Informatique fondamentale

Qualité logicielle

R.Gosswiller

1. Principes généraux
2. La documentation
3. Les tests

Principes généraux

Le code

"Quality, time, price : pick two".

Le code

Question

Qu'est-ce qu'un bon code ?

Réponse

Intégration de contraintes variées et parfois exclusives

Le code

- Vitesse de développement
- Lisibilité, maintenabilité
- Performance en temps
- Performance en ressources
- Complexité

Le code

Attention

Le nombre de lignes n'est en rien représentatif de la qualité d'un code !

Qualité

Documentation
Evaluation par des tests

La documentation

Documentation externe

Principe

La documentation externe regroupe tous les outils de description d'un programme regroupés dans des documents dédiés.

Exemples

Spécifications techniques, wiki en ligne, readme, changelogs, javadoc et documentations générées, tutoriaux, sites web, ...

Documentation interne

Principe

La documentation interne concerne tous les éléments de description et de détail internes au code

Exemples

Commentaires, commentaires interprêtés, nommage, normes, standards

Les commentaires

Principe

Les commentaires sont l'élément de documentation interne d'un code le plus important

Syntaxe

```
1       # Commentaire sur une ligne
2
3       """
4       Commentaire
5       multi-lignes
6       """
```

Les commentaires

Rôle

Les commentaires ont plusieurs rôles distincts
Documenter les fonctions et variables
Décrire un algorithme ou un point ponctuel
Rendre du code non-interprêtable

Attention

Attention : ne pas confondre code commenté et dead code !

Les commentaires

Un bon commentaire

- Explique pourquoi plutôt que comment
- Réponds à une question à laquelle le code ne peut pas répondre
- Explique un code condensé ou condense un bloc de code simple
- S'écarte de la machine pour se rapprocher de l'humain
- Brille d'autant mieux avec du code écrit vite que du code écrit bien

L'indentation

"It's easy to write code that a computer can understand. Good programmers write code that humans can understand."

Principe

L'indentation consiste à décaler des blocs de code les uns par rapport aux autres
Augmenter la lisibilité
Structurer la syntaxe

Syntaxe

```
1       indentation 0
2          indentation 1
3              indentation 2
```

L'indentation

Détails

Chaque bloc de code faisant partie d'un ensemble d'instruction doit être indenté

Fonctions, boucles, structures conditionnelles

Syntaxe

```
1       if(condition):
2         code
3
4       for i in range(1):
5         code
6
7       def func():
8         code
```

L'indentation

Détails

L'indentation n'est pas toujours prise en compte par le compilateur

En C

```
1   int main(){int num=1;while(num<=10){printf("%d\n",num);
2   num++;}return 0;}
3
4   int main()
5   {
6     int num=1;
7     while(num<=10)
8     {
9       printf("%d\n",num);
10      num++;
11    }
12    return 0;
13  }
```

Les tests

Types de tests

- Tests unitaires
- Tests fonctionnels
- Test d'intégration
- Tests de non-régression
- Test de performances

Autres

conformité, sécurité, ...

Tests unitaires

Principe

Vérifier les fonctionnalités de base de chaque fonction du code
Vérifier les cas classiques
Anticiper et tester les cas d'erreur

Détails

Valeurs types : cas nominaux d'utilisation
Valeurs limites : limites de l'intervalle de test, valeur extrêmes
Erreurs : typage, taille, hors intervalle

Tests fonctionnels

Principe

Souvent confondus avec les tests unitaires
Vérifier le comportement et la validité des fonctions du programme
Composition de tests unitaires

Détails

Valeurs types : cas nominaux d'utilisation
Valeurs limites : limites de l'intervalle de test, valeur extrêmes
Erreurs : typage, taille, hors intervalle

Tests de performances

Principe

Vérifier le temps de réponse/de calcul d'un programme

Détails

Délai des requêtes
Traitement d'une donnée, calcul d'un résultat
Normes imposées ou recherche de l'optimal
Etude de complexité

Tests d'intégration

Principe

Vérifier la conformité d'un module avec un projet dans son ensemble

Détails

Conformité à la logique de développement appliquée
Fonctionnalités attendues présentes

Tests de non-régression

Principe

Vérifier la conformité d'une version ou d'un module avec les versions précédentes

Détails

Pas de perte de fonctionnalités
Pas de perte d'un module en raison de l'intégration d'un autre
Rétro-compatibilité

Les bonnes habitudes à avoir

- Réfléchir avant de coder
- Définir clairement les conventions de nommage et de présentation du code
- Indenter rigoureusement
- Ne pas copier-coller des parties du code
- Commenter son code
- Rédiger une documentation externe
- Communiquer avec les autres membres d'équipe
- Maintenir son code et ses outils à jour
- Lire les logs et documentations d'équipe

Synthèse

- Qualité du code
- Documentation
- Tests