**Chapitre 1 – Préliminaires et concepts**

1. **Introduction**

Le développement de logiciels, lorsqu’il est mené à terme avec succès, se termine par la livraison d’un logiciel qui devrait satisfaire les exigences initiales du client. Par la suite, après sa mise en service, le logiciel devra évoluer pour répondre aux nouveaux besoins d’un environnement en constante évolution. De plus, c’est pendant l’exploitation du logiciel que l’organisation découvrira des anomalies, ainsi que de nouveaux besoins d’affaires qui feront surface, et qu’il faudra, après un certain nombre d’années, en modifier la plate-forme technologique. Enfin, il est bon de souligner que le cycle de vie de la maintenance ne devrait pas débuter lors de la mise en service du logiciel, mais en réalité, bien avant par une participation active des responsables de la maintenance tout au long du processus de développement, quand c’est possible.

1. **Définitions :**

**Evolution** : un processus de changement continu d'un niveau inférieur, plus simple, à un état supérieur, plus complexe, ou mieux.
**Maintenabilité** : la facilité avec laquelle la maintenance peut être effectuée.
**Maintenance** : le fait de garder une entité dans un état de réparation actuel, d'efficacité ou de validité, pour préserver de l'échec ou du déclin.
**Logiciels** : les programmes, la documentation et les procédures d'exploitation par lesquels les ordinateurs deviennent utiles à l'homme.

**La maintenance du logicie**l : modification d'un produit logiciel après la livraison, pour corriger les défauts, pour améliorer la performance ou d'autres attributs, ou pour adapter le produit à un environnement modifié.

Le cycle de vie du logiciel peut être divisé en deux parties distinctes selon Moriguchi [[1]](#footnote-1) :

1. Le développement initial du logiciel ;

2. La maintenance et l’opération du logiciel.

Un survol des définitions proposées pour la maintenance du logiciel est présenté au tableau 1 :

**Tableau 1. Les définitions généralement acceptées de la maintenance du logiciel**



Il est important de préciser que la maintenance du logiciel est nécessaire autant pour les logiciels applicatifs que pour les logiciels d’infrastructure (par exemple : de télécommunications, de systèmes d’opérations et de gestion des bases de données). Le logiciel applicatif, lui, contient des règles d’affaires traduites et insérées dans les fonctionnalités du logiciel. Ce sont ces règles d’affaires qui évoluent, changent et disparaissent pour supporter les opérations quotidiennes des organisations.

1. **Les bases fondamentales :**

Il est une fausse idée de croire que le logiciel (software) représente seulement les programmes.
Par exemple, lorsqu’on évoque des activités de maintenance du logiciel, on pense à des activités menées exclusivement sur les programmes.

La définition de Mc Dermid montre clairement que le logiciel comprend non seulement des programmes en code source et objet, mais aussi la documentation de toutes les facettes du programme, telles que l'analyse des besoins, la spécification, la conception, le système et les manuels d'utilisation, et
les procédures utilisées pour configurer et utiliser le système logiciel. Le Tableau 2 montre les composants d'un système logiciel et des exemples de chacun d'eux.

**Table 2**. Les Composants d’un système logiciel



Vous trouverez de nombreuses définitions différentes de la maintenance des logiciels dans la littérature Certains prennent une vue ciblée et spécifique. Typiquement parmi cet ensemble de définitions sont :

* La vue fixation de bugs : la maintenance est la détection et la correction d'erreurs
* La vue nécessité d'adapter : la maintenance est d'apporter des modifications au logiciel lorsque son environnement opérationnel ou ses besoins d'origine changent.
* La vue support de l'utilisateur : la maintenance est la fourniture de support aux utilisateurs.
1. **Différence entre la maintenance et le nouveau développement :**

Bien que la maintenance pourrait être considérée comme une continuation d'un nouveau développement (figure 1)



**Figure 1**. Modèle en cascade du cycle de vie d’un logiciel

Avant d'entreprendre tout développement ou travail de maintenance d’un système, une analyse d’impact doit être effectuée pour déterminer la ramification de la nouvelle modification du système sur l'environnement dans lequel il doit être introduit.

Il est tout d’abord nécessaire d’établir une distinction claire entre l’opération d’un logiciel et sa maintenance. Ainsi, il est précisé dans la norme ISO 14764 que les activités d’opération (copies de sécurité, administration des ordinateurs) sont effectuées par le personnel d’opération des systèmes informatiques et sont exclues de la portée de la norme ISO sur la maintenance des logiciels.

