagation d'une défaillance et l'autre a rédigé un premier programme générant un tableau AMDEC amélioré à partir de la méthode FBS.

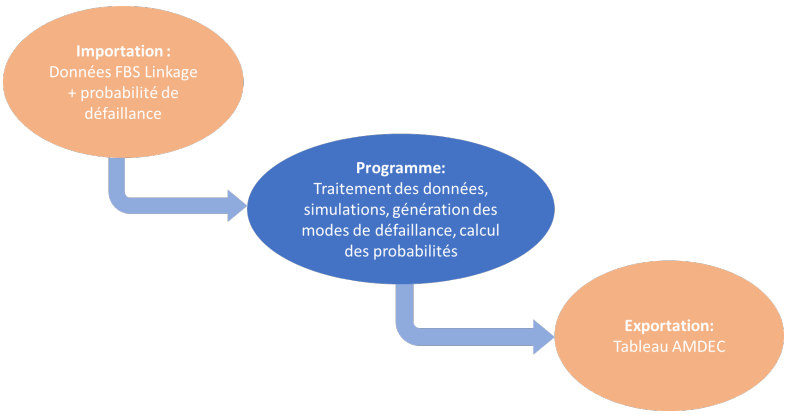
1. Objectifs

L'objectif de ce projet consiste donc à s'inspirer des méthodes AMDEC et FBS linkage afin de les améliorer pour créer un outil plus riche que les modèles classiques. En effet, ces méthodes ne sont, à l'origine, pas numériques. Pour y remédier, nous avons donc utilisé une simulation (multi-agent et épidémiologique SIR : *susceptible*, *infectious*, *recovered*) afin d'automatiser le processus de gestion de risque. Cet outil devra permettre de générer un tableau AMDEC à partir d'un réseau FBS donné :

⇒ Générer tous les modes de défaillance et les décrire dans un tableau AMDEC amélioré,
⇒ Générer leur probabilité d'occurrence à partir de simulations de propagation de défaillance,
⇒ Associer à chaque mode la (les) fonction(s) remplie(s) : les modes de défaillance doivent pouvoir être retracés depuis leur(s) origine(s) jusqu'à la fonction défaillante.

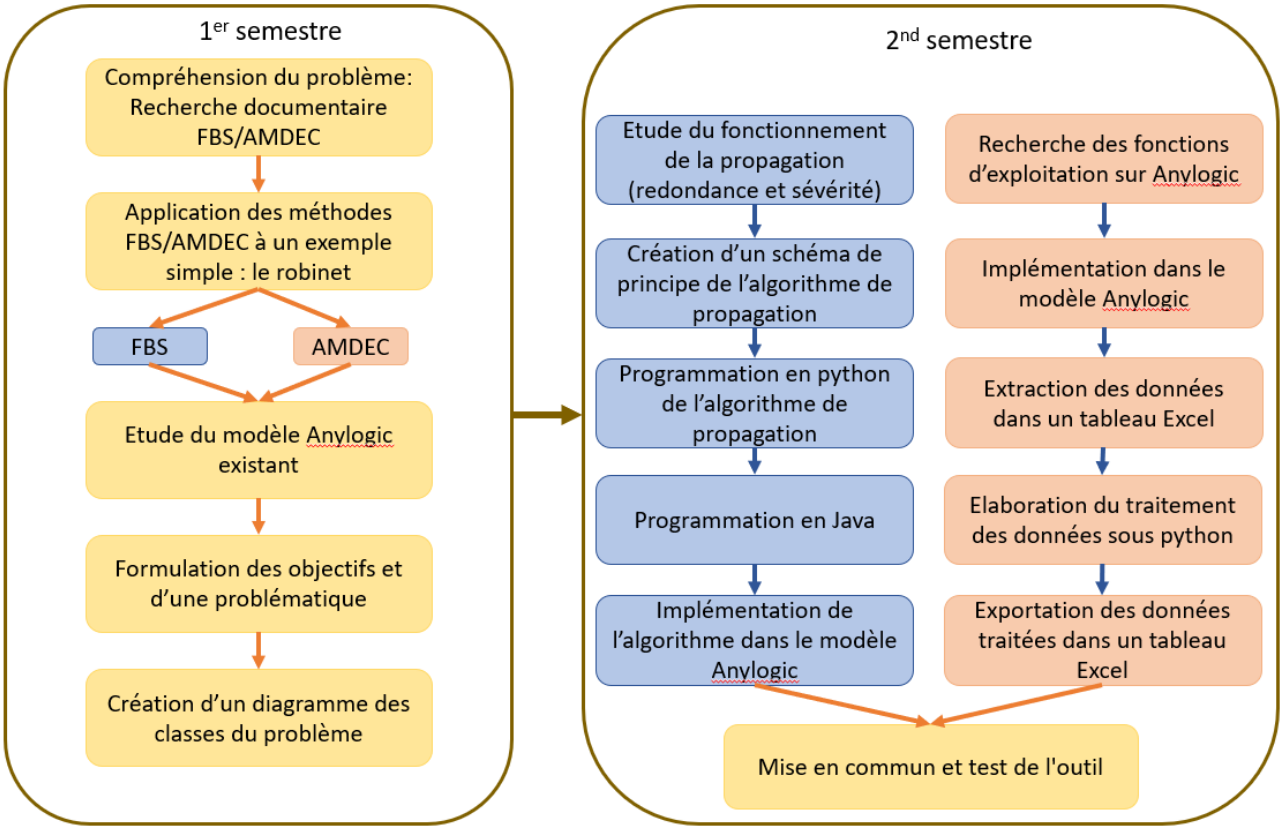
La Figure 1 schématise le processus que l'outil doit suivre :

- En entrée, on importe les données du réseau FBS d'un produit : celles-ci se présentent sous forme de tableaux type Excel (voir Annexe 1) contenant les informations suivantes :
  - Liste de tous les agents du produit (agents de structure, de comportement et de fonction)
  - Liste de tous les liens entre les agents donnés explicitement (équivalent à la matrice multi-domaine).
  - Probabilité de défaillance de chacun des agents de structure
- L'outil doit alors :
  - Reconstituer le réseau
  - Simuler, pour un nombre d'itérations donné, la propagation d'une défaillance en commençant par la défaillance d'un agent de structure. Chaque mode de défaillance ainsi généré (qui est donc toute la chaîne des agents défaillants) doit être répertorié et sa probabilité d'occurrence calculée à la fin des itérations
  - Récapituler tous les modes de défaillance ainsi trouvés avec leur probabilité d'occurrence dans un tableau en explicitant les liens entre eux (par exemple : deux modes de défaillance peuvent avoir la même cause et la même conséquence finale sans être passés par la même chaîne)
- En sortie, on doit pouvoir exporter le tableau AMDEC amélioré qui a été établi grâce aux simulations



2. Description des tâches réalisées et répartition des tâches

Nous avons construit un planning à l'aide du logiciel GANTT Project afin de s'organiser au mieux dans l'élaboration de notre outil (cf. Annexe 3). La **Erreur ! Source du renvoi introuvable.** résume les tâches qui ont été réalisées au cours de l'année.



Tout d'abord, nous nous sommes concentrés sur la compréhension du problème. Puis, afin de mieux comprendre les exigences (informations nécessaires) des différents modèles, nous nous sommes réparti les tâches : un groupe exploite le modèle FMECA/AMDEC (Laëtitia et Alex), l'autre le modèle FBS (Khoa-Minh et Leïla). Le cas particulier d'un robinet a été étudié pour définir les liens entre les deux modèles dans le but de construire un outil de génération du tableau AMDEC.

Ensuite, nous avons mis en commun les résultats de nos études afin de comparer les 2 méthodes et d'en tirer les différences notables. Par ailleurs, nous avons décidé d'analyser ensemble le modèle AnyLogic existant fourni par l'enseignant pour formuler la problématique et les objectifs. Enfin, nous avons créé un diagramme des classes du problème.

Durant le second semestre, nous avons gardé les mêmes binômes de travail. L'un des groupes (Khoa-Minh et Leïla) a travaillé sur la modification du mode de propagation ainsi que du calcul de la gravité tandis que l'autre (Alex et Laëtitia) s'est focalisé sur l'exportation des résultats de simulation ainsi que de leur traitement.

3. État de l'art

3.1. Méthode AMDEC

L'Analyse des Modes de Défaillance, de leurs Effets et de leur Criticité ou la méthode AMDEC (Failure Mode, Effects & Criticality Analysis, FMECA en anglais) est une méthode d'identification et d'évaluation quantitative et qualitative des défaillances d'un produit dans le but de les prévenir et les corriger. [1][2]

La méthode nécessite l'emploi des termes suivants :

- **Mode de défaillance :** manière dont le système peut dysfonctionner ;
- **Cause de la défaillance :** anomalie pouvant conduire à la défaillance ;
- **Effet de la défaillance :** conséquences subies par l'utilisateur ;
- **Criticité** : acceptabilité de la situation par la combinaison de plusieurs facteurs.

En supposant que le produit est connu, les étapes de la méthode AMDEC sont les suivants :

1) Identifier les modes de défaillances

L'étape consiste à se poser les questions suivantes :

Qu'est-ce qui ne fonctionne pas ?

Ce dysfonctionnement perturbe-t-il le système global ?

