Maintenance applicative
1. Rétroconception

Halim Djerroud

LISV
Laboratoire d'ingénierie
des systèmes de Versailles

UVSQ
université PARIS-SACLAY

révision : 0.1

Plan

- Introduction à la Maintenance Applicative
- Étapes Préliminaires et redocumentation
- Retroconception

Introduction à la Maintenance Applicative

La maintenance applicative consiste à maintenir, adapter et faire évoluer un logiciel existant pour garantir son bon fonctionnement, sa compatibilité avec de nouveaux environnements, et sa réponse aux besoins des utilisateurs. Elle englobe des actions correctives, adaptatives, évolutives et préventives pour assurer la pérennité et la performance de l'application.

- Types de maintenance :
  - Corrective.
  - Adaptative.
  - Évolutive.
  - Préventive.
- Exemples de contextes où la maintenance est essentielle.

Types de Maintenance Applicative (1)

- **Maintenance Corrective** :
  - Résolution des bugs et erreurs identifiés après le déploiement.
  - Exemples :
    - Correction d'une erreur dans un calcul.
    - Réparation d'une fonctionnalité qui ne répond pas correctement.
- **Maintenance Adaptative** :
  - Modification de l'application pour qu'elle fonctionne dans un nouvel environnement.
  - Exemples :
    - Migration vers un nouveau système d'exploitation ou matériel.
    - Adaptation à une nouvelle version d'une base de données.

Types de Maintenance Applicative (2)

- **Maintenance Évolutive** :
  - Ajout de nouvelles fonctionnalités ou améliorations pour répondre aux besoins des utilisateurs.
  - Exemples :
    - Ajout d'un tableau de bord analytique.
    - Intégration d'une API tierce pour étendre les capacités.
- **Maintenance Préventive** :
  - Amélioration de la stabilité et prévention des défaillances potentielles.
  - Exemples :
    - Réécriture d'un module pour éviter des problèmes de performance futurs.
    - Mise à jour d'une bibliothèque obsolète avant qu'elle ne pose problème.

Introduction aux Langages Obsolètes

- **Définition** :
  - Les langages obsolètes sont des technologies largement utilisées dans le passé mais qui ne sont plus activement développées ou soutenues.
- **Exemples courants** :
  - **COBOL** : Utilisé dans les systèmes bancaires et financiers.
  - **Fortran** : Populaire dans le calcul scientifique.
  - **VB6** : Développement d'applications Windows.
- Ces langages restent présents dans des systèmes critiques encore en production.

Problèmes Associés aux Langages Obsolètes

- **Rareté des compétences** :
  - Diminution du nombre de développeurs expérimentés.
  - Coût élevé de recrutement et de formation.
- **Limitations techniques** :
  - Absence de fonctionnalités modernes (parallélisme, sécurité, etc.).
  - Difficulté d'intégration avec des technologies récentes.
- **Manque de support** :
  - Arrêt des mises à jour officielles.
  - Faible communauté active pour résoudre les problèmes.

Exemples d'Applications Critiques

- **Systèmes bancaires et financiers** :
  - COBOL gère encore une grande partie des transactions bancaires mondiales.
- **Calcul scientifique et ingénierie** :
  - Fortran utilisé pour des simulations complexes (climatologie, mécanique des fluides).
- **Gestion interne d'entreprises** :
  - Applications écrites en VB6 pour l'inventaire, la facturation, etc.

Risques

Ces systèmes critiques ne peuvent souvent pas être remplacés rapidement en raison des coûts et des risques associés.

Solution : Migration vers un Langage Moderne

- Choisir le bon langage cible :
  - Critères : longévité, compatibilité, performance.
  - Exemples : COBOL → Java, VB6 → .NET.
- Techniques de migration :
  - Réécriture complète.
  - Migration incrémentale.
  - Utilisation de traducteurs automatiques.
- Gestion des risques.

Étapes Préliminaires

- Audit de l'application :
  - Évaluation du code source.
  - Identification des dépendances.
- Rassembler les spécifications (si disponibles).
- Inventaire des outils nécessaires pour la rétroconception.

Introduction aux Étapes Préliminaires

- Les étapes préliminaires sont essentielles pour :
  - Comprendre l'état actuel de l'application.
  - Identifier les risques et planifier efficacement la maintenance.
- Principaux objectifs :
  - Audit de l'application.
  - Rassemblement des spécifications.
  - Inventaire des outils nécessaires.

