# Algorithmique & Programmation

# **Comparaison de codes**

## **Peu importe le code pourvu qu'on ait l'exécution ?**

Fabien.Delecroix@univ-lille.fr

ABDELKADER Omar, BIRLOUEZ Martin, BONEVA Iovka, DELECROIX Fabien,
LE QUINIOU Erwann, MARSHALL-BRETON Christopher, REKIK Yosra, SECQ Yann,
SOW Younoussa, SUDHEENDRAN Megha, SUE Yue

*Saison 2025-2026 / DÉV*

# À chaque problème, **ses** solutions

- 2 + 2 = ?  4 !

→ arithmétique, solution unique, une seule bonne réponse

- Coder la Bataille Navale
- Coder la fonction factorielle

→ solutions multiples
Il n’y a pas qu’une manière de coder…
…mais toutes se valent-elles ?

# Un code peut-être plus ou moins…

- Concis
- Clair / Compréhensible
- Efficace
- Réutilisable
- Testé / Validé
- Décomposé / Modulaire
- …

# Concision

```
void a(int[][]g){String r="  012345678";int
i=0;while(i<81){if(i%9==0)r+="\n"+i/9+" ";
r+=g[i/9][i++%9];}println(r);}
```

**117 caractères, chapeau !**

**Mais au fait, que fait ce code ?**

# Clarté

**Ajoutons documentation, langue, variables, nommage, indentation, aération, commentaires, … et optons pour des boucles et conditions intuitives !**

```
/**
 *Procédure qui réalise l'affichage des entiers contenus dans une grille de 9x9
 * @grille un tableau à deux dimensions de taille 9x9 contenant des entiers
 */
void afficherGrilleEtCoordonnees (int[][] grille){
    int nombreLignes = 9;
    int nombreColonnes = 9;

    print("  ");//décalage correspondant à l'affichage du n° des lignes dans la suite
    println("012345678");//affichage des coordonnées des 9 colonnes

    for(int numeroLigne = 0; numeroLigne < nombreLignes; numeroLigne = numeroLigne + 1){
        for(int numeroColonne = 0; numeroColonne < nombreColonnes; numeroColonne = numeroColonne + 1){
            if (numeroColonne==0){
                print(numeroLigne+" ");
            }
            print(grille[numeroLigne][numeroColonne]);
        }
        println();
    }
}
```

**Voilà qui est plus clair !**
**… mais ne peut-on pas faire un algorithme plus efficace ?**

# Efficacité

**Optimisons ! Supprimons les alternatives et calculs évitables, les variables excédentaires, …**

```
//EFFICACITÉ
void afficher(int[][] grille){
    print("  012345678");

    println();
    for(int lig = 0; lig < 9; lig = lig + 1){
        print(lig);
        print(" ");
        for(int col = 0; col < 9; col = col + 1){
            print(grille[l][c]);
        }
        println();
    }
}
```

**Voilà qui est plus efficace !**
**… mais est-ce encore aussi lisible ? Est-ce qu'on y gagne vraiment beaucoup ?**
**→ les variables dispensables ne sont pas forcément à supprimer !**

**Autre chose : ce code est-t-il encore utilisable si on change un peu la situation ?**

# Réutilisabilité

**Élargissons les cas traités, généralisons les noms choisis.**

```
//RÉUTILISABILITÉ (généricité)
String toStringIdx(int[][] tab){
    String res ="  ";
    for(int c = 0; c < length(tab,1); c = c + 1){
        res+= c;
    }
    res+='\n';
    for(int l = 0; l < length(tab,1); l = l + 1){
        res+=l+" ";
        for(int c = 0; c < length(tab,2); c = c + 1){
            res+=tab[l][c];
        }
        res+='\n';
    }
    return res;
}
```

```
void afficherSudoku(int[][] grille){
    println(toStringIdx(grille));
}
```

```
void afficherVoisinage(int[][] voisinage){
    println("***Analyseur jeu de la vie***");
    println("Nombre de voisines pour chaque cellule : ");
    println(toStringIdx(voisinage));
}
```