2) Identifier les effets et les causes

Il peut avoir plusieurs effets ou causes par mode de défaillance.

3) Évaluer les défaillances

Le but est d'attribuer à chaque défaillance un indice de criticité qui résulte de trois facteurs : la gravité de la défaillance et de l'effet (G) qui entraîne une conséquence plus ou moins grave pour l'utilisateur, l'occurrence ou fréquence d'apparition de la défaillance (O), et la probabilité de non-détection (D).

Chaque mode de défaillance est identifié à l'aide d'experts qui connaissent le comportement du produit, et chaque indice est évalué individuellement sur une échelle variant généralement de 1 à 10 (voir Table 1).

Une fois tous les modes de défaillance possibles répertoriés, ils sont regroupés dans un tableau avec leurs indices et sont classés par ordre de criticité décroissante (voir exemple Table 2). Le concepteur doit ensuite proposer des changements au produit afin de traiter les modes de défaillance les plus graves d'abord, et les moins graves après. [3]

Table 1 - Exemples de table d'évaluation des indices

Note F	Fréquence ou probabilité d'apparition	Note G	Gravité	Note D	Probabilité de non-détection
10	Permanent	10	Mort d'homme ou catastrophe environnementale	10	Aucune probabilité de détection
5	Fréquent	5	Conséquences financières et/ou matérielles	5	Un système de détection est en place mais n'est pas infaillible
1	Invraisemblable	1	Pas grave	1	Le système de détection est infaillible

La méthode AMDEC possède d'importantes contraintes. En effet, elle se base fortement sur l'expérience et les connaissances des concepteurs et experts impliqués dans la conception du produit. Cela rend l'identification des modes de défaillance fastidieuse, et l'évaluation des indices grandement qualitative et subjective (NB : Elle présente également le défaut de ne pas faire de liens entre les modes de défaillance, alors que ceux-ci sont susceptibles d'exister dans le produit réel.

Table 2 - Exemple de tableau AMDEC

AMDEC : Analyse des Modes de Défaillances, leurs Effets et leur Criticité

Type de FMEA	☐ Concept ☐ Produit ☐Process	**Equipe**	
Projet / Client	XZ51 / Blue Automotive Industries (BAI)		
Date FMEA / Dernière MàJ	12/05/2010 21/06/2010		

#	Fct Externe	Fonction Interne	Mode de défaillance	Effet potentiel du mode de défaillance	Sévérité	SC/CC	Causes potentielles	Occurrence	Confirmations Effectuées	Detection	RPN	Confirmations Additionelles	Resp	Date	Gravité	Occurrence	Detection	RPN
1	FP1 : Eclairer	Fi 1.1 – Fixer la lampe au connecteur	Perte de la fixation mécanique	Perte d'éclairage	9	CC	Mauvaise déf. dim. du connecteur	3	Revue de conception	5	135	Etude dimensionelle, essais prototype	AAM	12/03	9	3	3	81
2					9	CC	Mauvaise déf. dim. de l'embase	2	Etude dim, , essais prototype, chaine de cotes	2	36							
3			Perte intermittente du contact mécanique	Risque de court circuit (par usure)	10	CC	Usure due à un jeu trop important	2	Retour d'expérience, essais endurance	2	40							
4					10	CC	Usure due à un mauvais choix matière	2	Retour d'expérience, essais endurance, Analyse	3	60							
5					10	CC	Usure par vieillissement	2	Retour d'expérience, essais vieillissement	2	40							
6			Perte de la fixation électrique	Perte d'éclairage	9	CC					0							
7			Perte intermittente du contact électrique	Perte intermittente d'éclairage	8	SC					0							
8		Fi 1.2 – Fixer le connecteur au bâti									0							
9											0							
10											0							
11											0							

Cette méthode été développée spécifiquement pour la gestion des risques. Il existe cependant d'autres méthodes utilisées dans divers domaines, tels que la conception de produit dans notre cas, qui peuvent être exploitées pour la gestion de risques. C'est le cas de la méthode FBS Linkage, qui servira de base pour la gestion de risques telle qu'elle est exploitée dans ce projet.

3.2. Méthode FBS et FBS Linkage

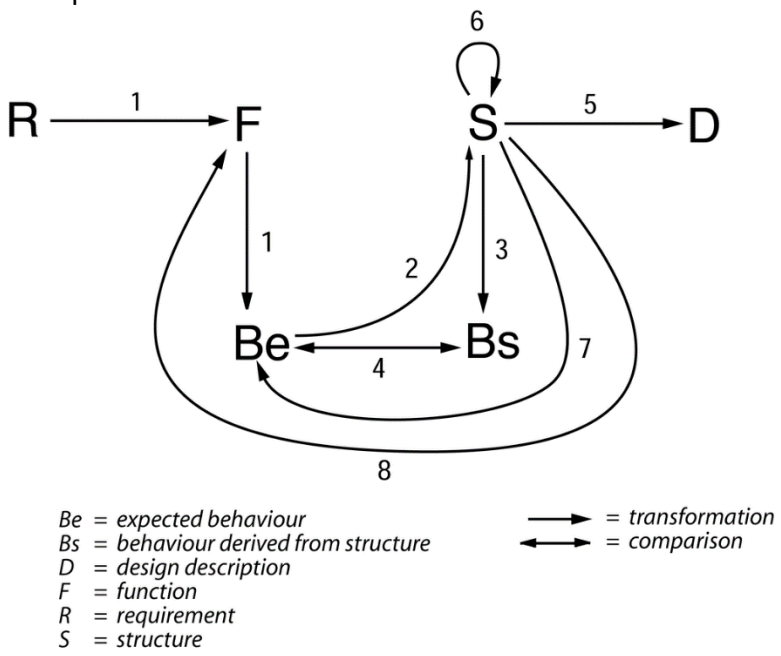
La méthode FBS Linkage est étroitement liée à la méthode FBS, c'est pourquoi cette partie s'attardera à expliquer ces deux méthodes.

FBS : Function Behaviour Structure

La méthode FBS est avant tout une démarche de conception de produit développée par John S. Gero et ses collègues [4]. Elle n'intervient habituellement pas dans la gestion des risques. Elle consiste à créer le modèle d'un produit basé sur sa décomposition en 3 types de propriétés : propriétés structurelles (en anglais « structure (S) »), comportementales (« behaviour (B) ») et fonctionnelles (« function (F) »).

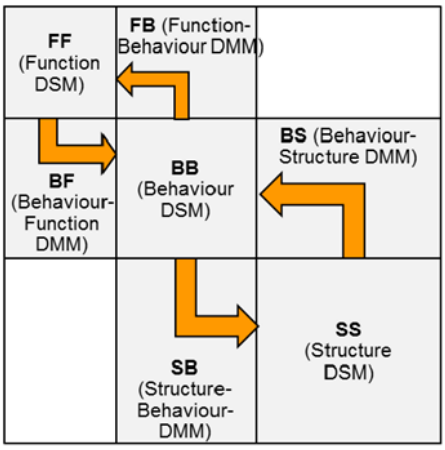
Son utilisation se décompose en plusieurs phases schématisées sur la Figure 3 :

1. **Formulation :** A partir des besoins exprimés, le concepteur commence par identifier les fonctions que le produit final doit remplir. Il quantifie ces fonctions en les associant à des comportements, qui se traduisent en général par des lois physiques régissant le système (ce sont les comportements attendus du système).
2. **Synthèse :** Une fois ces lois mises en place, on crée la structure du système.
3. **Analyse :** A partir de cette structure, on identifie les comportements qui en découlent directement (comportements « structurels »).
4. **Évaluation :** On compare les deux types de comportements déterminés précédemment afin d'évaluer si le système fonctionne comme prévu.
5. **Documentation :** On détaille la conception du produit à partir de la structure : les différentes pièces qui composent le système, leur géométrie, leur matériau, les interactions entre elles, etc…
6. **Reformulation type 1 :** ajustement de la structure à partir des étapes précédentes
7. **Reformulation type 2 :** ajustement des comportements à partir de la structure reformulée
8. **Reformulation type 3 :** ajustement des fonctions à partir de la structure reformulée et des comportements qui en découlent.



FBS Linkage : Réseau FBS

Théorisée par Dr Hamraz, cette méthode se base sur la méthode FBS de Gero ainsi que sur la méthode CPM (Change Prediction Method) de Clarkson [5]. Une fois le modèle FBS d'un produit mis en place, on établit des liens entre les différentes propriétés du système : on crée un réseau. Ce réseau comporte des liens S-S, S-B (inversement B-S), B-B, B-F (inversement F-B) et enfin F-F. Ces liens sont établis par méthode systématique détaillée dans la littérature. Ils sont déjà en partie établis par la méthode FBS comme expliqué précédemment, mais sont redéfinis de manière explicite. Pour chaque type de lien, on met en forme ce réseau sous la forme de matrices. Ces matrices sont ensuite regroupées dans une plus grande matrice nommée « matrice multi-domaine » (= multi-domain matrix MDM) schématisée sur la Figure .



La méthode FBS Linkage permet, grâce à la matrice multi-domaine, d'expliciter toutes les dépendances qui existent dans le système. Ainsi lorsqu'on souhaite modifier la conception d'un produit, on peut prédire efficacement toutes les conséquences qui découlent d'un changement qu'on effectue.

