TD N°1: Sur les techniques de test fonctionnel

Christoph Samuel – Jankowiak Matthias

I-PROBLÉMATIQUE :

1-L'ingénieur de test doit utiliser des techniques dans le cadre de son activité de test d'un logiciel. Ces techniques sont multiples et variés afin de tester le logiciel le plus efficacement possible. Cette première série de travaux dirigés, a pour but de mettre en œuvre dans la pratique certaines techniques de test fonctionnel introduites en cours, notamment les techniques fondées sur l'analyse partitionnelle, les tests aux limites et le graphe cause-effet.

2-Ces techniques de test fonctionnel ont un même point commun : elles s'appuient sur la spécification. En effet, un test fonctionnel, examine le comportement fonctionnel du logiciel, afin de détecter les erreurs d'omission et de conformité par rapport à la spécification. Pour ceci, l'étude de la spécification permet de générer des jeux de données de test (DT) représentatifs qui seront injectés en entrée du logiciel.

II-RÉALISATION :

A) Cas n°1 : test du logiciel « constructeur de triangles »

Pour ce premier exercice, nous allons étudier sur un constructeur d'objets de type triangle Q. À chaque appel de ce constructeur, il faut entrer trois paramètres correspondant à la longueur des 3 côtés du triangle à dessiner. En fonction de ces 3 paramètres, le résultat peut être le suivant :

- l’objet est un triangle scalène,
- l’objet est un triangle isocèle,
- l’objet est un triangle équilatéral,
- l’objet est un non-triangle,
- cas impossible.

Pour réaliser cet exercice, nous allons premièrement écrire un constructeur simple qui servira de logiciel sous-test, puis définir les classes d'équivalence (qui fait partie de l'analyse partitionnelle de l'exercice) qui nous servira pour élaborer une stratégie de test aux limites et enfin générer à partir des techniques de test qui précèdent un jeu de DT.

Étape 0 : Écrire un constructeur simple de logiciel sous-test :

Dans cette étape, nous avons réalisé un logiciel de sous-test afin de manipuler des triangles et tester facilement, notamment en écrivant un constructeur qui demande successivement la longueur des trois côtés d'un triangle (Exercice1-Christoph-Jankowiak.cpp).

```cpp
#include <iostream>

using namespace std;

class Triangle
{
  private:
    float a;
    float b;
    float c;

  public:
    Triangle();
    void displayTriangle();
    void typeTriangle();
};

Triangle::Triangle()
{
  cout<<"Donnez une longueur numéro 1 : ";
  cin>>(*this).a;
  cout<<"Donnez une longueur numéro 2 : ";
  cin>>(*this).b;
  cout<<"Donnez une longueur numéro 3 : ";
  cin>>(*this).c;
}
```

```cpp
void Triangle::typeTriangle()
{
  if(((a==0)&&(b==0)&&(c==0))||((a<0)||(b<0)||(c<0)))
    cout<<"Cas impossible"<<endl;
  else
  {
    float max;
    char p = 'a';
    bool faux = false;
    max = (*this).a;

    if((*this).b>max)
    {
      max = (*this).b;
      p = 'b';
    }
    if((*this).c>max)
    {
      max = (*this).c;
      p = 'c';
    }
```

En effectuant une liste de comparaisons entre les valeurs entrées (a, b et c) le logiciel test est capable de déterminer la nature du triangle.

```cpp
    if(!(faux))
    {
      if((*this).a==(*this).b)
      {
        if((*this).b==(*this).c)
        {
          cout<<"Triangle équilatéral"<<endl;
        }
        else
        {
          cout<<"Triangle isocèle"<<endl;
        }
      }
      else if((*this).b==(*this).c)
        cout<<"Triangle isocèle"<<endl;
      else if((*this).a==(*this).c)
        cout<<"Triangle isocèle"<<endl;
      else
        cout<<"Triangle scalène"<<endl;
    }
  }
}
```

Dans le constructeur typeTriangle(), nous demandons trois longueurs du triangle comme afficher ci-dessous sur la capture. L'utilisateur rentre alors les trois longueurs du triangle et le constructeur en calcule alors si c'est un :

- un triangle isocèle,
- triangle scalène,
- un triangle équilatéral,
- un non-triangle,
- cas impossible.

```
schristoph@scinfe171 ~/Bureau/stockage/TD1/EXERCICE1
./Exercice1-Christoph-Jankowiak.out
Donnez une longueur numéro 1 : 5
Donnez une longueur numéro 2 : 4
Donnez une longueur numéro 3 : 6
```

De plus, afin de faciliter la vérification du bon fonctionnement de l'entrée des données, nous avons implémenté une fonction displayTriangle() qui affiche les trois longueurs du triangle demandé.