Qu’en est-il des différences entre la maintenance et le développement du logiciel ? Le développement du logiciel possède, lui aussi, une interface avec la maintenance, mais elle est un peu plus difficile à différencier. Quelques auteurs ont étudié les activités qui sont uniques à la maintenance et qui ne se retrouvent pas dans un cycle de vie de développement de logiciels. Certaines des caractéristiques qui sont propres au domaine de la maintenance du logiciel sont :

* Les requêtes de modifications (RM) parviennent d’une manière plus ou moins aléatoire et ne peuvent pas être planifiées individuellement dans un processus annuel de budgétisation ;
* Les RM sont évaluées et classées par ordre de priorité ;
* La charge de travail de la maintenance n’est pas gérée par des techniques de gestion de projet, mais plutôt par l’utilisation des techniques de gestion des files d’attente, souvent supportées par un logiciel de bureau d’aide help desk.



**Figure 2**. Processus d’acceptation ou de refus du travail de la maintenance

Lorsque l’utilisateur fait une requête de modification, il est nécessaire d’estimer l’effort qui sera requis pour modifier le logiciel existant.

Il y a donc, pour la maintenance du logiciel, un processus unique d’acceptation ou de rejet du travail, pour les requêtes de modifications (RM) des logiciels opérationnels. Ce processus tient compte de la taille estimée de la modification et de la capacité à la réaliser à l’intérieur des contraintes de coûts et de qualité de fonctionnalité.

En résumé, la maintenance du logiciel possède un certain nombre de processus et d’activités qui ne sont pas effectués par les groupes de développement du logiciel. La maintenance du logiciel fait aussi appel à des processus et à des activités du développement du logiciel, et ce, plus particulièrement dans l’étape d’implémentation d’une modification au logiciel existant [ISO95, s5.5.3].

1. **Nécessité de la maintenance**

Il y a un certain nombre de facteurs qui fournissent la motivation pour la maintenance :

* Pour assurer la continuité du service : Les systèmes doivent continuer à fonctionner. Par exemple, le logiciel de contrôle des avions ou les systèmes de signalisation de train ne doivent pas s’arrêter si une erreur se produit. Il peut y avoir des conséquences graves à une défaillance du système, tels que des inconvénients sérieux ou des implications financières importantes. Les activités de maintenance visant à maintenir un système opérationnel comprennent la correction des bugs, la récupération suite à un échec, et l’adaptation des changements dans le système d'exploitation et le matériel.
* Pour soutenir les mises à jour obligatoires : Ce type de changement serait nécessaire à cause de modifications à la réglementation gouvernementale par exemple les changements dans les lois fiscales nécessiteront des modifications dans le logiciel utilisé par les services fiscaux (taxe).
* Pour répondre aux demandes d’amélioration des utilisateurs : Dans l'ensemble, plus le système est bon, plus il sera utilisé et plus les utilisateurs demanderont des améliorations de la fonctionnalité.
* Afin de faciliter les futurs travaux de maintenance : Il est souvent financièrement et commercialement justifié d'initier le changement uniquement pour faire l'entretien futur plus facile.

Si un système est utilisé, il n’est jamais terminé, car il aura toujours besoin d'évoluer pour répondre aux besoins de l'évolution du monde dans lequel il opère.

1. **Catégories de la maintenance**

Swanson [[2]](#footnote-2) est le premier, en 1976, à examiner en profondeur la phase de maintenance. Il met en valeur trois catégories différentes de ”difficultés” pouvant être résolues par un processus de maintenance. Ces trois catégories sont les suivantes :

– **Corrective** : Tout changement rendu nécessaire par des erreurs réelles (bugs induits ou résiduels dans un système) ;

– **Adaptative** : Tout effort lancé en raison de changements dans l’environnement dans lequel un système logiciel doit fonctionner ;

– **Perfective** : Tous les changements, insertions, suppressions, modifications, extensions, et perfectionnements faits à un système pour satisfaire les besoins d’évolution et/ou d’extension de l’utilisateur.

**6.1 Maintenance corrective :**

La maintenance corrective fixe les différents bogues d’un système. Le système peut ne pas fonctionner comme il devrait le faire selon ses plans de conception. L’objectif de la maintenance corrective est donc ici de localiser les spécifications originelles afin de déterminer ce que le système était initialement sensé faire.

**6.2 Maintenance adaptative**

La maintenance adaptative fixe les différents changements qui doivent être faits afin de suivre l’environnement du système en perpétuelle évolution. Par exemple, le système d’exploitation peut être mis à jour et des modifications peuvent être nécessaires afin de s’accommoder à ce nouveau système d’exploitation.

