Génie logiciel

Introduction

Caractéristiques du logiciel

- Logiciel est ubiquitaire dans tous les domaines de métier
- Logiciel s'étend à tous les aspects de la vie humaine
- Application qui prend en charge toutes les fonctions d'une organisation

Difficultés auxquelles font face les programmeurs

- Complexité accidentelle
  - Plus facile à résoudre
  - Due aux technologies utilisées
  - Imprévus de l'environnement
  - Problèmes transitoires
- Complexité essentielle
  - Difficultés inhérentes difficiles à résoudre
  - Complexité, invisibilité, versatilité, conformité, discontinuité

Conformité

- Gros logiciels composés de matériel hardware, d'usagers, d'interactions avec d'autres logiciels, etc.
- Logiciel intègre toutes ces parties dans un seul système
  - Préserve l'unité du domaine
  - Communique avec les composants hétérogènes du domaine
- Logiciel incarne le domaine
  - Propriétés du domaine s'infiltrent dans le logiciel
  - Logiciel doit se conformer au domaine (Besoin de connaissance du domaine)

Naissance du génie logiciel

- En 1968, la conférence de l'OTAN se réunit pour discuter d'un nouveau domaine: le génie logiciel (GL)
- Rendre GL une discipline à part entière
- Suivant les mêmes méthodologies que les disciplines de génie traditionnelles
- [Le génie logiciel est] l'établissement et l'utilisation de principes du génie afin d'obtenir un logiciel économique, fiable et qui fonctionne de manière efficace sur de vraies machines

Défis de l'ingénieur logiciel

- Logiciel doit correspondre aux besoins du client tout en fournissant une solution efficace au problème
- Logiciel doit être rentable
  - Coûts et temps de développement
  - Exigences croissantes
- La solution doit être de haute qualité
  - Réduire les efforts de maintenance

Les 5 grandes activités d'un ingénieur logiciel

1. Décrire
   a. Besoins
   b. Spécification de conception
   c. Documentation
2. Implémenter
   a. Conception
   b. Programmation
3. Évaluer
   a. Test
   b. Vérification
   c. Validation
   d. Révision
4. Gérer
   a. Planification
   b. Échelonnage
   c. Communication
5. Faire fonctionner
   a. Déploiement
   b. Installation
   c. Maintenance

Vie du logiciel

Vie d'un produit

- Développement
- Lancement
- Croissance
- Maturité
- Déclin
- Extension

Adoption d'une technologie

- Innovators : 2.5%
- Early adopters : 13.5%
- Early majority : 34%
- Late majority : 34%
- Laggards : 16%

Évolution d'un logiciel

- Développement initial
  - Processus de développement
    - Identification et analyse des besoins
    - Conception
    - Implémentation
    - Tests
    - Livraison
  - Décisions fondamentales
    - Technologie
      - Langage de programmation, conventions, librairies
    - Architecture
      - Composants, interactions
    - Connaissances du domaine du programme
      - Nécessaire à l'évolution
- Évolution
  - Adapte l'application aux changements
    - Nouvelles fonctionnalités
    - Nouvel environnement d'opération
    - Nouveaux utilisateurs
  - Correction de défauts et de malentendus
  - S'adapter aux nouveaux développeurs
  - Processus dédié à chaque type de changement
  - Le programme croît durant l'évolution

- Service
    - Programme n'évolue plus
        - Dégradation, stabilisation
    - Changements correctifs mineurs
        - Patch: rapide, moins dispendieux, mais cause la détérioration
    - Processus simplifié
        - Plus besoin d'ingénieurs seniors
        - Processus stable que l'on peut facilement mesurer et gérer
- Sortie graduelle
    - Logiciel n'est plus supporté, mais encore en production
    - Utilisateur doit vivre avec les défauts
- Retrait
    - Logiciel est discontinu
        - Durée de vie d'un logiciel réussi: 10 ~ 20 ans
    - Utilisateurs dirigés vers un remplacement
    - Stratégie de sortie
        - Rééducation pour le nouveau logiciel de remplacement
        - Que faire avec les données présentes

Modèles de développement

Développement de logiciel idéal

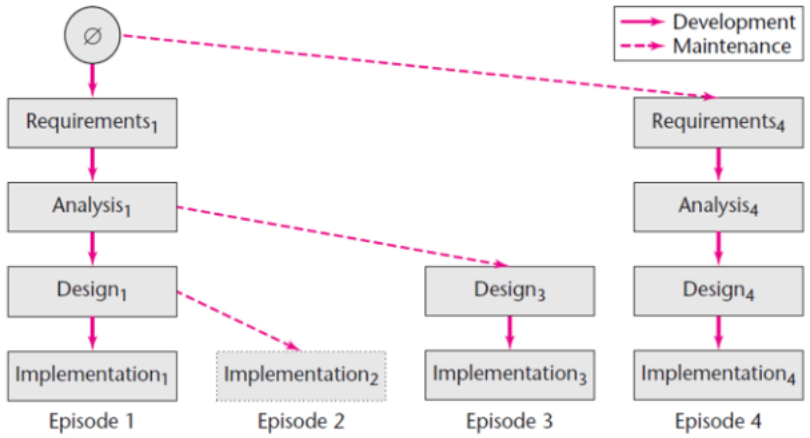
1. Requirements
2. Analysis
3. Design
4. Implementation
5. Delivery

- Linéaire
- Commence de rien
- Développement se termine à la livraison
- Correct du premier coup

Exemple de l'autobus

1. Première version implémentée
2. Faute trouvée

    a. Logiciel est trop lent à cause d'une erreur d'implémentation
    b. Changements dans le code débutent
3. Adoption d'un nouveau design
    a. Réutilisation d'un algorithme plus efficace
4. Les exigences changent
    a. Précision doit être augmentée
5. Épilogue : Quelques années plus tard, ces problèmes reviennent



Activités de développement

- Planification du projet
- Cueillette des exigences
- Analyse et spécification
- Conception
- Implémentation
- Vérification / Test
- Livraison / Déploiement
- Maintenance
- En continu :
    - Documentation
    - Vérification et validation
    - Gestion

Processus de développement de logiciels

- Description abstraite et idéalisée de l'organisation des activités du développement d'un logiciel
- Décrit un ensemble d'activités ordonnées

