Gestion de la maintenabilité

Marc Frappier

Département de mathématiques et d'informatique, Université de Sherbrooke, Sherbrooke, Québec, J1K 2R1,
tél.: 819-821-7096, fax: 819-821-8200, email: marc.frappier@dmi.usherb.ca

Résumé

De 50 à 75 % de l'effort logiciel est consacré à la maintenance de systèmes. La productivité des équipes de maintenance est fortement influencée par la maintenabilité. Il est donc essentiel pour un gestionnaire de systèmes de bien gérer la maintenabilité pour planifier et contrôler adéquatement les projets de maintenance. Puisqu'on ne peut gérer ce qu'on ne peut mesurer, il apparaît nécessaire de pouvoir quantifier la maintenabilité pour bien gérer la maintenance. Dans cet article, nous proposons une mesure de la maintenabilité qui peut être calculée durant la phase de maintenance. Nous identifions aussi une approche pour estimer la maintenabilité durant la phase de développement. Nous tentons également de clarifier le concept de maintenabilité, et de le distinguer des autres caractéristiques du logiciel.

Mots clés

Maintenabilité, coûts de maintenance, réingénierie.

1. Introduction

Plusieurs études de la maintenance du logiciel dans les organisations (Lientz, 78), (McKee, 84) montrent que de 50 à 75 % des ressources humaines affectées au logiciel sont consacrées à la maintenance. Cette situation s'explique aisément. Les organisations procèdent constamment à l'introduction de nouveaux logiciels. Le taux d'introduction de nouveaux logiciels étant généralement plus élevé que le taux d'élimination, on assiste à un accroissement du portefeuille logiciel des organisations. De plus, si on analyse la répartition de l'effort pour le cycle de vie d'un logiciel (Guimaraes, 83), on constate que la maintenance représente de 50 à 80 % de l'effort. Il est donc tout à fait normal d'affecter de 50 % à 75 % des ressources à la maintenance. Puisque la maintenance accapare autant de ressources, il est essentiel pour une organisation de bien contrôler les coûts de maintenance.

La maintenabilité est une caractéristique du logiciel. On définit la maintenabilité comme étant la facilité avec laquelle un logiciel peut être maintenu. La maintenabilité est donc un des facteurs qui influencent les coûts de maintenance. Il est essentiel pour les gestionnaires de systèmes de bien contrôler cet aspect de la qualité des logiciels. Pour faire une gestion rationnelle de la maintenabilité, il est primordial de disposer d'une mesure quantitative. Une telle mesure permettrait de définir des critères quantitatifs de maintenabilité pour les logiciels à développer et d'assurer un suivi durant la maintenance du logiciel. De plus, cette mesure pourrait servir à

identifier les systèmes nécessitant une réingénierie afin d'améliorer la maintenabilité et de réduire les coûts de maintenance.

Dans cet article, nous proposons une mesure de maintenabilité. Cette mesure est *calculable* après le début de la phase de maintenance. Elle permet donc de gérer la maintenabilité durant la maintenance. Comme il est aussi nécessaire de contrôler la maintenabilité durant le développement et à la livraison du logiciel, nous proposons également une technique pour *estimer* la maintenabilité.

La mesure que nous proposons n'est pas parfaite, car la maintenabilité est un concept difficile à mesurer avec précision. Nous indiquons donc les conditions dans lesquelles la mesure peut être utilisée et la manière dont elle doit être utilisée.

Cet article est structuré comme suit. Dans la section 2, nous situons la maintenabilité dans le contexte de la qualité du logiciel et nous la distinguons des autres caractéristiques de qualité. Dans la section 3, nous analysons la structure des coûts de maintenance pour bien caractériser comment la maintenabilité influence ces coûts. Dans la section 4, nous introduisons notre mesure de maintenabilité ainsi que les prémisses menant à sa définition. Dans la section 5, nous discutons de quelques méthodes d'estimation de la maintenabilité pour ensuite conclure par quelques remarques sur la nature de notre proposition.

2. Maintenabilité et la qualité du logiciel

La maintenabilité est un facteur de qualité du logiciel. Dans cette section, nous allons distinguer la maintenabilité des autres facteurs de qualité.

Plusieurs chercheurs ont proposé des modèles de qualité. Pour notre analyse, nous retenons un des premiers modèles proposés, celui de Boehm *et al* (Boehm 78), pour sa simplicité et sa complétude. Le modèle est illustré à la figure 1.

