Mail : yassine.elkhalki@outlook.fr
Nom : ELKHALKI Yassine
Groupe : INF3-FA

UVSQ université PARIS-SACLAY | IUT de Vélizy-Rambouillet
CAMPUS DE VÉLIZY-VILLACOUBLAY
CAMPUS DE RAMBOUILLET

Dépôt Git : https://github.com/YassouSensai/yassine_prog_avancee.git

---

Ce rapport a été réalisé en se basant sur les différents CMs & TPs/TDs

Pour vérifier certaines informations et améliorer ce rapport, certaines requêtes ont été demandées à ChatGPT (surtout pour la partie Cours).

Sommaire

Programmation Avancée

Cours

CM3

Deux modèles de programmation parallèle :

- Parallélisme de tâche : une tâche principale est décomposée en plusieurs sous tâches indépendantes (communication par message)
- Parallélisme de données : les données sont divisées en blocs qui seront traité par une tâche distincte

Plusieurs paradigmes de programmation parallèle :

Paradigme Master Worker :

- Master (Maître) : Coordonne le travail et distribue les tâches aux esclaves
- Worker (Esclave) : Exécute les tâches et renvoie les résultats au maître

Autres : Clients Serveur, itérations parallèles (queues en Java, pools en Python), …

Architectures :

- SISD (Single Instruction Multiple Data) : sur processeur monocoeur
- SIMD (Single Instruction Multiple Data) : GPU
- MIMD (Multiple Instruction Multiple Data) : systèmes multi coeurs

- Mémoire partagée : Tous les processus accèdent à la même mémoire (multiprocesseur)
- Mémoire distribuée : Chaque processus a sa propre mémoire (cluster)
- Systèmes hétérogène : CPU +GPU

CM4

API concurrent :

- Executor : Mécanisme qui permet de gérer les threads
- Future : Représente le résultat d'un calcul asynchrone (vérifie si la tâche est terminée et le récupère)

Evaluation des performances :

- Speedup (Sp) (Accélération) = Gain de vitesse obtenu en utilisant plusieurs processus par rapport à un seul. ***Sp = T1/Tp*** ou T1 est le temps d'exécution pour un seul processus et Tp pour plusieurs
- Scalabilité forte : on fixe la taille du problème tout en augmentant le nombre de processus. "Grossièrement", plus on a de processus et plus ça doit aller vite.
- Scalabilité faible : On augmente proportionnellement la taille du problème par rapport au nombre de processus.

