UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'UNIVERSITÉ DU QUÉBEC À TROIS-RIVIÈRES

COMME EXIGENCE PARTIELLE
DE LA MAÎTRISE EN GÉNIE INDUSTRIEL

PAR
RIADH ARIBI

LA FIABILITÉ DES LOGICIELS : DÉVELOPPEMENT DE SUPPORTS POUR L'ANALYSE DES ERREURS ET LEURS EFFETS SUR LE LOGICIEL ÉTUDE DE CAS

SEPTEMBRE 2007

SOMMAIRE

Au cours des dernières décennies, les systèmes logiciels ont envahi tous les domaines de la vie actuelle et sont devenus les leviers de l'économie. De ce fait, la complexité, la taille et les coûts des logiciels ont augmenté exponentiellement. Par conséquent, davantage d'intérêt a été accordé à ces systèmes, d'une part, pour minimiser les coûts, les délais et la complexité et d'autre part, augmenter la qualité, la performance et la fiabilité de ces derniers.

Cette recherche consiste à développer un cadre d'analyse des défaillances, des défauts et des erreurs tout au long du cycle de vie d'un logiciel. Ce cadre d'analyse pourrait être utilisé comme un support pour identifier et prévenir les modes de défaillances potentiels. Il pourrait aussi procurer aux personnes chargées des tests et de validation une synthèse de la criticité des modules du logiciel pour affiner leur démarche.

Dans un premier temps, une recherche bibliographique sur les concepts de la fiabilité des logiciels a été faite. Cette étude débouchera sur l'importance de la compréhension de la dynamique des erreurs, défauts et défaillance pour améliorer la fiabilité et minimiser les coûts d'un logiciel.

Dans un deuxième temps, la méthode "Analyse des Erreurs et de leurs Effets sur le Logiciel" a été appliquée sur un cas réel de développement d'un système de gestion intégrée amenant à développer des outils qui permettent d'améliorer la fiabilité des logiciels. Ces outils consistent en des listes et des listes de vérification (check-lists) basées sur une analyse des modes de défaillance les plus critiques et les plus fréquents. Cette méthodologie permettra de constituer

une grille d'analyse qui aide à identifier les défaillances potentielles et les relier aux causes et aux moments d'introduction des erreurs dans le programme.

L'application de la méthode "Analyse des Erreurs et de leurs Effets sur le Logiciel" sur l'étude de cas a donné de bons résultats dans l'ensemble. D'abord, le classement des modes de défaillances selon plusieurs familles, à savoir : Entrée, Sortie, Fonctionnalité, Paramétrage, Ergonomie, Informations, Données, Règles de gestion, Performances, Traitement des données et Divers. Ensuite, la détermination des modes de défaillance les plus fréquents et ainsi une liste des modes de défaillance génériques.

REMERCIEMENTS

Je tiens à remercier mon directeur de recherche, M. Georges ABDULNOUR, pour son appui précieux, l'orientation de mon projet et sa patience durant mes travaux de recherche. Je tiens également à remercier mon codirecteur M. Jocelyn DROLET, pour son intérêt, sa compréhension, ses conseils et surtout de m'avoir fourni les données nécessaires pour réaliser cette étude.

Je tiens à remercier tous ceux et celles qui m'ont encouragé et aidé, de loin ou de près, tout au long de mon cheminement de maîtrise.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SYMBOLES ET ABRÉVIATIONS

AEEL :	Analyse des Erreurs et leurs Effets sur le Logiciel
AMDE :	Analyse des Modes de Défaillance et de leurs Effets
AMDEC :	Analyse des Modes de Défaillance, de leurs Effets et leur Criticité
AVAIL :	Availability
COCOMO II :	COnstructive COst MOdel II
DCC :	Défaillances de Cause Commune
ERP :	Entreprise Resource Planning
IEEE :	Institute of Electrical and Electronics Engineers
KSLOC :	Kilo-SLOC, Thousand Source/Software Lines Of Code
MAJ :	Mise À Jour
MTBF :	Mean Time Between Failures
MTTF :	Mean Time To Failure
ND :	Non-Détection
NHPP :	Non-Homogeneous Poisson Process
PME :	Petites et Moyennes Entreprises
POFOD :	Probability Of Failure On Demand
ROCOF :	Rate Of Occurrence Of Failures
SRGM :	Software Reliability Growth Model
Std. :	Standard

CHAPITRE 1

INTRODUCTION GÉNÉRALE

Ce chapitre introduit l'étude, son contexte, sa problématique et son objectif. Il sera aussi question d'aborder la méthodologie adoptée, les résultats escomptés et les sources de la revue de la littérature.

1.1. Mise en contexte

Avec la conjoncture actuelle, plusieurs mutations économiques et sociales se produisent, notamment la mondialisation, le commerce électronique, les technologies de l'information et de la communication, etc. L'essor technologique est de plus en plus centré sur l'utilisation des logiciels qui jouent, aujourd'hui, un rôle essentiel dans toutes les activités commerciales. Le succès d'une entreprise dépend de sa capacité à intégrer, développer et maintenir les logiciels et les systèmes informatiques. Dans ce monde, où les ressources sont réparties et les problèmes de conformité à la réglementation sont courants, il est impératif de pourvoir à la gouvernance du développement, qui fait partie intégrante des processus commerciaux fondamentaux.

En effet, l'utilisation des ordinateurs et des logiciels a envahi toutes les activités et ils sont devenus les leviers de l'économie actuelle. Ils sont utilisés en télécommunications, aviation, automobiles, constructions, médecine, etc. De ce fait, la complexité et la taille des logiciels ont augmenté exponentiellement, ce qui a causé l'augmentation simultanée des coûts des logiciels. On estime qu'environ 80% du coût de développement des logiciels va pour l'identification et la correction des défauts [4].

Le coût des logiciels en tant que pourcentage du coût total d'un système informatique, continue à augmenter, contrairement aux coûts du matériel [5]. Un logiciel est mis au point par des humains, donc des erreurs sont possibles. L'évaluation quantitative de la qualité d'un logiciel peut être conduite par plusieurs approches. Cependant, il est parfois difficile pour les gestionnaires de projet de mesurer la qualité d'un projet ainsi que sa productivité. Néanmoins, la fiabilité est l'un des attributs les plus importants de la qualité des logiciels, vu qu'elle quantifie les défaillances du logiciel durant le processus de développement.

