IFT3912 – Développement et maintenance de logiciels

Évolution et maintenance

Bruno Dufour
Université de Montréal
dufour@iro.umontreal.ca

Modification des logiciels

- Les modifications sont inévitables
  - Des nouveaux besoins apparaissent lors de l'utilisation
  - L'environnement d'affaires change
  - Les erreurs doivent être réparées
  - Du nouvel équipement est ajouté au système
  - La performance ou la fiabilité du systèmes peuvent avoir à être améliorées
- Ces changements aux systèmes existants représentent un problème critique pour les organisations

2

Maintenance

3

Maintenance

- Définition : modification d'un logiciel après sa mise en service / déploiement
- Le terme « maintenance » est généralement utilisé pour les logiciels personnalisés. Les autres logiciels évoluent simplement de version en version
- La maintenance comporte rarement des changements majeurs à l'architecture
- Les changements apportés modifient ou ajoutent des composants

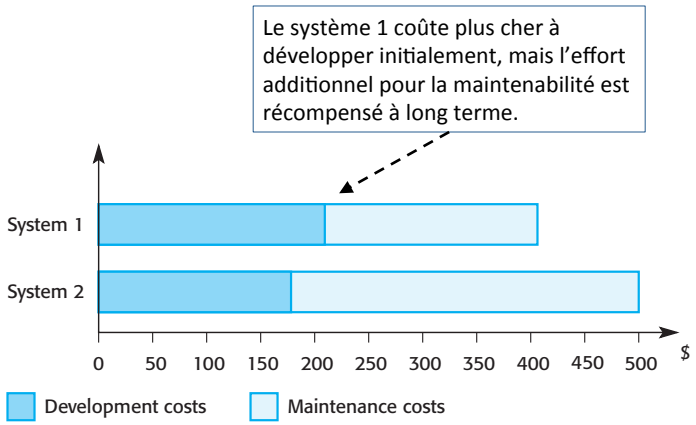
4

Types de maintenance

- Correction de fautes
  - Modifications qui visent à corriger les problèmes pour que le système rencontre sa spécification
  - = maintenance corrective
- Adaptation à un nouvel environnement
  - Modifications qui visent à supporter un environnement (matériel, OS, etc.) différent de celui utilisé lors de l'implémentation initiale.
  - = maintenance adaptive
- Ajout ou modifications de fonctionnalité
  - Modifications qui visent de nouveaux besoins au système.
  - = maintenance perfective
- Amélioration de la qualité du logiciel
  - Modifications qui visent à rendre les modifications ultérieurs plus faciles
  - = maintenance préventive

5

Coûts de développement et maintenance



Source: I. Sommerville, "Software engineering", Addison-Wesley 2011

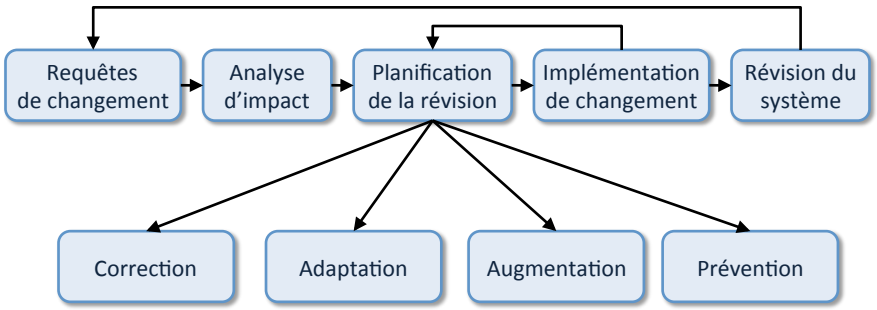
6

Facteurs de coût

- Plusieurs facteurs peuvent affecter le coût de la maintenance:
  - Stabilité de l'équipe : les coûts sont moindres lorsque les membres de l'équipe participent à la maintenance sur une longue période
  - Obligations contractuelles : les développeurs d'un système qui ne sont pas tenus d'en faire la maintenance ont peu de motivation à faciliter les changements dans leur conception.
  - Compétence des programmeurs : la maintenance est souvent assignés aux développeurs inexpérimentés.
  - Age et structure du programme : la structure d'un programme se dégrade avec l'âge, ce qui rend les modifications plus difficile.