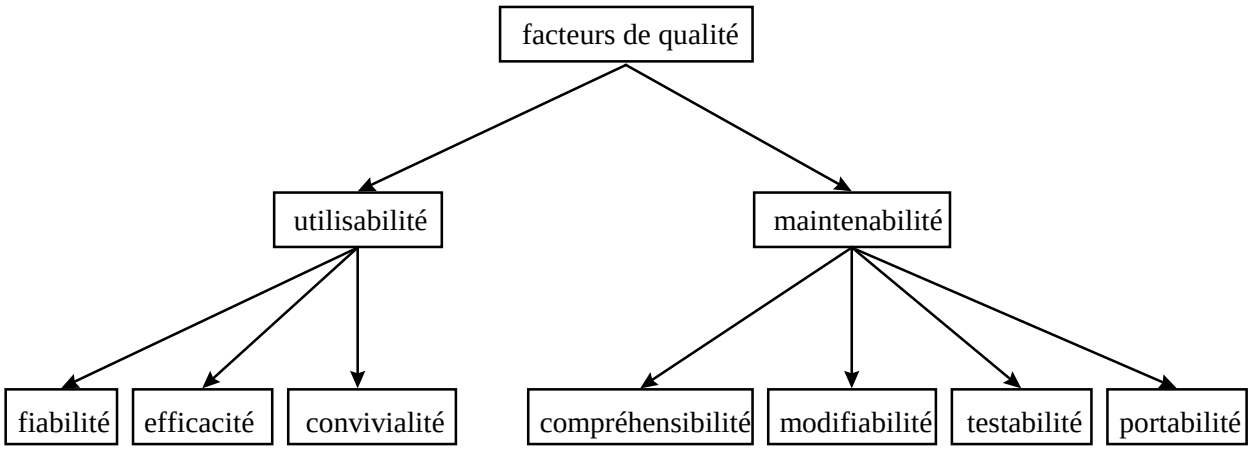

Figure 1 - Facteurs de qualité

Selon le modèle de Boehm *et al*, il y a deux facteurs principaux de qualité, soit l'utilisabilité, qui représente le point de vue de l'utilisateur, et la maintenabilité, qui représente le point de vue du concepteur. L'utilisabilité se décompose en trois facteurs:

fiabilité: le degré avec lequel un système s'exécute sans occurrence d'erreurs;

efficacité: le degré avec lequel un système exécute les fonctions requises sans gaspiller les ressources de l'ordinateur;

convivialité: le degré représentant la commodité et la facilité de l'usage d'un système.

La maintenabilité se décompose en quatre facteurs:

compréhensibilité: le degré représentant la facilité pour un humain de comprendre l'objet des éléments (divers produits logiciels, par exemple la spécification fonctionnelle, le design, le code source, etc.) d'un système;

modifiabilité: le degré de facilité de modification du système;

testabilité: le degré de facilité pour tester un système;

portabilité: le degré de facilité pour porter un système d'une plateforme à une autre.

Quelques auteurs (Martin, 83) incluent les facteurs d'utilisabilité dans la définition de la maintenabilité. Nous croyons que cette approche est inadéquate parce qu'elle ne correspond pas à la définition généralement acceptée de la maintenabilité, soit la facilité avec laquelle un système peut être maintenu. Les facteurs d'utilisabilité influencent plutôt la *quantité* de maintenance à faire: un système peu fiable, peu convivial ou inefficace nécessitera plus de changements qu'un système ayant bonne utilisabilité; la *facilité* de faire ces changement dépendra de la maintenabilité. En pratique, il est possible qu'il existe une corrélation entre l'utilisabilité et la maintenabilité, c'est-à-dire que les systèmes ayant une faible utilisabilité ont typiquement une faible maintenabilité, et vice versa. Toutefois, il existe aussi des systèmes ayant une bonne utilisabilité, mais qui ont une très faible maintenabilité. Notons également que certains facteurs de qualité sont en conflit. Par exemple, si on désire augmenter l'efficacité d'un système, on le fera peut être en utilisant des algorithmes plus complexes, donc plus difficiles à maintenir.