**6.3 Maintenance perfective**

La maintenance perfective ou d’amélioration comprend tous les changements faits sur un système afin de satisfaire aux besoins de l’utilisateur. Si quelque chose n’était pas dans la conception originelle ou dans les spécifications du système et que l’utilisateur désire l’y ajouter, on parlera de maintenance perfective. Toutes les nouvelles demandes des utilisateurs sont de ce type.

**6.4 Maintenance préventive**

En plus de ces trois catégories de maintenance, l’IEEE a défini un quatrième type de maintenance : la maintenance préventive. Elle peut être définie comme la maintenance exécutée afin d’empêcher des problèmes avant qu’ils surviennent. Ce type de maintenance peut être considéré comme très important dans le cas de systèmes critiques tels que ceux liés à l’aéronautique par exemple.

1. **Processus de la maintenance**

Les modèles de cycle de vie du logiciel sont assez bien connus mais peu de gens sont au courant de l’existence de modèles propres au processus de la maintenance. En 1993, l’IEEE sur une idée de Schneidewind [[3]](#footnote-3) a publié un processus itératif standard pour la gestion et l’exécution des activités de maintenance logicielle sous le nom de ”IEEE Standard for Software Maintenance”. Le standard initie le processus de maintenance dès l’étape de livraison du produit à un utilisateur. La figure 3 présente le processus de maintenance selon l’IEEE.



**Figure 3**. Processus de maintenance selon l’IEEE

**Classification et Identification**

Le processus démarre par la soumission d’une requête de modification (MR). L’entreprise de maintenance détermine tout d’abord le type de requête dont il s’agit (corrective, perfective ou adaptative). Une priorité est ensuite assignée à cette requête ainsi qu’un numéro unique. Toutes ces informations sont entrées automatiquement dans une base de données qui pourra être consultée tout au long du cycle.

**Analyse**

Une étude de faisabilité ainsi qu’une analyse détaillée sont effectuées durant cette phase. Le mainteneur prêtera attention aux impacts des modifications, il proposera plusieurs solutions alternatives et définira les coûts. L’analyse détaillée fournira différentes informations telles que des plans d’implémentation, des stratégies de test ou encore des identifications de fonctionnalités. A la fin de cette période d’analyse, l’ensemble des changements doit être totalement planifié.

**Conception**

Durant la phase de conception, l’ensemble des informations collectées pendant la phase précédente sont rassemblées et utilisées afin de construire le design de l’application. En fin de conception, la documentation du logiciel doit être mise à jour (plans de tests, design, analyse et exigences).

**Implémentation**

La phase d’implémentation comprend l’ensemble des processus de codage, de tests unitaires, d’intégration, d’analyse des risques et de revue de code. Toutes les documentations concernant le logiciel sont mises à jour.

**Test Système**

La phase des tests du système s’assure que l’ensemble des exigences originelles de l’utilisateur sont toujours rencontrées de même que les nouvelles ajoutées. La phase de tests est l’une des plus importantes du cycle car elle permet de s’assurer que le nouveau produit est toujours satisfaisant et que les nouveaux changements ont été implémentés correctement.

**Test d’acceptation**

La phase de test d’acception s’établit sur un système totalement intégré et est réalisée par les clients, utilisateurs ou par un tiers parti. Les testeurs ont pour objectif de rapporter les erreurs et de déterminer si les fonctionnalités du système sont en accord avec leurs exigences.

**Livraison**

Durant cette phase finale, le fournisseur livre le nouveau système aux utilisateurs. Il effectue la configuration du système et la formation des utilisateurs. Après cette phase, le système est prêt à fonctionner de manière autonome.

1. **Les problèmes et la mesure en maintenance du logiciel**

**8.1 Problématiques principales de la maintenance**

Afin d’assurer une maintenance efficace, un certain nombre de problématiques doivent être traitées.

La figure 4 présente les différents sujets groupés.



**Figure 4**. Problématiques principales en maintenance du logiciel

**8.1.1 Problèmes techniques**

**Compréhension limitée**

Par ”compréhension limitée”, on entend la rapidité avec laquelle quelqu’un peut comprendre ou faire un changement ou une correction dans un logiciel sur lequel cet individu n’a pas travaillé. Ainsi les mainteneurs ont généralement une compréhension limitée du logiciel et un effort non négligeable doit être fait pour accroitre leur propre compréhension du logiciel.

**Essai**

Réaliser un essai complet sur un morceau majeur de logiciel est généralement significatif en terme de temps et de coûts. On constatera, d’ailleurs, l’impossibilité fréquente d’effectuer des tests sur des fonctions critiques d’un logiciel nécessitant la mise hors-ligne du système.