7

Processus de maintenance



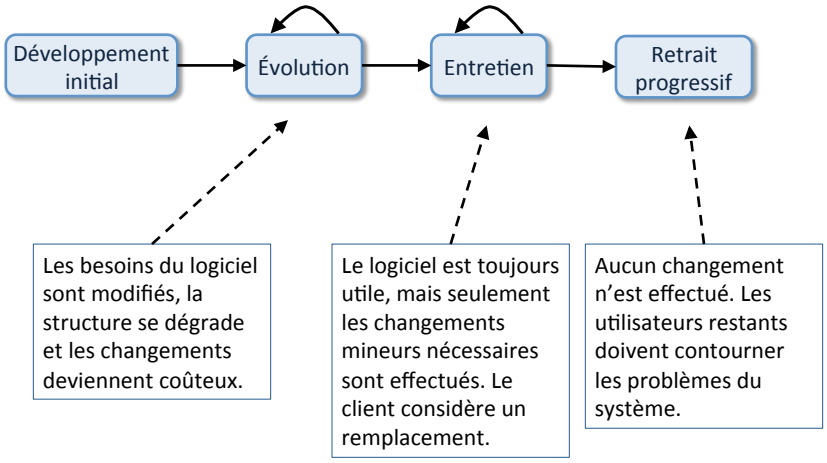
8

Évolution

Importance de l'évolution

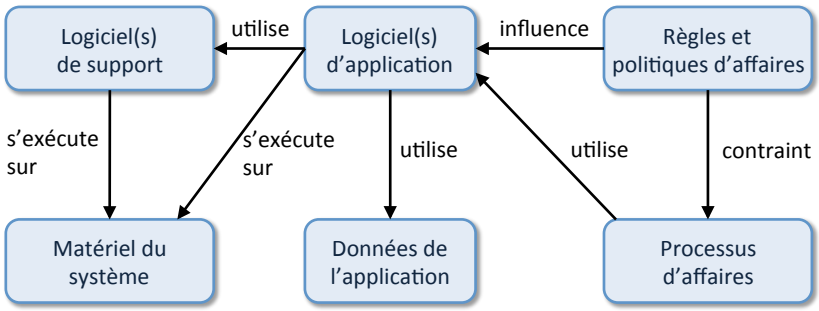
- Les entreprises investissent de larges sommes dans la création de logiciels
  - Les logiciels doivent donc servir plusieurs années par souci de rentabilité
  - Les logiciels d'affaires sont ont souvent plus de 10-20 ans, d'autres logiciels (ex: contrôle aérien) ont des durées de vie de plus de 30 ans
- Plus d'argent est dépensé par les grandes compagnies pour la maintenance et l'évolution que pour le développement de nouveaux systèmes.
- Plusieurs scénarios
  - La même compagnie développe toutes les versions
  - Une compagnie développe le logiciel initial, le client se charge de la maintenance / évolution
  - Une compagnie développe le logiciel initial, une autre s'occupe de la maintenance / évolution

Évolution et entretien



11

Systèmes hérités



12

Coûts des systèmes hérités

- Les systèmes hérités deviennent plus coûteux à maintenir avec l'âge
  - Pas de style de programmation cohérent
  - Plusieurs contributeurs, mais personne ne comprend le système en entier
  - Utilisation de langages de programmation désuets
  - Dépendance sur de la technologie ancienne
  - Documentation inadéquate ou dépassée
  - Structure corrompue par plusieurs années de maintenance *ad hoc*
  - Optimisation pour l'espace / temps d'exécution et non pas la compréhension du code
  - Les données peuvent être séparées en plusieurs fichiers qui ont des structure incohérentes. Les données elle-même peuvent aussi être incohérentes, dupliquées, désuètes, etc.

13

Remplacement des systèmes hérités

- Replacer un système hérité pose des risques :
  - Il n'existe généralement pas de spécification du système en entier sur laquelle se baser pour développer un nouveau système
  - Le processus d'affaire est souvent étroitement relié aux logiciels utilisés
  - Les logiciels peuvent contenir des règles d'affaires non documentées
  - Développer un nouveau système comporte des risques
    - Retards, coûts plus élevés que prévu, etc.

14

Le dilemme des systèmes hérités

- Continuer à utiliser le système accroît invariablement les coûts
- Remplacer le système coûte cher, et le nouveau système peut être moins efficace que l'ancien

- ∴ Plusieurs entreprises cherchent à allonger la durée de vie de leurs logiciels
  - Rétroingénierie
  - Réingénierie