- Doit être «personnalisé» pour l'entreprise de façon à définir l'ordonnancement idéal des activités
    - Que doit-on produire ?
        - Types de documents, format, échéancier
    - Qui fait quoi ?
    - Comment superviser l'évolution du projet ?
        - Mesurer les résultats, prévoir plans futurs
    - Comment gérer les changements ?
        - Du processus ou du logiciel

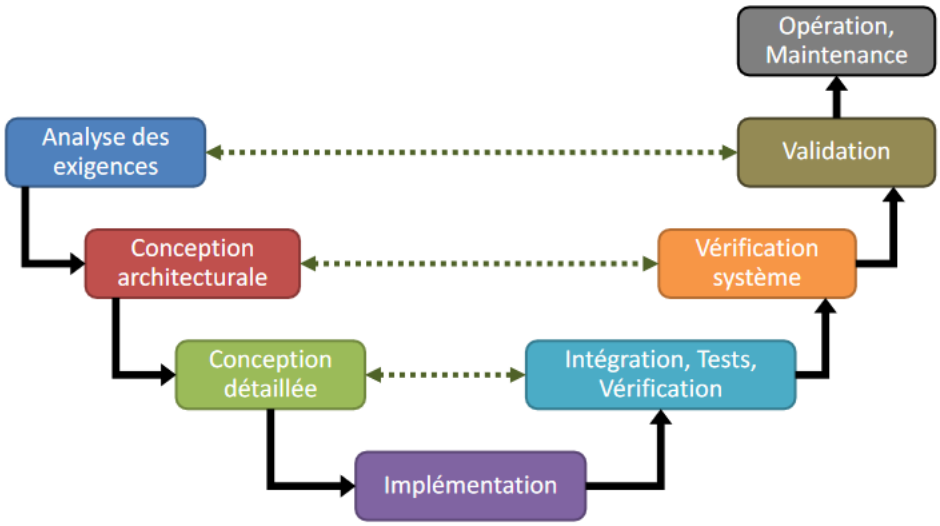
Modèle en cascade

- Exigences
- Analyse
- Conception
- Implémentation
- Vérification
- Déploiement
- Maintenance

Critique d'un processus en cascade

- Avantages
    - Simple et facile à suivre
    - Axé sur la documentation
    - Permet une conception bien pensée
- Inconvénients
    - Linéaire: chaque étape doit être complétée avant de passer à la suivante
    - Trop rigide: suppose que les exigences ne changent pas durant le développement
    - Pas de feedback du client avant la livraison
    - Vérification tardive
    - A contribué à la crise du logiciel

Modèle en V

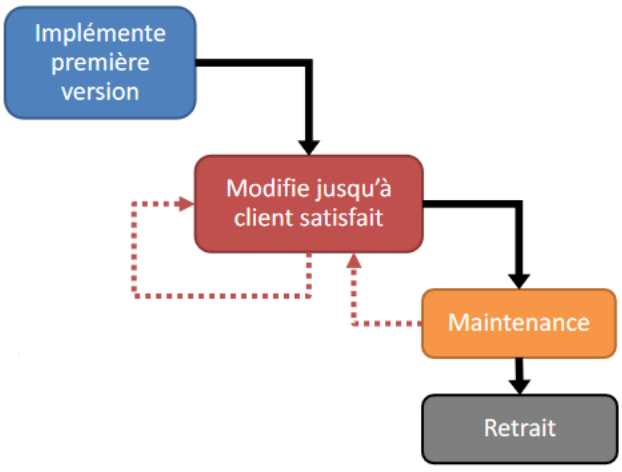


Développement itératif

- Le processus de développement logiciel est à la base itératif
- Chaque version a pour but de se rapprocher du système cible plus que la version précédente
- Architecte la coquille du produit complet, puis améliore chaque composant
- Intégration facilitée, moins de raffinement

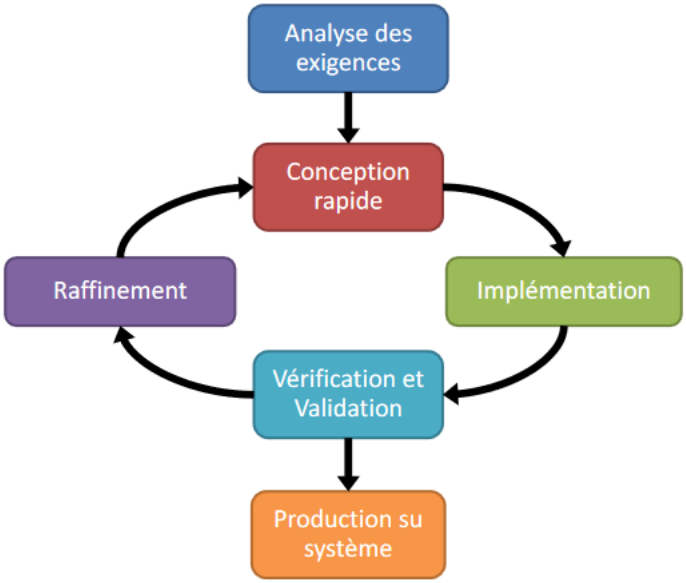
Modèle code-et-modifie

- Moyen le plus facile de développer un logiciel
- Mais aussi le plus cher sur le long terme
    - Pas de conception
    - Pas de spécification
        - Cauchemar pour la livraison
- Pertinent pour un petit logiciel interne utilisé par peu de personnes



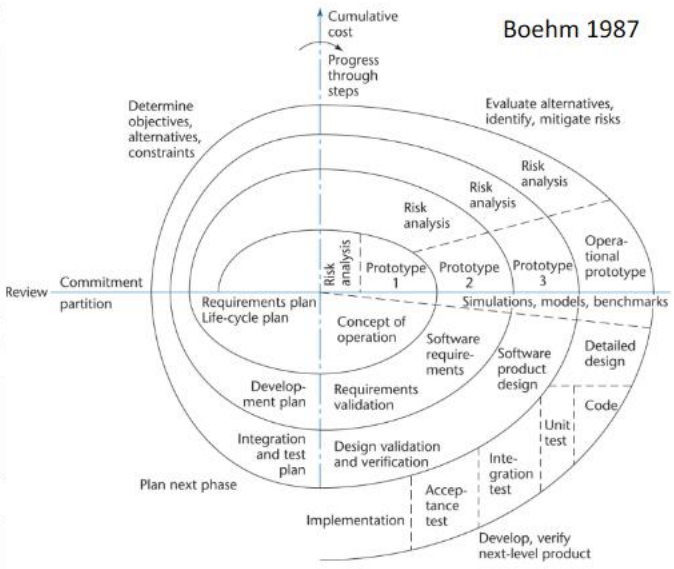
Processus par prototypage rapide

- Pertinent pour les projets où
    - Exigences pas clairement définies
    - Exigences susceptibles de changer durant le développement
- Prototype: programme implémenté rapidement
    - Jetable
        - Compréhension du client
        - Évaluation d'alternatives
    - Évolutif
        - réutilisé à chaque itération jusqu'au produit final