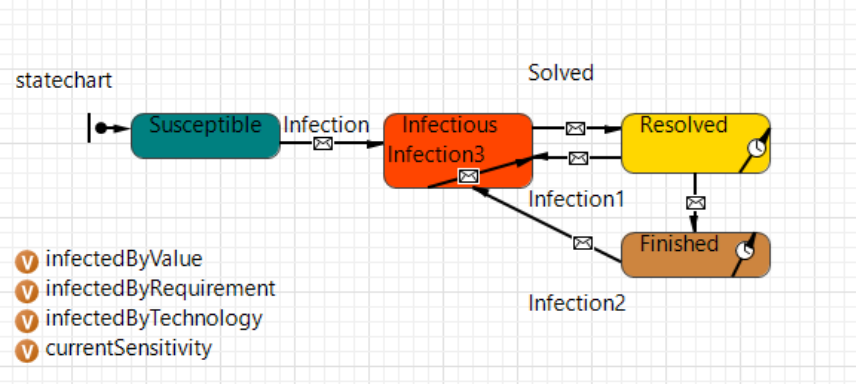
Ainsi, nous tâcherons de combiner les 2 méthodes décrites précédemment à l'aide du modèle Anylogic fourni par le professeur encadrant.

3.3. Modèle AnyLogic

Description du modèle fourni

Un modèle AnyLogic permettant de simuler la propagation des risques dans le développement d'un sèche-cheveux nous a donc été fourni. Pour construire ce modèle, des tables Excel élaborées avec la méthode FBS Linkage ont été utilisées. Il s'agit d'un modèle multi-agent basé sur le principe de propagation épidémiologique. Le sèche-cheveux est décrit par sa structure (composants, sous-composants…), ses comportements et ses fonctions. Il y a donc 3 types d'agents différents. Chaque élément existe dans l'un des 4 états suivants (voir Figure) :

- *Susceptible* : l'élément n'est pas infecté mais est susceptible de le devenir,
- *Infected* : l'élément est infecté (risque causé par un autre élément),
- *Resolved* : l'élément a été amélioré en faisant plusieurs itérations, il a 20% moins de chances d'être sous risque.
- *Finished* : l'élément n'est plus sous risque mais peut l'être à nouveau selon les concepteurs

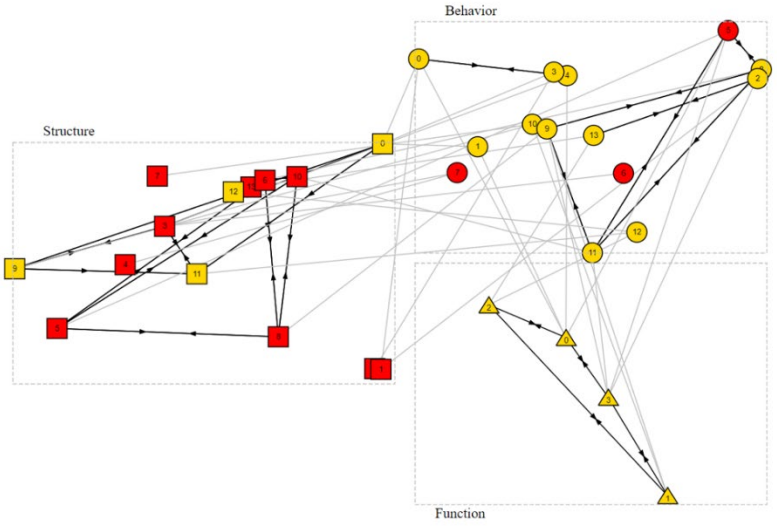


Avant le lancement d'une simulation, tous les agents sont dans l'état *susceptible*. Lorsqu'on lance une simulation, un agent passe dans l'état *infected* : il devient sous risque. Il propage ensuite son risque à l'un des agents auquel il est directement lié (ce lien est établi grâce au réseau FBS) (cf. Figure). La propagation se poursuit de la même manière dans tout le système. Le composant va ensuite alterner entre les états *infected* et *resolved* dans le but de réduire le risque d'infection à 20%.

Dans l'état *resolved*, le concepteur tente d'améliorer l'agent afin qu'il soit moins sensible aux défaillances. Si le problème est résolu, l'agent passe à l'état *finished*. Il n'est plus à risque mais si l'amélioration réalisée par le concepteur ne suffit pas il peut de nouveau être contaminé et retourner à l'état *infected*. Il peut également devenir à risque pour une autre cause que celle qui avait été résolue par le concepteur.

La propagation du risque se fait de manière uniforme, les éléments en lien avec un élément *infected* ont une probabilité uniforme d'être infecté. Par exemple, si un nœud infecté est connecté à 4 autres nœuds, il y a une chance de 0.25 de les infecter.

La Figure 6 montre la simulation à une étape intermédiaire. On peut identifier l'état de chaque nœud à l'instant t. Les carrés, triangles et cercles colorés représentent respectivement les classes Structure, Fonction et Comportement (*Structure, Behavior* et *Function*).



Les résultats montrent que les éléments ont tendance être à l'état *infected* et *resolved*, les risques sont donc très fréquents et les concepteurs cherchent à les réduire au maximum.

Comparaison modèle/méthode AMDEC et modèle souhaité

Le modèle ne gère pas la propagation des modes de défaillances, mais seulement la propagation des risques. Or on cherche bien à générer des modes de défaillance. Cependant, il permet de faire des liens de cause à conséquence entre les agents, et donc d'établir, avec les modifications adaptées, des liens entre les modes de défaillance qu'on ne pourrait pas établir manuellement. Nous chercherons donc à exploiter le mode de propagation utilisé dans le modèle pour élaborer notre outil.

Comparaison modèle/ méthode FBS Linkage et modèle souhaité

Le modèle AnyLogic utilise bien la méthode FBS Linkage comme base. En effet, le fichier importé pour lancer la simulation contient : les agents, qui sont dans l'une des trois catégories *Structure*, *Behaviour* ou *Function*, et leur mise en réseau les uns par rapport aux autres en respectant les hypothèses de la méthode (à savoir : il n'y a pas de liens entre les structures et les fonctions). C'est bien ce que nous souhaitons implémenter dans notre modèle.

Modèle à réaliser

Dans notre cas, nous cherchons seulement les modes de défaillance, non pas la propagation de risques. Nous ne souhaitons donc pas obtenir une solution mais les différents résultats intermédiaires des simulations, ainsi les blocs qui nous intéressent sont donc : susceptible et infecté. Cependant, le modèle AnyLogic que nous utilisons n'est pas adapté à la propagation des modes de défaillance. Ces données sont donc à extraire de la simulation. Ainsi, ces résultats intermédiaires représenteront chacun un mode de défaillance et définira une ligne du tableau.

Aussi, le modèle sera également à améliorer avec la prise en compte des redondances et la propagation par défaillance générale du système. Pour finir, une simulation générera un mode de défaillance.

4. Travail préliminaire à l'élaboration d'une solution

4.1. Appropriation du sujet et des méthodes au travers de l'exemple d'un robinet

Afin de mieux comprendre le fonctionnement et l'utilisation des méthodes AMDEC et FBS Linkage, nous les avons mises en application pour un système relativement simple : un robinet à levier. Pour cela, nous nous sommes divisés en deux binômes, chacun appliquant l'une des méthodes au système. Les travaux effectués sont présentés en Annexe 4, Annexe 5 et Annexe 6. Le but de cet exercice est de s'approprier le raisonnement derrière chaque méthode, et non pas de

chercher à créer des modèles réalistes ou utilisables (car nous ne sommes pas concepteurs de robinets), c'est pourquoi les travaux peuvent présenter des incohérences entre eux.

Cet exemple nous a également permis de souligner les différences qui existent entre ces deux méthodes. Nous avons ainsi pu mieux identifier les exigences de chaque méthode. Dans les deux cas il est nécessaire d'avoir une connaissance approfondie du système, mais plus particulièrement pour la méthode AMDEC. La méthode FBS Linkage se veut plus systématique, dans le sens où elle établit des liens logiques entre des entités définis dès le départ de la conception. La méthode AMDEC, quant à elle, nécessite que l'on sache prédire le comportement du système, « l'intuiter ». Ce sont bien les exigences que nous avions relevées lors de nos recherches bibliographiques.

4.2. Diagramme des classes

Dues aux différences relevées sur les méthodes FBS linkage, AMDEC et le modèle Anylogic et donc à la difficulté de les mettre en lien, nous avons donc réalisé une 1ère version du diagramme des classes qui sera amené à être modifié selon les exigences du projet (voir Figure 7).