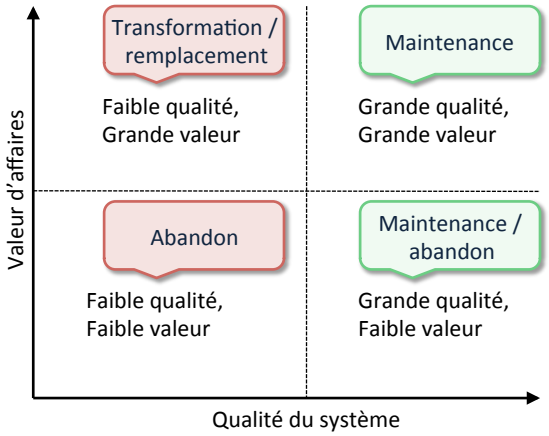
15

Évaluation des systèmes hérités

- 4 options possibles:
  - Abandonner le système complètement
  - Continuer à maintenir le système
  - Transformer le système pour faciliter la maintenance
  - Remplacer le système hérité par un nouveau système
- Les systèmes complexes peuvent utiliser une combinaison de plusieurs options

16

Évaluation des systèmes hérités



17

Évaluation de la valeur d'affaires

L'évaluation devrait prendre en compte plusieurs opinions :

- **Utilisateurs**: Quelle est l'efficacité du système à supporter le processus d'affaires? Quelle portion de la fonctionnalité est utilisée?
- **Clients**: Le système est-il transparent pour les utilisateurs? Le système cause-t-il de l'attente? Les erreurs ont-elles un impact direct sur les clients?
- **Gestionnaires financiers**: Le système contribue-t-il au succès de leur unité? Les coûts d'opération du système sont-t-ils justifiés? Les données manipulées par le système sont-elles critiques au fonctionnement de leur unité?

18

Évaluation de la valeur d'affaires

- **Gestionnaires techniques**: Est-il difficile de trouver du personnel pour travailler sur le système? Le système consomme-t-il des ressources qui pourraient déployées plus efficacement sur d'autres systèmes?
- **Cadres supérieurs**: Quelle est la contribution du système et de ses processus d'affaires aux objectifs de l'entreprise?

19

Autres considérations

- Utilisation du système
  - Si un système est utilisé peu fréquemment ou par peu d'utilisateurs, sa valeur d'affaires est probablement faible.
- Processus supporté
  - Si un système force l'utilisation d'un processus inefficace, sa valeur d'affaires est probablement faible.
- Fiabilité
  - Si un système n'est pas fiable, et que les problèmes affectent les clients directement, ce système à une faible valeur d'affaires.
- Sorties
  - Si une organisation dépend des sorties d'un système, ce système a une valeur d'affaires élevée.

20

Évaluation de la qualité du système

- Évaluation du processus d'affaires
  - Un mauvais processus mène au changement
  - Y a-t-il différents processus pour la même fonction?
  - Le processus a-t-il été adapté pour faciliter le travail?
  - Quels sont les relations avec les autres processus, et sont-elles nécessaires?
  - Le processus est-il supporté efficacement par le système hérité?
- Évaluation de l'environnement
  - Nombre de fautes matérielles pour une période donnée, âge du matériel, performance, interopérabilité, stabilité des fournisseurs, etc.
- Évaluation du logiciel d'application
  - Nombre de requêtes de changement, nombre d'interfaces utilisées, volumes de données utilisées, etc.

21

Évaluation de la qualité du système

- Évaluation de l'environnement
  - Stabilité des fournisseurs : Existe-t-il toujours? Est-il stable? Une autre compagnie assure-t-elle la maintenance?
  - Nombre de fautes matérielles pour une période donnée
  - Âge du matériel et des logiciels : il peut être avantageux de remplacer le matériel trop vieux mais s'il fonctionne toujours
  - Interopérabilité : Y a-t-il des problèmes à interfacer avec d'autres systèmes? L'émulation du matérielle est-elle nécessaire?
  - Support : Quels sont les coûts du support local ?
  - Maintenance : Quels sont les coûts de maintenance du matériel et de support des licences?
- Évaluation du logiciel d'application
  - Nombre de requêtes de changement, nombre d'interfaces utilisées, volumes de données utilisées, etc.

22

Évaluation de la qualité du système

- Évaluation du logiciel d'application
  - Compréhension : Est-il difficile de lire le code source? Quelle est la complexité des structures de contrôle ? Les variables sont elles nommées d'une façon claire et qui reflète leur utilisation ?
  - Documentation : La documentation est-elle disponible, complète, cohérente et à jour ?
  - Données : Existe-t-il un modèle de données pour le système ? Combien de duplication de données y a-t-il ?
  - Performance : Est-elle adéquate ? Une faible performance affecte-t-elle les utilisateurs ?
  - Langage de programmation : Le langage utilisé est-il moderne et activement utilisé pour le développement ?
  - Personnel : Le personnel possède-t-il les qualifications et/ou l'expérience requises pour modifier le système ?
  - Tests : Des données de tests existent-elles pour le système ?

