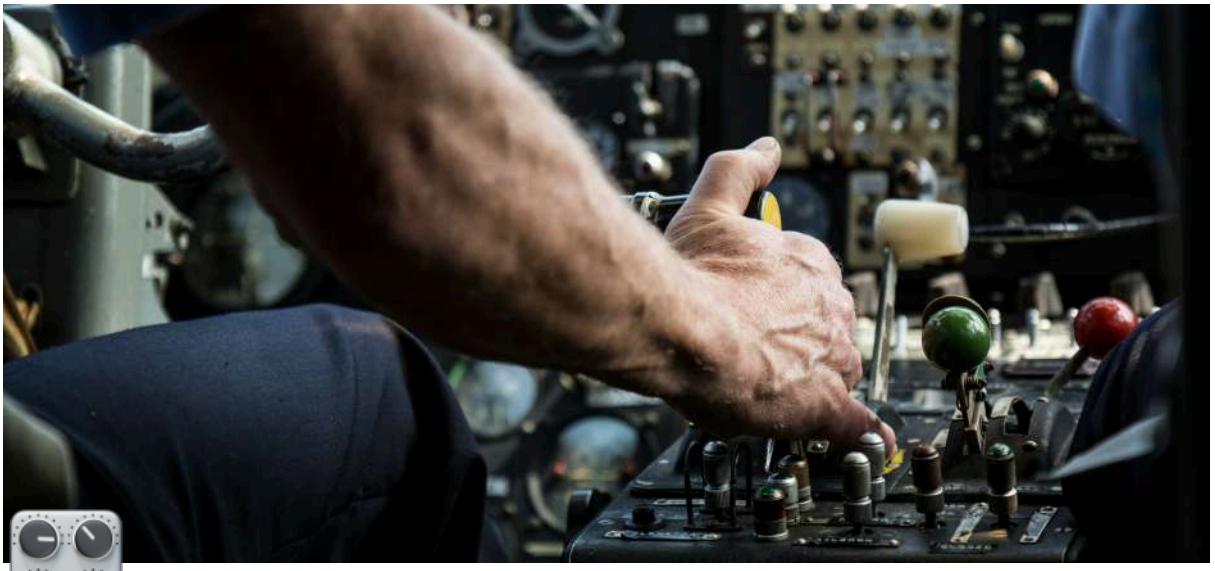


Devoir Maison 3

1. Quoi ?

Tout d'abord, la **complexité cyclomatique** est une métrique permettant de mesurer la complexité d'un programme informatique en comptant les chemins potentiellement empruntés dans ses structures de contrôles. Cette donnée quantitative permet, théoriquement, de déterminer si un programme est simple ou non à lire, tester et re-factoriser[1].

Seulement, d'après une étude de M. Shepperd, la base théorique de cette donnée n'est pas suffisamment solide, ce qui la rend peu adapté au développement logiciel. Il est tout particulièrement souligné qu'aucune observation ne peut justifier son utilité[2].

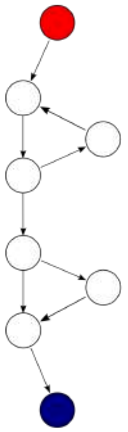
Ensuite, la **couverture de code** est une métrique permettant de déterminer la proportion de code source convert par une suite de tests[3]. Un programme avec une couverture de code élevée - i.e. le plus proche possible de 100% - suggère qu'il a moins/peu de chances qu'il contienne des bogues non détectés.

2. Comment ?

Pour commencer, la **complexité cyclomatique** peut se calculer en étudiant le graphe de flot de contrôle d'un programme[1] (Voir ci-contre).

Un nœud du graphe correspond à un groupe d'instructions indivisibles et les flèches connectent deux nœuds si il peuvent s'exécuter successivement (Voir ci-contre).