3. Maintenabilité et coûts de maintenance

La qualité du logiciel influence les coûts de maintenance. Pour bien comprendre comment la qualité influence les coûts de maintenance, il est utile de diviser la maintenance en trois catégories:

- la maintenance *corrective*, qui consiste en la correction de fautes;
- la maintenance *adaptative*, qui consiste à modifier le design (l'implantation) d'un système tout en préservant ses fonctions (ex: améliorer la performance, améliorer la maintenabilité, porter le système sur une autre plateforme)

- la maintenance *perfective*, qui consiste à modifier la fonction d'un système, typiquement pour s'adapter à de nouveaux besoins des utilisateurs.

Les coûts de maintenance pour une période donnée peuvent être estimés, grossièrement, à l'aide de la formule suivante:

$$COÛT_{MNT} = COÛT_{MOD} * (NB_{COR} + NB_{ADP} + NB_{PER}), \quad (1)$$

où $COÛT_{MNT}$ représente les coûts de maintenance, $COÛT_{MOD}$ représente le coût d'une unité de changement, et NB_{COR}, NB_{ADP} et NB_{PER} représentent respectivement le nombre d'unités de changements correctifs, adaptatifs et perfectifs pour la période donnée. Nous discuterons dans la prochaine section de la définition d'une unité de changement. Pour l'instant, nous faisons appel à l'intuition du lecteur pour considérer qu'une unité de changement représente une unité de travail à effectuer dans la phase de maintenance.

L'équation (1) nous permet d'analyser comment les facteurs de qualité influencent les coûts de maintenance. La maintenabilité influence la variable $COÛT_{MOD}$: moins un système est maintenable, plus le coût d'une unité de changement sera grand. Soulignons qu'il y a d'autres facteurs reliés au processus de maintenance et aux ressources qui influencent $COÛT_{MOD}$, comme l'habileté de l'équipe de maintenance et l'usage d'outils de GLAO (ex: générateurs de tests, outils de vérification, outils de gestion des configurations). La fiabilité influence la variable NB_{COR} : moins un système est fiable, plus il y aura de changements correctifs à faire. De même, l'efficacité influence la variable NB_{ADP} et la convivialité influence la variable NB_{PE}. Notons que ces variables ne sont pas influencées uniquement par les les facteurs de qualité du logiciel; la qualité du processus de maintenance, la qualité des ressources et la volatilité des exigences des utilisateurs les influencent aussi.

A la lumière de cette équation, il est assez facile d'analyser les causes de coûts de maintenance lorsqu'ils sont jugés trop élevés. Par exemple, il est courant d'énoncer qu'un système coûte cher à maintenir et d'en déduire que le système a une faible maintenabilité. L'équation (1) nous force à user de plus de discernement dans l'analyse des causes. Un système qui coûte cher à maintenir n'est pas nécessairement un système ayant une faible maintenabilité; la cause peut aussi être une faible utilisabilité, un processus de maintenance mal structuré, des ressources de faible qualité ou un grand nombre de changements perfectifs demandés par les utilisateurs.

4. Définir et évaluer la maintenabilité

Dans cette section, nous proposons une définition théorique de la maintenabilité, puis nous identifions un indicateur permettant de l'évaluer durant l'opération du système.

4.1. Définition rigoureuse de la maintenabilité

En introduction, nous avons défini de manière informelle la maintenabilité comme étant la facilité avec laquelle un logiciel peut être maintenu. Cette définition est au mieux imprécise, et nous allons essayer de la clarifier. Nous définissons la maintenabilité comme étant une relation d'ordre sur les systèmes: on dit que le système *A* est moins maintenable que le système *B*, que

nous notons par $A \leq_m B$, si et seulement si *A* et *B* fournissent les mêmes fonctions et, pour tout changement *c*, le coût de modification de *A* est supérieur au coût de modification de *B*. Si on exprime cette définition de manière formelle, nous obtenons la formule suivante:

$$A \leq_m B \Leftrightarrow \mathit{fonction}(A) = \mathit{fonction}(B) \wedge \forall c : \mathit{coût}(c, B) \leq \mathit{coût}(c, A). \qquad (2)$$

Cette relation nous permet de comparer les systèmes, de les ordonner selon la maintenabilité. On remarque que certains systèmes ne peuvent être comparés. Si *A* et *B* n'offrent pas les mêmes fonctions, alors on ne peut dire si *A* est plus (ou moins) maintenable que *B*; cela n'a pas de sens, parce qu'on ne peut appliquer les mêmes changements à *A* et à *B* pour pouvoir comparer la difficulté de faire un changement. Par exemple, on ne peut dire si un système de gestion du personnel est plus (ou moins) maintenable qu'un système de comptabilité. Pour pouvoir les comparer, il faudrait leur appliquer les mêmes changements et comparer ensuite le coût requis pour effectuer les changements. Puisqu'ils ont des fonctions différentes, on ne peut leur appliquer les mêmes changements. Toutefois, il est possible de comparer les *coûts de maintenance* de ces systèmes pour une période donnée. Cela illustre la distinction entre la notion de maintenabilité et la notion de coûts de maintenance, comme nous l'avons souligné à la section 3.

