Notions de base de la maintenance de logiciels

G. Tremblay

Été 2004

1 Introduction : Qu'est-ce que la maintenance?

- Définition de l'IEEE :

  > *Modification of a software product after delivery, to correct faults, to improve performance or other attributes, or to adapt the product to a modified environment.*

- La maintenance d'un système consiste donc à modifier ce système *après* qu'il ait été livré et installé, après qu'il ait été mis en opération.

Pourquoi la maintenance est inévitable

De nombreuses raisons font que de la maintenance doit être effectuée :

- Pour corriger des erreurs.
- Pour interfacer le système avec d'autres systèmes.
- Pour faire des améliorations pour mieux supporter les besoins des usagers.
- Pour adapter le système face à des changements divers (par exemple, changements aux règles gouvernementales, concurrence des compétiteurs).
- Pour faire des changements dans les fichiers ou bases de données.
- Pour améliorer la conception.
- Pour convertir les programmes pour qu'ils fonctionnent dans d'autres environnements (matériel, logiciel, système d'exploitation, réseau de télécommunications, etc.)..

Deux des "lois" de Lehman sur la dynamique d'évolution des systèmes

- Changement continu : un programme qui est utilisé dans un environnement réel devra nécessairement s'adapter et changer, sous peine de devenir progressivement moins utile dans cet environnement.
- Complexité croissante : au fur et à mesure qu'un programme évolue, sa structure tend à devenir plus complexe. Des ressources additionnelles doivent être consacrées à la préservation et à la simplification de sa structure.

2 Principaux types de maintenance

Les quatre catégories de maintenance selon l'IEEE

- Corrective : pour réparer des défauts (erreurs conception, de codage, de logique, etc.).

- Adaptative : pour adapter le logiciel à un environnement changeant, (règles d'affaires, règles et politiques gouvernementales, procédures de travail, plates-formes logicielles et matérielles).
- Perfective : pour ajouter ou modifier les fonctionnalités d'un système existant (pendant qu'il est en utilisation, les usagers on tendance à découvrir de nouveaux besoins).
- Préventive : pour prévenir la détérioration du logiciel face aux changements, pour améliorer sa "maintainabilité" future.

Il est important de bien catégoriser les types de changements car cela permet de bien gérer la maintenance, en fixant des niveaux appropriés de priorités pour les diverses demandes de changements :

- Maintenance adaptative et perfective apportent des *améliorations* (*enhancements*).
- Maintenance corrective, corrige des problèmes, des bogues.

Donc : la maintenance ne consiste pas uniquement *à corriger des bogues*.

3 Coûts de la maintenance

Importances des coûts de maintenance

- Étude faite dans les années '90 conclut que les dépenses en logiciels ont atteint plus de 100 milliards $. Or, plus de 70 milliards $ furent consacrés à la maintenance de systèmes existants, alors que seulement 30 milliards $ furent consacrés au développement de nouveaux systèmes.
- Diverses études sur le pourcentage des coûts de la maintenance par rapport à l'ensemble des coûts ont montré que la maintenance accapare généralement beaucoup plus de la moitié des coûts totaux : 60 %, 40–80 %, 60–70 %, 75 %, 65 %, 60–80 %.
- Répartition des différents types de maintenance :
  - Corrective : 20 %
  - Adaptative : 25 %
  - Perfective : 55 %

  Donc, près de 80 % des coûts sont liés à des améliorations, et non à des corrections de problèmes!

Pourquoi la maintenance est coûteuse

- Différence fondamentale entre activités de développement et de maintenance : contraintes que le système existant imposent sur la maintenance.

  Analogie : construire une nouvelle maison vs. ajouter une pièce à une maison existante — la nouvelle pièce ne doit pas affaiblir, ne doit pas mettre en péril la construction existante.
- La maintenance a des coûts élevés parce qu'il est nécessairement plus difficile d'ajouter une fonctionnalité *après* que le système est en opération que de réaliser la même fonctionnalité durant le développement initial.

  À ajouter : une figure sur le ratio des coûts de correction d'une erreur à différentes étapes du cycle de vie.

  Diverses raisons expliquent ce phénomène :

- Instabilité de l'équipe : Lorsqu'un système est livré, l'équipe qui l'a développé est souvent dispersée. Ceux qui font le travail de maintenance doivent alors comprendre le système existant, souvent à partir de zéro.
- Responsabilités contractuelles : Le contrat de maintenance est souvent donné à une organisation différente, à l'interne (autre département) ou à l'externe (autre organisation, par exemple, développement externe puis maintenance interne). L'équipe de développement n'a pas nécessairement avantage à passer beaucoup de temps à rendre le logiciel modifiable.

  Avant de pouvoir modifier un programme développé par quelqu'un d'autre, il faut tout d'abord le lire, le comprendre, ce qui peut demander un effort important : voir Section 4.
- Niveau d'expérience des équipes de maintenance : La maintenance est souvent effectuée par des gens moins expérimentés, peu familiers avec le domaine d'application ou les outils et logiciels utilisés.
- Vieillissement du programme : Au fur et à mesure qu'un programme évolue, sa structure se dégrade, donc devient de plus en plus difficile à comprendre et à changer.

Pourquoi on maintient des vieux systèmes, parfois désuets, (*legacy systems*) plutôt que les remplacer

- Contraintes économiques : Les coûts de maintenance peuvent être amortis à court terme, alors que les coûts de développement d'un nouveau système seraient très élevés et ne pourraient être amortis qu'à long terme.
- Erreurs résiduelles dans le nouveau système : Le nouveau système contiendra inévitablement des erreurs, qui devront elles aussi être corrigées. En fait, rien n'assure qu'à court ou moyen terme, le nouveau système fonctionnera mieux que l'ancien.
- Base existante de connaissances : Le système existant représente un repositoire important de connaissances, d'expertise acquise au fil des années.