Le présent mémoire est constitué de 7 chapitres.

Le premier chapitre introduit l'étude, son contexte, sa problématique et son objectif et aborde la méthodologie de recherche adoptée et les résultats escomptés, tout en présentant les sources de la revue de la littérature.

Les chapitres 2 et 3 sont consacrés à une étude approfondie des concepts, mesures et méthodes d'estimation de la croissance de fiabilité et la modélisation des coûts des logiciels.

Le chapitre 4 consistera en une étude sur les erreurs, les défauts et les défaillances. Dans un premier temps, l'accent est mis sur les liens de causes à effet entre ces derniers pour aboutir, dans un deuxième temps, à la présentation de la démarche de développement de l'"Analyse des Erreurs et de leurs Effets sur le Logiciel" (AEEL).

Les chapitres 5 et 6 sont consacrés à une étude de cas ainsi que la présentation et l'analyse des résultats. En effet, l'application de la méthode AEEL sur un cas réel débouche sur l'élaboration d'une liste de modes de défaillance génériques, défaillances critiques et des grilles d'analyse.

Le chapitre 7 est consacré à la conclusion générale, les limites et les éventuelles perspectives de cette étude.

1.2. Problématique

Au fil des années, le coût des systèmes matériels est à la baisse contrairement à celui des logiciels qui augmente continuellement. Un grand pourcentage du coût du logiciel va, de plus en plus, vers la maintenance, l'identification et la correction des défauts dans les logiciels (figure 1).

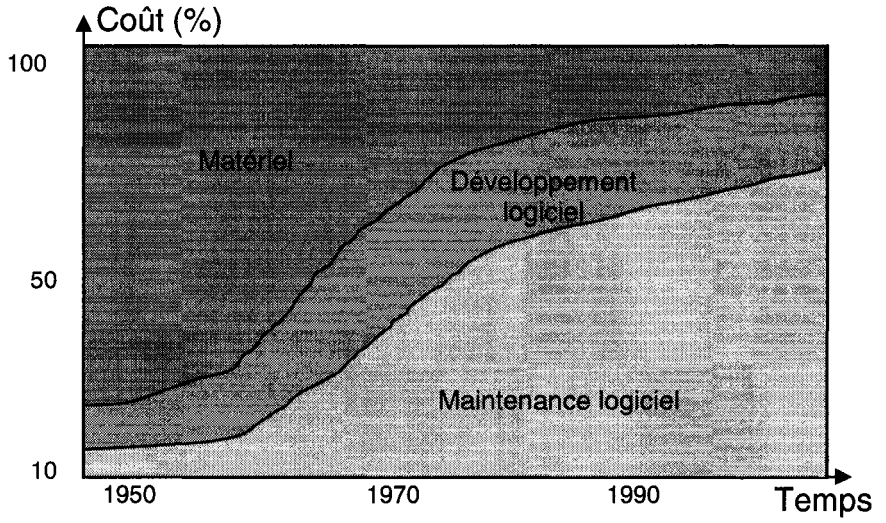

Figure 1 : Coût matériel versus coût logiciel [6].

Afin de minimiser le coût d'un système informatique, il s'avère plus intéressant de minimiser les coûts de logiciels. Il existe une forte corrélation entre le développement et la maintenance des logiciels. En effet, un logiciel "mal développé" nécessitera un plus grand effort de maintenance pour regagner ses fonctionnalités, d'où un effort supplémentaire qui générera une augmentation du coût total. Il paraît donc plus rentable de se doter des techniques appropriées pour mener à bien la phase de développement. Ces techniques peuvent être spécifiques aux logiciels ou adaptées des outils d'analyse du matériel comme l'"Analyse des Modes de Défaillance et de leurs Effets" (AMDE) et l'"Analyse des Modes de Défaillance, de leurs Effets et leur Criticité" (AMDEC).

1.3. Objectifs

L'objectif de cette étude est de développer un cadre d'analyse des défaillances, des défauts et des erreurs tout au long du cycle de vie d'un logiciel débouchant sur des supports pour identifier et prévenir les modes de défaillances potentiels. Il s'agit de proposer aux personnes chargées des tests et de validation une synthèse de la criticité des modules du logiciel analysé ce qui leur permettrait certainement d'affiner leur démarche.

1.4. Résultats escomptés

Les résultats visés par cette recherche sont :

- Mieux développer la méthode Analyse des Erreurs et de leurs Effets sur les Logiciels.
- Classer les modes défaillances selon des familles.
- Déterminer les modes de défaillances les plus fréquents et ainsi une liste des modes de défaillance génériques.
- Présenter, sous forme de listes de vérification, d'une part, les liens entre les défaillances et leurs causes, et d'autre part, relier les causes aux phases de cycle de vie du logiciel où l'erreur pourrait être introduite.

1.5. Méthodologie de la recherche

Le type de recherche employée dans cette étude est la recherche action. La recherche action est une méthode dont la finalité est la recherche et l'action [26], ce qui entraîne une formation qui se manifeste par l'explication de phénomène et/ou l'application sur le terrain. Le processus de la recherche action est présenté dans la figure 2.