Si deux systèmes offrent les mêmes fonctions, il est possible qu'on ne puisse comparer leur maintenabilité. Par exemple, il est assez facile d'imaginer deux systèmes, *A* et *B*, qui fournissent les mêmes fonctions tout en ayant des conceptions (implantations) différentes. Pour certains changements, le système *A* peut être plus facile à maintenir que le système *B*, et pour d'autres changements, le système *B* peut être plus facile à maintenir que le système *A*. Dans ce cas, on ne peut conclure que *A* est plus (ou moins) maintenable que *B*. On peut comparer la maintenabilité de *A* et *B* à condition de se restreindre à un ensemble de changements. Pour ce faire, nous définissons la notion de *maintenabilité relative*. On dit que, pour un ensemble de changements *C*, que le système *A* est moins maintenable que le système *B*, que nous notons par $A \leq_{m|C} B$, si et seulement si *A* et *B* fournissent les mêmes fonctions et, pour tout changement *c* de *C*, le coût de modification de *A* est supérieur au coût de modification de *B*. Si on exprime cette définition de manière formelle, nous obtenons la formule suivante:

$$A \leq_{m|C} B \Leftrightarrow \mathit{fonction}(A) = \mathit{fonction}(B) \wedge \forall c \in C : \mathit{coût}(c, B) \leq \mathit{coût}(c, A). \qquad (3)$$

Lorsqu'on conçoit un système, on cherche à définir le design qui maximisera la maintenabilité, soit d'un point de vu absolu, en considérant tous les changements possibles, ou soit d'un point de vu relatif, en considérant un ensemble de changements que l'on estime très probables. On cherche donc un système *A* qui est un maximum selon la relation d'ordre de maintenabilité.

Notre définition de la maintenabilité se démarque de celles généralement utilisées dans la littérature scientifique. En général, on observe deux tendances: soit la maintenabilité est réduite à une caractéristique organique du logiciel comme le flot d'information ou le graphe de contrôle, ou soit la définition choisie comprend d'autres facteurs que la maintenabilité. Par exemple, dans (Coleman, 94), on propose deux mesures de maintenabilité qui sont basées sur le guide AFOTEC

(USAF, 89). Le guide AFOTEC permet de calculer la maintenabilité en évaluant une centaine d'aspects organiques jugés représentatifs de la maintenabilité. Les aspects sont évalués par un expert sur une échelle de 0 à 5. La maintenabilité est représentée par la somme des valeurs. Aucune étude empirique indique que cette mesure permet de comparer la difficulté de faire un changement. De plus, une étude (Gibson, 89) montre que les programmeurs sont incapables de discerner les différences entre les systèmes. Il faut donc utiliser les évaluations par des personnes avec prudence et appliquer des méthodes pour contrôler la subjectivité.

4.2. Indicateur de maintenabilité

Les définitions (2) et (3) données dans la section précédentes sont intéressantes d'un point de vue théorique pour bien cerner le concept de maintenabilité. Toutefois, elles n'aident pas beaucoup le praticien dans la gestion de la maintenabilité puisqu'elles sont difficilement calculables. Dans cette section, nous proposons une mesure que nous qualifions d'*indicateur de maintenabilité*.

Ayant mentionné dans la section 3 que la maintenabilité influence la valeur de la variable $COÛT_{MOD}$, il semblerait naturel d'utiliser cette variable comme indicateur de maintenabilité. Toutefois, il y a plusieurs problèmes à cette approche. En premier lieu, il faut définir la notion d'*unité de changement*. Une unité de changement devrait représenter une certaine quantité de travail à faire. La mesure en jours-personnes de l'effort requis pour effectuer une unité de changement devrait être constante pour des systèmes de même maintenabilité. Actuellement, il n'existe pas, à notre connaissance, de définition adéquate d'une unité de changement. Plusieurs mesures ont été suggérées, comme les mesures de taille du logiciel telles les lignes de code source ou les points de fonction. Ces mesures comportent plusieurs faiblesses. Par exemple, le nombre de lignes de code est influencé par plusieurs facteurs dont la complexité du domaine d'application, le style de programmation et le langage de programmation; le nombre de points de fonction ne tient pas compte, entre autres, de la difficulté d'implanter une fonction.