Étape 1 : Définir les classes d'équivalences :

On peut commencer par définir les classes d'équivalences selon les différents cas des triangles suivant :

Classes des triangles :	Conditions des triangles :
Triangle isocèle	isocèle si a≥b et a≥c et b+c≥a et (b=c) ≠ a isocèle si b≥a et b≥c et a+c≥b et (a=c) ≠ b isocèle si c≥a et b≥c et a+b≥c et (a=b) ≠ c
Triangle scalène	scalène si a>b et a>c et b+c≥a et b ≠ c ≠ a scalène si b>a et b>c et a+c≥b et a ≠ c ≠ b scalène si c>b et c>a et b+a≥c et b ≠ a ≠ c
Triangle équilatéral	équilatéral si a=b=c
Non-triangle	Non-triangle si a≥b et a≥c et b+c < a Non-triangle si b≥a et b≥c et a+c < b Non-triangle si c≥b et c≥a et b+a < c
Cas impossible	cas impossible si a, b ou c <0 cas impossible si a, b ou c est un caractère ≠ à un nombre cas impossible si a, b ou c est vide

Pour résumer, en définissant quatre classes d'équivalences nous pouvons obtenir, (avec A, B et C les trois longueurs d'un triangle) un tableau démontrant tous les cas possibles pour le *test du logiciel « constructeur de triangles »* tel que

Classe	Entrées	Validité
C1 : Entrées négatives	A v B v C < 0	Invalide
C2 : Entrées nulles	A v B v C = 0	Invalide
C3 : Entrées correctes	0 < A v B v C < MAX Entier + 1	Valide
C4 : Entrées trop grandes	MAX Entier < A v B v C	Invalide

Étape 2 : Élaborer une stratégie de test aux limites :

Le principe de la technique consiste à s'intéresser aux bornes des classes lorsque ces classes partitionnent le domaine D des données d'entrées. Le test aux limites est considéré comme l'une des techniques fonctionnelles les plus efficaces. On constate que souvent les défauts sont dus au fait qu'on n'a pas prévu le comportement d'un logiciel, pour les valeurs limites.

Les DT limites résultent de l'étude des limites des champs de définition des données d'entrées. Le fait d'envisager des valeurs qui dépassent les limites pourrait paraître injustifié. En effet, le logiciel n'a, en principe, pas été conçu pour traiter des données hors spécifications.

De manière générale, les tests aux limites consistent à identifier les domaines de définition de chaque variable d'entrée. Ces valeurs sont combinées pour produire des jeux de DT. Nous allons donc prendre des valeurs qui respectent les conditions des classes d'équivalence et générer un jeu de DT pour démontrer cette stratégie de test aux limites.

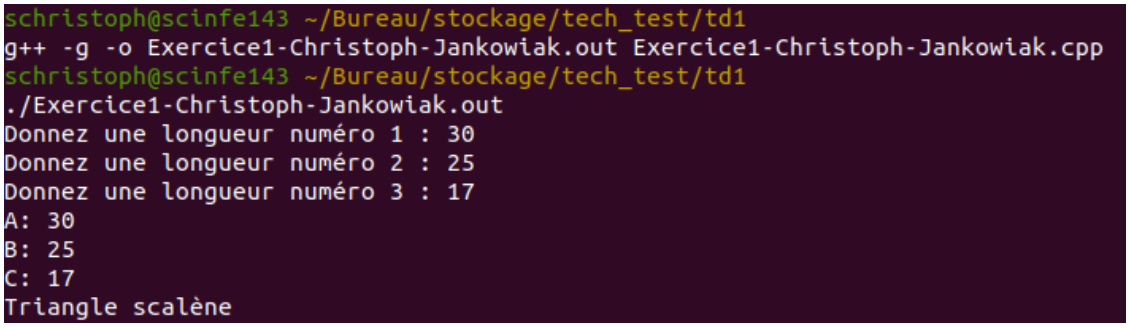
Classe	Limites	Validité
Bornes entre les classes C1,C2	-4 -3 0	Invalide Invalide Invalide
Bornes entre les classes C2, C3	0 1 4 MAX Entier -1	Invalide Valide Valide Valide
Bornes entre les classes C3, C4	MAX Entier -1 MAX Entier MAX Entier+1	Valide Valide Invalide