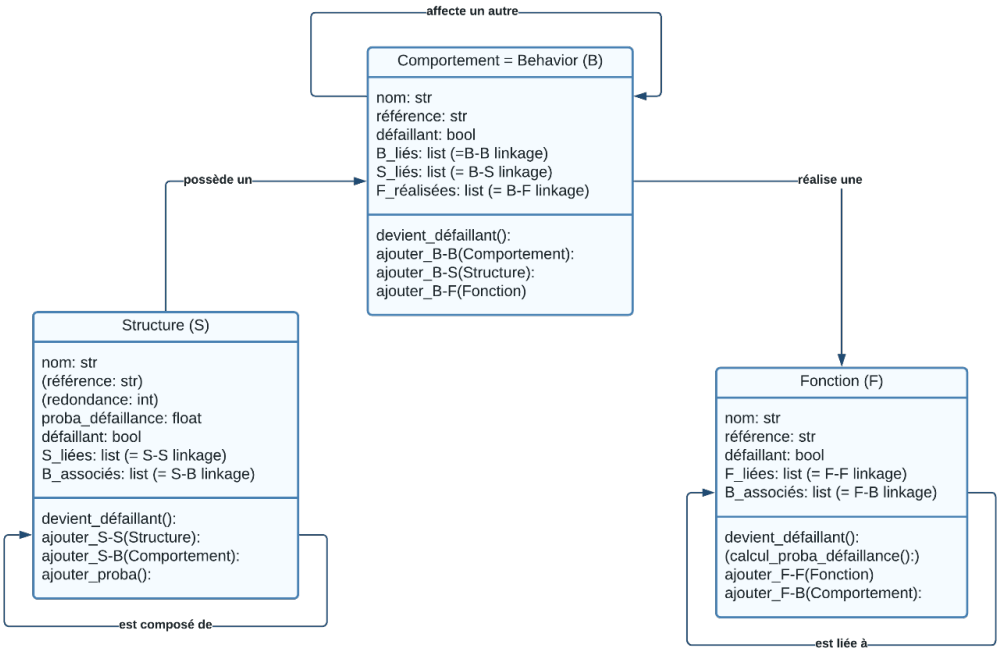


Figure 7 - Première version du diagramme des classes

5. Modification du modèle AnyLogic préexistant

Afin de construire notre outil de génération des modes de défaillance, nous avons d'abord apporté des modifications au fichier AnyLogic fourni par la professeure encadrante. Nous avons donc commencé par supprimer toutes les fonctionnalités qui ne nous intéressaient pas dans le cadre de notre projet, à savoir les états *Resolved* et *Finished*, avec toutes les fonctions, variables et liens qui y sont rattachés (voir Figure 8 et Figure 9).

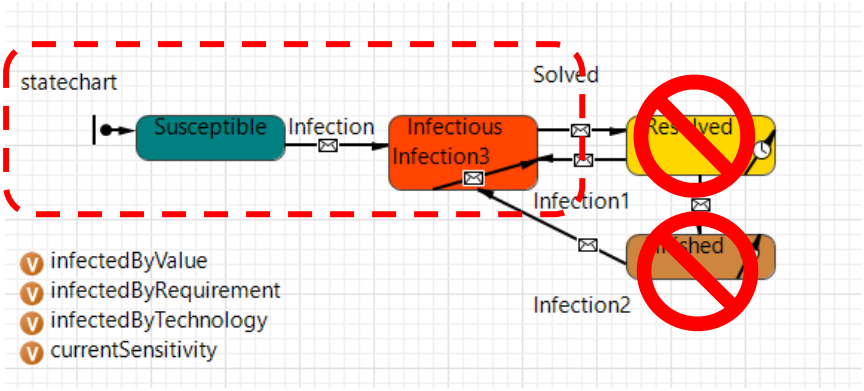


Figure 8 - Modèle AnyLogic réduit

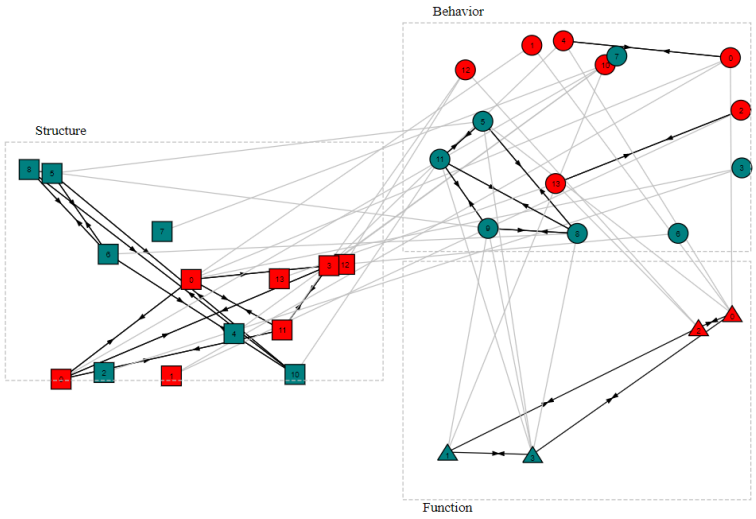


Figure 9 - Résultat de la simulation du modèle adapté

Nous avons conservé le mode de propagation épidémiologique, ainsi que la possibilité d'importer un fichier comprenant les données relatives au produit construit à partir de la méthode FBS. Puis, afin de se rapprocher du cadre de notre projet, nous avons choisi de développer la propagation des défaillances et non pas des risques (de conception) comme le modèle d'origine. Ensuite, nous avons réalisé un modèle plus complet de propagation comprenant les redondances et sévérités de chaque élément. Enfin, nous avons modifié le programme Java afin que celui-ci renvoie un tableau regroupant les résultats de simulation.

5.1. Nouvelle modélisation de la propagation

5.1.1. La gravité

La gravité (ou sévérité selon les traductions) est une donnée qui nous intéresse, car elle permet d'évaluer les modes de défaillance et de les hiérarchiser dans le but de définir ceux à traiter en priorité.

La gravité est donnée par la formule ci-dessous (et illustrée par la Figure 10 - Schématisation de la gravité) :

$$\text{gravité} = \text{sensibilité}_{\text{noeudvictime}} \times \text{sensibilité}_{\text{noeudattaquant}} \times \text{force}_{\text{lien}}$$

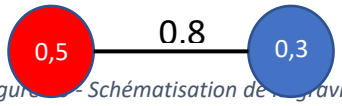


Figure [illegible] - Schématisation de la gravité

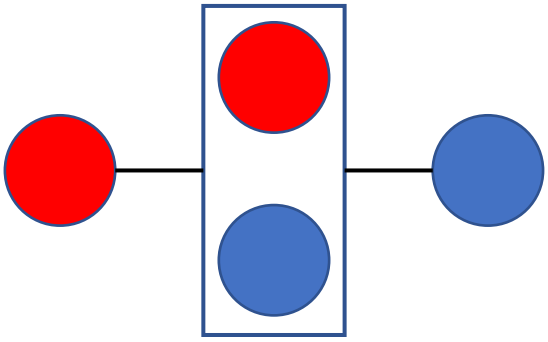
Le nœud attaquant est celui à l'origine de la propagation. Le nœud victime est celui qui va être infecté par la défaillance. Les trois paramètres qui servent à calculer la gravité sont fournis par le fichier Excel importé.

Lors de la propagation, un nœud ne passe à l'état infecté que si la gravité de la contamination est inférieure à son seuil minimal de gravité, lui aussi donné dans le fichier d'importation.

5.1.2. *Les redondances*

Dans le cas des structures, nous allons considérer aussi les cas de redondance, c'est-à-dire qu'une structure peut exister en plusieurs fois. Cette caractéristique est illustrée par le paramètre « redondance » qui est importé du fichier initial grâce à une nouvelle colonne que nous avons ajouté.

Les redondances servent à augmenter la résistance aux défaillances du système. Si un composant est infecté mais qu'il possède une ou plusieurs redondances saines, les fonctions et comportements ne seront pas impactés par la défaillance (voir Figure 11).



5.1.3. *Propagation par défaillance du système*

Notre modèle va prendre en compte les défaillances globales du système, c'est-à-dire que nous allons considérer qu'un agent peut devenir défaillant si un trop grand nombre de ses voisins sont défaillants. Pour cela, nous ajoutons une variable *vois_défaillant* qui correspond au nombre actuel de voisins défaillants d'un agent et un paramètre *vois_défaillant_max* qui est importé grâce au fichier Excel. Pour chaque agent, si le nombre de voisins défaillants dépasse le nombre de voisins défaillants maximum, l'agent devient automatiquement lui aussi défaillant (voir Figure 12).

Dans ce cas particulier, nous avons considéré que la sensibilité du nœud infecté deviendrait égale à celle du dernier nœud de son voisinage à avoir été infecté.