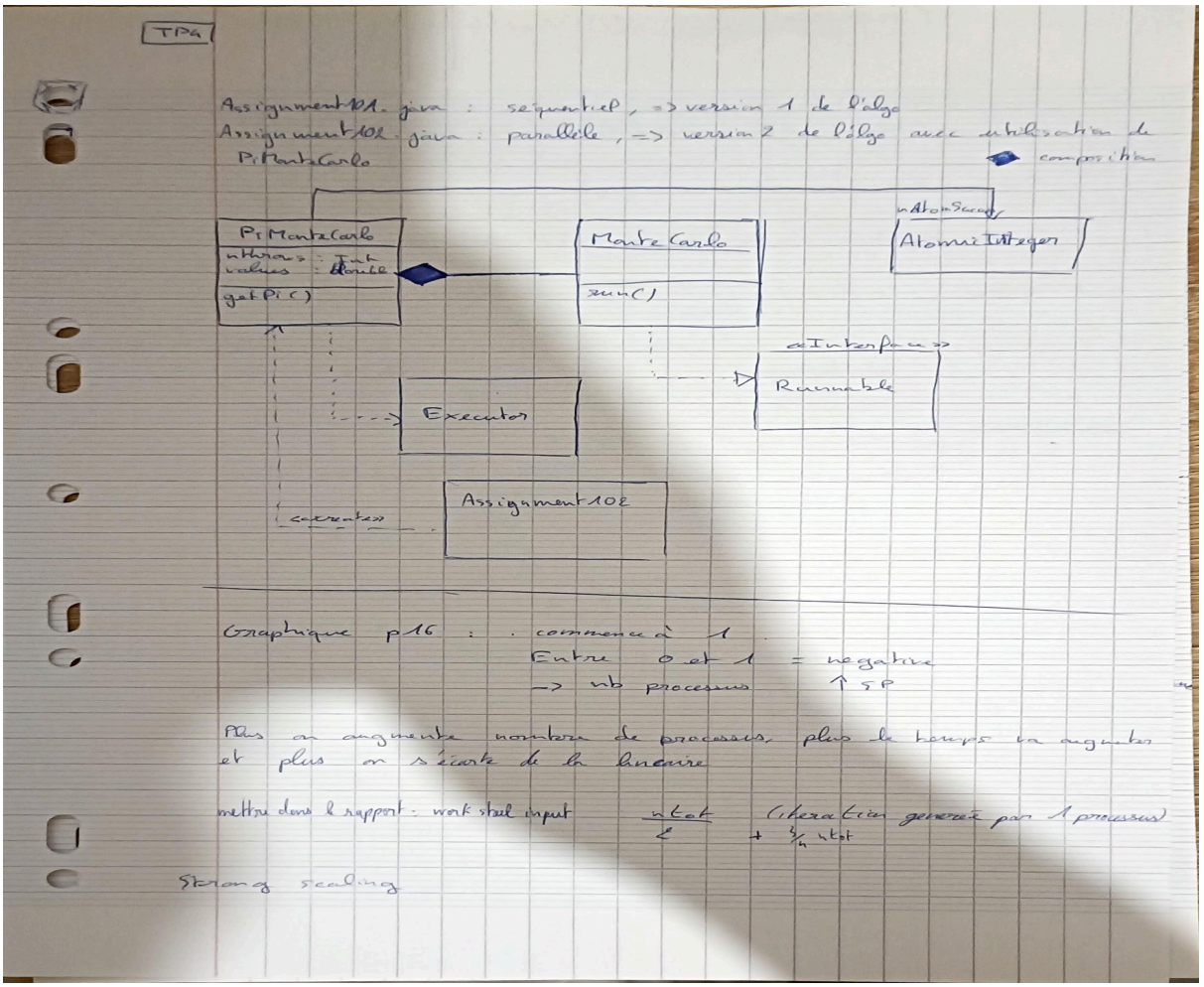
La scalabilité forte permet d'évaluer si l'on va plus vite avec plus de processus = Question de temps.

La scalabilité faible permet d'évaluer si l'on peut faire plus avec plus de processus = Question de "production" (Mon mot 😬)

Evaluation du code source

Pi et Assignement102

Assignment102 (Première version de PI avec MonteCarlo)



Calcul de π

Calcul de π par une méthode de Monte Carlo

Méthode de Monte Carlo

Les méthodes de Monte Carlo sont des techniques de simulation basées sur des tirages aléatoires pour résoudre des problèmes numériques, notamment en physique, finance et en optimisation.

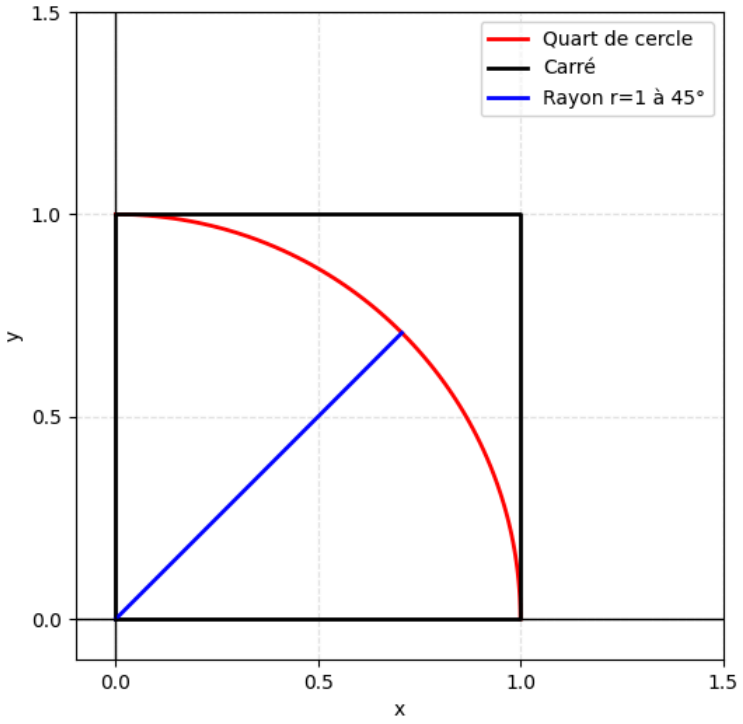
Exemples :