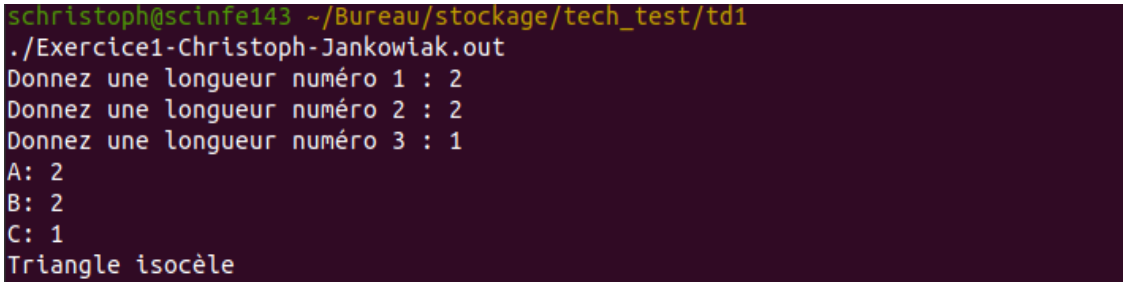
Étape 3 : Générer à partir des techniques de test qui précède un jeu de DT :

Grâce aux techniques de test que nous avons préalablement décrite sur les trois premières étapes, nous pouvons générer un jeu de DT sous forme de tableau ci-dessous :

N° DT	DT de tests limites:	Remarques:
1	30,25,17	Triangle scalène
2	15,16,20	Triangle scalène
3	12,30,24	Triangle scalène
4	2,2,1	Triangle isocèle
5	3,2,2	Triangle isocèle
6	4,4,1	Triangle isocèle
7	2,2,2	Triangle équilatéral
8	3,3,3	Triangle équilatéral
9	20,20,20	Triangle équilatéral
10	1,1,5	Non-triangle
11	4,1000,1	Non-triangle
12	0,0,0	Non-triangle
13	-1,2,2	Cas impossible
14	A,2,2	Cas impossible
15	« »,1,1	Cas impossible

Pour démontrer nos techniques de tests et prouver que le logiciel de sous test est juste, nous avons inclus ci-dessus un cas pour chaque cas de triangles (scalène, isocèle, équilatéral, non-triangle et cas impossible) dans un terminal.





```
schristoph@scinfe143 ~/Bureau/stockage/tech_test/td1
./Exercice1-Christoph-Jankowiak.out
Donnez une longueur numéro 1 : 2
Donnez une longueur numéro 2 : 2
Donnez une longueur numéro 3 : 2
A: 2
B: 2
C: 2
Triangle équilatéral
```

```
schristoph@scinfe143 ~/Bureau/stockage/tech_test/td1
./Exercice1-Christoph-Jankowiak.out
Donnez une longueur numéro 1 : -1
Donnez une longueur numéro 2 : 2
Donnez une longueur numéro 3 : 2
A: -1
B: 2
C: 2
Cas impossible
```

```
schristoph@scinfe143 ~/Bureau/stockage/tech_test/td1
./Exercice1-Christoph-Jankowiak.out
Donnez une longueur numéro 1 : 1
Donnez une longueur numéro 2 : 1
Donnez une longueur numéro 3 : 5
A: 1
B: 1
C: 5
Non triangle
```

B) Cas n°2 : test du logiciel « placement des nouveaux diplômés »

Une association d'anciens étudiants en Informatique développe une application dans le but d'automatiser un processus de placement (offre d'emploi) en faveur des étudiants nouvellement diplômés d'un master.

Un emploi est caractérisé par :

- une classe : niveau des salaires pratiqués
- un type : pour indiquer la corrélation entre la spécialisation de l'étudiant et la nature de l'activité de l'entreprise qui recrute.

