UTOPIOS
TRANSFORMATION DIGITALE INTELLIGENTE

JAVA CLEAN CODE Partie 1

Programme

1. Introduction à la qualité logicielle
2. Métriques dans un environnement Java
3. Découverte des outils de qualimétrie dans le monde Java
4. Utilisation des outils de qualimétrie dans un environnement Java
5. Développement en TDD
6. Développement en BDD
7. Comment réaliser et utiliser les rapports qualimétriques
8. Mise en relation des métriques fournies et des bonnes pratiques
9. Stratégie de mise en œuvre de la qualimétrie

Introduction à la qualité logicielle

- Notion de dette technique / Problèmes de la non-qualité.
- Qualité, normes et certification.
- Importance de la qualité du code dans le développement logiciel.
- TP : Analyse de quelques exemples de code non qualitatis

Introduction à la qualité logicielle

Notion de dette technique / Problèmes de la non-qualité

- La dette technique est un concept en génie logiciel qui reflète le coût implicite de l'ajout de fonctionnalités supplémentaires à un logiciel, dû au choix de solutions faciles (mais incorrectes) au lieu d'utiliser une meilleure approche qui prendrait plus de temps.
- L'analogie de la dette est appropriée parce qu'une dette a un coût : l'intérêt.
- De même, choisir une solution rapide et facile peut avoir un coût supplémentaire plus tard, lorsque vous devrez refactoriser votre code, le déboguer ou le documenter. Plus vous attendez pour "rembourser" cette dette, plus les "intérêts" s'accumulent sous forme de coûts de maintenance supplémentaires.

Introduction à la qualité logicielle

Notion de dette technique / Problèmes de la non-qualité

- La dette technique peut se manifester de différentes façons :

1. **Code mal structuré** : Le code peut être difficile à lire, mal organisé, ou ne pas respecter les conventions de codage standard. Cela peut rendre le code difficile à comprendre et à maintenir.
2. **Duplication de code** : Au lieu de réutiliser le code existant ou de créer des fonctions réutilisables, le même code peut être répété à plusieurs endroits. Cela peut rendre le code plus difficile à maintenir et plus sujet aux bugs.
3. **Manque de tests** : Si le code n'est pas bien testé, il peut contenir des bugs non détectés. Les tests sont également importants pour vérifier que les modifications du code n'introduisent pas de nouveaux bugs.
4. **Manque de documentation** : Sans une bonne documentation, il peut être difficile pour d'autres développeurs de comprendre comment le code fonctionne et comment l'utiliser ou le modifier.

Introduction à la qualité logicielle

Notion de dette technique / Problèmes de la non-qualité

- Demo : Un contrôleur Spring

Introduction à la qualité logicielle

Qualité, normes et certification

- La qualité du logiciel est une évaluation de la façon dont le logiciel se conforme aux exigences fonctionnelles et non fonctionnelles, aux normes de développement et aux caractéristiques désirables telles que la maintenabilité, la performance et la facilité d'utilisation.

1. **Fonctionnalité** : Le logiciel doit répondre à toutes les exigences fonctionnelles spécifiées. Il doit accomplir les tâches pour lesquelles il a été conçu de manière précise et fiable.
2. **Performance** : Le logiciel doit être efficace dans son utilisation des ressources système et doit maintenir une vitesse d'exécution acceptable même lorsque la charge est élevée.
3. **Fiabilité** : Le logiciel doit être capable de fonctionner de manière constante et correcte dans diverses conditions, et doit être capable de se récupérer gracieusement des erreurs.

Introduction à la qualité logicielle

Qualité, normes et certification

4. **Utilisabilité** : Le logiciel doit être facile à utiliser et intuitif, minimisant la courbe d'apprentissage pour les nouveaux utilisateurs.
5. **Maintenabilité** : Le logiciel doit être conçu de telle manière qu'il soit facile à comprendre, à modifier et à étendre. Un code bien structuré, lisible et documenté est essentiel pour la maintenabilité.
6. **Évolutivité** : Le logiciel doit être capable de s'adapter et de croître en fonction des exigences changeantes et de l'augmentation de la charge.