- Simulation de la propagation du feu (abordée en discussion en entreprise) (Physique)
- Estimation de la valeur future d'un actif (Finance)

- Utilisation d’algorithmes d’optimisation comme avec scipy en Python

Détermination de l’algorithme de Monte Carlo pour π

Partie réalisée en cours - Réécriture au propre - Voir Annexe



- Soit un carré de côté 1
- Soit un quart de disque de rayon r = 1
- L’aire du carré s’écrit Ac = r^2 = 1*1 = 1
- L’aire du quart de disque s’écrit Ad4 = π*r^2 / 4 = π/4
- On effectue n_total tirages

En suite, on sait que la probabilité qu’un point Xi soit dans le quart de disque si sa valeur di est comprise dans le quart de disque :

$$P(Xi \mid di < 1) = Ad4 / Ac = (\pi/4) / 1 = \pi/4$$

En sachant que chaque point Xi a deux coordonnées xi et yi et que sa valeur di est obtenue par di = √(xi^2 + yi^2)

On sait également que plus n_total est grand, plus la valeur estimée de π sera proche de la valeur réelle de π.

On peut donc estimer la probabilité que Xi soit dans le quart de disque est :

$$P(Xi \mid di < 1) = n\_cibles / n\_total$$

Avec n_cibles, le nombre de points qui se trouvent dans le quart de disque.

Finalement, on peut déduire que la valeur approximative de π est :

$$\pi/4 = n\_cibles / n\_total$$
$$\Leftrightarrow \quad \pi = 4 * n\_cibles / n\_total$$

L'algorithme en code

Voir version manuscrite (prise de note)

```
n_cible = 0;
for (p = 0; n_tot > 0; n_tot--) {
    x_p = rand();
    y_p = rand();
    if ((x_p * x_p + y_p * y_p) < 1) {
        n_cible++;
    }
}
pi = 4 * n_cible / n_tot;
```

Parallélisation - Paradigme Master Worker

Explication des Étapes :