Processus en spirale

1. Déterminer les objectifs
2. Spécifier les contraintes
3. Produire des alternatives
4. Identifier les risques
5. Résoudre les risques
6. Développer et vérifier
7. Planifier prochain cycle



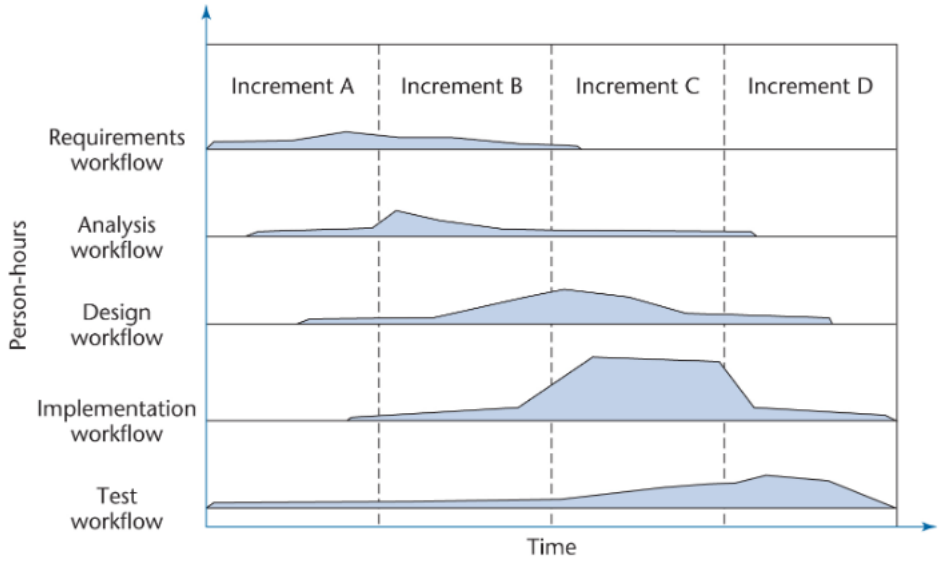
Critique d'un processus itératif

- Avantages
    - Développer en itérations
    - La réutilisation de prototypes
    - Produit visible dès le début
    - Souci de vérification et validation du client anticipé
- Inconvénients
    - Retravaille chaque itération
    - Pas de plan de maintenance
    - Produit livré à la fin

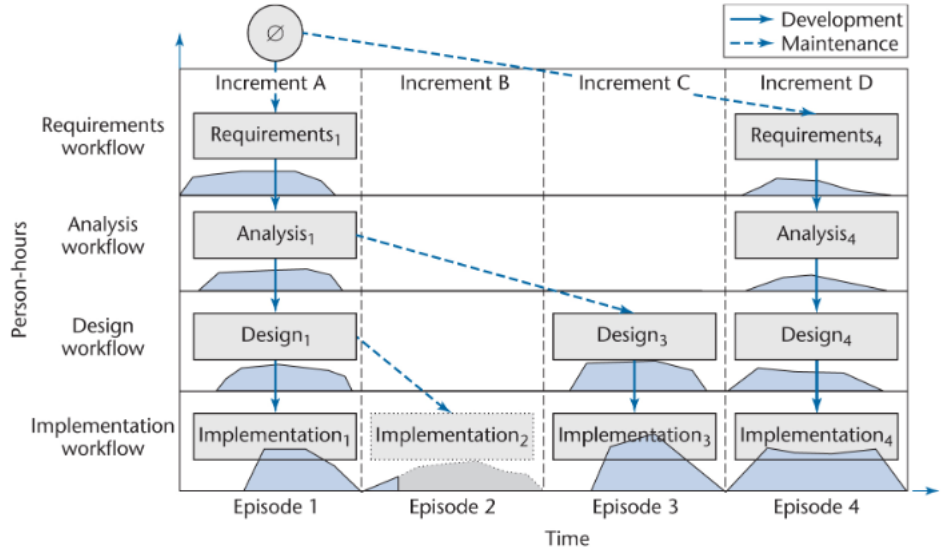
Développement incrémental

- Pour gérer de plus grandes informations, utiliser le raffinement par étapes (stepwise refinement)
    - Se concentrer sur les aspects les plus importants à ce moment
    - Laisser les moins critiques pour plus tard
    - Chaque aspect sera géré, dans l'ordre d'importance actuelle
- Chaque incrément abouti à une livraison
    - Voir une fonctionnalité complétée tôt dans le processus
    - Chaque incrément est le prototype du suivant
- Plus facile d'évaluer le progrès

Processus incrémentaux



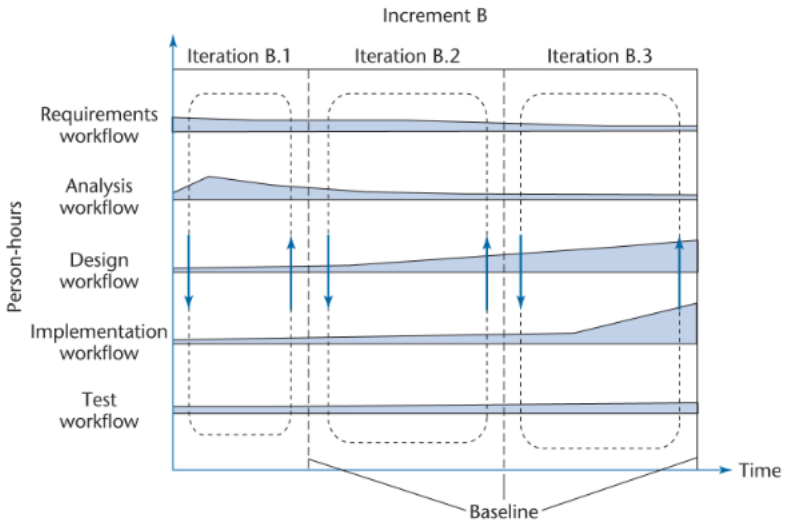
Exemple de l'autobus



Critique d'un processus incrémental

- Avantages
    - Développer par ordre de priorité
    - Livraisons de composants rapides
    - Facilement mesurer le progrès
- Inconvénients
    - Pas de processus visible et clair à suivre
    - Tâche d'intégration prend plus d'importance
    - Pas de plan de maintenance

