Chapitre 5 : Qualité logicielle

« Le débogage est deux fois plus difficile que l'écriture de code. Donc, si vous écrivez du code de la manière la plus intelligente possible, vous n'êtes, par définition, pas assez intelligent pour le déboguer. »
Brian Kernighan

1. Objectif du chapitre

Ce chapitre est consacré à la présentation de la notion de qualité logicielle avec un focus particulier sur le point de vue des méthodes agiles.
À l'issue de ce chapitre :

- Vous vous rendrez compte que, pour le développeur, la qualité logicielle résulte plus d'une démarche artisanale que de l'application de normes.
- Vous serez sensibilisé à la notion de qualité de code.
- Vous saurez tirer parti des indicateurs de qualité pour renforcer la qualité de vos codes.

2. Notion de qualité logicielle et assurance qualité (QA)

2.1. Définitions

La **qualité** englobe l'ensemble des caractéristiques d'un logiciel qui affecte sa capacité à satisfaire des besoins exprimés ou implicites en termes de fonctionnalités, délais et coûts.
Une appréciation globale de la qualité tient autant compte de :

- facteurs externes, directement observables par l'utilisateur,
- facteurs internes, observables par les ingénieurs de développement.

L'**assurance qualité** est un ensemble d'activités planifiées et systématiques de toutes les actions nécessaires pour fournir une assurance que la qualité du logiciel est conforme aux exigences et aux attentes établies.

2.2. Normes de qualité

La qualité logicielle est régie par des normes, issues de la famille ISO 9000.

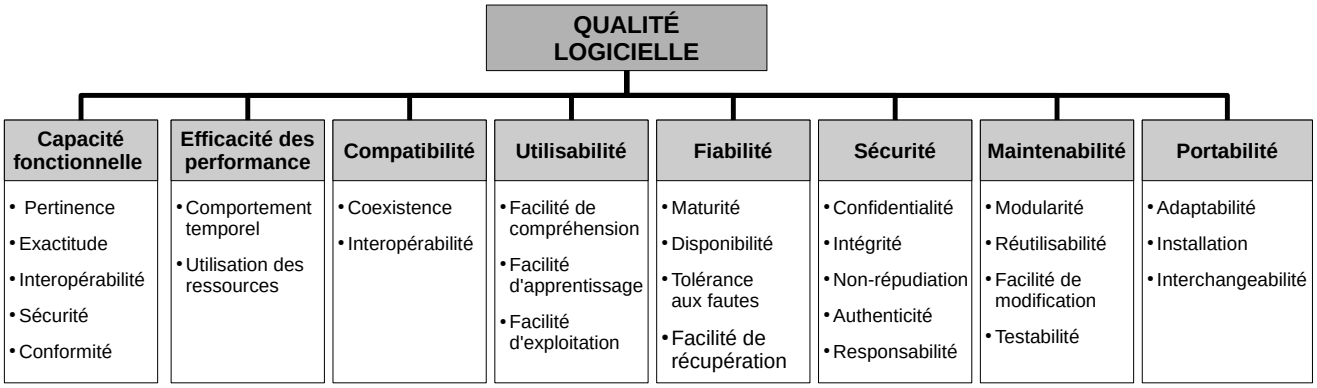
2.2.1. Normes de qualité par des facteurs internes

Citons deux normes emblématiques :

- La norme ISO/CEI 90003 est la ligne directrice de l'application de la norme ISO 9001 à l'acquisition, la fourniture, le développement, l'exploitation et la maintenance de logiciels informatiques et aux services de soutien connexes. Elle se base entièrement sur l'approche prédictive de la gestion de projet.
- La norme ISO/CEI 12207 décrit un modèle pour le processus du cycle de vie du logiciel qui reprend globalement le modèle en V.

2.2.2. Normes de qualité par des facteurs externes

- La norme ISO/CEI 25010 décrit la qualité d'un logiciel à partir par 35 indicateurs de performance (*Key Performance Indicator*, KPI) couvrant 8 caractéristiques :



- La capacité fonctionnelle représente la mesure dans laquelle un produit ou un système fournit des fonctions qui répondent aux besoins exprimés et implicites lorsqu'il est utilisé dans des conditions spécifiées.
- L'efficacité des performances représente la performance par rapport à la quantité de ressources utilisées dans des conditions indiquées.
- La compatibilité est le degré avec lequel un produit, un système ou un composant peut échanger des informations avec d'autres produits, systèmes ou composants, et/ou exécuter ses fonctions requises, tout en partageant le même environnement matériel ou logiciel.
- L'utilisabilité est la mesure dans laquelle un produit ou un système peut être utilisé par des utilisateurs spécifiés pour atteindre des objectifs spécifiés avec efficacité, efficience et satisfaction dans un contexte d'utilisation spécifié.
- La fiabilité est le degré avec lequel un système, un produit ou un composant exécute des fonctions spécifiées dans des conditions spécifiées pendant une période de temps spécifiée.
- La sécurité est le degré avec lequel un produit ou un système protège les informations et les données afin que les personnes ou autres produits ou systèmes disposent d'accès aux données en fonction de leurs types et niveaux d'autorisation.
- La maintenabilité représente le degré d'efficacité et d'efficience avec lequel un produit ou un système peut être modifié pour l'améliorer, le corriger ou l'adapter aux changements de l'environnement et des exigences.
- La portabilité est le degré d'efficacité et d'efficience avec lequel un système, un produit ou un composant peut être transféré d'un matériel, d'un logiciel ou d'un autre environnement opérationnel ou d'utilisation à un autre.

2.3. Bilan

Malheureusement, en pratique ces normes de qualité n'aident en rien à la production de logiciel de qualité.