Pour réaliser cet exercice, nous allons premièrement développer une application très simple qui servira de logiciel sous test, puis utiliser la technique de test basée sur les graphes cause-effet, pour cela nous allons dans un premier temps identifier les causes : conditions d'entrée, identifier les effets : conditions de sortie ou transformations du système, établir le graphe cause-effet pour ensuite compléter le graphe par des contraintes entre causes et/ou entre effets, convertir le graphe en table de décision et pour finir nous générer une DT par ligne pour tester le logiciel de placement des diplômés.

Étape 0 : Développer une application de sous test très simple qui servira de logiciel sous test :

Dans cette étape, on va créer un constructeur simple de logiciel sous test (Exercice2-Christoph-Jankowiak.cpp). Dans ce constructeur, l'utilisateur se verra demander une liste de questions par lesquelles il faudra répondre par oui (o) ou par non (n) pour pouvoir être placé en faveur des étudiants nouvellement diplômés d'un master pour obtenir une offre d'emploi et qui traduit le graphe cause-effet qui se trouve dans l'étape 2. En fonction de ses réponses, le logiciel affichera de nouvelles questions, et ce, jusqu'à ce que le processus soit terminé.

```cpp
#include <iostream>

using namespace std ;

int main()
{
  char rep;
  cout<<"\nRépondre à toutes les questions par oui (o) ou par non (n)"<<endl;
  cout<<"\nAvez-vous obtenu une offre d'emploi de classe A1 ?"<<endl;
  cout<<"Oui (o) ou non (n) : ";
  cin>>rep;
  if(rep == 'o')
  {
    cout<<"Vous êtes exclus du processus de classement"<<endl;
  }
  else
  {
    cout<<"Avez-vous accepté une 2eme offre d'emploi ?"<<endl;
    cout<<"Oui (o) ou non (n) : ";
    cin>>rep;
    if(rep == 'o')
    {
      cout<<"1ère offre annulée"<<endl;
      cout<<"Vous êtes exclus du processus de classement"<<endl;
    }
```

```cpp
    else
    {
      cout<<"Avez-vous refusé 3 offres différentes pour un 2eme emploi ?"<<endl;
      cout<<"Oui (o) ou non (n) : ";
      cin>>rep;
      if(rep == 'o')
      {
        cout<<"Vous êtes exclus du processus de classement"<<endl;
      }
      else
      {
        cout<<"Avez-vous obtenu un emploi de type B1 ?"<<endl;
        cout<<"Oui (o) ou non (n) : ";
        cin>>rep;
        if(rep == 'o')
        {
          cout<<"Vous ne pouvez viser qu'un emploi B1 pour une deuxième offre"<<endl;
        }
        else
        {
          cout<<"Avez-vous obtenu un stage S1 ?"<<endl;
          cout<<"Oui (o) ou non (n) : ";
          cin>>rep;
          if(rep == 'o')
          {
            cout<<"Vous êtes éligible aux offres de classe A1"<<endl;
          }
```

```cpp
          else
          {
            cout<<"Avez-vous obtenu un stage S2 ?"<<endl;
            cout<<"Oui (o) ou non (n) : ";
            cin>>rep;
            if(rep == 'o')
            {
              cout<<"Vous êtes éligible qu'aux offres de classe A2"<<endl;
            }
          }
        }
      }
    }
  }
  cout<<"Processus de placement terminé"<<endl;
  return 0;
}
```

Le logiciel de sous-test fonctionne selon les réponses de l'utilisateur. Si la réponse et négative (n) l'utilisateur sera confronté à une autre question, a contrario si la réponse est positive il se verra attribuer une offre (de classe A1 ou A2) ou une exclusion du processus de placement.

Étape 1 : Déterminer les causes et les effets :

Nous allons dans cette partie déterminer les causes et les effets. En analysant ces spécifications, nous relevons six causes c1, c2, c3, 4, c5, c6 :

-C1 : Première offre dans un emploi de classe A1 (de type B1 ou B2).
-C2 : Première offre dans un emploi A2B1.
-C3 : Première offre dans un emploi A2B2.
-C4 : Première offre dans un emploi S1.
-C5 : Première offre dans un emploi S2.
-C6 : Deuxième offre d'emploi accepté.