Donc, on peut calculer la complexité cyclomatique M avec E le nombre d'arêtes, N le nombre de nœuds et P le nombre de sous-graphes[1]:



Auteur: JulesH.

$$M = E - N + 2P$$

Puis, la **couverture de code** se détermine via plusieurs critères. Les principaux sont[3]:

- **Couverture de fonction:** Est-ce que toutes les fonctions ont-elles été appelées ?
- **Couverture d'instruction:** Est-ce que toutes les instructions ont-elles été exécutées ?
- **Couverture de condition/de prédicat:** Est-ce que toutes les expressions booléennes ont-elles été évaluées à leurs deux états possibles (`true` et `false`) ?
- **Couverture d'arête:** Est-ce que toutes les arrêtes du graphe de flot de contrôle (vu précédemment) ont-elles été exécutées ? Ou plus précisément, avec la **couverture de branche**, est-ce que toutes les branches de chaque structure de contrôles ont-elles été exécutées ?

En établissant toutes les parties du code source à vérifier et à analyser, il est possible de déterminer un pourcentage de couverture pour une suite de tests donnée.

3. C'est à dire ?

Soit ce petit programme `gmn` permettant de jouer au jeu du "Guess My Number":

```c
/* @author Xibitol */

#include <stdlib.h>
#include <time.h>
#include <stdio.h>

#define EXEC_NAME "gmn"

#define MIN 1
#define MAX 100

int main(void){
    printf(
        "Guess which the number I'm thinking of, between %d and %d, \
          and win nothing...\n",
        MIN, MAX
    );

    srand(time(NULL));
    unsigned int num = rand()%(MAX - MIN) + MIN;
    unsigned int guess;

    do{
        printf("What's your guess ? ");

        if(scanf("%u", &guess) == EOF)
            perror(EXEC_NAME);
        else if(guess < num)
            printf("Greater...\n");
        else if(guess > num)
            printf("Lesser...\n");
        else
            printf("Got it! This was fun, right?\n");
    }while(guess != num);

    printf("Bye.\n");
```

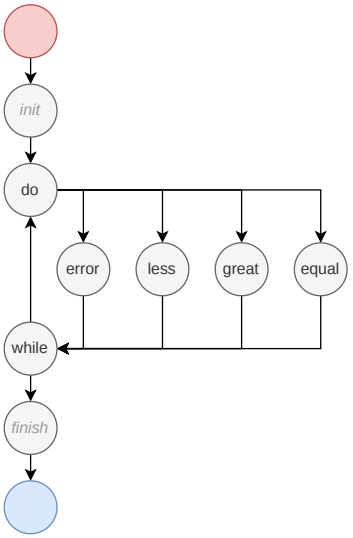
```
    return EXIT_SUCCESS;
}
```

Nous n'allons évidemment pas rentrer dans le détail des librairies standards.

Depuis le graphe de flot de contrôle ci-contre du programme ci-dessus, il est facile de calculer sa complexité cyclomatique M_0:

$$M_0 = 12 - 10 + 2 \times 1 = 4$$

D'après la catégorisation de McCabe, notre programme est simple avec un léger risque ($1 \leq M_0 \leq 10$)[1].



Graphe de flot de contrôle du programme gmn d'exemple.

Références

1. McCabe (December 1976). "A Complexity Measure". *IEEE Transactions on Software Engineering*. **SE-2** (4): 308–320. doi:10.1109/tse.1976.233837. S2CID 9116234.
2. M. Shepperd, « A critique of cyclomatic complexity as a software metric », *Software Engineering Journal*, IET, vol. 3, no 2, @December 30, 87@December 30, 87@December 30, 87, p. 30-36 (ISSN 0268-6961, résumé [archive], lire en ligne [archive]).
3. Brader, Larry; Hilliker, Howie; Wills, Alan (March 2, 2013). "Chapter 2 Unit Testing: Testing the Inside". *Testing for Continuous Delivery with Visual Studio 2012*. Microsoft. p. 30. ISBN 978-1621140184. Retrieved 16 June 2016.