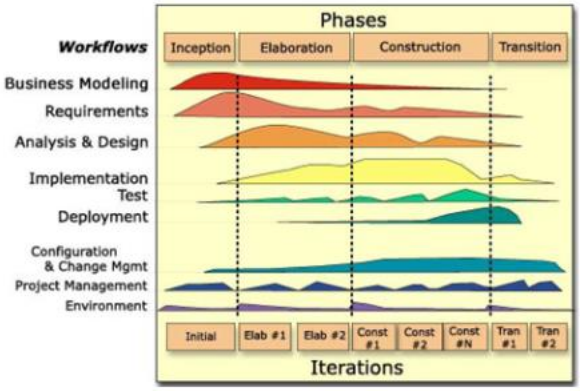
Itération et incrémentation



Avantages des processus I&I

- Tous les flux d'activités (workflow) sont impliqués dans chaque incrément, mais certains vont dominer plus
- Plusieurs opportunités de tester, recevoir du retour du client et s'ajuster
- Robustesse de l'architecture peut être déterminée tôt dans le développement
- Livrables spécifiques pour chaque incrément et chaque workflow
- On peut atténuer et résoudre les risques plus tôt
    - Il y a toujours des risques impliqués dans le développement et la maintenance d'un logiciel

Modèle du processus unifié



Processus agile

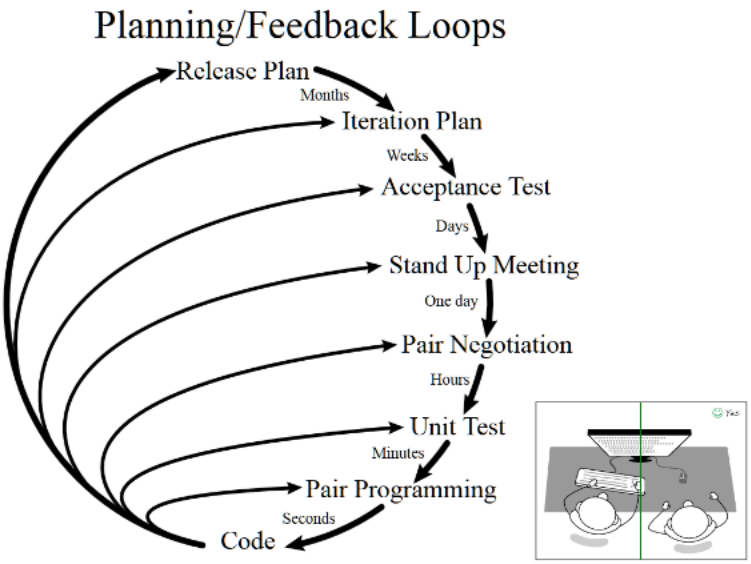
- Dirigé par la description des spécifications du client: scénarios
    - Client toujours impliqué durant le processus

- Reconnaît que les plans ne sont pas toujours respectés
    - Favorise la communication entre développeurs
    - Client fait partie de l'équipe
- Développe le logiciel itérativement avec plus d'emphase sur les activités de construction
    - Équipe de développement contrôle le travail à faire
- Livre plusieurs incréments du logiciel
- S'adapte rapidement quand un changement se produit
- Ces principes sont énumérés dans le manifeste agile par Kent Beck et al. 2001

Quelques méthodes agiles

- Extreme Programming
- Test-driven Development
- Scrum
- Crystal Methods
- Feature-Driven Development
- Lean Development
- Dynamic Systems Development Methodology
- Adaptive Software Development

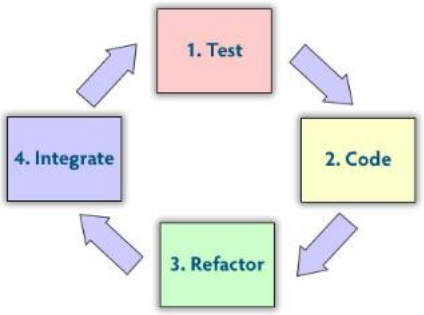
Programmation extrême (XP)



Développement dirigé par les tests

- Production de tests automatisés pour diriger la conception et la programmation

- Test utilisé comme spécification
- Processus en petites étapes



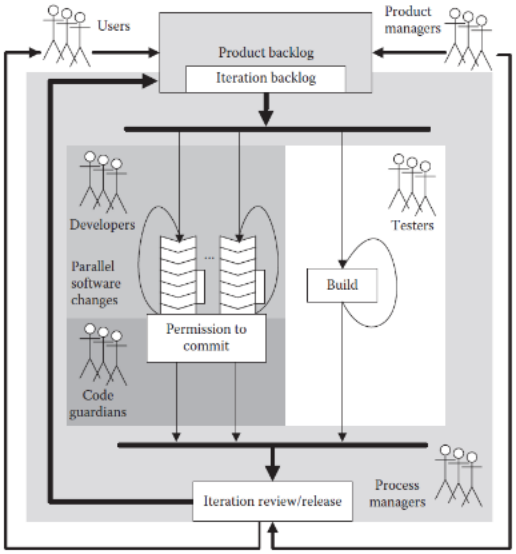
Modèle Scrum

Équipe Scrum

- Propriétaire du produit
    - Représentant des clients et utilisateurs
- Scrum master
    - Leader au service de l'équipe
- Équipe de développement
    - 7 ±2 personnes
    - Pluridisciplinaire pour avoir toutes les compétences nécessaires
    - Une fois l'engagement d'un sprint pris, elle a pleine autorité
        - Travail à faire, heures d'ouvrage, responsabilités
- Environnement de travail

- Salle ouverte, dégagée
- Murs couverts de tableaux
- Réseau sans-fil
- Meubles mobiles