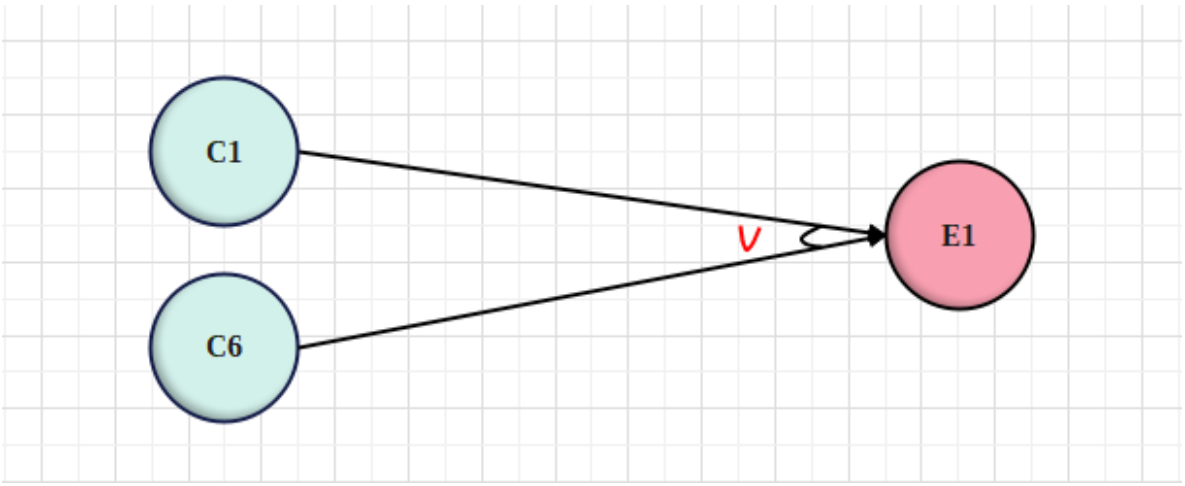
Ensuite, les effets de ces causes sont :

-E1 : Exclu du processus de placement.
-E2 : Première offre d'emploi annulée.
-E3 : Maximum 3 tentatives, seulement pour un emploi de type B1 (A1 ou A2).
-E4 : Maximum 3 tentatives pour un emploi de type B1 ou B2 (A1 ou A2).
-E5 : Maximum 3 tentatives pour un emploi de classe A2 (B1 ou B2).

Étape 2 : Dessiner le graphe cause-effet :

Cette méthode consiste à élaborer, un graphe non orienté, un réseau qui relie les effets d'exécution d'un logiciel (sorties/résultats) aux causes les (entrées) qui sont à leur origine. Un tel graphe est constitué par des nœuds représentant des variables d'états et des arêtes qui symbolisent les liaisons logiques entre ces variables.

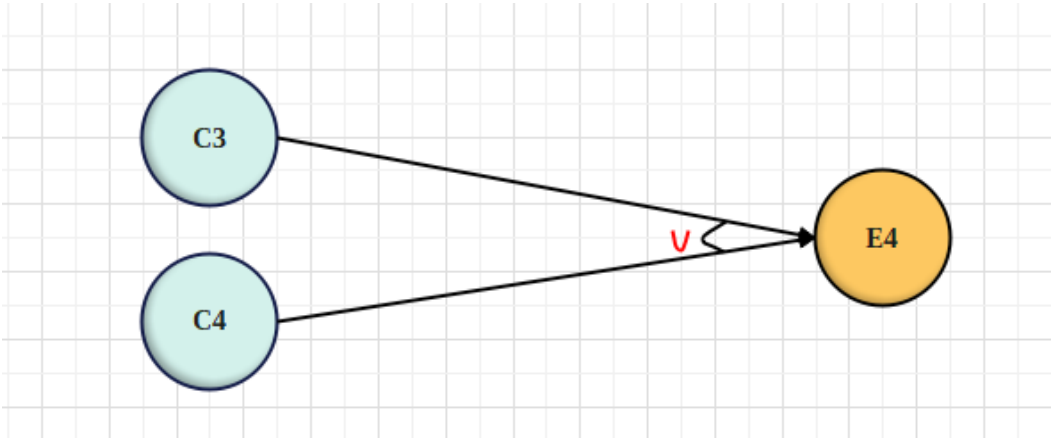
L'élaboration du graphe s'appuie sur une analyse minutieuse de la spécification du programme. Dans notre cas, voici à quoi ressemblerait les graphiques cause-effet :



Première étape : Exclu du processus de placement.

Deuxième étape : Première offre d'emploi annulée.