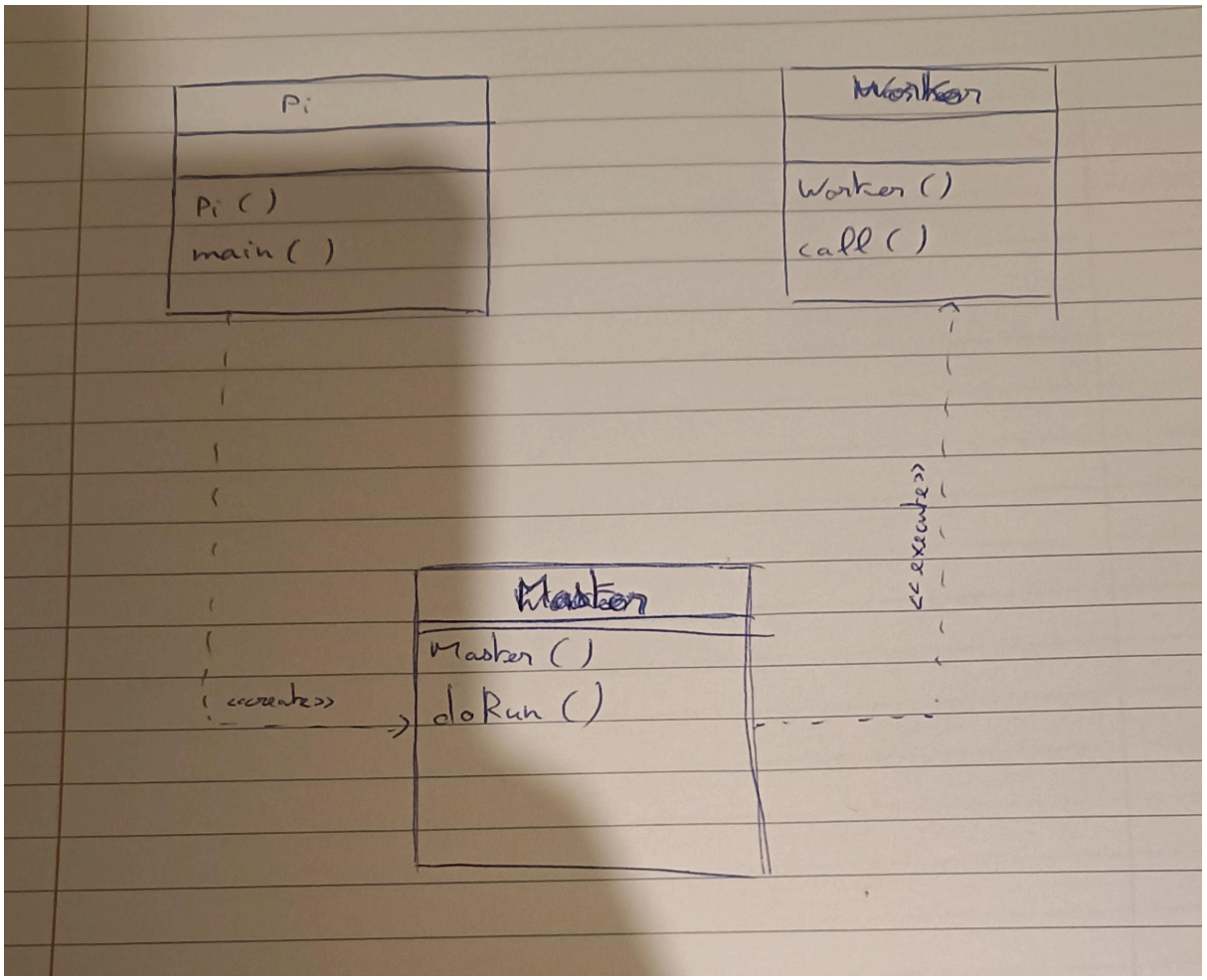
- Dans la méthode main de la classe Pi, on détermine si l'on veut calculer la scalabilité faible ou forte

```
boolean scalabilite = true; // Scalabilité forte si true,
scalabilité faible sinon
```

- S'il s'agit de la scalabilité forte, on divise la taille de l'expérience par le nombre de worker.

```
if (scalabilite) {
   master.doRun(totalCount / worker, worker, filename);
} else {
   filename = filename.replace(".txt", "_" + totalCount + ".txt");
   master.doRun(totalCount, worker, filename);
```

- L’appel de la méthode doRun() va créer une liste de tâches et chaque tâche étant un Worker. Les tâches recevront le nombre de tir à effectuer en fonction de la scalabilité évaluée.
- Les tâches sont ensuite exécutées grâce à un pool ce qui permettra de les exécuter en parallèle.
- Une fois exécutées, On remplis les fichiers textes avec l’erreur, la taille de l’expérience, le nombre de worker, et le temps mis pour chaque calcul de PI