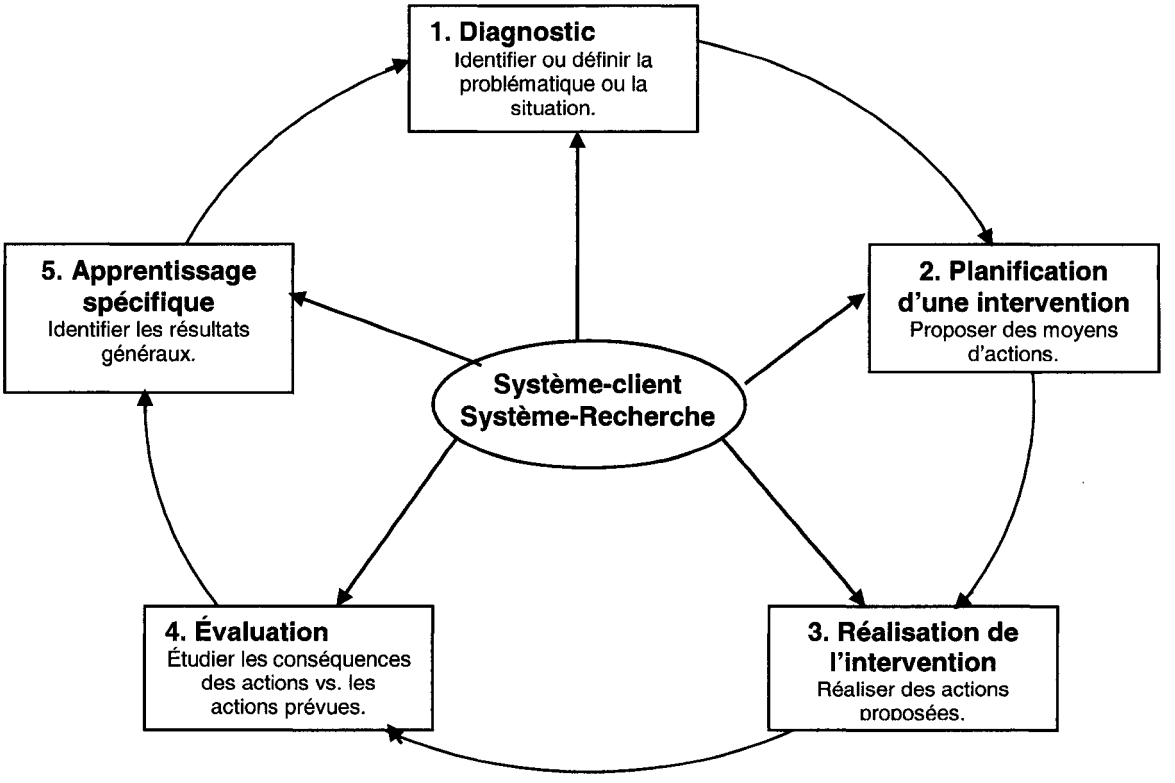

Figure 2 : Processus de la recherche-action [26].

Selon la figure 2, le processus de la recherche action est cyclique et est formé de 5 sous-processus, à savoir :

1. Le processus diagnostic : le système-client et le système-chercheur collaborent pour déterminer les caractéristiques ou les dimensions d'un phénomène ou d'une situation.
2. Le processus planification : le système-client et le système-chercheur collaborent au diagnostic et à la planification d'un projet, d'une intervention ou d'une stratégie.
3. Le processus réalisation : le système-client et le système-chercheur collaborent pour l'application effective de ce qui a été planifié.
4. Le processus évaluation : le système-client et le système-chercheur collaborent pour formuler un jugement « avant et après » sur une intervention, pour en évaluer la pertinence et les effets, le plus souvent en terme d'effectivité, d'efficience et d'efficacité.
5. Le processus apprentissage spécifique : le système-client et le système-chercheur collaborent à toutes les phases du cycle afin d'élaborer une expérience qui définit la situation, planifie des actions, les réalise et évalue leurs effets.

Dans le cadre de cette étude, le processus diagnostic consiste à faire un recensement des écrits dans le domaine de la fiabilité des logiciels et une étude du monde réel. Plus spécifiquement, rechercher tout ce qui est en relation étroite avec les analyses de défaillances, erreurs et défauts durant le cycle de vie d'un logiciel. Quant au processus de planification, il consiste à proposer un support d'aide à la prévention des défaillances potentielles dès la phase de gestation, de sorte à augmenter la fiabilité tout en respectant un budget et des délais donnés. Les processus réalisation et évaluation visent à appliquer dans un cas réel ce qui a été planifié, et de mesurer les écarts avec les objectifs prédéterminés. Enfin, le processus apprentissage consiste à ajuster les résultats et à élaborer une expérience spécifique dans le cas d'étude.

1.6. Sources de la revue de la littérature

Les documents consultés se limitent à l'ensemble des ouvrages en anglais et en français qui font partie de la collection des écrits physiques et virtuels des bibliothèques universitaires et nationales nord-américaines. Les principales sources d'information sont : les journaux académiques, les mémoires et dissertations, les périodiques professionnels, les manuels scolaires ainsi que les traités professionnels.

La recherche documentaire a nécessité l'accès aux banques de données informatisées et aux bibliothèques universitaires du réseau de l'Université du Québec. Les banques de données consultées sont principalement : "Science Direct", "Willey Science", "Emerald", "Kluweronline" et "ABI Inform". Les principaux volumes et périodiques consultés sont : "Journal of Systems and Software", "European Journal of Operational Research" et les Standards IEEE.

Pour chacune des bases de données recensées, le travail consiste à identifier des écrits scientifiques sur le cadre conceptuel étudié. Lors de la recherche sur les banques de données, les mots clés utilisés sont : "Software + Reliability" et "Software + engineering" avec les différentes combinaisons des mots : "Cost", "fault", "error", "failure" et "AEEL". De même, ces termes sont traduits en français surtout pour la recherche sur Internet et sur le réseau de l'Université du Québec (Manitou).

Les articles scientifiques et documents jugés les plus pertinents dans le cadre de cette recherche sont : « Software engineering » [29], « AMDEC/AMDE/AEEL : l'essentiel de la méthode » [8], « IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software » [9] et « Fiabilité du logiciel : concepts, modélisation, perspectives » [7].

CHAPITRE 2

LA FIABILITÉ DES LOGICIELS : CONCEPTS DE BASE ET TERMINOLOGIE

Dans ce chapitre, les éléments suivants seront présentés : les concepts de la fiabilité des logiciels, les différences entre la fiabilité des systèmes matériels et des systèmes logiciels, le cycle de vie et les modèles de développement des logiciels, les attributs de la fiabilité et les concepts de la complexité de logiciels.

2.1. Définitions de base

Quelques définitions de termes reliés à la fiabilité des logiciels sont avancées. Ces définitions sont traduites, en général, de la norme IEEE Std 982.2-1988 [9].