La définition d'une unité de changement qui nous semble la *moins pire* est la suivante: une unité de changement consiste en la correction d'une faute. La génération des fautes étant un facteur relativement aléatoire et indépendant des utilisateurs, la distribution de la taille des fautes devrait être similaire d'un système à un autre. De plus, le processus de maintenance corrective est essentiellement le même que le processus de maintenance adaptative et perfective. Il est donc raisonnable de supposer que la difficulté de faire la maintenance corrective est représentative de la difficulté de faire la maintenance adaptative et perfective.

Notre choix d'unité de changement nous amène à proposer $COÛT_{COR}$, le coût moyen de correction d'une faute, comme indicateur de maintenabilité. Si la valeur de $COÛT_{COR}$ augmente, la maintenabilité devrait diminuer, et vice versa. Pour calculer $COÛT_{COR}$, on calcule le temps moyen pour exécuter le processus illustré à la figure 2 pour corriger une faute.

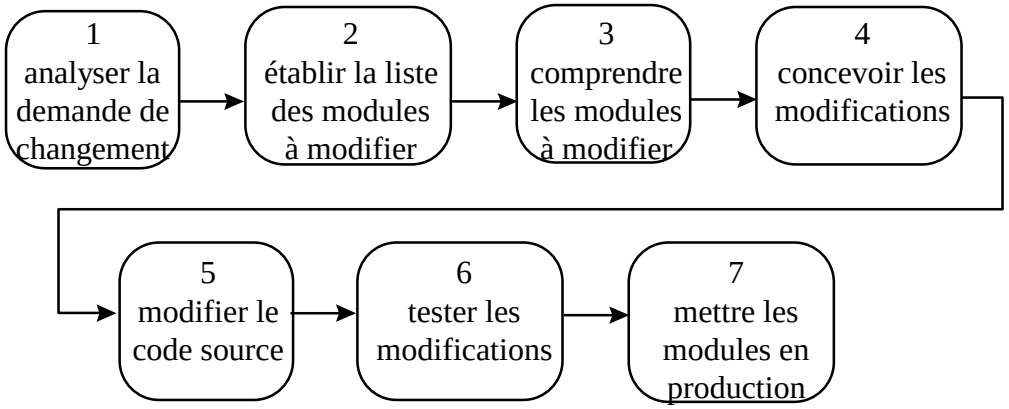

Figure 2 - Processus de maintenance

Nous qualifions $COÛT_{COR}$ d'indicateur de maintenabilité parce qu'il existe plusieurs facteurs autres que la maintenabilité qui l'influencent, comme la qualité du processus de maintenance et la qualité des ressources. Ceci étant dit, les remarques de la section 4.1 sur la définition de la maintenabilité et la nature d'indicateur de $COÛT_{COR}$ nous amène à faire les précisions suivantes: il n'est pas valide d'utiliser $COÛT_{COR}$ pour comparer la maintenabilité de systèmes ayant des fonctions différentes, des processus logiciels différents ou étant maintenus par des ressources de qualité différente; de plus, l'indicateur $COÛT_{COR}$ ne reflète pas la notion de maintenabilité relative telle que définie à la section 4.1.

Le gestionnaire de logiciels peut utiliser l'indicateur $COÛT_{COR}$ pour effectuer les tâches suivantes:

- suivre l'évolution de la maintenabilité d'un système;

  En calculant $COÛT_{COR}$ pour des intervalles réguliers (ex: six mois ou un an), le gestionnaire peut suivre l'évolution dans le temps, et décider du moment opportun pour faire la réingénierie du système, afin d'améliorer la maintenabilité, ou de faire la réingénierie du processus de maintenance, afin d'augmenter la productivité, ou de procéder à la restructuration de l'équipe de maintenance.