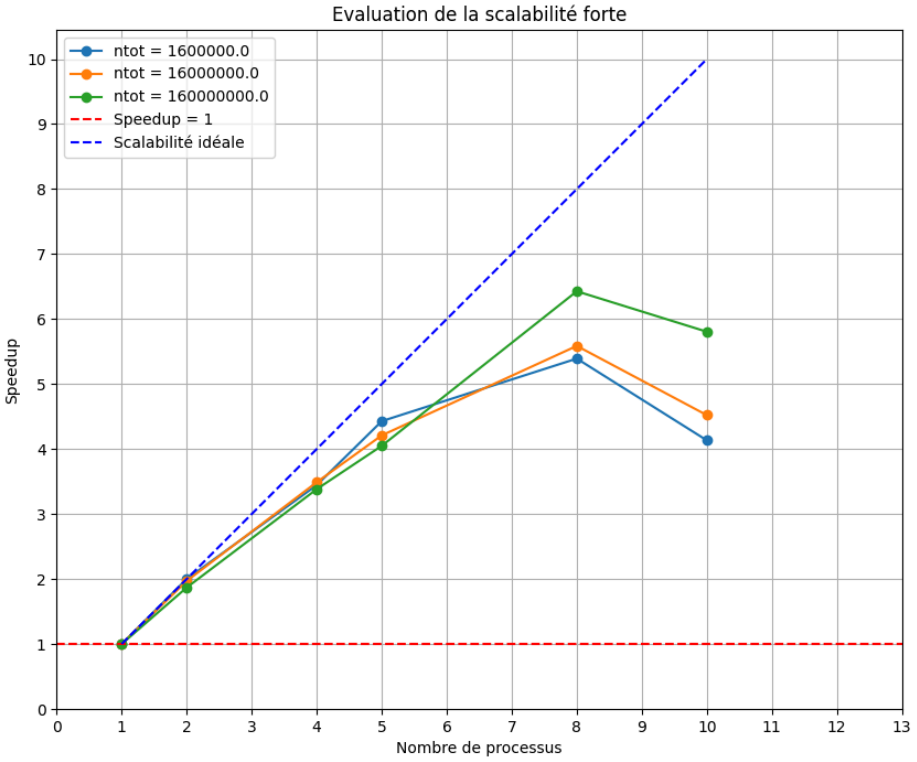
Evaluation des performances (PC G26)

Sur PC : Le processeur Intel Core I7-7700 possède 4 coeurs physiques hyper threadé à 8.

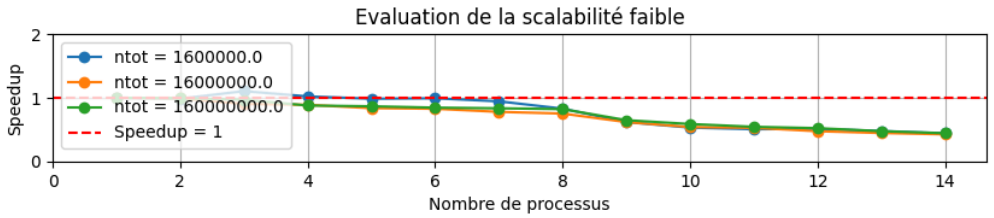
Dans le graphique de scalabilité forte ci-dessous, on observe une performance assez proche de l'idéal jusqu'à 5-6 processus. Ensuite, cela se stabilise jusqu'à 8 (Le maximum) jusqu'à s'éloigner totalement car au-delà de 8 processus, les processus suivants utilisent les mêmes ressources que les premiers.

On observe également que plus la taille de l'expérience est élevée (ntot), meilleure est la scalabilité.

On en déduit qu'une plus grande charge de travail permet de mieux exploiter la parallélisation.



Dans le graphique de scalabilité faible ci-dessous, on observe que jusqu'à 8 processus, les courbes restent assez proches de 1 puis se rapprochent de 0 au-delà. On peut en déduire que l'augmentation proportionnelle de la taille du problème par rapport au nombre de processus ne dégrade pas les performances à part au delà de 8 processus, ce qui est normal.