La fiabilité de logiciels : La probabilité qu'un logiciel ne cause pas une défaillance du système durant un temps spécifié et sous des conditions données. Elle peut s'exprimer par :

- La probabilité de non-défaillance pendant une durée t,
- Le taux de défaillance,
- Le temps moyen entre deux défaillances,
- La durée moyenne de fonctionnement après réparation,
- Le temps moyen de fonctionnement avant la première défaillance ou avant défaillance.

Le management de la fiabilité des logiciels : Le processus d'optimisation de la fiabilité des logiciels par un programme qui appuie la prévention des erreurs, la détection et la suppression des défauts et l'utilisation des métriques pour

maximiser la fiabilité, et ce, en respect des contraintes du projet telles que les ressources, les procédures et la performance.

La disponibilité de logiciels : aptitude d'un système logiciel à être en état d'accomplir une fonction requise dans des conditions de temps déterminées

La maintenabilité : aptitude du système logiciel au diagnostic et à la localisation d'un défaut ainsi qu'à sa remise en état de fonctionnement. Elle se mesure par :

- Le taux de réparation,
- Le temps moyen avant la remise en service,
- Le temps moyen de réparation.

Défaut : non-satisfaction aux exigences de l'utilisation prévue. Un défaut peut rester non détecté pour une longue période de temps.

Erreur : différence entre une valeur ou un état calculé, observé ou mesuré et la valeur ou l'état réel, spécifié, qui est en principe correct. L'erreur est générée par le concepteur ou le programmeur.

Défaillance : est un écart entre le résultat attendu et le résultat obtenu lors de l'utilisation. L'utilisateur est emmené à déterminer la sévérité du niveau de la défaillance selon son effet sur le système. Il est sous-entendu ici que la criticité de la défaillance varie d'un système à l'autre et d'une application à l'autre.

Mode de défaillance : Effet par lequel une défaillance est observée.

Défaillance de cause commune : défaillance de deux composants ou plus due à une seule cause.

Complexité : caractéristique du logiciel portant sur la difficulté à être analysé et à être compris.

2.2. Différence entre la fiabilité des hardware et software

La fiabilité des logiciels est inspirée en grande partie de la fiabilité des systèmes matériels. Ceci est dû au fait que le domaine de la fiabilité des systèmes matériels est en phase de maturité et a fait l'objet de multiples études. Cependant, plusieurs métriques utilisées pour la mesure de la fiabilité des systèmes matériels ne sont toujours pas utilisées pour les logiciels vu la différence de la nature des défaillances pour les logiciels et les systèmes matériels [29]. En effet, une défaillance pour un système matériel entraîne l'arrêt de fonctionnement du composant et requiert une réparation. Quant au logiciel, une défaillance se manifeste par un ensemble d'entrées, le logiciel peut continuer à fonctionner.

Les différences intrinsèques entre un système matériel et un système logiciel viennent du fait qu'un système matériel est monté à partir d'un nombre d'objets physiques différents qui sont fabriqués avec une certaine tolérance. Un système matériel quittant une ligne de production est différent de ces voisins sortant de la même ligne. Cependant, un logiciel est produit uniformément. De ce fait, il est nécessaire de bien travailler le prototype durant les phases de design, codage et test.

Le tableau 1 résume la comparaison entre un produit matériel et un produit logiciel. La comparaison est faite par rapport à la durée d'utilisation et à l'origine de la défaillance [7].

Tableau 1 : Différence entre produit "Matériel" et produit "Logiciel"

Différences	Produit "Matériel"	Produit "Logiciel"
Par rapport à la durée d'utilisation	La fiabilité décroît avec le temps si on ne répare pas Les causes sont engendrées différemment dans le temps	
		La fiabilité croit avec le temps si on corrige les défauts
Par rapport à l'origine de la défaillance	Les causes de défaillances prennent naissance pendant la conception et la production	
	Les causes de défaillances pour un matériel en utilisation sont dues aux dégradations physico-chimiques.	
	L'obsolescence peut être la conséquence des effets des défaillances de ceux de la mode ou bien du progrès technologique.	

2.3. Cycle de vie d'un logiciel

En général, les étapes du cycle de vie d'un logiciel se résument comme suit [7] :

Phase 0 : Élaboration d'un cahier des charges fonctionnel, qui est un document dans lequel les demandeurs, clients, utilisateurs expriment leurs besoins,

Phase 1 : Élaboration des spécifications, les spécifications reflètent les engagements du réalisateur sur les performances attendues ou révisées en accord avec le client et l'utilisateur avant la phase développement,

Phase 2 : conception préliminaire, mettre en évidence les fonctions requises sous forme modulaire,

Phase 3 : conception détaillée, détaille les modules présentés dans la phase précédente,

Phase 4 : codage, traduction des données et des algorithmes dans le dossier de conception détaillée dans un langage de programmation prévu,

Phase 5 : tests unitaires, test des composants logiciels séparément avant leur intégration dans l'unité logicielle,

Phase 6 : test d'intégration, introduction des composants logiciels, un par un, fonction par fonction, dans l'unité logicielle. Chacune des fonctions est testée séparément puis globalement,

Phase 7 : intégration dans un environnement cible simulé, validation du logiciel, banc de test,

Phase 8 : installation sur site opérationnel et qualification, le logiciel est soumis à de nouvelles agressions, notamment au travers d'éléments matériels pouvant eux-mêmes être défaillants,

Phase 9 : exploitation, le logiciel est livré. Les acteurs sont les utilisateurs finaux et les développeurs. Les utilisateurs sont intéressés par la disponibilité et le bon fonctionnement du logiciel alors que les développeurs portent leur attention sur les erreurs et les fautes qui sont à l'origine des défaillances.

Dans la littérature, les modèles de développement de logiciels sont classés en deux grandes catégories : Spirale et Waterfall. Dans la catégorie Waterfall, plusieurs modèles sont avancés [11] : modèle en cascade, modèle en cascade avec retour, modèle incrémental, modèle par processus, etc.