- bâtir une norme afin de comparer les systèmes entre eux;

  On peut bâtir un historique qui comporterait les informations suivantes pour chaque système: caractérisation des fonctions, caractérisation du processus logiciel, caractérisation de l'équipe de maintenance et valeur de $COÛT_{COR}$. Un gestionnaire peut ensuite comparer des systèmes de mêmes caractéristiques pour établir des critères de valeurs désirées pour $COÛT_{COR}$, c'est-à-dire qu'il peut bâtir une norme de maintenabilité.

- sélectionner les logiciels pour la réingénierie

  L'objectif de la réingénierie est d'améliorer la maintenabilité afin de réduire les coûts de maintenance. Lorsqu'on dispose d'une norme fondée sur un historique tel que décrit au point précédent, on peut ensuite choisir plus rationnellement les logiciels pour la

réingénierie. Par exemple, supposons qu'un gestionnaire dispose du tableau suivant pour son portefeuille de logiciels.

Système	Norme $COÛT_{COR}$ (j-p par faute)	valeur actuelle $COÛT_{COR}$ (j-p par faute)	réduction potentielle $COÛT_{COR}$	coûts annuels de maintenance	économie annuelle potentielle
A	4	6	33 %	300K $	100K $
B	3	4	25 %	100K $	25K $
C	2	2,5	20 %	800K $	200K $

Tableau 1 - Table d'aide à la décision de réingénierie

La deuxième colonne du tableau 1 donne la valeur de $COÛT_{COR}$ en jours-personnes selon la norme établie pour un système. La troisième colonne donne la valeur actuelle de $COÛT_{COR}$. La quatrième colonne donne la réduction potentielle de la valeur de $COÛT_{COR}$ après avoir effectué une réingénierie. En posant comme hypothèses que $COÛT_{COR}$ est directement proportionnel à $COÛT_{MOD}$ et que le nombre de changements à faire demeure constant d'une année à l'autre, on peut, grâce à l'équation (1), calculer l'économie annuelle des coûts de maintenance après une réingénierie. Dans l'exemple illustré au tableau 1, le système A offre le meilleur potentiel d'accroissement de la maintenabilité (33 %). Toutefois, le système C offre le meilleur potentiel d'économie de coûts de maintenance parce qu'il a un plus gros volume de changements. Pour finaliser son choix de réingénierie, le gestionnaire pourra prendre en considération d'autres facteurs comme le coût du projet de réingénierie, la valeur corporative du système, sa durée de vie et les coûts d'opération.

5. Estimation de l'indicateur de maintenabilité

L'indicateur $COÛT_{COR}$ que nous proposons peut être calculé seulement après que le système ait été en opération et qu'un nombre significatifs de fautes aient été corrigées. Néanmoins, un estimé de cet indicateur serait fort précieux pour la phase de développement. Il permettrait d'évaluer la maintenabilité du logiciel durant sa construction, et de guider l'équipe de développement dans le choix d'alternatives de conception afin d'obtenir un certain niveau de maintenabilité.

On peut tenter d'estimer $COÛT_{COR}$ à partir de mesures des caractéristiques du logiciel, du processus de maintenance et des ressources. Pour ce faire, il faut identifier des mesures X_i du logiciel, du processus de maintenance et des ressources. Ensuite, on doit bâtir un échantillon de systèmes où on peut mesurer $COÛT_{COR}$ ainsi que les autres mesures X_i. Finalement, on tente de dériver de l'échantillon un modèle (une fonction) qui permet d'estimer $COÛT_{COR}$ à partir des X_i.

Il existe plusieurs techniques pour créer des modèles. Certaines proviennent des statistiques (régression linéaire, régression logistique), d'autres proviennent de la recherche en apprentissage automatique (réseau de neurones, arbre de classification, réduction optimisée d'ensembles).

Comme le lecteur s'en doute, la construction d'un modèle d'estimation est plutôt ardue. Les difficultés sont nombreuses. D'une part, on ne sait pas encore très bien quelles mesures permettent d'estimer $COÛT_{COR}$. Les candidats semblent nombreux. Nous référons le lecteur à (Frappier, 94), où plusieurs mesures de logiciel sont identifiées en se basant sur des études empiriques, et à (Briand, 96), (Evanco, 95) et (Jorgensen, 95). Les facteurs reliés au processus de maintenance et aux ressources sont tout aussi difficiles à mesurer: comment peut-on caractériser les habilités d'une équipe de maintenance, la structure d'un processus logiciel, les outils logiciels?