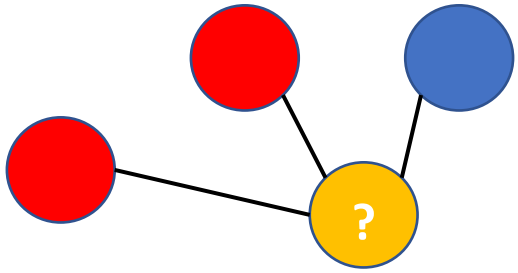
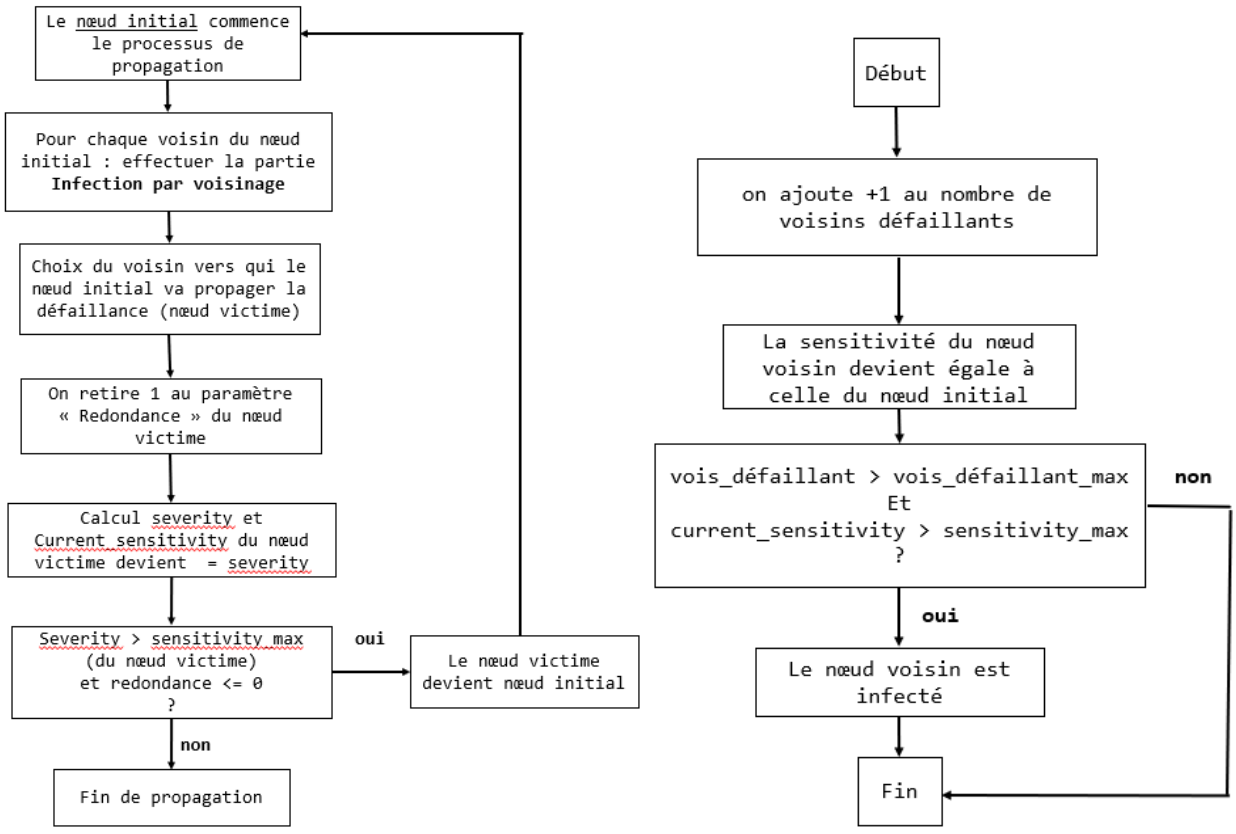


Figure 12 - Propagation par défaillance générale. L'état de l'agent en orange dépend du nombre de voisins à partir duquel il est automatiquement infecté

5.1.4. Modélisation finale

Par soucis de clarté, le modèle final est divisé en deux parties comme présenté en Figure 13. La propagation par défaillance du système, qui se trouve à droite, est appelée par le modèle global, à gauche.



5.1.5 Implémentation du modèle sur AnyLogic

Après avoir réalisé cet algorithme, nous avons cherché à le mettre en forme dans le langage Python (voir Annexe), puis nous l'avons implémenté en Java dans le modèle AnyLogic avec l'aide de notre professeure encadrante.

Afin qu'il soit fonctionnel, nous avons dû ajouter des colonnes au fichier importé (voir Table 3). Ces colonnes contiennent les informations nécessaires pour réaliser la propagation telle qu'elle est décrite ci-dessus, et elles doivent être renseignées par le concepteur.

Table 3 - Table d'importation modifiée : en orange les colonnes qui ont été ajoutées

id	name	value	requirement	technology	**vois_max**	**sensitivity_max**	**Redundancy_structure**
0	F.Geometry	1,00	0,90	1,000	10	1,000	0
1	F.Material	1,00	1,00	1,000	10	1,000	0
,,,	,,,	,,,	,,,	,,,	,,,	,,,	,,,

Nous avons ensuite ajouté des paramètres au programme AnyLogic qui extraient ces valeurs des tables tels qu'ils sont visibles dans la Figure 14.

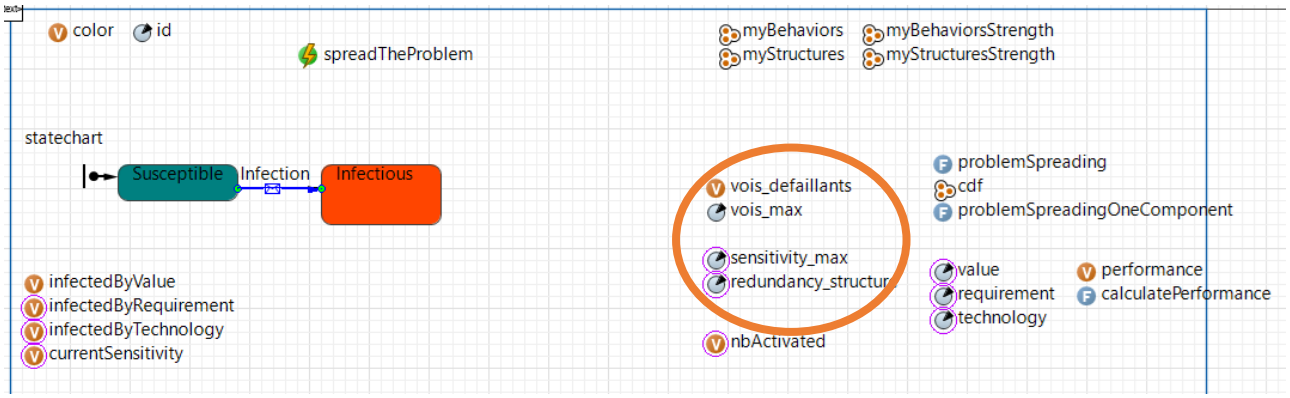


Figure 14 - Ajout des paramètres dans le programme

En exécutant une simulation, on peut rendre compte du bon fonctionnement de la propagation comme illustré dans la Figure 15.

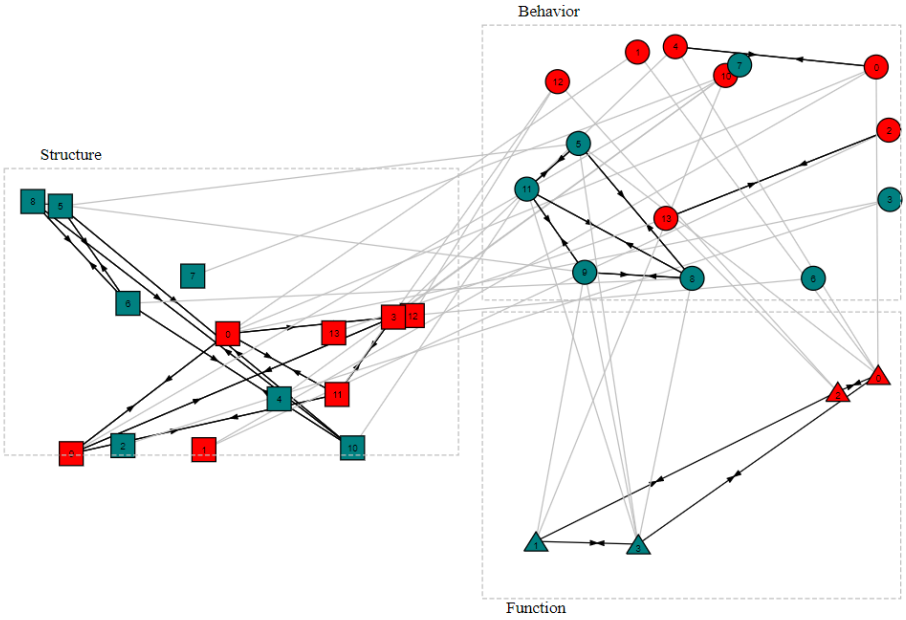


Figure 15 - Résultat d'une simulation

5.2. Exportation des résultats du modèle AnyLogic

En premier lieu, nous avons modifié le programme Java afin que celui-ci renvoie lors de la simulation la liste des infections produites, autrement dit qu'il affiche les éléments défaillants et leur cause associée sous la forme : « Effet *infected by* Cause ». Nous avons ensuite utilisé les fonctions d'écriture de fichier Excel du logiciel Anylogic en appelant le tableau Excel dans le logiciel. De plus, nous avons créé une fonction *writeOutput* qui permet au logiciel d'écrire directement les résultats des simulations sur la dernière feuille du fichier Excel *FBSdatabase.xlsx*. Nous avons choisi d'utiliser directement le fichier *FBSdatabase.xlsx* car il contient toutes les données du produit, les interactions entre les différents éléments (structure, comportement et fonction) décrites par la méthode FBS Linkage, et leur sensibilité. Ainsi, cela nous permet de mieux traiter les données. Par ailleurs, nous avons choisi d'afficher les résultats du fichier AnyLogic sous forme d'un tableau à 4 colonnes :

- La nature de la cause
- L'ID de l'élément causant la défaillance
- La nature de l'élément défaillant
- L'ID de l'élément défaillant

5.3. Code VBA sur Excel puis Python pour le traitement des données

Après avoir exporté les données sur une feuille Excel, nous avions d'abord pensé à utiliser le code VBA pour traiter les données. Il fallait commencer par supprimer les doublons que nous avions pu avoir en retournant les résultats de la simulation. Pour cela, on a utilisé la fonction *RemoveDuplicates* dans le code VBA. Cependant, il s'avère que le code VBA ne nous permettait pas

de modifier le fichier d'origine à notre guise, en outre, nous ne pouvions pas faire de retour en arrière après avoir fait fonctionner le code.