Introduction à la qualité logicielle

Qualité, normes et certification

- Les normes de codage, également connues sous le nom de conventions de codage, sont un ensemble de lignes directrices, de règles et de spécifications que les développeurs de logiciels suivent pour écrire le code source d'un programme.
- Ces règles régissent la structure, l'apparence, les conventions de nommage et d'autres aspects du code.

1. **Lisibilité et compréhension du code** : Les normes de codage aident à garder le code propre, organisé et facile à lire, ce qui est crucial pour la compréhension et la maintenabilité du code.
2. **Cohérence** : Elles assurent une cohérence dans le code, ce qui signifie que le code semble avoir été écrit par une seule personne, même s'il a été écrit par une équipe.
3. **Productivité** : En rendant le code plus facile à comprendre, les normes de codage peuvent augmenter la productivité des développeurs.
4. **Qualité** : Les normes de codage aident à prévenir les erreurs, en particulier en forçant les bonnes pratiques.

Introduction à la qualité logicielle

Qualité, normes et certification

Exemples de normes de codage en Java :

1. **Conventions de nommage** : Par exemple, les noms de classe en Java devraient commencer par une majuscule, et si le nom est composé de plusieurs mots, chaque mot supplémentaire doit également commencer par une majuscule (ex : ClassName). Les variables et les méthodes doivent commencer par une minuscule et suivre la règle camelCase (ex : variableName).
2. **Directives de formatage** : Par exemple, l'indentation doit être de 4 espaces, et les accolades doivent suivre le style K&R ("Egyptian brackets").
3. **Commentaires** : Toutes les classes et méthodes devraient avoir des commentaires Javadoc expliquant leur but, leurs paramètres et leur valeur de retour. Les commentaires de ligne ou de bloc peuvent être utilisés pour expliquer des parties de code particulièrement complexes.

Introduction à la qualité logicielle

Qualité, normes et certification

- La certification de qualité logicielle est un processus par lequel une organisation indépendante évalue les processus de développement de logiciels d'une entreprise pour s'assurer qu'ils respectent certaines normes de qualité. L'objectif est de valider que le logiciel est développé de manière contrôlée et efficace, réduisant ainsi les risques d'erreurs, de défaillances et de problèmes de sécurité.

Différentes certifications de qualité logicielle existent :

1. **ISO 9000** : Il s'agit d'une série de normes internationales qui établissent des critères pour les systèmes de gestion de la qualité. L'ISO 9000 est applicable à tous types d'organisations, pas seulement aux entreprises de logiciels. Il s'agit essentiellement de s'assurer que les organisations ont des processus standardisés et contrôlés pour fournir un produit ou un service de qualité.
2. **CMMI (Capability Maturity Model Integration)** : C'est un modèle de maturité qui fournit des directives pour améliorer les processus d'une organisation, avec un accent particulier sur le développement de produits, y compris les logiciels. Les organisations peuvent être évaluées sur une échelle de 1 à 5 pour déterminer leur niveau de "maturité" en matière de processus.

Introduction à la qualité logicielle

Qualité, normes et certification

Concernant la certification Clean Code, celle-ci est centrée sur le développement de logiciels. Elle vise à valider que les développeurs comprennent et appliquent les principes de développement "propre" - c'est-à-dire un code qui est facile à lire, à comprendre et à maintenir.
La certification Clean Code couvre une variété de sujets, tels que :

- Les principes SOLID
- Les conventions de codage et de formatage
- Les bonnes pratiques de commentaires
- L'importance des tests unitaires et de l'intégration continue
- Comment éviter les anti-modèles et le code spaghetti

Introduction à la qualité logicielle

Les principes SOLID

- Le principe SOLID est un ensemble de principes de conception logicielle qui visent à créer des systèmes évolutifs, flexibles et faciles à maintenir. Ces principes sont souvent mentionnés dans le contexte du Clean Code,

1. Principe de responsabilité unique (Single Responsibility Principle - SRP) : Ce principe stipule qu'une classe ou un module ne devrait avoir qu'une seule raison de changer. En respectant le SRP, on s'assure que chaque classe a une responsabilité claire et bien définie, ce qui facilite la compréhension, la maintenance et les modifications du code.
2. Principe d'ouverture/fermeture (Open/Closed Principle - OCP) : Ce principe énonce que les entités logicielles (classes, modules, etc.) devraient être ouvertes à l'extension, mais fermées à la modification. Cela signifie que vous devriez pouvoir ajouter de nouvelles fonctionnalités en ajoutant de nouveaux modules ou en dérivant de classes existantes, sans avoir à modifier le code existant. Cela permet d'éviter les effets de bord et de maintenir la stabilité du système.