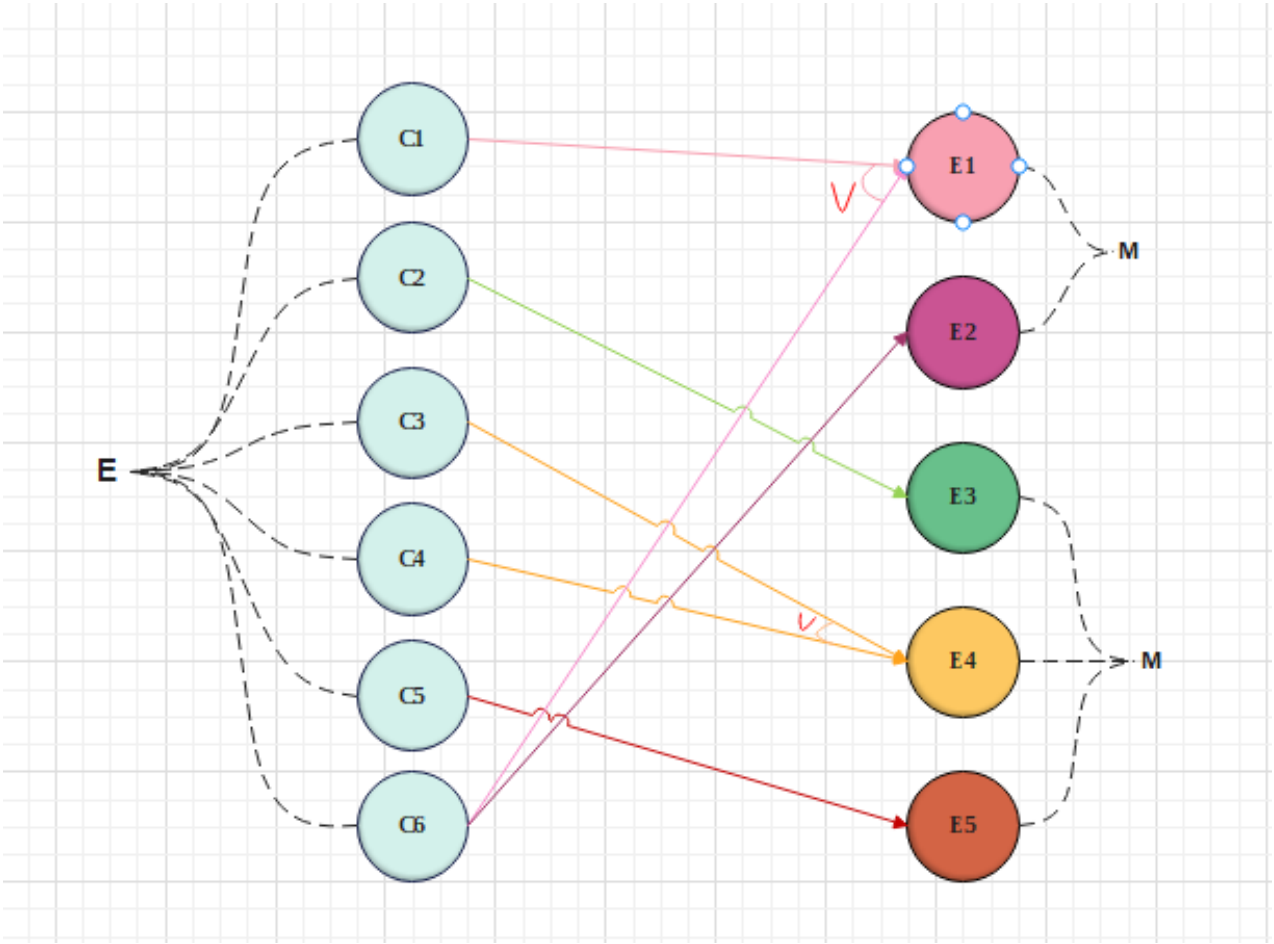
Troisième étape : Maximum trois tentatives, seulement 1 emploi de type B1 ou B2 (A1 ou A2).



Quatrième étape : Maximum 3 tentatives pour un emploi de type B1 ou B2 (A1 ou A2).

Enfin, la cinquième étape : Maximum 3 tentatives pour un emploi de classe A2 (B1 ou B2).