Par conséquent, nous avons décidé d'effectuer le traitement des données en Python (cf. code en Annexe 7), langage que l'on maîtrise mieux. Le programme Python que nous avons élaboré a plusieurs fonctionnalités. En premier lieu, il retire les doublons des résultats de simulation, afin de faciliter la lisibilité des données.

Puis, afin de pouvoir créer un tableau AMDEC amélioré, *i.e.* un tableau AMDEC indiquant des informations supplémentaires comme les relations entre les modes de défaillances, nous avons exporté depuis le modèle AnyLogic un tableau regroupant l'historique de propagation des défaillances d'une seule simulation. La colonne de gauche recense effectivement les éléments qui ont causé la défaillance des éléments de la colonne de droite, et ceux par ordre chronologique de leur infection. Le code consiste à relever toutes les propagations issues d'une défaillance structurelle, en partant de la fonction la plus récemment infectée (fin du tableau). On remonte ensuite le tableau afin d'y retrouver les origines de la propagation des défaillances pour chaque fonction. Il suffit de rechercher dans la colonne Effets, la cause associée à la fonction, et ainsi de suite jusqu'à tomber sur un élément de type fonction ou structure. Ce chemin de propagation représente ainsi une ligne du tableau. Un exemple est donné en vert sur la Figure 16, nous sommes partis de la fonction 3, qui a été défaillante en raison de la défaillance des comportements 9, 11, 8 puis 5, causée par la fonction 0.

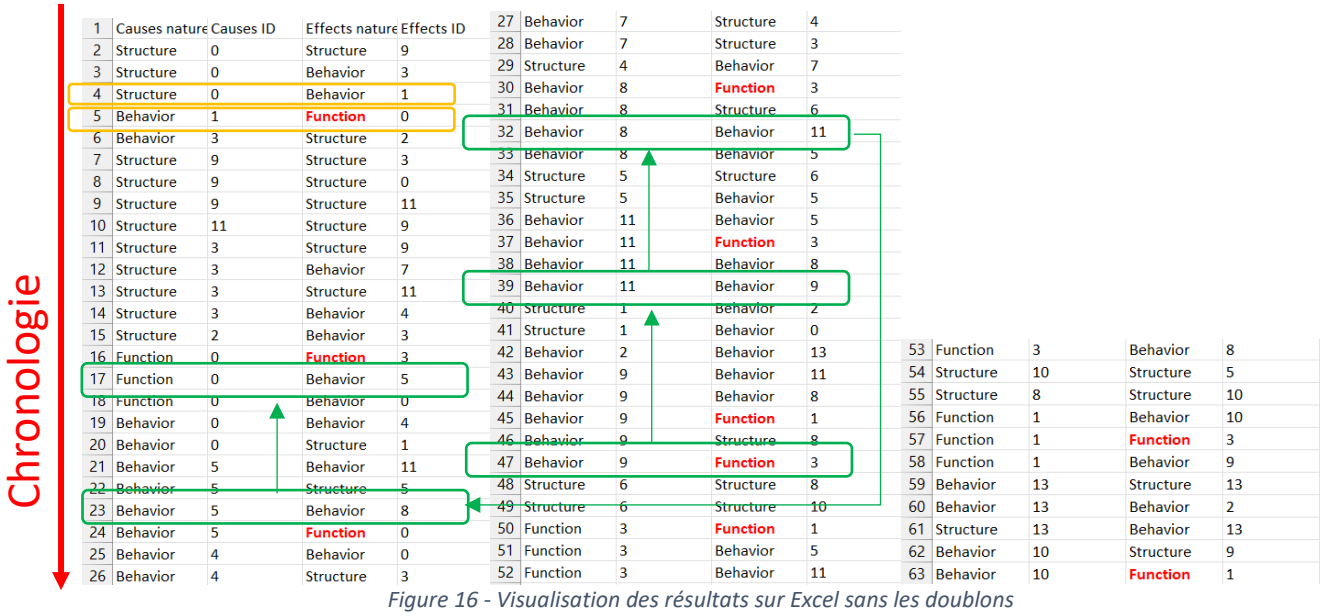


Figure 16 - Visualisation des résultats sur Excel sans les doublons

En réitérant cet algorithme sur chaque fonction se trouvant dans la colonne Effets, on relève tous les modes de défaillances et donc plusieurs lignes du tableau AMDEC (cf. Figure 17). L'idéal serait d'obtenir une défaillance de structure pour origine d'une défaillance de fonction. Par exemple, en orange sur la Figure 16, la fonction 0 est défaillante car le comportement 1 l'est à cause de la structure 0. Ce schéma représente ce qui est indiquée généralement sur les tableaux AMDEC. Néanmoins, le fait de trouver comme origine une fonction, met en évidence les relations de dépendance entre les différentes lignes du tableau AMDEC, ce qui n'est à l'origine pas le cas.

Fonctions défaillantes **Historique des propagations**

	0	1	2	3	4	5
0	Function 1	Behavior 10	Function 1	None	None	None
1	Function 3	Function 1	None	None	None	None
2	Function 1	Function 3	None	None	None	None
3	Function 3	Behavior 9	Behavior 11	Behavior 8	Behavior 5	Function 0
4	Function 1	Behavior 9	Behavior 11	Behavior 8	Behavior 5	Function 0
5	Function 3	Behavior 11	Behavior 8	Behavior 5	Function 0	None
6	Function 3	Behavior 8	Behavior 5	Function 0	None	None
7	Function 0	Behavior 5	Function 0	None	None	None
8	Function 3	Function 0	None	None	None	None
9	Function 0	Behavior 1	Structure 0	None	None	None

Figure 17 - Tableau python présentant l'historique de propagation des défaillances

Par ailleurs, afin que le tableau Figure 17 se rapproche de la forme d'un tableau AMDEC, nous avons choisi de transformer le tableau afin que chaque ligne ne contienne que 3 colonnes : la fonction défaillante, le mode de défaillance et la source de la défaillance. Le tableau est ensuite exporté dans un tableau Excel pour plus de lisibilité. L'idéal serait de l'exporter dans le même fichier comportant les données et informations du produit étudié.

Fonctions défaillantes **Modes de défaillance** **Cause de la défaillance**

	0	1	2
0	Function 1	Behavior 10	Function 1
1	Function 3	Function 1	None
2	Function 1	Function 3	None
3	Function 3	Behavior 9	Function 0
4	Function 3	Behavior 11	Function 0
5	Function 3	Behavior 8	Function 0
6	Function 3	Behavior 5	Function 0
7	Function 1	Behavior 9	Function 0
8	Function 1	Behavior 11	Function 0
9	Function 1	Behavior 8	Function 0
10	Function 1	Behavior 5	Function 0
16	Function 0	Behavior 5	Function 0
17	Function 3	Function 0	None
18	Function 0	Behavior 1	Structure 0

Figure 18 - Tableau Python modifié

	A	B	C	D
1		**0**	**1**	**2**
2	**0**	Function 1	Behavior 10	Function 1
3	**1**	Function 3	Function 1	
4	**2**	Function 1	Function 3	
5	**3**	Function 3	Behavior 9	Function 0
6	**4**	Function 3	Behavior 11	Function 0
7	**5**	Function 3	Behavior 8	Function 0
8	**6**	Function 3	Behavior 5	Function 0
9	**7**	Function 1	Behavior 9	Function 0
10	**8**	Function 1	Behavior 11	Function 0
11	**9**	Function 1	Behavior 8	Function 0
12	**10**	Function 1	Behavior 5	Function 0
13	**16**	Function 0	Behavior 5	Function 0
14	**17**	Function 3	Function 0	
15	**18**	Function 0	Behavior 1	Structure 0

Figure 19 - Exportation du tableau sur Excel

6. Prochains objectifs

Toutefois, il reste encore des objectifs à atteindre et des fonctionnalités à implémenter dans l'extraction des données. En effet, pour la propagation, il nous faut ajouter la possibilité de réinfecter un élément et mettre à jour sa sensibilité. Concernant l'exportation du tableau, il faut tout d'abord revoir la mise en forme en utilisant le code VBA Excel afin de regrouper les lignes par fonction en fusionnant les cellules ainsi que de rechercher les désignations associées aux éléments et les remplacer dans le tableau Excel. Par ailleurs, il est nécessaire d'ajouter les informations complémentaires dans le tableau, autrement dit d'associer à chaque mode de défaillance sa gravité (sévérité) et de réaliser une simulation Monte Carlo pour déterminer la probabilité d'occurrence des modes de défaillance dans le modèle AnyLogic. Enfin, il serait également judicieux de simplifier et de basculer le code python en java sur AnyLogic ou en code VBA sur Excel.