Introduction à la qualité logicielle

Les principes SOLID

3. Principe de substitution de Liskov (Liskov Substitution Principle - LSP) : Ce principe met l'accent sur l'héritage et stipule que les objets d'une classe dérivée doivent pouvoir être substitués par des objets de leur classe de base sans altérer la cohérence du système. En respectant le LSP, on garantit une utilisation cohérente des sous-classes et une meilleure modularité du code.
4. Principe de ségrégation des interfaces (Interface Segregation Principle - ISP) : Ce principe recommande de découpler les interfaces en des ensembles de méthodes cohérentes plutôt que de créer des interfaces monolithiques. Cela permet aux clients d'utiliser uniquement les méthodes dont ils ont besoin, évitant ainsi les dépendances inutiles. En appliquant le ISP, on facilite la maintenance et la réutilisation du code.
5. Principe d'inversion des dépendances (Dependency Inversion Principle - DIP) : Ce principe indique que les modules de haut niveau ne devraient pas dépendre directement des modules de bas niveau, mais plutôt d'abstractions. Cela favorise la modularité, la flexibilité et la testabilité du code. En respectant le DIP, on peut réduire les dépendances directes et rendre le code plus facilement extensible.

Introduction à la qualité logicielle

Qualité, normes et certification

- L'amélioration continue de la qualité est une approche systématique pour améliorer les processus de travail et les produits de manière continue. Elle souligne que la qualité ne doit pas être considérée comme un objectif unique, mais comme un processus en constante évolution visant à augmenter la satisfaction des clients et à améliorer les performances opérationnelles.
- La méthodologie Kaizen, originaire du Japon, est l'un des cadres les plus populaires pour l'amélioration continue. Kaizen, qui signifie "amélioration continue" en japonais, se concentre sur l'identification et l'élimination des gaspillages - c'est-à-dire des processus ou des activités qui n'ajoutent aucune valeur.

Introduction à la qualité logicielle

Qualité, normes et certification

- En matière de développement logiciel, la revue de code et les tests sont deux pratiques essentielles pour maintenir et améliorer la qualité du code :

1. **Revue de code** : Cette pratique implique que d'autres développeurs examinent le code pour vérifier la logique, la clarté, la qualité et la conformité avec les conventions de codage. Les revues de code peuvent aider à identifier les bugs, les erreurs de logique ou de conception et d'autres problèmes avant qu'ils ne deviennent trop coûteux ou difficiles à résoudre.
2. **Tests** : Les tests de logiciels sont un autre pilier de l'amélioration de la qualité. Les tests unitaires, les tests d'intégration, les tests de charge, les tests de sécurité et d'autres formes de tests peuvent aider à identifier les problèmes avant qu'ils ne soient découverts par les utilisateurs. L'utilisation de techniques telles que l'intégration continue et le déploiement continu peut également aider à garantir que les tests sont effectués régulièrement et automatiquement.

Introduction à la qualité logicielle

Importance de la qualité du code dans le développement logiciel

- La qualité du code est cruciale dans le développement logiciel pour de nombreuses raisons.
- Un code de haute qualité est plus facile à lire, à comprendre, à maintenir et à modifier. Cela facilite la détection des bugs, la mise en œuvre de nouvelles fonctionnalités, l'adaptation à l'évolution des exigences et l'extension du logiciel à mesure qu'il grandit et évolue. De plus, un code de qualité supérieure est souvent plus performant, plus sûr et plus fiable.
- **Facilité de maintenance** : Un code de haute qualité est plus facile à maintenir et à modifier. Cela est particulièrement important à mesure que le logiciel évolue et que de nouvelles fonctionnalités sont ajoutées ou que des modifications sont apportées.
    - Exemple en Java Spring : Si vous avez une application Spring Boot bien structurée, où le code est divisé en services modulaires, chaque service pouvant être développé, déployé et mis à jour indépendamment des autres, cela facilite la maintenance. Par ailleurs, l'utilisation d'une architecture propre et d'un codage cohérent améliore la lisibilité et facilite la maintenance.