4 Lecture et compréhension de programmes

- Avant de faire des changements, quelles qu'ils soient, il est essentiel de bien comprendre le logiciel et de bien saisir l'impact qu'auront ces changements. Ceci demande :
  - d'avoir une idée générale de ce que fait le logiciel, de ses relations avec l'environnement ;
  - de bien identifier où le système doit être modifié pour effectuer les changements requis ;
  - de bien comprendre comment les différentes parties du logiciels qui doivent être corrigées ou modifiées sont inter-reliées.

  Tout ces aspects sont liés à bien comprendre le logiciel, une tâche majeure du processus de maintenance.

- Les trois (3) principales étapes dans la compréhension d'un logiciel :
  1. Lire à propos du programme = lire les documents d'analyse (pour comprendre ce que fait le programme), les documents de conception (pour comprendre sa structure générale), etc.
  2. Lire le code source pour comprendre sa structure, les types et structures de données utilisés, les algorithmes employés, etc.

     Différents outils peuvent être utilisés :

- Outils de références croisées = pour générer un index de l'utilisation des procédures ou variables.
- *Program slicers* = pour identifier toutes les sections de code qui utilisent une ou plusieurs variables (pour bien comprendre l'impact de changements à cette variable).
- *Dataflow analysers* = pour comprendre les liens entre les parties de programme et, donc, l'analyse des impacts.

3. Exécuter le programme pour bien comprendre son comportement dynamique.

- Différents facteurs peuvent influencer la compréhension d'un programme
  - Expertise du programmeur qui effectue la maintenance.
  - Qualité de la mise en oeuvre du programme :
    - Convention utilisée pour l'écriture du programme : choix des identificateurs, mise en page et formatage, etc.
    - Commentaires.
    - Modularité et mécanismes de décomposition utilisés dans le programme.
  - Disponibilité et qualité de la documentation.
  - Organisation et présentation des programmes (indentation, espaces, utilisation d'une approche *style livre*, etc.)
  - Disponibilité d'outils d'aide à la compréhension et à l'analyse de programmes.
- Principales stratégies de compréhension de programmes — technique utilisée pour se former un modèle mental du programme, à partir de la documentation et du code source :
  - Approche descendante.
  - Approche ascendante.
  - Approche opportuniste.

5 Gestion des demandes de modification

Logiciel = programme + documentation

> *It is a common misconception to believe that software is programs. [...] A more comprehensive view is [that software] consists of the programs, documentation and operating procedures by which computers can be made useful to man [ldots]. [TG96]*

Contrôle des demandes de modification

- Le processus de contrôle des demandes de modification s'occupe de gérer la séquence d'événements qui débute avec une requête de modification d'un système existant et se termine soit avec le rejet de la requête, soit avec l'approbation de la demande de modification et son incorporation dans le système.

  Objectif du processus de contrôle des demandes de modification = assurer que les modifications au système sont faites de façon contrôlée de façon à éviter les effets de bord imprévus et indésirables.
- Activités clés du processus de contrôle des modifications :
  - Sélectionner les items prioritaires
  - Reproduire le problème (si c'est le cas)
  - Analyser le code (et les spécifications, lorsque disponibles)

- Effectuer les modifications de la conception et développer les tests appropriés
- Incorporer les modifications appropriées dans le code
- Effectuer le processus d'assurance qualité

```
DEBUT
  Compléter une requête (formulaire) de demande de modification
  Analyser la requête
  SI la modification demandée est valide ALORS
    Déterminer de quelle façon la modification pourrait être mise en oeuvre
    Évaluer le coût de la modification
    Enregistrer la demande de modification dans la base de données
    Soumettre la demande de modification au comité de contrôle des modifications
      (à moins que la modification soit relativement simple et mineure)
    SI la demande de modification est acceptée ALORS
      REPETER
        effectuer des modifications au logiciel
        enregistrer la modification en établissant la référence à la demande de modification
        soumettre le logiciel modifié au groupe d'assurance-qualité
      JUSQUE la qualité du logiciel soit adéquate
      créer une nouvelle version du système
    SINON
      Rejeter la demande de modification
  SINON
    Rejeter la demande de modification
FIN
```

Figure 1: Aperçu du processus de contrôle des modifications

Aperçu du processus de gestion des changements: voir Figure 1 (tiré de [Som01]).

- Il faut aussi s'assurer que si des changements sont effectués dans le code, ces changements sont reflétés dans la documentation, les spécifications, le manuel de l'usager, etc.

  Remarque importante: En fait, le processus de contrôle des modifications prend effet *dès que le logiciel* (au sens défini précédemment) est mis sous le contrôle du processus de *gestion de la configuration.*

Bibliographie

[BD99] B. Bruegge and A.H. Dutoit. *Object-Oriented Software Engineering — Conquering Complex and Changing Systems*. Prentice Hall, 1999.

[Pig97] T.M. Pigoski. *Practical Software Maintenance — Best Practices for Managing Your Software Investment*. John Wiley & Sons, Inc., 1997. [QA76.76S64P55].

[Som01] I. Sommerville. *Software Engineering (Sixth Edition)*. Addison-Wesley, 2001.

[Spi03] D. Spinellis. *Code Reading — The Open Source Perspective*. Addison-Wesley, 2003.

[TG96] A.A. Takang and P.A. Grubb. *Software Maintenance — Concepts and Practice*. International Thomson Computer Press, 1996. [QA76.76S64T25].