Modèle agile en pratique



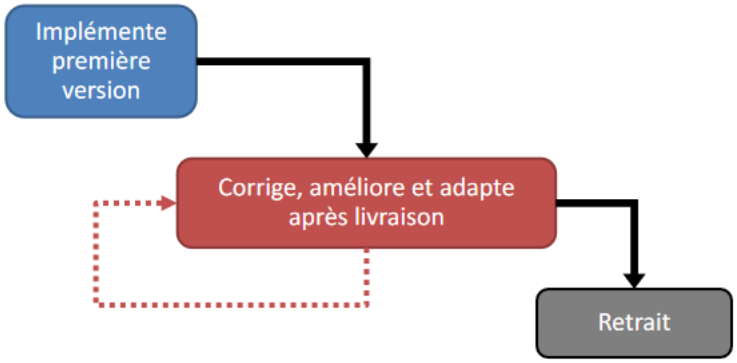
Modèle des logiciels libres (open source)

- Deux phases formelles

1. Un individu développe la version initiale
   a. Déploie et rend disponible sur internet, ex: SourceForge, GitHub
2. S'il y a assez d'intérêt dans le projet
   a. Version initiale téléchargée abondamment
   b. Utilisateurs deviennent des co-développeurs
   c. Logiciel évolue

- Les individus travaillent bénévolement dans leurs temps libres
- La 2^{e} phase est de la maintenance après livraison seulement

Processus des logiciels libres



Groupes d'utilisateurs des logiciels libres

- Groupe principal
    - Petit groupe de soutien qui a la volonté, le temps et les habiletés nécessaires de soumettre résoudre les rapports d'erreur
    - Prennent la responsabilité de gérer le projet
    - Ont l'autorité d'installer les correctifs
- Groupe périphérique
    - Utilisateurs qui soumettent des rapports de défaut de temps en temps

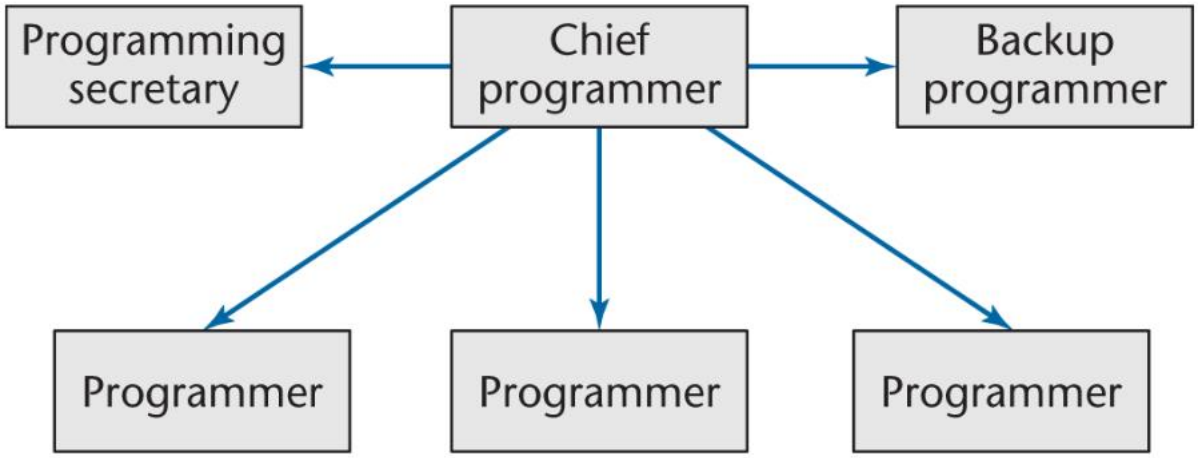
Équipe de développement

Approche démocratique

- Programmeurs sont très attachés à leur code
    - Nomment leurs modules d'après eux
    - Ils voient leurs modules comme une extension d'eux-mêmes
- Concept de base: programmation non-égocentrique
    - Encourage les membres de l'équipe à trouver des fautes dans le code
    - Une faute doit être considérée comme un événement normal et accepté
- Équipes démocratiques sont extrêmement productives
- Meilleures quand le problème est difficile
- Fonctionnent bien dans un environnement de recherche
- Difficultés pour le gestionnaire
    - Équipe démocratique difficile à introduire dans un environnement qui ne l'est pas

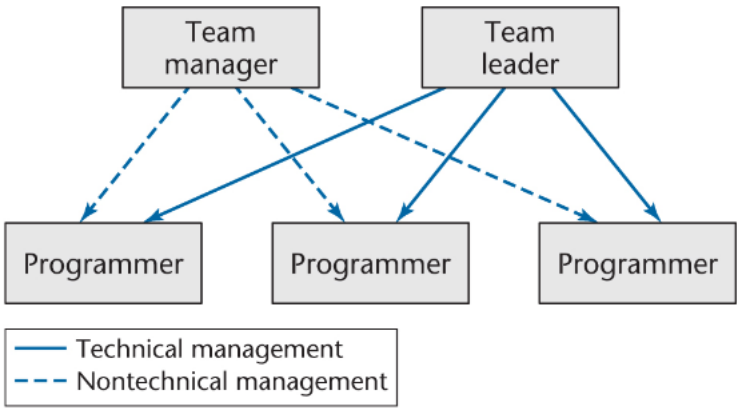
Équipe de programmeur chef classique : irréaliste

- Programmeur chef doit être un programmeur hors norme ET un excellent gestionnaire
    - Très rare de trouver l'un ou l'autre
    - Qualités d'un excellent programmeur rarement présente chez un excellent gestionnaire, et vice-versa
- Programmeur en réserve doit être aussi bon que le chef et en connaitre autant sur le projet
    - Mais doit attendre dans l'ombre que quelque chose arrive au chef
    - Jamais un programmeur ou gestionnaire top ne ferait ça
- Programmeur secrétaire doit s'occuper de toute la documentation du projet, les programmeurs ne font que programmer
    - Tout ingénieur logiciel déteste la paperasse!