Introduction à la qualité logicielle

Importance de la qualité du code dans le développement logiciel

- **Détection des bugs** : Un code bien organisé et cohérent rend plus facile la détection des bugs. Cela est d'autant plus vrai lorsque le code est accompagné de tests unitaires et d'intégration bien conçus qui peuvent être exécutés automatiquement.
    - Exemple en Java Spring : Dans Spring Boot, l'utilisation de frameworks de test comme JUnit et Mockito facilite la rédaction de tests unitaires et d'intégration. Ces tests permettent de détecter rapidement les bugs avant qu'ils ne deviennent un problème en production.
- **Performances** : Un code de haute qualité est souvent plus performant. En évitant les goulots d'étranglement, en optimisant les algorithmes et en utilisant les ressources de manière efficace, un code de qualité supérieure peut souvent améliorer les performances d'une application.
    - Exemple en Java Spring : Si vous utilisez Spring Data JPA pour gérer l'accès aux données dans votre application Spring Boot, une utilisation efficace des requêtes personnalisées, des requêtes nommées, des spécifications ou des critères peut aider à améliorer les performances en minimisant le nombre de requêtes SQL et en récupérant uniquement les données nécessaires.

Introduction à la qualité logicielle

Importance de la qualité du code dans le développement logiciel

- **Sécurité** : Un code de haute qualité est souvent plus sûr. En évitant les vulnérabilités courantes, en utilisant des pratiques de codage sécurisées et en prenant en compte la sécurité dès le début du développement, vous pouvez rendre votre code plus résistant aux attaques.
  - Exemple en Java Spring : Dans Spring Boot, l'utilisation de Spring Security peut aider à sécuriser votre application en fournissant des fonctionnalités pour l'authentification, l'autorisation, la protection contre les attaques CSRF, etc. En suivant les meilleures pratiques de codage sécurisé et en tenant compte de la sécurité dès le début, vous pouvez rendre votre code plus sûr.
- **Fiabilité** : Enfin, un code de haute qualité est souvent plus fiable. En évitant les erreurs de codage courantes, en gérant correctement les exceptions et en utilisant des pratiques de codage robustes, vous pouvez rendre votre code plus résistant aux erreurs et aux défaillances.
  - Exemple en Java Spring : En gérant correctement les exceptions dans votre application Spring Boot, en utilisant par exemple le contrôleur de conseils (@ControllerAdvice) pour gérer les exceptions de manière centralisée, vous pouvez rendre votre application plus robuste et plus fiable. De plus, l'utilisation de stratégies de reprise après erreur, telles que les réessais et les circuits ouverts avec Spring Retry et Hystrix, peut aider à améliorer la fiabilité de votre application.

Introduction à la qualité logicielle

TP

- Vous êtes un nouvel ingénieur logiciel chez TechCorp Inc. L'une de vos premières tâches est de revoir et d'améliorer le code d'une application web développée avec Spring Boot. Cette application web présente actuellement quelques problèmes de performances et de sécurité.

Instructions du TP

- Analysez chaque extrait de code et identifiez les problèmes de qualité qui s'y trouvent.
- Discutez de l'impact potentiel de chaque problème identifié. Comment ces problèmes peuvent-ils affecter les performances, la sécurité, la maintenance et la fiabilité de l'application ?
- Proposez des solutions pour améliorer la qualité du code dans chaque extrait. Réécrivez le code pour résoudre les problèmes que vous avez identifiés.
- Présentez vos résultats et vos solutions aux autres stagiaires et discutez des différentes approches pour résoudre ces problèmes.