**Notre fonction va pouvoir être réutilisée dans d'autres contextes.**
**Mais au fait, est-on bien sûr qu'elle fonctionne avec des entrées autrement dimensionnées ?**

# Testabilité

**Avantage d'être passé d'une procédure d'affichage à une fonction : on peut tester !**
**Alors allons-y : écrivons des tests...**

```
//TESTÉ/VALIDÉ
void testToString(){
    int[][] grille = new int[][]{
        {1,2},
        {3,4}
    };
    assertEquals("  01\n0 12\n1 34\n",toStringIdx(grille));
    grille = new int[][]{{5}};
    assertEquals("  0\n0 5\n",toStringIdx(grille));
}
```

**… et exécutons-les !**

```
delecroi@labeo:~/S1/ap/2019/cours$ ijava StylesCode
No function 'void algorithm()' found, will try to run tests ...
1> testToString
1 test(s) verified on 1 tests (100% success).
```

**Attention à ne pas en conclure trop vite que ça marche avec tous les paramètres.**
**Simplement que pour ceux-ci, ça répond bien aux attentes.**
**Bien, mais si on veut se resservir d'une partie du code mais pas tout ?**

# Décomposition / Modularité

**Si on veut pouvoir se resservir d'une partie du code, dommage de la dupliquer. Pour éviter ça, décomposons ! En plus, ça rend le code plus facile à comprendre, maintenir et tester.**

```
//DÉCOMPOSITION / MODULARITÉ
String toStringNum(int[][] tab){
    String res ="  ";
    res+= numColonnes(length(tab,2));
    res+= '\n';
    for(int numL = 0; numL < length(tab,1); numL = numL + 1){
        res+= numL+" ";
        res+= toString(tab[numL]);
        res+='\n';
    }
    return res;
}
```

```
String numColonnes(int nb){
    String res = "";
    for(int idx = 0; idx < nb; idx++){
        res+= idx;
    }
    return res;
}
```

```
String toString(int[] tab){
    String res = "";
    for(int c = 0; c < length(tab); c = c + 1){
        res += tab[c];
    }
    return res;
}
```

**Autres avantages : code plus facile à comprendre, à maintenir et à tester**

# Généricité

**Si on veut des lettres pour les colonnes ou les lignes, on change tout ?**
**Pour éviter ça, ajoutons des paramètres !**

```
//GÉNÉRICITÉ
String toStringCoord(int[][] tab, char coordDebutCol, char coordDebutLig){
    String res ="  ";
    res+= suiteCaracteres(coordDebutCol,length(tab,2));
    res+= '\n';
    for(int numL = 0; numL < length(tab,1); numL = numL + 1){
        res+= (char) (coordDebutLig+numL)+" ";
        res+= toString(tab[numL]);
        res+='\n';
    }
    return res;
}
```

```
String suiteCaracteres(char debut, int nb){
    String res = "";
    for(int idx = 0; idx < nb; idx++){
        res+= (char) (debut+idx);
    }
    return res;
}
```

**Et on peut encore aller plus loin dans la généricité… hexasudoku ?**
**...mais gare à ne pas aller trop loin quand même ;-)**
**Alors finalement, quel est le meilleur code ?**