23

Rétroingénierie

26

Rétroingénierie

- Objectif: produire une spécification manquante à partir du code source et / ou des données d'un système
  - Alternative: à partir d'un ensemble d'exécutions du système
- Souvent utilisée pour aider un ingénieur à comprendre un système avant la réingénierie
- Les spécifications peuvent aussi être utilisées pour le développement d'un système de remplacement ou pour faciliter la maintenance
- Démarche souvent automatisée
  - Des annotations manuelles peuvent améliorer les résultats

27

Rétroingénierie du code

- Exemples de techniques de rétroingénierie du code :
  - Décomposition en sous-systèmes
  - Reconstruction de l'architecture
  - Analyse des dépendances dans le code (statiques) et à l'exécution (dynamiques)
  - Exploration et visualisation du code
- Problème : le code ne contient pas toute l'information
  - Les compromis de conception, les contraintes techniques et les concepts du domaine d'application n'existent souvent que dans l'esprit des développeurs
  - Ces concepts disparaissent avec le temps

28

Rétroingénierie des données

- Tente de déterminer quelle information est disponible et comment cette information peut être utilisée dans un contexte différent.
- Activités principales :
  - Analyse : permet de produire un modèle précis et complet des données
  - Abstraction : vise de passer du modèle logique obtenu par analyse à un modèle de conception (ex: modèle orienté-objet)
- Problème : connaissances fragmentaires
  - Les utilisateurs, la documentation existante et le code peuvent ensemble contribuer suffisamment de connaissances pour la rétroingénierie
  - L'analyse des données d'une base de données est plus simple que lorsque des formats de fichiers *ad hoc* sont utilisés.

29

Réingénierie

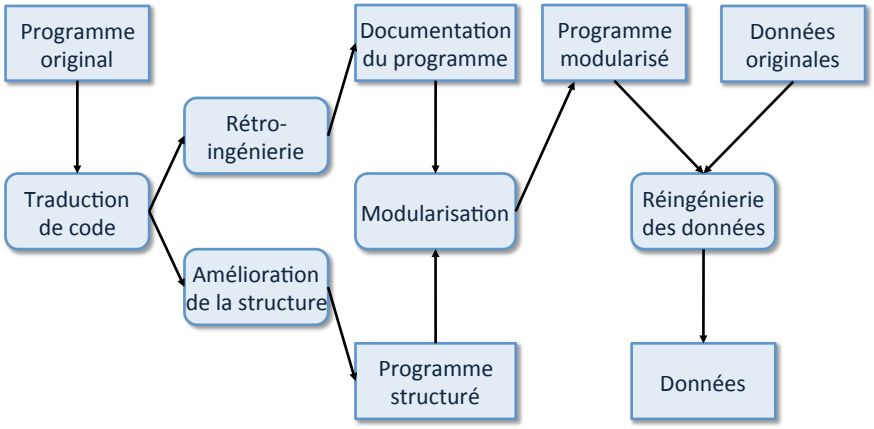
30

Réingénierie

- Avantages:
  - Réduit les risques: redévelopper un système critique introduit beaucoup de nouveaux risques
  - Réduit les coûts: redévelopper un nouveau système coûte plus cher que la modification d'un système existant
- La réingénierie est souvent appliquée aux systèmes hérités, mais les concepts et outils sont de plus en plus populaires pour le développement de systèmes modernes

31

Démarche de réingénierie



32

Traduction de code

- Traduction (semi-)automatique d'un langage de programmation à un autre
- Causes principales:
  - Mise à jour du matériel
  - Manque d'expertise du personnel
  - Changement de l'entreprise
  - Manque de support (ex: logiciels)

33

Amélioration de la structure

- Objectif: simplification de la logique du code pour faciliter la lecture et la compréhension
- Les structures complexes sont souvent le résultat d'activités de maintenance et d'optimisations de l'utilisation des ressources
  - Ex: goto, conditions complexes
  - `if not (A > B and (C < D or not (E > F)))`
    devient
  - `if (A <= B) or (C >= D and E > F)`
- Peut être appliquée sélectivement à différents parties du programme

34

Amélioration de la structure

- Inconvénients
  - Perte de commentaires
  - Perte de documentation
  - Calculs complexes
  - Pour les systèmes centrés sur les données, une modularisation peut être nécessaire pour facilité la lecture du code

35

Modularisation

- Objectif: réorganisation du code de sorte que les parties qui sont logiquement reliées soient groupées en un seul module
- Différents types de modules peuvent être créés:
  - Abstractions de données
  - Pilotes de matériel
  - Modules fonctionnels (ex: validation des entrées)
  - Modules de support du processus d'affaires
- Souvent une démarche (semi-)manuelle
  - Les outils peuvent aider à la transformation, mais l'ingénieur doit identifier les modules