Conclusion

Afin de prévenir les risques, l'analyse des différents modes de défaillances lors de la conception d'un produit est cruciale. Cependant, la méthode AMDEC souvent utilisée n'est pas suffisamment objective, étant basée sur l'expérience des experts. Ainsi, il serait intéressant de créer un outil permettant de générer un tableau AMDEC enrichi à partir d'un réseau FBS Linkage donné par le concepteur, et qui a pour but d'automatiser la gestion de risques.

Nous avons atteint le premier objectif du projet qui consiste à créer une première version de l'outil de génération de tableau AMDEC amélioré. Celui-ci prend en compte le mode de propagation que nous avons élaboré ainsi que les calculs des paramètres pertinents pour la génération du tableau AMDEC amélioré. Il permet également d'exporter les données liées à la propagation, de les trier et de les mettre en forme dans un tableau. Il reste cependant à perfectionner cet outil, en lui permettant d'effectuer une simulation de Monte Carlo et de la traiter ensuite, et de mettre en forme de manière plus graphique les modes de défaillances.

Ce projet a été pour nous une introduction à la conception de produit et à l'analyse des risques. Au travers des recherches bibliographiques que nous avons effectuées, nous avons su assimiler des notions nouvelles que nous n'avions jamais abordées auparavant. Il nous a également permis d'apprendre à utiliser des fonctionnalités du logiciel AnyLogic que nous ne connaissions pas, ainsi qu'à découvrir le langage de programmation Java.

Bibliographie

[1] «AMDEC : définition et mise en oeuvre,» 18 Novembre 2021. [En ligne]. Available: https://www.manager-go.com/management-de-la-qualite/amdec.htm.

[2] «Analyse des modes de défaillance, de leurs effets et de leur criticité,» [En ligne]. Available: https://fr.wikipedia.org/wiki/Analyse_des_modes_de_d%C3%A9faillance,_de_leurs_effets_et_de_leur_criticit%C3%A9.

[3] «Failure Mode, Effects & Criticality Analysis (FMECA),» Consulté le 26 décembre 2021 à 21:27. [En ligne]. Available: https://quality-one.com/fmeca/.

[4] J. S. Gero, «Design prototypes : a knowledge representation schema for design,» 1990.

[5] B. Hamraz, N. H. M. Caldwell, T. W. Ridgman and P. J. Clarkson, "FBS Linkage ontology and technique to support engineering change management," 2015.

[6] B. Hamraz, N. H. M. Caldwell et P. J. Clarkson, «A Multidomain Engineering Change Propagation Model to Support Uncertainty Reduction and Risk Management in Design» 2012.

[7] A. Dong et I. Y. Tumer, «Creating Faultable Network Models of Complex Engineered Systems,» 2014.

[8] T. R. Browning, "Sources of Performance Risk in Complex System Development," 1999.

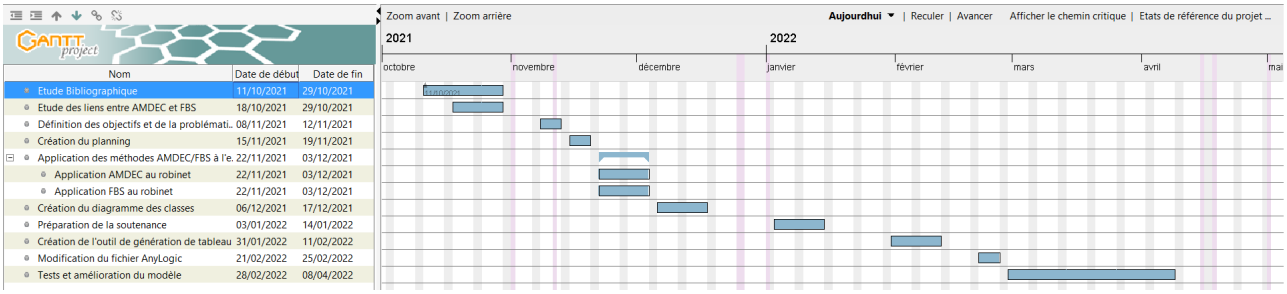
Annexes

Structure_id	Connected to structure_id	Strength	Sensitivity
0	3	1,00	1,00
0	9	1,00	1,00
0	11	1,00	1,00
3	0	1,00	1,00
3	9	1,00	1,00
3	11	1,00	1,00
5	6	1,00	1,00
5	8	1,00	1,00
5	10	1,00	1,00
6	5	1,00	1,00
6	8	1,00	1,00
6	10	1,00	1,00
8	5	1,00	1,00
8	6	1,00	1,00
8	10	1,00	1,00
9	0	1,00	1,00
9	3	1,00	1,00
9	11	1,00	1,00
10	5	1,00	1,00
10	6	1,00	1,00
10	8	1,00	1,00
11	0	1,00	1,00
11	3	1,00	1,00
11	9	1,00	1,00

Annexe 1 - Tableau de mise en réseau des agents de Structure (modèle AnyLogic)

ID	Function name	No	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36	37	38	39	40
F1	Start engine	1		0.3		0.3	0.3										0.3																									
F2	Import electricity	2	0.3						0.3															0.3																			
F3	Control torque	3															0.3																									
F4	Import liquid fuel	4	0.3											0.3																												
F5	Import air	5	0.3							0.3																																
F6	Import oil	6										0.3																														
F7	Convert elect. energy to rot. ene	7		0.3																0.3																						
F8	Transport air to chamber	8					0.3			0.3		0.3															0.3															
F9	Translate piston down	9							0.3		0.3					0.5				0.5																0.3		0.5					
F10	Transport oil to moving parts	10						0.3			0.3					0.3					0.3								0.3									0.3		0.3			
F11	Decrease air temparature	11							0.3						0.3							0.3																				
F12	Transport fuel to injector	12				0.3											0.3				0.3																				
F13	Control air flow	13											0.3					0.3															0.3									
F14	Translate piston up	14									0.5	0.3					0.5			0.5								0.5							0.3							
F15	Control fuel flow	15	0.3		0.3									0.3												0.3																
F16	Compress air	16													0.3	0.5			0.5						0.5																	
F17	Stop therm. energy	17																0.5							0.3			0.5													
F18	Convert rotary motion of cranksh	18							0.3		0.5					0.5													0.5		0.3						0.5				
F19	Convert rot. energy to hydr. ene	19										0.3		0.3												0.3				0.3					0.3			0.3			
F20	Convert rot. energy to pneum. e	20											0.3																	0.3		0.3								
F21	Convert rot. energy to electr. En	21																												0.3											0.3
F22	Convert elect. energy to therm.	22		0.3																					0.3																
F23	Increase air temparature	23																0.5	0.3					0.3				0.8													
F24	Distribute fuel to chamber	24															0.3				0.3							0.8													
F25	Convert therm. energy to pneum	25								0.3																													0.3			
F26	Convert chem. energy of fuel to	26																	0.5						0.8	0.8		0.8				0.3	0.3								
F27	Translate piston down	27										0.3				0.5				0.5								0.8		0.8					0.3		0.5				
F28	Convert linear motion of piston t	28																										0.8		0.3											
F29	Stop vibration	29													0.3					0.3	0.3	0.3	0.3							0.3				0.3							0.3	
F30	Dissipate acoustic energy	30																										0.3													
F31	Remove exhaust bypass	31																				0.3						0.3						0.3							
F32	Control exhaust flow	32																													0.3		0.3				0.3				
F33	Collect oil	33									0.3					0.3												0.3									0.3		0.3			
F34	Decrease system temperature	34																			0.3																			0.3		
F35	Translate piston up	35									0.5	0.3								0.5								0.5							0.3				0.3			
F36	Export exhaust	36																									0.3							0.3			0.3				
F37	Cool oil	37										0.3									0.3														0.3						
F38	Export Heat	38																																		0.3					
F39	Export torque	39																													0.3										
F40	Export electricity	40																					0.3																		

Annexe 2 - Function DSM (Design Structure Matrix), matrice des liens structurels pour l'exemple d'un moteur diesel



Annexe 3 - Description du planning sur le logiciel GANTTProject

Fonction	Mode de défaillance	Causes possibles	Effet de la défaillance	Moyens de détection	Gravité (1 à 10)	Détectabilité (1 à 10)	Probabilité (1 à 10)	Criticité	Actions de maîtrise des risques
Distribuer l'eau (chaude/froide)	Eau ne coule pas	Détérioration tuyau (fuites, casse)	Arrêt distribution	Ouverture robinet	2	10	3	6	Réparation tuyauterie
	L'eau ne coule plus par le tuyau, mais au niveau des joints	Mousseurs dans robinet défectueux, présence de calcaire ou corps	Utilisation difficile	Visuel	3	7	4	12	réparation joints et robinet
	Robinet fuit même robinet fermé à fond	Serrage	gaspillage eau et nuisance sonore	À l'ouïe	2	6	4	8	Resserrage robinet et joints
Réguler la température	Impossibilité d'obtenir de l'eau chaude	Source extérieure	Température eau froide	Au toucher	2	4	8	16	Permettre l'arrivée d'eau
		Mécanisme interne tourne mal	Température eau froide	Au toucher	3	4	2	6	Réparer
	Plus d'eau froide, que de l'eau bouillante	Chauffe eau défectueux	Eau très chaude	Au toucher, à la vapeur d'eau	6	4	3	18	Réparation chauffe-eau
Contrôler le débit	Débit d'eau faible	Détérioration tuyau (fuites minimes)	Faible débit	Ouverture robinet	2	5	4	8	Réparation tuyauterie
	débit d'eau trop fort	Pression dans canalisation	Gaspillage et utilisation difficile	Visuel, toucher	3	4	3	9	Régulation pression dans les canalisations