La fiabilité du logiciel se construit tout au long du cycle de vie du logiciel. Trois différents concepts peuvent être identifiés [7] :

- De la phase 0 jusqu'à la phase des tests unitaires, la fiabilité est apparentée à la fiabilité humaine. En effet, les tâches sont typiquement intellectuelles et créatives.
- De la phase test d'intégration jusqu'à la phase de qualification sur le site d'utilisation, le produit est "vivant" avec ses caractéristiques évolutives, "rapides" et dont les valeurs sont précises.
- Au cours de l'exploitation, les caractéristiques peuvent se dégrader dans leur ensemble. Cette dégradation n'est pas une fonction du temps et est due aux erreurs résidentielles et aux changements apportés au logiciel lors de la phase de la maintenance [19]. Les changements peuvent avoir lieu pour corriger les défauts ou modifier le code pour tenir compte de

nouvelles exigences, spécifications ou pour améliorer la performance du logiciel.

Après analyse du cycle de vie d'un logiciel, il est possible de le résumer en cinq phases, et ce, selon les activités : analyse, design, programmation, test et opération. Une étude a montré que les phases : analyse, design, programmation et test consomment, respectivement, 25%, 18%, 36% et 21% de l'effort de développement [31].

Approximativement, les coûts de développement représentent 60% du coût d'un logiciel, 40% sont associés à des coûts de test et d'intégration [29]. Les coûts des activités varient selon le type du système à développer et les exigences telles que performance et fiabilité du système. Selon Sommerville, la distribution des coûts dépend du modèle de développement adopté comme indiqué dans la figure 3.

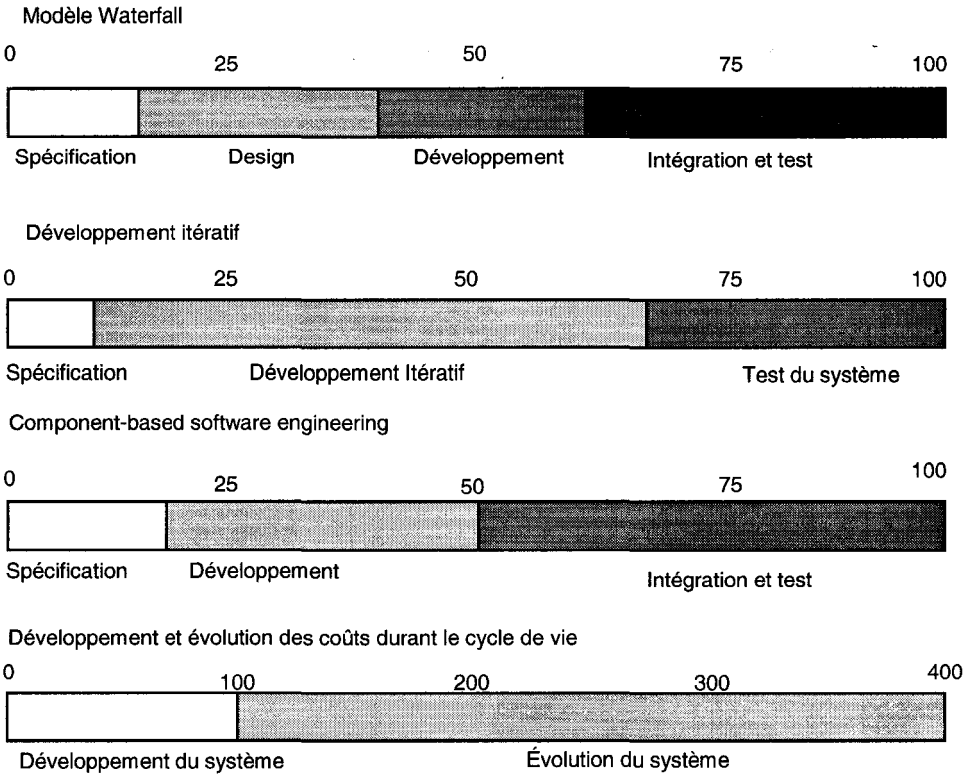

Figure 3 : Distribution des coûts par activité [29].

2.4. Modèles de développement des logiciels

Le processus de développement des logiciels est un processus très complexe et dynamique [19]. En effet, il implique entre autres des humains, des systèmes fondamentaux et des applications de nature différente qui varient d'un projet à l'autre. Il est communément admis qu'il est nécessaire d'augmenter la fiabilité de logiciel tout en éliminant des erreurs faites pendant le développement de logiciel. Les recherches académiques et les industries ont répondu à ce besoin en améliorant les méthodes de développement connues sous le nom de génie logiciel (Software Engineering), et en utilisant les contrôles systématiques pour détecter les erreurs dans le logiciel pendant le processus de développement.

La fiabilité des logiciels est influencée par le processus de développement utilisé [29]. Les processus répétitifs orientés vers l'action d'éviter les défauts sont susceptibles de développer un logiciel fiable. Lee et al. se sont basés sur le Processus de Hiérarchie Analytique (AHP) pour développer une méthodologie de développement des logiciels basée sur le concept de l'intervalle de comparaison et une optimisation stochastique [13].

Une spécification formelle et une preuve ne garantissent pas que le logiciel soit fiable [7, 12]. En effet, une spécification peut ne pas refléter les besoins réels des utilisateurs du système, la preuve peut contenir des erreurs et peut assumer un modèle d'utilisation incorrect.

Les métriques utilisées pour la spécification de la fiabilité des logiciels sont données dans le tableau 2.

Tableau 2 : Métriques utilisées pour la spécification de la fiabilité des logiciels [29].

Métrique	Signification
La probabilité de défaillance sur demande (POFOD)	Mesure de la probabilité de défaillance d'un système sur demande de fonctionnement.
Taux de l'occurrence des défaillances (ROCOF)	Mesure de la fréquence de l'occurrence avec laquelle le comportement inattendu est susceptible de se produire.
Temps moyen entre défaillances (MTBF)	Mesure du temps entre deux défaillances successives observées.
Disponibilité (AVAIL)	Mesure de la façon dont (probablement) le système doit être disponible pour l'usage.

Le choix de ces métriques peut varier d'un module à un autre pour un même programme.

2.5. Les attributs de la fiabilité des logiciels

Selon la norme IEEE Std 982.1-1988, la prévention des erreurs et des défauts peut-être soutenue par [9] :

- ✓ Un personnel habilité et compétent,
- ✓ Bonne étude des besoins exprimés et explicites des utilisateurs potentiels,
- ✓ Utilisation d'un prototype,
- ✓ Utilisation de techniques modernes (méthodes, langages et outils automatisés) pour améliorer la productivité et la qualité tout au long de cycle de vie du logiciel.