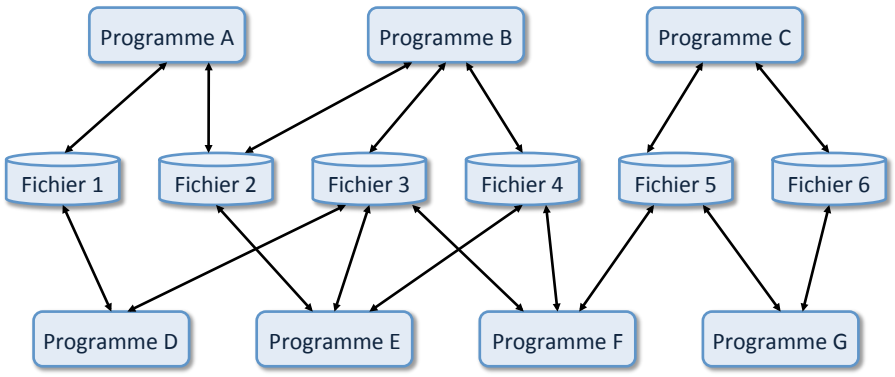
36

Réingénierie des données

- Objectif: analyse et réorganisation des structures de données (ou des parfois valeurs) utilisées par un système
- Peut être nécessaire même lorsque la fonctionnalité d'un système est inchangée:
  - Dégradation des données
  - Limites imposées par le système
  - Évolution de l'architecture
- Démarche presque toujours semi-manuelle

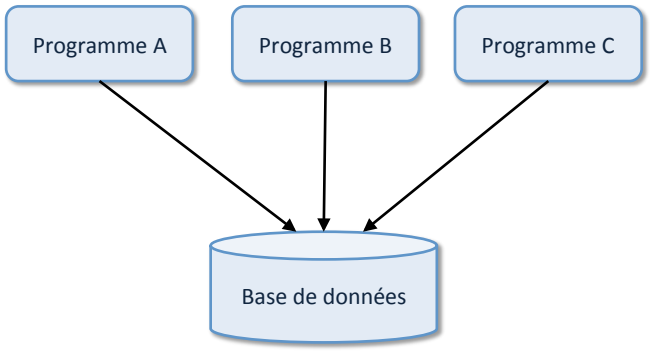
37

Données en fichiers partagés



38

Données centralisées



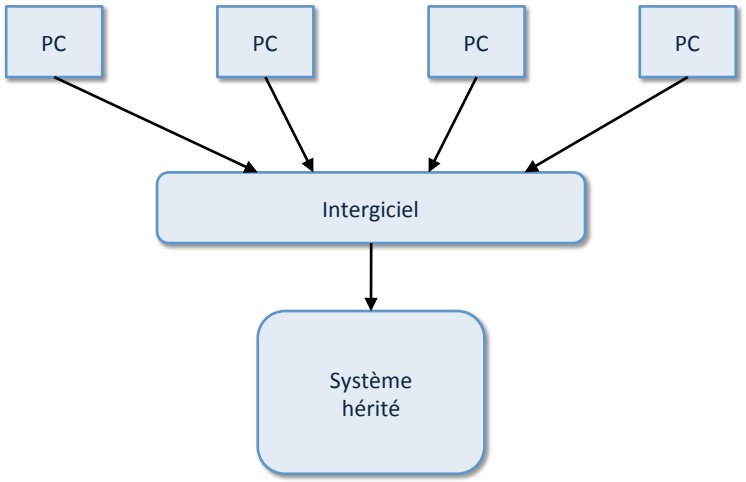
39

Évolution de l'architecture

- Les architectures client-serveur sont plus rentables que les architectures centralisées d'autrefois
- Plusieurs entreprises doivent donc faire évoluer leurs systèmes en raison de plusieurs facteurs:
  - Coût du matériel
  - Attentes de l'interface
  - Accès distribué au système
- La structure existante du système se prête rarement à l'évolution
  - Un couche d'intergiciel peut être nécessaire pour traduire

40

Distribution d'un système hérité



41

Distribution de l'interface

- Les systèmes à interface textuelles peuvent s'exécuter en utilisant un émulateur de terminal
- Les services de l'interface graphique (présentation, contrôle des intéractions, validation) peuvent être migrés vers les PCs des clients
  - Le serveur contient les données et l'application
  - Les interfaces modernes communiquent avec le serveur de la même façon que l'interface du terminal
  - L'interface peut être native ou celle d'un fureteur

42