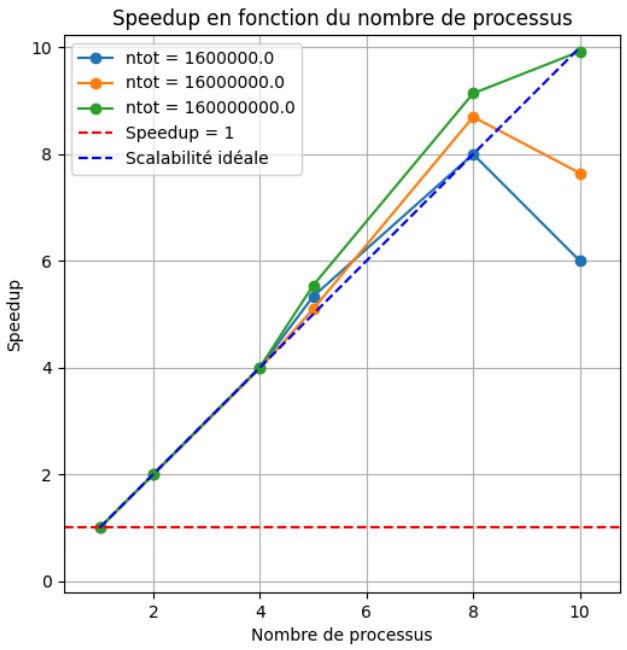
Evaluation des performances (Mac M3)

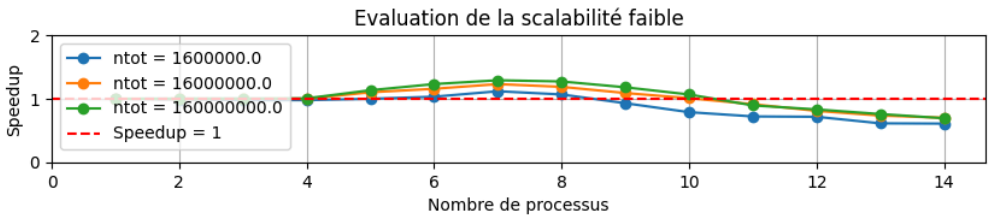
Sur puce M3 (8 coeurs CPU 10 coeurs GPU) (4 coeurs de performance, 4 coeurs d'efficacité)

L'exécution du même code sur mon mac montre des résultats inattendus pour l'évaluation de la scalabilité forte et faible.

En effet, dans le premier graphique, jusqu'à 8 processus, j'observe des courbes au-dessus de la scalabilité idéale ce qui voudrait dire que je ne tire pas partie de la parallélisation.

Dans le deuxième également, les courbes sont proches de 1 puis au-dessus de 1 et finalement en-dessous.





Distribution - MasterSocket / WorkerSocket

Configuration des machines CentOS
Installation de java + fermeture du pare feu pour l'ouverture des ports

```
sudo yum install java-devel
sudo systemctl stop firewalld
```

Pour tester, on peut lancer les scripts d'avant en se plaçant au niveau du package du code :

```
javac tp4/Pi.java
java tp4.Pi
```

Une fois que le code fonctionne, on peut compiler les deux autres fichiers qui sont MasterSocket et WorkerSocket du package tp4_socket.

***Remarque :** Le code de WorkerSocket utilise le code de Worker du package tp4 ce qui permet d'appliquer le critère **"Reusability"** vu en qualité de développement.*

Séance du 03/03/2025

Lors de cette séance, nous avons réalisé des expériences pour calculer π en utilisant la méthode de Monte Carlo de manière distribuée. L'ordinateur d'Alexis a été désigné comme Master, tandis que les 16 autres ordinateurs (PC) ont été utilisés comme Workers. Nous avons pu mener l'expérience avec jusqu'à 64 Workers, chacun utilisant 4 threads pour le calcul.

Les résultats des expériences pour un total_count de 6.4E+10 ont été entré en commun dans le fichier que vous trouverez ici :

https://docs.google.com/spreadsheets/d/1l2Fgtc7R8JHEdoIeecSQAJ-IqZJ2F923Wb9ysvsyzUI/edit?gid=0#gid=0

Les résultats obtenus étaient ceux attendus et l'erreur était moindre. Je ne peux pas proposer de courbe de scalabilité car pas assez de points calculés ont été retenus sur le document.