Finalement, nous pouvons donc traduire de toutes les étapes et graphes cause-effet précédents par un seul graphe cause-effet que voici :



Étape 3 : Dresser la table de décision réduite :

Le réseau précédent se traduit par une table de décision. La table de décision met en évidence toutes les combinaisons des valeurs de vérité de l'ensemble des causes pour mettre en évidence tous les effets produits par ces causes.

D'où la table de décision :

cause / effet	E1	E2	E3	E4	E5
C1	1	0	0	0	0
C2	0	0	1	0	0
C3	0	0	0	1	0
C4	0	0	0	1	0
C5	0	0	0	0	1
C6	1	1	0	0	0

Étape 4 : Générer les jeux de DT pour tester le logiciel de placement des diplômes :

Les tests aux limites consistent à identifier les domaines de définition de chaque variable d'entrée. Ces valeurs sont combinées pour produire des jeux de DT. Voici notre tableau qui génère les jeux de TD pour le cas du placement des nouveaux diplômés.

N°DT	Cause(s)	Entrée (cause)	Résultat attendu (effet)
1	C1=1; C6=1;	1re offre : emploi rémunéré jusqu'à 69 k€/an	Exclu du processus de placement
2	C6=1;	2ième offre acceptée	1ière offre annulée
3	C2=1;	1ère offre : emploi rémunéré jusqu'à 42 k€/an chez « Ingénierie Technologies de l'internet »	Maximum 3 tentatives pour emploi B1 : A1B1 ou A2B1
4	C3=1; C4=1;	Stage S1	Maximum 3 tentatives pour emploi B1 ou B2 A1B1 ou A2B1 ou A1B2 ou A2B2
5	C5=1;	1re offre S2	Maximum 3 tentatives pour emploi A2 : A2B1 ou A2B2

III-BILAN/CONCLUSION :

Durant cette série de TD nous avons pu étudier différentes approches pour réaliser un test fonctionnel dont la réalisation concrète d'un constructeur simple qui nous a servi de logiciel sous-test. Ainsi, la réalisation de classes d'équivalences des DT l'élaboration d'une stratégie de test aux limites pour finir par générer à partir des techniques de test un jeu de DT mais également de déterminer des causes et des effets qui nous ont servi pour élaborer et dessiner un graphe cause-effet. Nous avons alors, mieux compris comment et pourquoi adapter les techniques de test à appliquer suivant le domaine de définition. Et cela nous a permis d'appliquer les techniques de tests suivantes à des cas concrets de tests aux limites, d'analyse partitionnelle et de technique de graphes.

- Pour la génération des jeux de DT : Procéder par étapes tout en ayant réfléchi à celles-ci paraît être la bonne solution pour travailler efficacement. Il faut trouver le bon domaine pour générer le jeu de DT, et enfin pour pratiquer un test efficace, il faut réfléchir à la manière d'aborder celui-ci et avoir une bonne approche du problème.

- Pour la stratégie de test au limites : Il faut avancer méthodiquement et minutieusement afin d'établir une bonne stratégie, bien cerner le problème dans le but d'envisager tous les cas. C'est pourquoi cette technique révèle des failles dans les logiciels, mais aussi des erreurs de logique. En effet, le comportement du logiciel étant poussé jusqu'aux limites de celui-ci, certains défauts sont révélés par le test. Les erreurs de logique sont présentes, car lors de la conception du logiciel, le développeur n'envisage pas tous les cas possibles, ce qui entraînerait des résultats inattendus, sans pour autant avoir un dysfonctionnement du logiciel.

<u>- Pour finir, la stratégie de graphe cause-effet :</u> La stratégie de test du graphe cause-effet produit des résultats très clairs et sont lisibles et facilement exploitables, notamment la table de décision réduite. C'est la technique la plus basée sur la théorie, hors de toute implémentation. Cette stratégie permet de se rendre compte de toutes les spécificités d'un problème, ce qui la rend extrêmement pratique, et très utile en complément d'autres techniques.

La méthode fonctionnelle est basée sur la détection d’erreurs d’émission et de conformité par rapport à la spécification, elle n’examine pas le contenu logiciel, mais son comportement fonctionnel, elle utilise donc en perspective de moins de combinaisons en comparaison avec d’autres techniques. Nous avons ainsi pu voir que la technique de tests fonctionnelle était fiable et efficace en combinant plusieurs techniques de tests. Cette série nous a permis d'approfondir nos connaissances théoriques et de mettre en œuvre plusieurs techniques de test fonctionnel vues en cours.