- La norme ISO/CEI 90003 préconise une méthode de gestion de projet prédictive, mais nous l'avons vu elle est inadaptée au cas du logiciel.
- La norme ISO/CEI 12027 se base sur un modèle de développement en V. Mais, nous l'avons vu plus tôt, ce modèle n'offre pas les meilleures garanties de succès des projets.
- La norme ISO/CEI 25010 se présente comme une liste de caractéristiques à mesurer sur le produit fini.

Mais, en développement logiciel, il n'existe pas d'indicateurs objectifs et incontestables pour mesurer la valeur de ces caractéristiques et donc pas d'outils permettant de graduer la qualité d'un logiciel. Finalement, la mesure de ces indicateurs (KPI) est largement empirique et donc la qualité reste globalement subjective, ce qui va à l'encontre de l'ambition des normes.

3. Le point de vue de l'agilité

L'agilité renonce à l'application de ces normes qu'elle considère comme contre-productives pour se tourner vers les valeurs humaines dans les approches du développement.

- Facteurs externes : Le cycle de vie est basé sur un cycle itératif et incrémental qui repose sur une collaboration étroite entre l'équipe de développement et le client qui est le juge de la qualité.
- Facteurs internes : La qualité d'un logiciel résulte de la qualité de son code. Il ne suffit pas qu'un logiciel soit efficace, il faut en plus qu'il soit bien conçu. Cette vision est mise en pratique de façon emblématique par la méthode Agile *eXtreme Programming* (XP). En agilité, l'ingénieur développeur se voit comme un artisan du logiciel qui prône l'amour du travail bien fait. Ainsi, les caractéristiques énumérées dans la norme ISO/CEI 25010 fournissent les points de focalisation de l'attention du développeur en recherche de qualité de code mais en aucun cas des mesures de qualité (il n'y a plus de notion de KPI).

3.1. Artisanat du logiciel (*Software craftsmanship*)

En l'absence de mesures objectives, l'artisanat du logiciel met l'accent sur les compétences de codage des développeurs. Le code produit par l'équipe est considéré comme un « chef-d'œuvre ».
Les facteurs de qualité du code à privilégier :

- **Simplicité** (Principe KISS : *Keep it Simply*, *Stupid*)
  - Le code doit être « évident ».
- **Propreté**
  - Le code doit se lire comme sa propre documentation.
- **Testabilité**
  - Le code doit être couvert par des tests automatiques.

3.2. Atelier de l'artisan

Pour aider à la qualité du code, il faut que l'équipe de développement se dote :

- d'un environnement de développement professionnel,
- des outils d'aide à l'analyse de la propreté du code.

3.2.1. Environnement de développement

Le développement nécessite des environnements de développement sophistiqués. Cela inclut de façon non exhaustive :

1. IDE : Environnement de Développement Intégré (e.g., à l'école : Intellij, CLion, Android Studio, Visual Studio)
   - Permet une édition avancée du code.
   - Permet la compilation interactive.
   - Permet le management de code.
2. Logiciel de gestion de versions (e.g., à l'école : Git et *gitlab.ecole.ensicaen.fr*).

- Partage de la production.
- Suivi de versions.

3. Moteur de production (i.e., super makefile) (e.g., pour le Java : Gradle ou Maven).
   - Compilation.
   - Déploiement.
4. Outils d'intégration continue (e.g., à l'école GitLab-CI/CD)
   - Vérification automatique de l'opérationnalisation du logiciel.
5. Des outils de déverminage et de profilage de code.

3.2.2. Outils d'aide à l'analyse de la propreté du code

Pour aider à la propreté du code, les informaticiens se sont dotés d'outils d'analyse automatique de code. S'il est impossible, à l'heure actuelle, de mesurer la qualité d'un logiciel, il est possible de critiquer la qualité d'un code. En Java, les outils les plus utilisés sont :

- **Checkstyle** : vérification des règles et conventions de codage.
- **PMD** : similaire à Checkstyle mais plus focalisé sur les problèmes potentiels de codage comme le code non utilisé ou sous-optimisé, la taille et la complexité du code.
- **FindBugs** : détection des bogues potentiels, des problèmes de performance, ou des mauvaises habitudes de codage.
- **SonarQube** : une référence dans le domaine de l'analyse de code (une combinaison de tous les outils précédents).

Ces outils d'analyse sont intensément utilisés pour identifier les parties de code qui doivent être retravaillées. Ils peuvent être ajoutés sous forme de plugins dans les IDE. Je vous recommande d'installer au moins le plugin SonarQube.

Il existe des outils d'analyse de code nativement dans l'IDE IntelliJ IDEA :

- Voir le menu `Analyze::inspect code`.
- Il faut le configurer avec ses préférences (voir la plateforme pour un exemple de fichier avec une configuration reprenant les directives données en ODL).

4. Que retenir de ce chapitre

- La qualité logicielle est impossible à mesurer.
- Les normes de qualité issues de la famille ISO 9000 n'apportent aucune aide pour développer des logiciels de qualité.
- Si l'on s'intéresse à la qualité du point de vue des développeurs, les facteurs de qualité sont :
  - La simplicité
  - La propreté du code
  - La testabilité
- En l'absence de mesure objective de ces facteurs, l'assurance qualité consiste à :
  - Adopter une posture d'artisan qui considère que la qualité du logiciel résulte de la qualité du code.
  - Utiliser des environnements de développement sophistiqués qui permettent d'obtenir des critiques de la qualité du code.
  - Les indicateurs de performance (*KPI*) ne doivent être considérés que comme des alarmes de parties de code à retravailler et pas comme des mesures de la qualité du logiciel.