**Analyse d’impact**

Une analyse d’impact décrit comment mener, à un coût efficace, une analyse complète des impacts d’un changement dans un logiciel existant. Par une connaissance de la structure et du contenu d’un logiciel, les mainteneurs réalisent l’analyse d’impact, identifient tous les produits des systèmes et des logiciels affectés par une demande de modification de logiciel et établissent une estimation des ressources nécessaires au changement. Arthur [4] définit les objectifs de l’analyse d’impact comme :

– Détermination de la portée d’un changement pour établir un plan et implanter le travail ;

– Développement d’estimations justes des ressources nécessaires pour effectuer le travail ;

– L’analyse de coûts/bénéfices du changement demandé ;

– Communication aux autres de la complexité d’un changement donné.

**Maintenabilité**

L’IEEE définit la maintenabilité comme l’aisance avec laquelle un logiciel peut être maintenu, amélioré, adapté ou corrigé pour satisfaire aux exigences spécifiées. Ces sous-caractéristiques de maintenabilité doivent être spécifiées, revues et contrôlées durant les activités de développement logiciel pour réduire les coûts de maintenance. Un succès amenant la qualité du logiciel à s’améliorer est difficile à atteindre car les sous caractéristiques de maintenabilité ne sont généralement pas respectées durant le processus de développement.

**8.1.2 Problèmes de management**

**Alignement avec les objectifs de l’organisation**

On considère les objectifs de l’organisation comme le retour sur investissement des activités de maintenance.

**Embauche du personnel**

La maintenance n’est généralement pas vue comme un travail séduisant. Le personnel de la maintenance est souvent vu comme un personnel de seconde classe.

**Processus**

Par la notion de processus, on comprend l’unicité de certaines activités de maintenance. On notera les nombreuses activités en commun avec le développement de logiciel mais néanmoins il existe plusieurs activités qu’on ne trouve pas dans le développement logiciel et posant un certain nombre de défis au manager.

**Aspects organisationnels de la maintenance**

Les aspects organisationnels décrivent comment identifier quelle organisation et/ou fonction qui sera responsable pour la maintenance du logiciel.

**Externalisation**

Un des défis majeurs pour ceux qui fournissent le service externalisé est de déterminer la portée des services de maintenance requis et les détails contractuels. Un autre défi identifié est la transition du logiciel vers le fournisseur de services externalisés.

**8.1.3 Estimation des coûts de la maintenance**

**Estimation des coûts**

ISO/IEC 14764 énonce que ” les deux approches les plus populaires pour l’estimation des ressources pour la maintenance de logiciel sont l’utilisation de modèles paramétrés et l’utilisation de l’expérience”.

**Modèles paramétrés**

Les modèles paramétrés se basent sur les données des projets et discutent de tous les aspects de l’estimation des coûts, incluant les points de fonction.

**Expérience**

L’expérience, par le jugement d’expert est une approche utilisée pour enrichir les modèles paramétrés. On mettra en avant que la meilleure approche pour l’estimation de la maintenance est une combinaison d’expérience et de données empiriques fournies par différents programmes de mesure.

**8.1.4 Mesure de la maintenance du logiciel**

En général, il y a trois entités liées à la maintenance logicielle dont les attributs peuvent être soumis à la mesure : processus, produits et ressources.

* Un processus représente toute activité liée au logiciel, comme l’analyse des changements, la spécification, la conception le codage et les tests.
* Une ressource est une entrée pour le processus, comme par exemple le personnel, le hardware et le software.
* Un produit représente le résultat de sortie final ou intermédiaire d’un processus logiciel comme par exemple la documentation d’un système, le listing d’un programme,…

La relation entre ces trois entités est représentée par le diagramme de la figure 5.



**Figure 5.** Relation entre processus ressource et produit.

Deux facteurs peuvent être utilisés pour évaluer la maintenabilité d’un logiciel, (1) la compréhension et (2) la modifiabilité. La compréhension est le temps, T1, nécessaire par module pour déterminer les changements nécessaires. La modifiabilité est le temps nécessaire par module pour effectuer ces modifications.

Pour garantir un logiciel de qualité tout en assurant des coûts de test et de maintenance faibles, la complexité d'un logiciel devrait être mesurée dès le début du codage. Ainsi lorsque les valeurs recommandées sont dépassées, le développeur peut intervenir rapidement.

Pour quantifier la complexité d'un logiciel, on se sert de métriques. Les métriques peuvent être classées en trois catégories :

• celles mesurant le processus de développement ;

• celles mesurant des ressources ;

• et celles de l‘évaluation du produit logiciel.