Annexe 4- Tableau AMDEC de l'étude du robinet

Type	ID	Description	Value	Requirements	Technology
Function	F0	Distribuer l'eau	0,95	1,00	1,00
Function	F1	Contrôler la température	0,91	1,00	1,00
Function	F2	Contrôler le débit	0,94	1,00	1,00
Behavior	B0	Jet d'eau	0,95	1,00	1,00
Behavior	B1	Rotation du robinet de débit	0,91	1,00	1,00
Behavior	B2	Rotation du robinet de température	0,94	1,00	1,00
Behavior	B3	Température du corps de robinet	0,90	1,00	1,00
Behavior	B4	Bruit du robinet de température	0,95	1,00	1,00
Behavior	B5	Bruit du robinet de débit	0,91	1,00	1,00
Behavior	B6	Hydrodynamisme du bec	0,94	1,00	1,00
Behavior	B7	Approvisionnement en eau	0,90	1,00	1,00
Behavior	B8	Étanchéité du joint	0,95	1,00	1,00
Structure	S0	Géométrie du col de robinet	0,95	1,00	1,00
Structure	S1	Matériau du col de robinet	0,91	1,00	1,00
Structure	S3	Géométrie des robinets	0,90	1,00	1,00
Structure	S4	Matériau des robinets	0,95	1,00	1,00
Structure	S5	Géométrie du joint	0,91	1,00	1,00
Structure	S6	Matériau du joint	0,94	1,00	1,00
Structure	S7	Géométrie du mitigeur	0,90	1,00	1,00
Structure	S8	Matériau du mitigeur	0,95	1,00	1,00

Annexe 5 - Tableau répertoriant toutes les entités FBS de l'étude du robinet

F_no	B_no
F0	B7
F0	B8
F0	B6
F0	B0
F1	B2
F2	B1

Annexe 6 - Liens F-B du robinet établis sous forme de tableau

Annexe 7 – Programme Python générant un tableau AMDEC

```
import pandas as pd #pour les tableaux
import openpyxl #pour importer et exporter sous fichier Excel

# In[0] -- Importation des colonnes du fichier Excel dans des listes

data =
pd.read_excel(r'C:\Users\Hoover48\Desktop\GIM2\PJT2A\EITM_FBSRiskSimulation\EITM_FBSRiskSimulatio
n\FBSdatabase - Copie.xlsx', sheet_name='FMECA')

#Suppression des lignes duppliquées
filedata = data.drop_duplicates()
#print(filedata)

#Importation des colonnes dans des listes : Nature et ID
List_CausesN= filedata['Causes nature'].values.tolist()
List_CausesID= filedata['Causes ID'].values.tolist()
List_EffectsN= filedata['Effects nature'].values.tolist()
List_EffectsID= filedata['Effects ID'].values.tolist()

NbInfect = len(List_CausesN)

#Concaténation des listes pour faciliter les recherches et comparaison
List_Causes = []; List_Effects = []
for k in range(NbInfect):
    List_Causes.append(str(List_CausesN[k]) + ' ' + str(List_CausesID[k]))
    List_Effects.append(str(List_EffectsN[k]) + ' ' + str(List_EffectsID[k]))

#Inverser les listes pour partir du plus récemment infecté et remonter à l'origine
List_Causes.reverse()
List_Effects.reverse()

#In[1] -- Recherche des propagations (mode de défaillance)


def findpath():
    """
    Recherche la fonction le plus récemment infectée puis retourne la propagation liée (le
"chemin" d'infection) dont la source est une fonction ou une structure"
    """

    ###Initialisation###
    e=0 #compteur à 0 pour balayer la liste dont les éléments causes et effets sont classés par
ordre anti-chronologique
    cause = List_Causes[e] #première cause de défaillance
    path = [] #chemin de propagation "path"
```

```
    func_or_not = True #True si on cherche une fonction, c'est le cas pour le 1er élément
recherché
    ElLeft = len(List_Causes) #nombre d'éléments restants "nb elements left = El Left"

    ###Recherche dans l'historique###
    while cause != 'Structure 0' and e!=ElLeft:
        eff=List_Effects[e] #effets causée par "cause"
        if eff.split()[0] == "Function" and cause.split()[0] != 'Function' and func_or_not:
            '''effet est une fonction/cause n'est pas une fonction/on cherche une fonction'''
            firstfunct=e+1 #indice de la fonction la plus récemment infectée

            #ajout de l'effet puis de la cause associée (ordre antichronologique)#
            path.append(eff)
            path.append(cause)

            #recherche des origines de la cause (=dernière fonction infectée)#
            ec = e #on part de l'indice de la fonction infectée

            #on recherche les origines de la cause en réitérant la recherche #
            while List_Effects[ec]!= cause and ec!=ElLeft: #cause devient effet : on recherche la
cause de cet effet
                ec+=1

            #ajout de la nouvelle cause dans la liste path (chemin de propagation)
            cause=List_Causes[ec]
            path.append(cause)
            e=ec
            func_or_not = False #nous ne cherchons plus une fonction : les opérations qui nous
intéressent sont ci-dessous

        elif eff.split()[0] == "Function" and cause.split()[0] == 'Function' and func_or_not:
            '''effet est une fonction & cause est une fonction & on cherche une fonction'''

            firstfunct=e+1
            path.append(eff)
            path.append(cause)
            del List_Causes[:firstfunct]
            del List_Effects[:firstfunct]
            return path
            func_or_not = True
            del path[:]

        elif eff == cause and eff.split()[0] !='Function' and not func_or_not and e!=ElLeft:
            '''la cause est l'effet d'un autre élément& l'effet n'est une fonction& on ne cherche
pas une fonction'''
            ec = e
            while List_Effects[ec]!= cause and ec!=NbInfect:
```

```
                ec+=1
            eff=List_Effects[ec]
            cause=List_Causes[ec]
            path.append(cause)
            e=ec

        else:
            '''la cause n'est pas l'effet d'un autre élément, on passe'''
            e+=1
            if func_or_not :
                eff = List_Effects[e]
                cause = List_Causes[e]
    del List_Causes[:firstfunct]
    del List_Effects[:firstfunct]

    return path

def findallpath():
    """
    Recherche tous les chemins de propagation (find all path)
    """
    AllPath = [] #liste de toutes les propagations de défaillance
    lenght = len(List_Effects) #taille de la liste initiale de l'historique
    lenmin = len(List_Effects)-1 #taille de liste tronquée finale (indice de la première fonction
infectée)
    while List_Effects[lenmin].split()[0] !='Function' and lenmin >= 0 :
        lenmin -= 1
    lenmin+=1
    while len(List_Causes) >= lenght - lenmin +1 :
        AllPath.append(findpath())
    return AllPath


# In[3] -- Mise en forme du tableau

foundpath = findallpath() #liste de toutes les chemins de propagation de défaillance
print(pd.DataFrame(foundpath))

def formatting(foundpath):
    """
    Met en forme le tableau en mettant les comportements sur une colonne en fonction de la
fonction"
    """
    nblinei = len(foundpath) #nombre de chemins (= nombre de ligne dans le tableau initial)
    nblinef = 0 #nombre de ligne dans le tableau final - à calculer
    for line in range(nblinei):
        '''Calcul du nombre de ligne final = le nombre de behavior dans tout le tableau'''
```

```
        nblinef += len(foundpath[line])-2

    for line in range(nblinef):
        '''Mettre les behavior sur chaque ligne'''

        if len(foundpath[line]) > 3:
            addpath = [foundpath[line][0]] #création d'une nouvelle ligne avec comme seul élément
qu'est la fonction
            endpath = [foundpath[line][-1]] #liste contenant le dernier élément pour le rajouter
            col = 2 #indice de la colonne du tableau initial
            element = foundpath[line][col] #element se situant dans la ligne line et la colonne
col
            nbcol=len(foundpath[line]) #nombre de colonne associée à la ligne line

            while (element.split()[0]!='Function' or element.split()[0]!='Structure') and
col<=nbcol-2:
                addpath.append(foundpath[line].pop(col-1)) #ajouter d'un élément behavior
                foundpath = foundpath[:line] + [addpath+endpath] + foundpath[line:] #liste
contenant fonction, behavior, cause
                addpath = addpath[:1] #réinitialisation de la liste avec le seul élément qu'est
la fonction
                nbcol = len(foundpath[line]) #nombre d'élément associée à la ligne line
            nbline = len(foundpath)
    return foundpath

foundpath = formatting(foundpath) #tableau formatté
print(foundpath)

#In[4] -- Append in the Excel File



df = pd.DataFrame(foundpath) #mise en forme du tableau
dataexport = df.drop_duplicates() #suppression des éléments dupliqués
print(dataexport)

#path =
"C:\Users\Hoover48\Desktop\GIM2\PJT2A\EITM_FBSRiskSimulation\EITM_FBSRiskSimulation\FBSdatabase -
Copie.xlsx"

with pd.ExcelWriter('FMECA.xlsx') as writer:
    '''exportation du tableau dans un tableur Excel'''
    dataexport.to_excel(writer, sheet_name='FMECA')
```