# Le code en questions

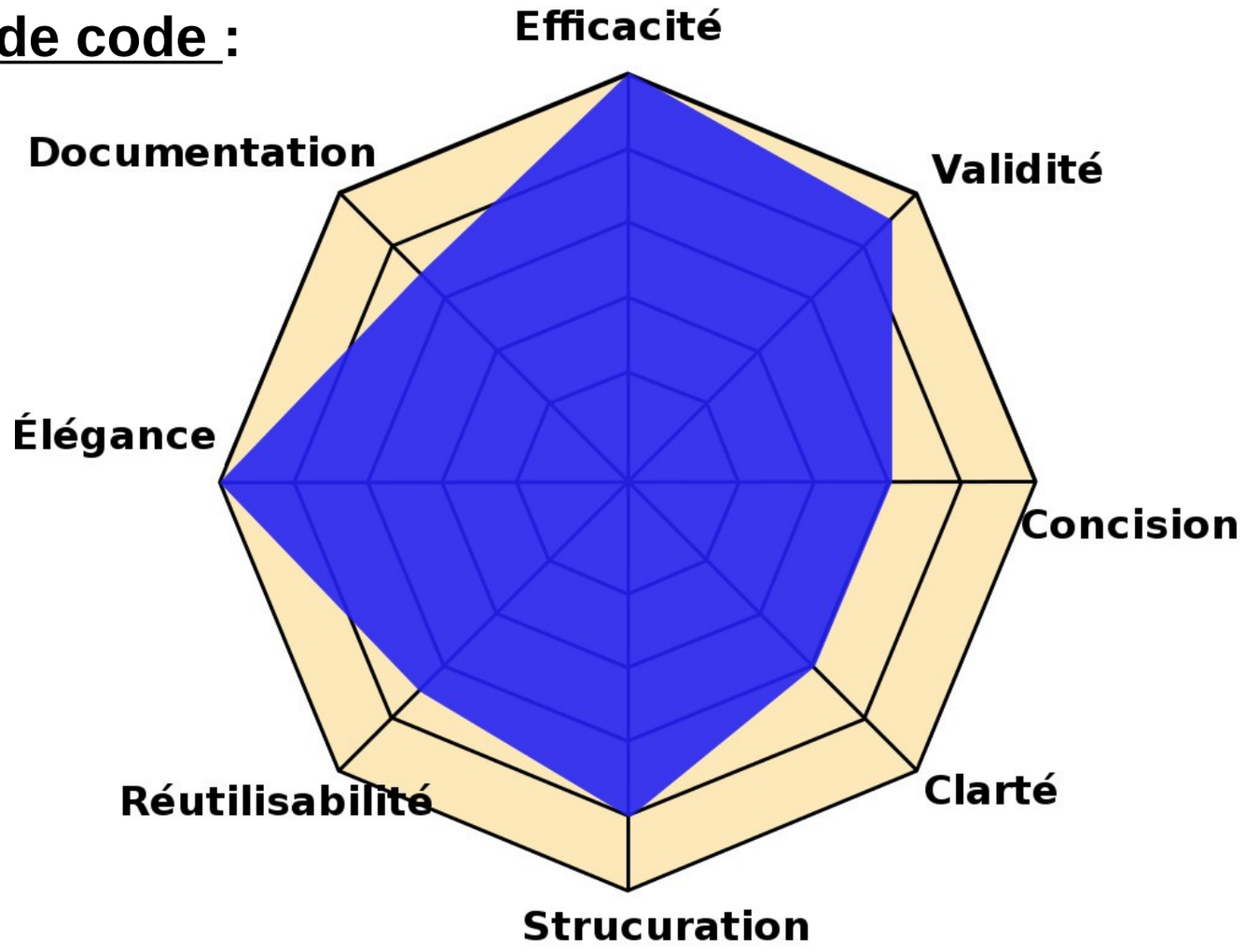
**Il n’y a pas de meilleur code en soi, ça dépend !**

- *Pourquoi je code ?*
  - pour essayer une idée ? pour un rendu (projet) ? pour un enjeu sensible ?
- *Pour qui je code ?*
  - moi ? mon groupe ? une communauté ?
- *Pour quand je code ?*
  - 5 minutes ? la semaine ? l’année ? toujours ?
- *Pour quel ordinateur je code ?*
  - Puissance ? Mémoire ? Architecture ?
- …