Audit de l'Application

- **Évaluation du code source** :
  - Lisibilité, modularité et documentation.
  - Identification des sections critiques.
- **Identification des dépendances** :
  - Bibliothèques et frameworks utilisés.
  - Risques liés aux dépendances obsolètes ou non supportées.
- Outils courants : SonarQube, PMD.

Rassembler les Spécifications

- **Pourquoi ?**
  - Comprendre les exigences initiales.
  - Identifier les écarts entre le comportement attendu et actuel.
- **Comment ?**
  - Recherche dans la documentation existante.
  - Consultation avec les parties prenantes :
    - Développeurs initiaux.
    - Utilisateurs finaux.

Inventaire des Outils

- **Outils nécessaires** :
  - Analyse statique (ex. : SonarQube, code2flow, Sourcetrail).
  - Rétroconception (ex. : Enterprise Architect).
  - Gestion de versions (ex. : Git).
- **Défis** :
  - Documentation insuffisante.
  - Dépendances complexes.
  - Coûts et temps liés à un audit complet.
- Une bonne préparation est clé pour une maintenance et une modernisation réussies.

La Redocumentation

- **Définition** :
  - Processus de création ou de mise à jour de la documentation d'une application à partir de son code source existant.
- **Objectifs** :
  - Faciliter la compréhension du système.
  - Identifier les fonctionnalités et les interactions des composants.
  - Soutenir les étapes de rétroconception et de maintenance.
- **Quand ?** :
  - En l'absence de documentation initiale.
  - Lorsque la documentation est obsolète ou incomplète.

Méthodes de Redocumentation

- **Analyse statique du code** :
  - Extraction des structures (fonctions, classes, modules).
  - Identification des dépendances et des flux de données.
- **Utilisation d'outils spécialisés** :
  - Outils de génération de documentation : Doxygen, Sphinx.
  - Générateurs UML : Enterprise Architect, Visual Paradigm.
- **Techniques manuelles** :
  - Lecture du code pour identifier les fonctionnalités clés.
  - Ajout ou mise à jour de commentaires dans le code.

La Rétroconception

- **Définition et objectifs** :
  - Transformer le code source en documentation exploitable.
- **Techniques de rétroconception** :
  - Génération de diagrammes UML (composants, classes, séquence, etc.).
  - Analyse des dépendances.
  - Détection des algorithmes critiques.
- **Outils de rétroconception** :
  - Exemple : Enterprise Architect, Visual Paradigm.

Introduction aux Diagrammes de Composants

- **Définition** :
  - Diagramme UML décrivant la structure d'un système en termes de composants logiciels.
- **Rôle principal** :
  - Montrer comment les différentes parties d'un système interagissent.
  - Représenter les dépendances et interfaces entre composants.
- **Avantages** :
  - Permet une vision modulaire du système.
  - Simplifie l'analyse et la refonte de l'architecture.

Éléments d'un Diagramme de Composants

- **Composant** :
  - Représente une unité modulaire du système.
  - Exemples : modules, bibliothèques, services.
- **Interfaces** :
  - Définissent les points d'interaction entre composants.
- **Connecteurs** :
  - Représentent les relations ou dépendances entre composants.
- **Artefacts associés** :
  - Fichiers, bases de données ou configurations utilisées par les composants.

Exemple de Diagramme de Composants

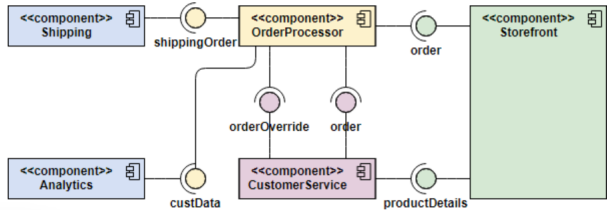

Figure { Exemple diagramme de composants.

- Les composants sont reliés par des interfaces définissant leurs interactions.
- Chaque composant peut inclure des artefacts associés.

Importance des Diagrammes de Composants dans la Maintenance

- Simplifient la compréhension de la structure existante.
- Aident à identifier les zones critiques ou fortement couplées.
- Facilitent la planification de modifications ou de migrations.
- Soutiennent la collaboration entre équipes de développement et de maintenance.