D'autre part, il est difficile de construire un échantillon de taille significative où des mesures fiables et rigoureuses peuvent être calculées. Il n'est pas tout de rassembler la documentation de plusieurs systèmes. Il faut s'assurer que les mesures sont calculées de la même façon pour chaque système, que les mêmes définitions ont été utilisées. Pour les systèmes développés et maintenus il y a plusieurs années, il peut être difficile de faire cette vérification. De plus, le coût de cette opération de collecte des mesures n'est pas négligeable. A titre indicatif, la littérature cite des coûts de programmes de mesure effectués durant la phase de développement qui varient entre 7 % et 15 % des coûts de développement (Sommerville, 96).

Quelques modèles d'estimation ont été proposés dans la littérature. Leur précision varie beaucoup, allant d'erreurs relatives de 5 % à 50 % dans les meilleurs cas. De plus, ces modèles sont difficilement applicables dans des environnements différents de celui où ils ont été développé, ce qui démontre qu'il est compliqué de caractériser correctement le domaine d'application, le processus logiciel et la qualité des ressources.

6. Conclusion

La maintenabilité est un facteur de qualité du logiciel qui influence les coûts de maintenance. Il est important de faire la distinction entre la maintenabilité et les coûts de maintenance. La maintenabilité représente la facilité de maintenir un système, alors que les coûts de maintenance représentent l'effort consacré pour effectuer un certain nombre de changements.

Nous avons proposé une définition théorique de la maintenabilité qui est basée sur le coût pour effectuer un changement. Cette définition implique qu'on ne peut comparer la maintenabilité de systèmes offrant des fonctions différentes, parce que cela n'a pas de sens. Nous identifions un indicateur de maintenabilité, le coût moyen de correction d'une faute, qui peut être calculé lorsque le système est en opération. Pour évaluer cet indicateur durant la phase de développement, on doit bâtir un modèle qui estime la maintenabilité à partir de mesures du logiciel, du processus de maintenance et des ressources. Cette tâche est plutôt ardue, et peu de modèles de précision adéquate ont été présentés dans la littérature jusqu'à maintenant.

Notre indicateur de maintenabilité peut être utilisé pour suivre l'évolution de la maintenabilité d'un logiciel, et décider du moment opportun pour effectuer une réingénierie. Il

permet également de bâtir une norme, de comparer les logiciels entre eux, et de sélectionner les candidats les plus profitables pour la réingénierie.

7. Bibliographie

BOEHM, B. *et al* (1978): *Characteristics of Software Quality*, TRW Series of Software Technology, North-Holland.

BRIAND, L. C., S. MORASCA, V.R. BASILI (1996): Property-Based Software Engineering Measurement, *IEEE Transactions on Software Engineering*, **22**(1), 68-86.

COLEMAN, D., D. ASH, B. LOWTHER, P. OMAN (1994): Using Metrics to Evaluate Software System Maintainability. *IEEE Software*, August 1994, 44-49.

EVANCO, W.M. (1995): Modeling the Effort to Correct Faults, *Journal of Systems and Software*, **29**, 75-84.

FRAPPIER, M., S. MATWIN and A. MILI (1994): Software Metrics for Predicting Maintainability. Software Metrics Study, Tech. Memo. 2, Canadian Space Agency, St-Hubert, Canada, http://www.dmi.usherb.ca/personnel/profs_info/html/Frappier/tm2.html.

GIBSON, V.R., J.A. SENN (1989):: System Structure and Software Maintenance Performance. *Communications of the ACM*, **32**(3), 347-358.

GUIMARAES, T (1983): Managing Application Program Maintenance Expenditures. *Communications of the ACM*, **26**(10), 739-746.

JORGENSEN, M. (1995): Experience with the Accuracy of Software Maintenance Task Effort Prediction Models, *IEEE Transactions on Software Engineering*, **21**(8), 674-681.

LIENTZ, W, B. SWANSON (1978): *Software Maintenance Management*. Addison Wesley.

MCKEE, J.R. (1984): Maintenance as a Function of Design. In *Proc. AFIPS National Computer Conference*, Las Vegas, 187-193.

SOMMERVILLE, I. (1996): *Software Engineering*. Addison Wesley.

USAF (1989): Software Maintainability - Evaluation Guide, AFOTEC Pamphlet 800-2 (updated), HQ Air Force Operational Test and Evaluation Center, Kirkland Air Force Base, NM, USA, vol. 3.