Pour quantifier la complexité d’un logiciel, les mesures les plus utilisées sont les lignes de code (LOC acronyme de « lines of code »)

**Quelles sont les limites acceptables ?**

La longueur des fonctions devrait être de 4 à 40 lignes de programme. Une définition de fonction contient au moins un prototype, une ligne de code, et une paire d'accolades, qui font 4 lignes. En règle générale, une fonction plus grande que 40 lignes de programme doit pouvoir s’écrire en plusieurs fonctions plus simples. A toutes règles son exception, les fonctions contenant un état de sélection avec beaucoup de branches ne peuvent pas être décomposées en fonctions plus petites sans réduire la lisibilité. La longueur d’un fichier devrait contenir entre 4 et 400 lignes de programme, ce qui équivaut déjà à un fichier possédant entre 10 et 40 fonctions.

**Le nombre cyclomatique de Mc Cabe : v(G)**

La complexité Cyclomatique (complexité de McCabe), introduite par Thomas McCabe en 1976, est le calcul le plus largement répandu des métriques statiques. Conçue dans le but d’être indépendante du langage, la métrique de McCabe indique le nombre de chemins linéaires indépendants dans un module de programme et représente finalement la complexité des flux de donnés.

Il correspond au nombre de branches conditionnelles dans l’organigramme d’un programme.

Le nombre cyclomatique évalue le nombre de chemins d’exécution dans la fonction et ainsi donne une indication sur l’effort nécessaire pour les tests du logiciel. Pour un programme qui consiste en seulement des états séquentiels, la valeur pour v(G) est 1.

v(G)=e-n+2

Où n= le nombre de nœuds ; e= le nombre d’arcs ; et v(G) est la complexité cyclomatique.

Exemple



e=11 ; n=8 ,V(G)=5

**Quelles sont les limites acceptables pour le nombre cyclomatique v(G) ?**

Une fonction devrait avoir un nombre cyclomatique inférieur à 15. Si une fonction a plus que 15 chemins d'exécution il est difficile à comprendre et à tester. Pour un fichier le nombre cyclomatique ne devrait pas dépasser 100.

**Les Métriques de Halstead**

Les métriques de complexité de Halstead qui procurent une mesure quantitative de complexité ont été introduites par l’américain Maurice Halstead. Ils sont basées sur l’interprétation du code comme une séquence de marqueurs, classifiés comme un opérateur ou une opérande.

Toutes les métriques de Halstead sont dérivées du nombre d’opérateurs et d’opérandes :

• nombre total des opérateurs uniques (n1)

• nombre total des opérateurs (N1)

• nombre total des opérandes uniques (n2)

• nombre total des opérandes (N2)

Sur la base de ces chiffres on calcule :

• La Longueur du programme(N) : N = N1 + N2.

• La Taille du vocabulaire(n) : n = n1 + n2

A partir de là, on obtient le Volume du Programme(V) en multipliant la taille du vocabulaire par le logarithme 2 : V = N \* log2(n).

Le Niveau de difficulté (D) ou propension d'erreurs du programme est proportionnel au nombre d’opérateurs unique (n1) dans le programme et dépend également du nombre total d’opérandes (N2) et du nombre d'opérandes uniques (n2). Si les mêmes opérandes sont utilisés plusieurs fois dans le programme, il est plus enclin aux erreurs.

D = ( n1 / 2 ) \* ( N2 / n2 )

L'Effort à l’implémentation (E) est proportionnel au volume (V) et au niveau de difficulté (D). Cette métrique est obtenue par la formule suivante :

E = V \* D

Halstead a découvert que diviser l'effort par 18 donne une approximation pour le Temps pour implémenter (T) un programme en secondes.

T = E / 18

Les outils Testwell CMT++ et Testwell /CMTJava permettent de mesurer la complexité du code. Grace à l’analyse des fichiers non-préprocessés les outils Testwell sont extrêmement rapide. Même des grands projets peuvent être analysés en quelques minutes.

1. S.Moriguchi, (1996) Software Excellence: A Total Quality Management Guide, Productivity Press. [↑](#footnote-ref-1)
2. |  |
| --- |
|  E.B, Swanson. Proceeding |
| ICSE '76 Proceedings of the 2nd international conference on Software engineering  |
| Pages 492-497 IEEE Computer Society Press Los Alamitos, CA, USA ©1976 |

 [↑](#footnote-ref-2)
3. Schneidewind, N. F. (1993). Report on the IEEE standard for a software quality metrics methodology. Proceeding of the Conference on Software maintenance, 104-106. [↑](#footnote-ref-3)