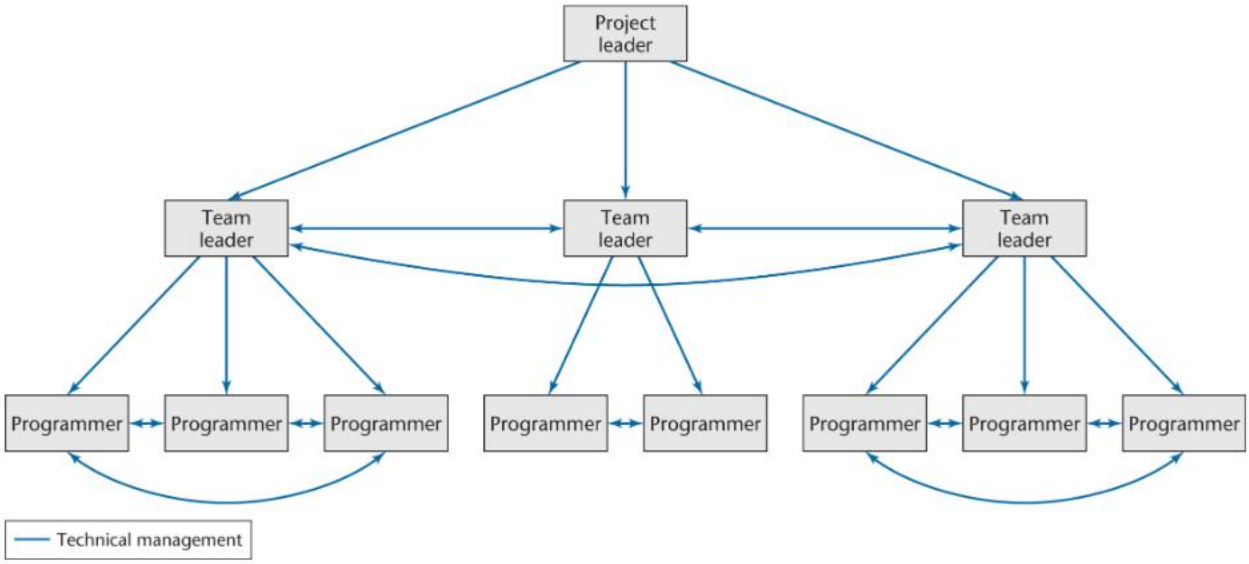
Séparation des pouvoirs

- Organisation qui combine les modèles démocratique et programmeur chef
    - Attitude positive
    - Réduit le rôle de gestion du programmeur chef



Pour les projets plus grands : Décentralisation des prises de décision

- La partie non-technique reste identique
- Ajouter plus de couche pour des projets encore plus grands



Équipe agile Scrum

- Propriétaire du produit
  - Représentant des client et utilisateurs
  - Seul qui dirige l'équipe de développement
  - Responsable de création et maintien du carnet du produit
- Scrum master
  - Leader au service de l'équipe
    - Entraineur, facilitateur, très expérimenté
  - S'assure du respect du processus
- Équipe de développement
  - 7 ±2 personnes
  - Pluridisciplinaire pour avoir toutes les compétences nécessaires
  - Une fois l'engagement d'un sprint pris, elle a pleine autorité
    - Travail à faire, heures d'ouvrage, responsabilités, propriété du code

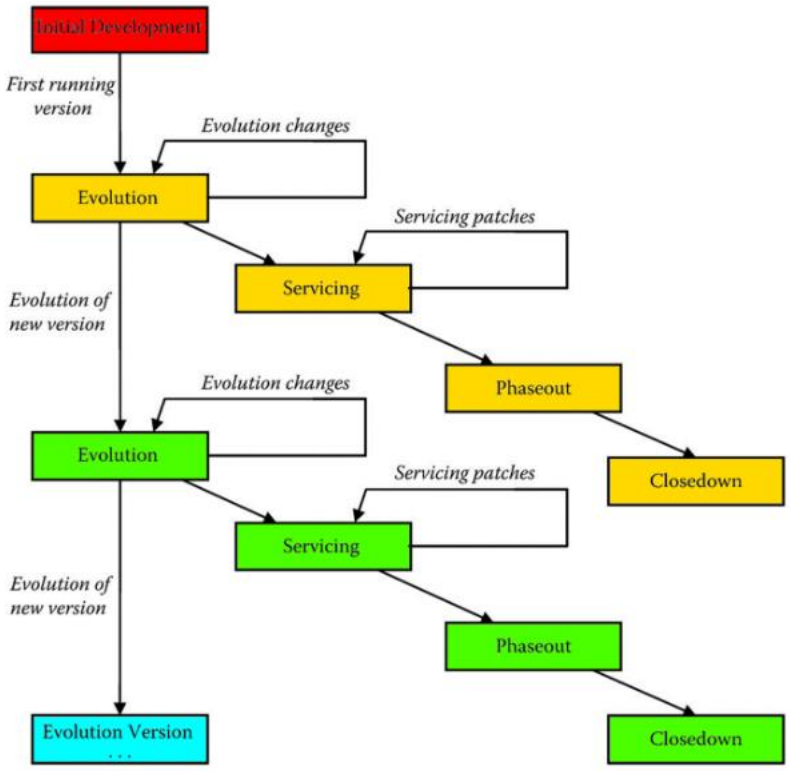
Contrôle de versions du logiciel

Versions du logiciel

- Plus d'un programmeur dans le projet ⇒ collaboration

    - Travailler en parallèle
    - Code de chacun doit être consistant avec le reste
- Besoin de sauvegarde sécuritaire du code source (back-up)
- Besoin de garder une trace des changements dans le code source
    - Besoin de revenir à une version antérieure de mon propre code
- Développement incrémental et itératif: plusieurs versions du code
    - Ex: Tests faits sur itération N, pendant que programmeur travail sur itération N+1
    - Documentation, artéfacts de conception, spécifications, etc.

Évolution est un pilier du logiciel



Système de contrôle de révisions

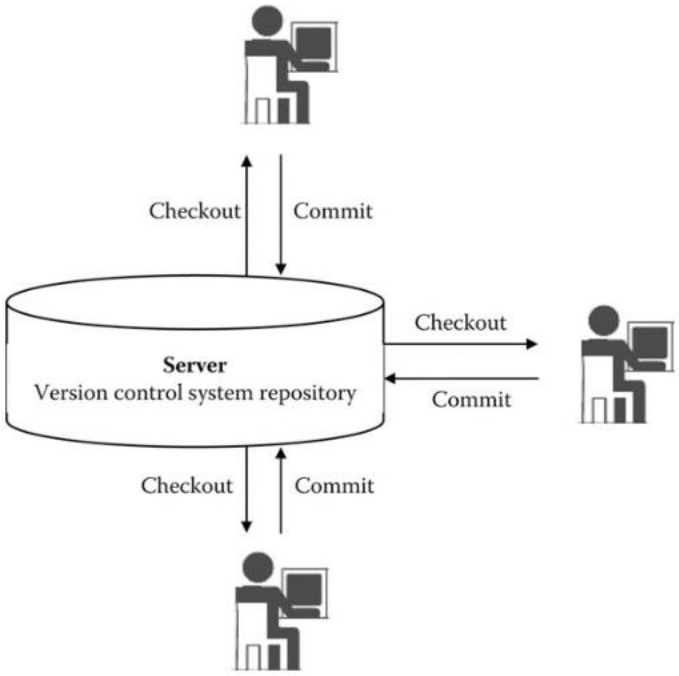
- Gérer les changements apportés à des fichiers
- Conserver une trace
    - Quels sont les changements?
    - Qui a fait quel changement?
    - Voir l'arbre d'évolution complet
- Collaborer entre individus

    - Partager le travail dans une hiérarchie de fichiers
    - Synchronisation garantie
- Protéger
    - Contre la perte de données, documents et code source
    - Ignorer un changement et revenir à une version antérieure ou ultérieure

Contrôler les versions de quoi?