Dans une étude exploratoire faite par Pham et Zhang, les facteurs affectant le plus la fiabilité des logiciels sont : la complexité du programme, la compétence du programmeur, l'effort de test, la couverture des tests, l'environnement du test et la fréquence de changement des spécifications du programme [23, 30].

La norme IEEE Std 982.2-1988 propose un modèle qualité et une liste des facteurs qualité influençant la fiabilité des logiciels par phase du cycle de vie [9].

Le tableau 3 résume les attributs qualité ayant un impact sur la qualité du logiciel. La considération appropriée de ces facteurs lors du développement des logiciels permettrait la prévention et la détection des erreurs et défauts [9].

Tableau 3 : Les facteurs qualité influençant la fiabilité des logiciels.

Phase	Facteurs
Concept (idée générale)	Les besoins/objectifs de l'utilisateur (fonctionnalité, performance, perfection, consistance) Documentation standard.
Exigences	Adhérence au besoin. Architecture. Environnement opérationnel. Perfection. Facilité d'utilisation (ergonomie) Documentation standard.
Design	Complexité. Modularité. Interfaces. Expansibilité. Ranger selon la taille. Documentation standard. Perfection (état complet)
Réalisation	Complexité. Interfaces. Standards de développement. Perfection (état complet). Maintenabilité.
Test	Couverture fonctionnelle (information, processus). Couverture topologique. Interfaces (logiciel-logiciel, logiciel-matériel). Mesures des performances.
Opération et maintenance	Intégrité des changements Couverture du test Facilité de l'étude, Facilité d'utilisation
Retirement	Transférabilité, conversion, migration

2.6. La complexité des logiciels

La complexité est une caractéristique du logiciel portant sur la difficulté à être analysé et à être compris [16, 28]. La complexité peut se manifester à différents niveaux : architecture, algorithmique/logique, technologique, interfaçage, forme textuelle et vocabulaire, présentation des données, etc.

Menzies et al. donnent une liste des mesures et des indicateurs de la complexité [17]. McCabe définit, en se basant sur la théorie des graphes, des mesures de la complexité des logiciels [16]. Les mesures proposées pour un graphe "g" sont :

- ✓ La complexité de structure $Cs=\frac{m}{n}$
- ✓ Le nombre cyclomatique du graphe $V(g)=m-n+1$
- ✓ La densité de contrôle $Dc=\frac{V(g)}{n}$

avec : m : nombre d'arcs.
n : nombre de sommets.

Un programme en cours d'exécution distribue ses activités sur l'ensemble de ses modules et fonctions. Le balayage de ces derniers ne se fait pas d'une manière égale lors de l'exécution des modules, ce qui définit trois types de complexité dynamique : fonctionnelle, d'exécution et modulaire [3, 18].

- ✓ La complexité fonctionnelle

Un programme est destiné à achever un ensemble de fonctions. L'utilisateur ne fait pas appel aux différentes fonctions à la même fréquence. Généralement, l'exécution des fonctions se fait mutuellement.

- ✓ La complexité d'exécution

Une fonctionnalité dans un programme est accomplie par un ou plusieurs modules. En effet, pour achever une fonctionnalité, le programme est amené à balayer les différents modules, d'où le concept de la complexité d'exécution.

- ✓ La complexité modulaire

C'est la probabilité non conditionnelle qu'un module particulier soit exécuté selon la conception du programme.

Les trois types de complexité doivent être considérés lors du développement d'un logiciel, vu que la probabilité de défaillance d'un module est reliée à sa complexité [18]. La fréquence des défaillances est plus importante pour les fonctions principales du logiciel, vu qu'elles sont plus sollicitées par les utilisateurs. D'où l'intérêt de minimiser la complexité fonctionnelle. La décomposition du système logiciel en modules et fonctions et la définition minutieuse des différentes interfaces entre ces derniers permettent de contrôler la complexité d'exécution et la complexité modulaire et facilitent la localisation des erreurs suite à une défaillance.

CHAPITRE 3

LA MODÉLISATION DE LA FIABILITÉ DES LOGICIELS

Il s'agit dans ce chapitre de présenter les mesures, les méthodes d'estimation de la croissance de la fiabilité des logiciels et les modèles d'estimation du coût d'un logiciel.

La modélisation de la fiabilité de logiciels est utilisée pour l'estimation et la prédiction de la fiabilité des logiciels. La détermination ou l'évaluation de la fiabilité d'un logiciel aide à prendre des décisions telles que l'arrêt des tests et la délivrance du logiciel. Les modèles utilisent les caractéristiques du processus de développement des logiciels dès la phase de gestation à la phase test et extrapolent ces informations pour prédire le comportement du logiciel durant son exploitation [2].

Les modèles de fiabilité sont un outil puissant pour la prédiction, le contrôle et l'évaluation de la fiabilité. Plusieurs modèles pour la mesure de la fiabilité des logiciels ont été développés [4, 14, 15, 16, 20, 21, 22, 24, 28, 30].

3.1. Mesure de la fiabilité des logiciels

De nos jours, les logiciels sont devenus plus complexes. Ils consistent en plusieurs composants et modules qui peuvent être testés et réparés séparément avant le test du système. Les modules sont intégrés selon l'architecture du programme pour former des sous-systèmes qui, à leur tour, sont intégrés à d'autres composants et modules pour donner à la fin le système logiciel (figure 4). Les modules peuvent être développés à l'interne ou être procurés de

l'externe. La décomposition du système en modules et sous-systèmes permet de diminuer la complexité du logiciel et de faciliter l'analyse des erreurs et les tests.