# Une question de compromis

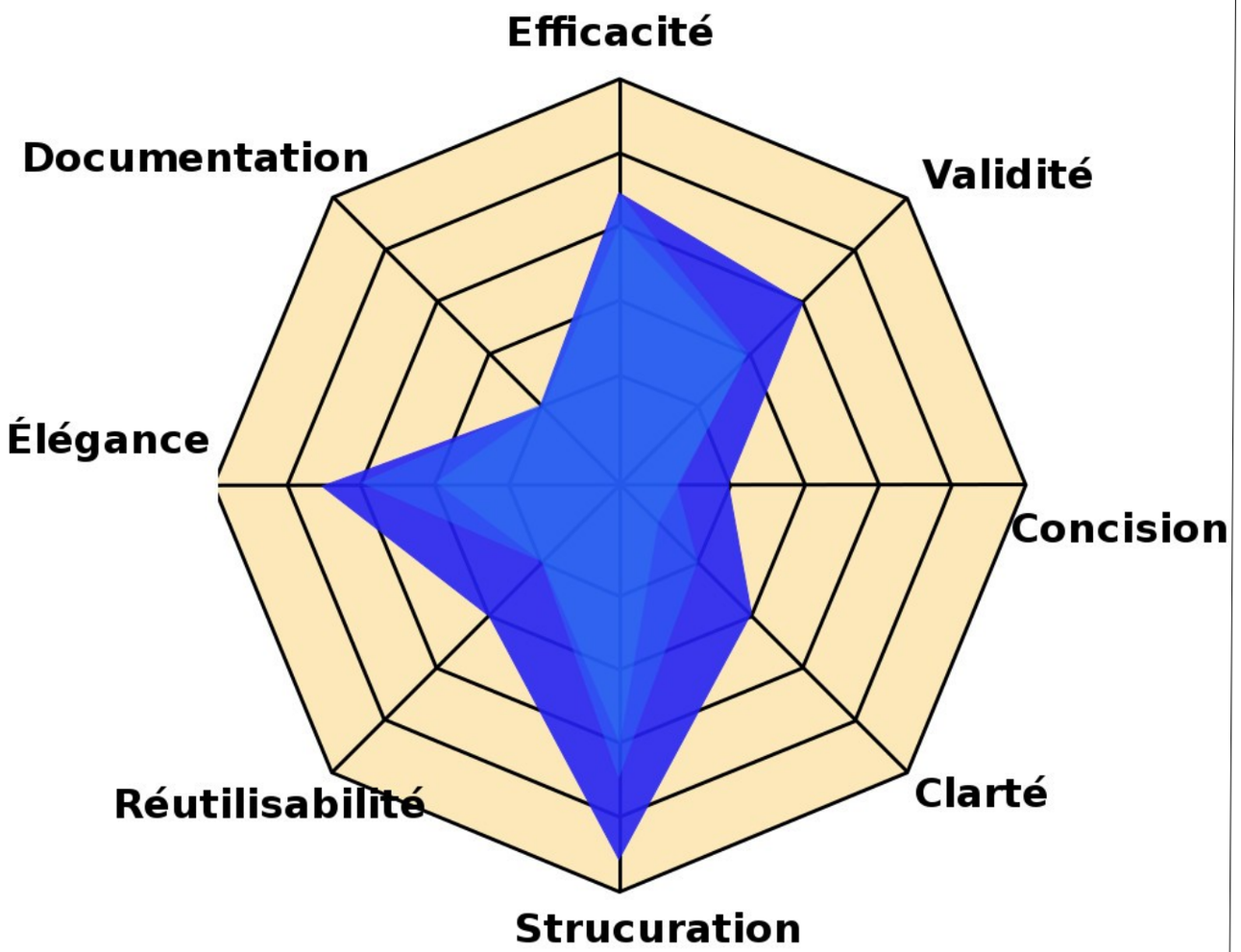
**À chaque situation, ses critères et un compromis à trouver.**