- Fichier texte (source?)
    - Version de changement ligne par ligne
        - Conserve uniquement les changements
    - Visualisation des différences entre versions
- Fichier binaire (compilé?)
    - Granularité au niveau du fichier
    - Chaque version conserve le fichier au complet

Système de contrôle de révisions centralisé



Dépôt (Repository)

- Conserve tout le contenu associé à un projet
    - Sous forme de base de donnée
- Contient tout l'historique du projet

    - Contenu initial et modifié
    - Auteur des modifications
    - Date/heure d'une modification
- Dépôt centralisé
    - Le dépôt est sur le serveur en forme canonique
- Dépôt décentralisé
    - Chaque client a sa propre copie du dépôt provenant du serveur

Copie de travail (working copy)

- Programmeur qui veut apporter des modifications doit obtenir une copie de travail
    - Opération check out ou clone du dépôt pour la première fois
    - Opération update ou pull du dépôt pour avoir les nouveaux changements
- Programmeur apporte des modifications sur sa copie de travail, pas directement sur le dépôt
- Protège le contenu du dépôt jusqu'à créer une nouvelle révision

Suivi des changements (track change)

- Système fait le suivi des changements apportés à une copie de travail en arrière-plan
    - Possible à tout moment d'annuler les modifications apportées sur la copie de travail d'un ou plusieurs fichiers
- Changements suivis
    - Ajout, suppression, modification, déplacement, renommage un fichier ou dossier, etc.
- Processus transparent jusqu'à ce que les changements soient prêts à former la prochaine version

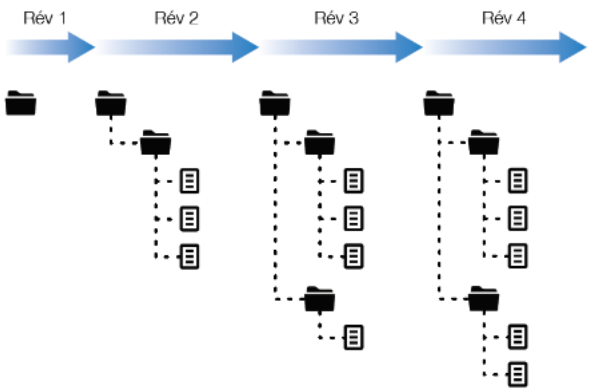
Soumission (commit)

- Lorsque les modifications sont complétées et enregistrés en local sur la copie de travail, le client soumet ses changements au dépôt
- Soumission d'un ou plusieurs changements dans une seule transaction atomique
- Une soumission provoque la création d'une nouvelle révision pour l'ensemble des changements
    - Calcule les différences entre la version de chaque fichier dans la copie locale avec celle du dépôt
    - Applique les changements à la nouvelle révision du dépôt
- La nouvelle révision devient la révision courante HEAD

Création de révisions

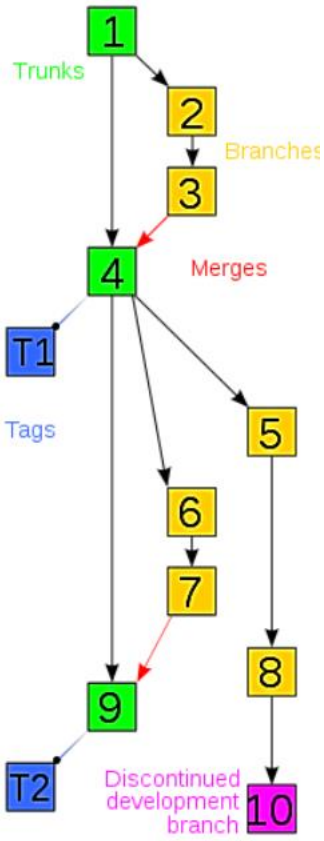
- Une nouvelle révision obtient un identifiant unique qui permet d'y accéder et d'y revenir dans le futur

    - Numéro séquentiel (1, 2, 3, ...) ou code de hachage (5d368c0635de...)
- Majorité des systèmes supportent l'étiquetage des révisions
    - Nom plus parlant
- Lors de la soumission, le client peut ajouter un message descriptif du changement



Troncs, branches et tags

- Tronc (trunk)
    - Branche principale du contenu
    - Dernière version en cours de développement
- Branches
    - Copies du tronc pour une révision expérimentale qui risque de briser le tronc
    - Changements peuvent y être soumis pendant que d'autres changements sont soumis au tronc
    - Version en développement encore instable
- Tags
    - Branches en lecture seule, images du projet à un moment donné
    - Version stable d'un livrable
    - Autre mécanisme d'étiquetage

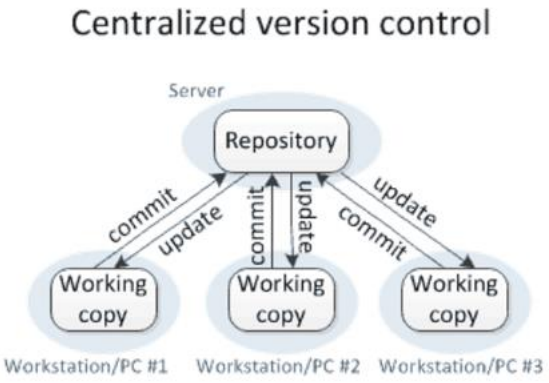


Fusion de fichiers (merge)