Qualité de développement

Notes pas à pas

SQuaRE : System and Software Quality Requirements and Evaluation
25010 : voir page 14 (system) pour les explications

Quality News : Experience Utilisateur
Quality Product : Technique

Pages 7 à-11 :

Functional suitability = A quel point je vérifie mes cas d'utilisation (influence sur les utilisateurs principaux)

Performance efficiency = Performance du logiciel en fonction des ressources (matérielles : CPU/GPU, …) à disposition (influence sur les utilisateurs principaux + information de qualité qui concernent les différentes parties prenantes (clients ou autres) => gains de productivité)

Compatibility : Un composant peut utiliser/appeler un autre composant (important pour la maintenance et l'évolution) => Conception architecturale 2eme année

Portability : Compatibilité d'un code d'une machine à une autre (influence sur la maintenance)

Maintainability : (justesse + efficacité) Degré de modification (==Flexibilité) possibilité => N'influence pas l'expérience utilisateur car le rendu est sûrement le même. Pas les autres parties prenantes car ils ne sont pas intéressé par ça mais plutôt les résultats
(Ex : Il y a un bug => a quelle vitesses je peux trouver les bugs et les corrigés facilement)

Throughput rates : (débit) temps de communication d'un produit ou d'un système

Capacity : au maximum combien (ex : nombre de personnes maximum connectés sur un serveur)

[effectiveness : justesse
efficiency : efficacité]

Modularité : Degré avec lequel un système est composé de composants tel que si l'un est modifié, l'impact est moindre. => application modulaire.

Reusability : Réutilisation d'un composant par d'autres composants

Analysability : Couverture par des tests/prints

Modifiability : À quel point je peux modifier sans impacter les autres critères de qualité

Important / À retenir

Le premier document étudié est une norme canadienne qui adopte la norme internationale ISO/IEC 25010:2011.

ISO = International Organization for Standardization
IEC = International Electrotechnical Commission
25010:2011 =

- 250 = Fait partie de la famille SQuare
- 10 = Numéro de la norme
- 2011 = année de publication de la norme

Le document se concentre sur deux aspects principaux :

1) La Qualité en usage (Quality Use)

- ***Effectiveness (Efficacité) :*** Capacité à atteindre les objectifs avec précision
- ***Efficiency (Efficience) :*** Ressources utilisées pour atteindre les objectifs
- ***Satisfaction :*** Degré de satisfaction des utilisateurs lors de l'utilisation du système en terme de/d' ***(Utilité, Confiance, Plaisir, Confort) (Usefulness, Trust, Pleasure, Comfort)***
- ***Freedom from risk (Absence de Risque) :*** Capacité du système à atténuer les risques
- ***Context coverage (Couverture du contexte) :*** Capacité du système à être utilisé dans différents systèmes + ***Flexibility***

2) La Qualité du produit (Quality Product)

- ***Functional Suitability (Adéquation Fonctionnelle) :*** Capacité à fournir des fonctions qui répondent aux besoins exprimés et implicites
- ***Performance Efficiency (Efficacité des Performances) :*** Capacité à fournir des performances optimales par rapport aux ressources utilisées. On peut mesurer le/la/les ***(Comportement temporel, Utilisation des ressources, Capacité) (Time behaviour, Resource utilization, Capacity)***
- ***Compatibility :*** Capacité à coexister et à interagir avec d'autres systèmes. Sous caractéristiques comme :
    - ***Co-existence (Coexistence) :*** Fonctionnement sans impact négatif sur d'autres systèmes
    - ***Interoperability (Interopérabilité) :*** Echange d'informations entre systèmes

- ***Usability (Utilisabilité) :*** Facilité d'utilisation, d'apprentissage et de contrôle du système. Rentre en compte les facteurs d'accessibilité, l'esthétique et d'autres caractéristiques.
- ***Reliability (Fiabilité) :*** Capacité à fonctionner correctement dans des conditions spécifiques. Évaluation en fonction de la tolérance aux pannes, la récupérabilité (retour à un état normal à la suite d'une panne, la disponibilité (l'accessibilité opérationnelle) et la maturité (la fiabilité en condition normale).
- ***Security (Sécurité) :*** Protection des données et des informations contre les accès non autorisés (confidentialité, intégrité authenticité, …)
- ***Maintainability (Maintenabilité) :*** Facilité de modification, d'analyse et de test du système (modularité, réutilisabilité, testabilité, …)
- ***Portability (Portabilité) :*** Capacité à être transféré d'un environnement à un autre.