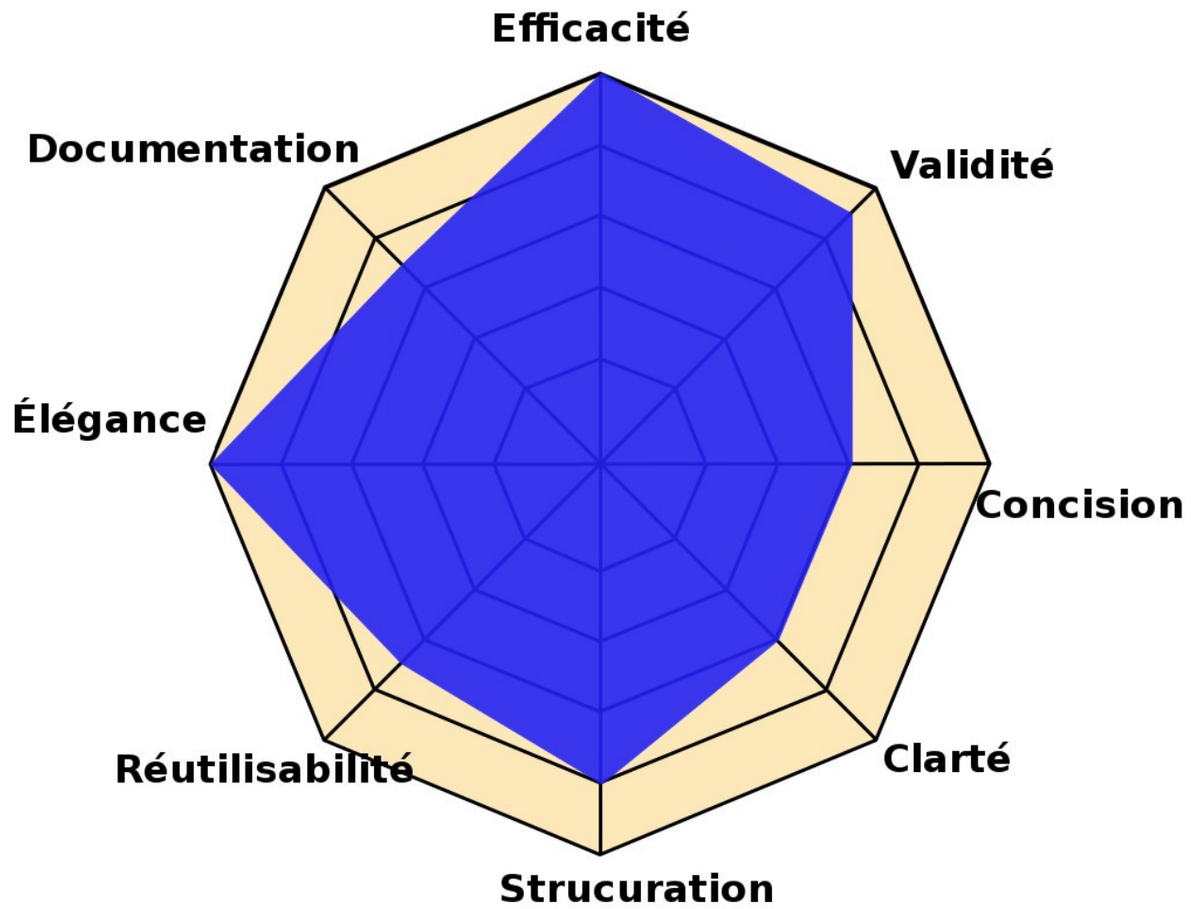
**<u>Exemple de profil de code</u> :**




**Code élégant et efficace, pas très clair ni concis → bien si c’est pour présenter un nouvel algorithme de tri, moins si c’est pour un tutoriel**

# Donc tous les codes se valent ?




- Meilleur en tout point (ou presque)

  … donc non !

# Conclusion

- Ce qu’est un bon code dépend notamment du contexte …
- … et, à la marge, aussi un peu de la personnalité de chacun

# Rendez-vous aux paradigmes

- ✔ Programmation impérative (Dév)
- ➔ Programmation orientée objet (Dév objets → S2)
- ◆ Programmation bas niveau (C → S2)
- ◆ Programmation événementielle (IHM → S2)
- ◆ Programmation déclarative (logique, fonctionnelle)
- ◆ Programmation concurrente
- ◆ …