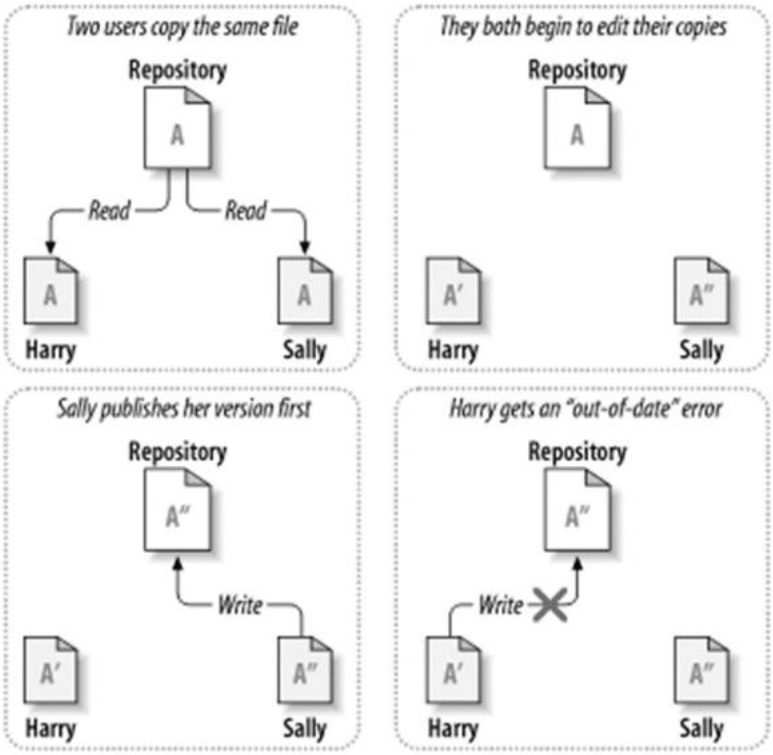
- Système connait les changements apportés à chacun des fichiers depuis leur dernier ancêtre commun
- Fusion automatique quand il n'y a pas de conflits
- En cas de conflit, un humain doit décider de la fusion manuellement
    - Aidé par une vue des différences pour faciliter la fusion manuelle

Mise à jour

- Des changements importants peuvent avoir été apportés au tronc pendant qu'un développeur travaille sur sa branche
- Pour importer les derniers changements, on fait une mise à jour
    - Faites souvent une mise à jour pour éviter des conflits lors de la fusion
- La mise à jour copie les fichiers du dépôt en local
    - Programmeurs changent la copie locale
    - Protège les données dans le dépôt jusqu'à la prochaine fusion
- Checkout/clone: première fois
- Update/pull: toutes les autres fois

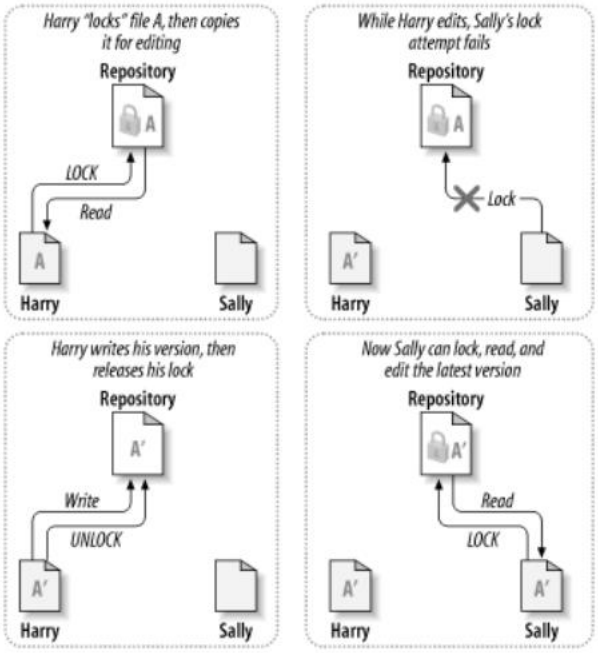


Scénario de collaboration



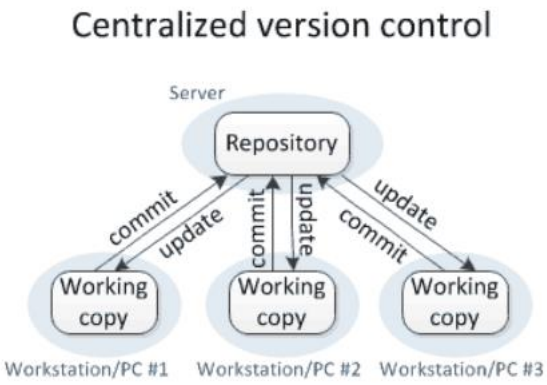
Solution : verrouiller (lock)

- Un fichier verrouillé ne peut être modifié que par un client à la fois
- Les autres doivent attendre qu'il soit déverrouillé
- Soumettre un fichier verrouillé le déverrouille automatiquement
- Limitations
    - Changements concurrents sur des fichiers différents seulement
    - Oubli de déverrouiller un fichier
    - Deux clients verrouillent deux fichiers interdépendants

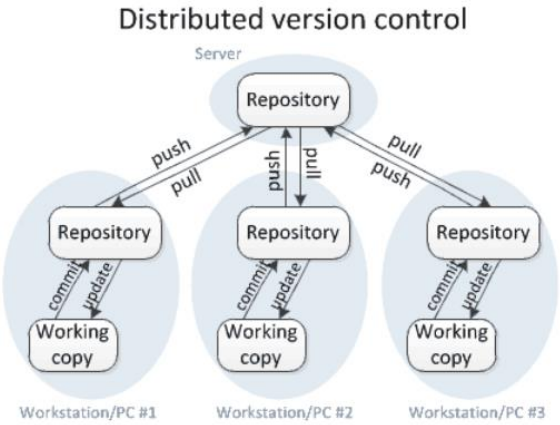


Systèmes de gestion de révisions

- Centralisé
    - SVN, CVS



- Décentralisé
    - GIT, Mercurial



Subversion (SVN)

- Système centralisé le plus populaire, depuis 2000
- Support natif pour les fichiers binaires
- Modèle simple, facile à apprendre et à utiliser
- Doit être connecté au serveur de dépôt pour soumettre
- Opérations peuvent être lentes
- Branches et fusions difficiles à utiliser
- Outils par ligne de commande ou graphique (TortoiseSVN)

GIT

- Système décentralisé le plus populaire, depuis 2005
- Serveur optionnel
- Opérations rapides car locales
- Branches efficaces utilisées fréquemment
- Modèle distribué plus complexe
- Beaucoup de fusion, donc plus de possibilités de conflits
- Pas d'outil graphique, seulement par ligne de commande