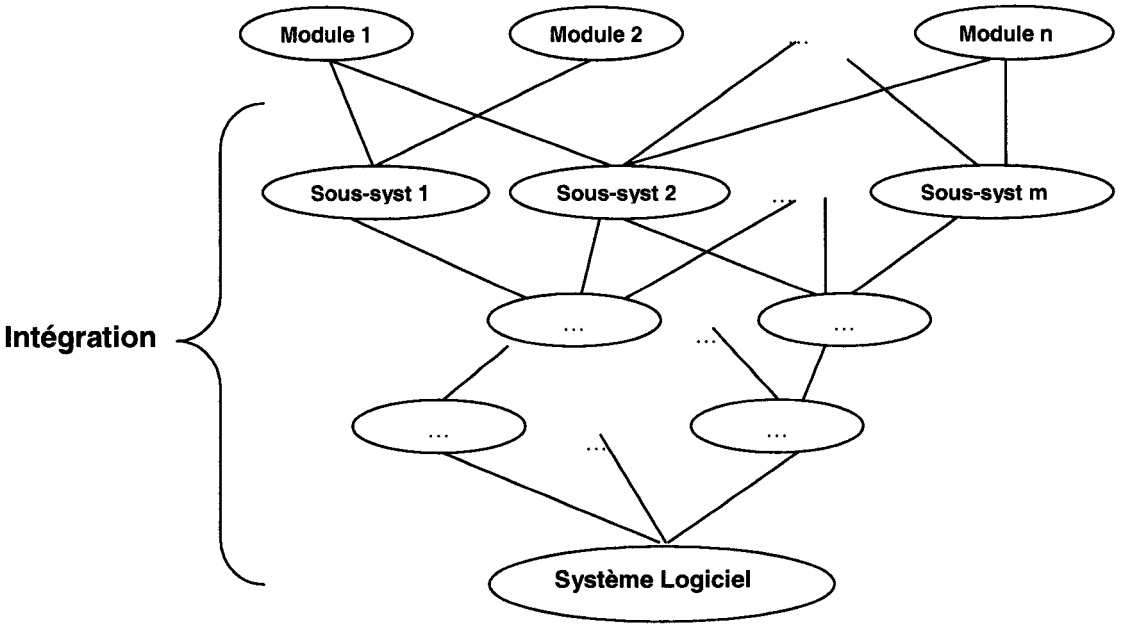

Figure 4 : Intégration des différents modules d'un système logiciel.

Le test est utilisé pour vérifier l'exactitude et la fiabilité du logiciel dans son environnement opérationnel prévu, basé sur la façon dont les utilisateurs l'emploient. Le processus de test d'un logiciel inclut l'unité, l'intégration et le test du système. Le logiciel est commercialisé après avoir été testé, réparé, validé et assuré qu'il répond aux spécifications et aux exigences inscrites dans le cahier de charges.

Les mesures de la fiabilité peuvent être effectuées sur le produit ou sur le processus. Les différentes "mesures produits" et "mesures processus" sont données dans le tableau 4. La qualité du produit dépend de la qualité de l'outil de mesure incluant la manière dont il est utilisé. La fiabilité d'un logiciel est influencée, en grande partie, par la fiabilité de l'outil (logiciel, langage de programmation, etc.) utilisé pour le développer.

Tableau 4 : Les mesures produit et les mesures processus [9].

Les mesures produit	Les mesures processus
✓ Erreurs, défauts, défaillances ✓ Temps moyen de défaillance, taux de défaillance ✓ Croissance de fiabilité ✓ Les défauts résiduels ✓ Perfection et uniformité ✓ Complexité	✓ Mesures de contrôle de gestion ✓ Mesure de la couverture du test, ✓ Risques, avantages et évaluation des coûts

Dans le développement des logiciels, le coût, l'échéancier et la fiabilité sont trois facteurs corrélés. Après un certain point, il est difficile d'augmenter la fiabilité sans augmenter soit le coût soit la durée ou les deux à la fois [29]. Plus la fiabilité augmente, plus le coût du système augmente exponentiellement comme indiqué dans la figure 5.

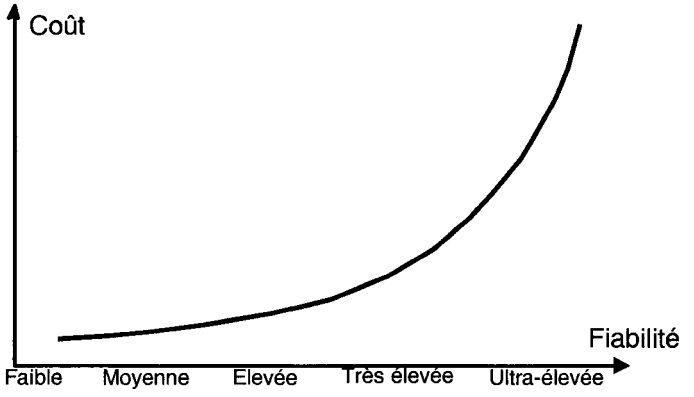

Figure 5 : Variation du coût en fonction de la fiabilité [29].

Augmenter la fiabilité nécessite de payer une pénalité. En effet, un logiciel fiable nécessite d'ajouter des codes au programme, parfois redondant, pour améliorer le contrôle et le test pour des cas exceptionnels du fonctionnement. Cependant, la fiabilité doit toujours précéder l'efficience pour les raisons suivantes [11] :

- ✓ Un logiciel non fiable peut atténuer la réputation de l'entreprise, ce qui endommage les futures ventes,
- ✓ Le coût des défaillances peut-être considérable,
- ✓ Les systèmes non fiables sont difficiles à améliorer,
- ✓ Un système non fiable peut causer une perte de données et/ou informations.

3.2. Les méthodes d'estimation de la croissance de la fiabilité des logiciels

Les modèles de prévision applicables sur les logiciels en phase de test ou en phase d'exploitation utilisent des méthodes différentes selon que le logiciel est ou n'est pas dans une période de croissance de fiabilité. Ils font appel à l'application des lois de probabilité appliquées à des phénomènes que les variables statistiques appropriées permettent d'étudier. Ces modèles se basent sur le comportement de défaillance pendant le test et l'extrapolent pour déterminer le comportement durant l'exploitation [2]. Les modèles applicables pendant une période de croissance de fiabilité sont les modèles de Poisson par morceau et les modèles basés sur le processus de Poisson non homogène.

3.2.1. Les modèles de Poisson par morceau

Ces modèles sont utilisés lorsque l'étude s'appuie sur les relations entre les taux de défaillance successifs.

- ✓ Modèles probabilistes : introduction d'une séquence de signaux aléatoires en entrée [7]. Par exemple : Modèle de Littlewood, Modèle de Littlewood-Veral [7, 14].

- ✓ Modèles déterministes : on fait une hypothèse sur les défauts à trouver et les tests sont effectués par simulation ou par génération aléatoire de défauts [7]. Par exemple : Modèle de Jelinski-Moranda, Modèle de Musa.

Ces modèles s'appuient sur 2 hypothèses :

- ✓ Le taux de défaillance instantané $\lambda(t)$ est proportionnel au nombre total de défauts résiduels,
- ✓ Les intervalles de temps entre défaillances sont statistiquement indépendants et obéissent à une loi de probabilité exponentielle.

La loi de poisson pour la croissance de fiabilité :

Hypothèses : Accroissement des intervalles indépendants,

Continuité, le nombre d'événements dans $[t_i, t_{i+1}]$.

Paramètres : $m=\lambda T$ moyenne des défauts par unité de temps.

$$\theta=\int_{i}^{t_{i+1}} \lambda(t)dt$$

La solution est combien de défaillances peuvent avoir lieu entre t_i et t_{i+1}.

3.2.2. Le Processus de Poisson Non Homogène (NHPP)

C'est un processus aléatoire dont la distribution de la probabilité varie avec le temps. Selon Pham, les modèles NHPP sont des outils qui ont tout à fait réussi dans la mesure de la fiabilité pratique de logiciels [19]. Depuis le début des années soixante-dix, plusieurs modèles de croissance de la fiabilité des logiciels ont été proposés pour l'évaluation de la croissance de la fiabilité durant le processus de développement des logiciels [15, 30].

Les hypothèses des modèles NHPP [7] :

- ✓ Le nombre de défaillances est une variable aléatoire,
- ✓ Les défauts sont détectés aléatoirement et indépendamment les uns des autres (probabilité identique de trouver chaque défaut),

La densité de la probabilité du nombre de défaillances, observée par intervalle de temps, est régie par la loi de Poisson dont la moyenne varie d'un intervalle à l'autre. Le nombre de défaillances observées sur un intervalle de temps donné est indépendant du nombre de défaillances observées sur tout autre intervalle de temps. Le temps qui s'écoule entre la (i-1)ème défaillance et la ième défaillance dépend du temps écoulé jusqu'à la (i-1) défaillance. La notion d'indépendance est attribuable à deux causes :

  - Les activations des défauts provoquées par des erreurs, quelles qu'en soient les origines, ne sont rencontrées qu'aléatoirement,
  - Les entrées successives ne sont pas corrélées.

- ✓ Il n'y a pas de corrélation entre le processus de défaillance et le processus de correction. Tous les défauts sont corrigés.
- ✓ Le nombre moyen de défaillance par intervalle décroît d'un intervalle à l'autre, selon une loi exponentielle.
- ✓ Le taux de défaillance est proportionnel au nombre de défauts résiduels.

Les principaux résultats du NHPP sont la fiabilité du système après n défaillances, le temps moyen et le nombre moyen (Espérance mathématique) de défaillances sur un intervalle et l'intensité de défaillance. Le NHPP a pour propriété que le taux de défaillance associé à un événement i a la même expression que le taux de défaillance et que seul l'instant de début de l'observation diffère [20]. Le tableau 5 donne une liste des résultats par modèle.

Tableau 5 : Les modèles NHPP et leurs résultats

Modèle	Résultats
Goel-Okumoto	La fiabilité après n défaillances. Le nombre de défaillances résiduelles au temps t. Le temps d'attente pour la prochaine défaillance.
Schneidewind	Le nombre moyen de défauts pour les intervalles postérieurs à un temps t. Le nombre de défauts détectés antérieurement à un intervalle de temps t. Le nombre total de défauts.
Shanthikumar	La fiabilité après n défaillances observées. Le nombre de défaillances résiduelles au temps t. Le nombre de défaillances détectées au temps t. Le temps pour trouver les défaillances non encore observées.
Jelinski-Moranda	Le temps de trouver les défaillances résiduelles. Le MTTF(1) après détection d'un certain nombre de défauts.
Musa	La fiabilité après n défaillances. Le MTTF(1), MTBF(2) Le temps pour retrouver les défaillances restantes. La fiabilité des composants logiciels en série. La fiabilité des composants logiciels en parallèle.
Keiller-Littlewood, Littlewood-Verall	La fiabilité après n défaillances. Le MTTF(1). Le nombre moyen des défaillances détectées à un instant t.
Shick-Wolverton	La fiabilité après n défaillances. Le temps d'attente pour les défaillances restantes.
Cox	La fiabilité après n défaillances.
Hyperexponentiel	La fiabilité après n défaillances. L'estimation du nombre d'erreurs pendant l'espace de temps futur connaissant le nombre de défaillances observées entre le temps i et j passés.

(1) Temps moyen avant la première défaillance (Mean Time To Failure)
(2) Temps moyen entre 2 défaillances successives (Mean Time Between Failure)

La majorité des modèles de croissance de la fiabilité (SRGM) est basée sur les attributs comme le nombre de défaillances, temps d'occurrence, sévérité des défaillances, intervalle entre deux défaillances consécutives, etc. tandis que d'autres modèles décrivent la relation entre les tests planifiés, l'effort de test et le nombre de défauts détectés durant le test. Parfois, les SRGMs démontrent de bonnes performances en terme de prédictibilité de la fiabilité des logiciels et parfois pas. Ceci pourrait être causé par l'insuffisance des informations sur la manière dont le logiciel a été développé, maintenu et opéré. Les SRGMs sont des modèles concaves ou en forme de S (S-shaped). Si la courbe décroît, le

modèle est concave. Si la courbe augmente et décroît, le modèle est en forme de S.

Un gestionnaire de projet de développement de logiciel doit choisir le modèle qui convient le mieux à son cas d'étude. En effet, chaque modèle a des données d'entrées et des hypothèses à satisfaire. Puisqu'il est difficile de satisfaire 100% des hypothèses, il est préférable de choisir le modèle avec le plus haut pourcentage de validité. Il n'existe pas un modèle générique qui peut être appliqué à tous les cas [2].

La figure 6 donne une classification des modèles de fiabilité selon le cycle de vie de développement des logiciels.

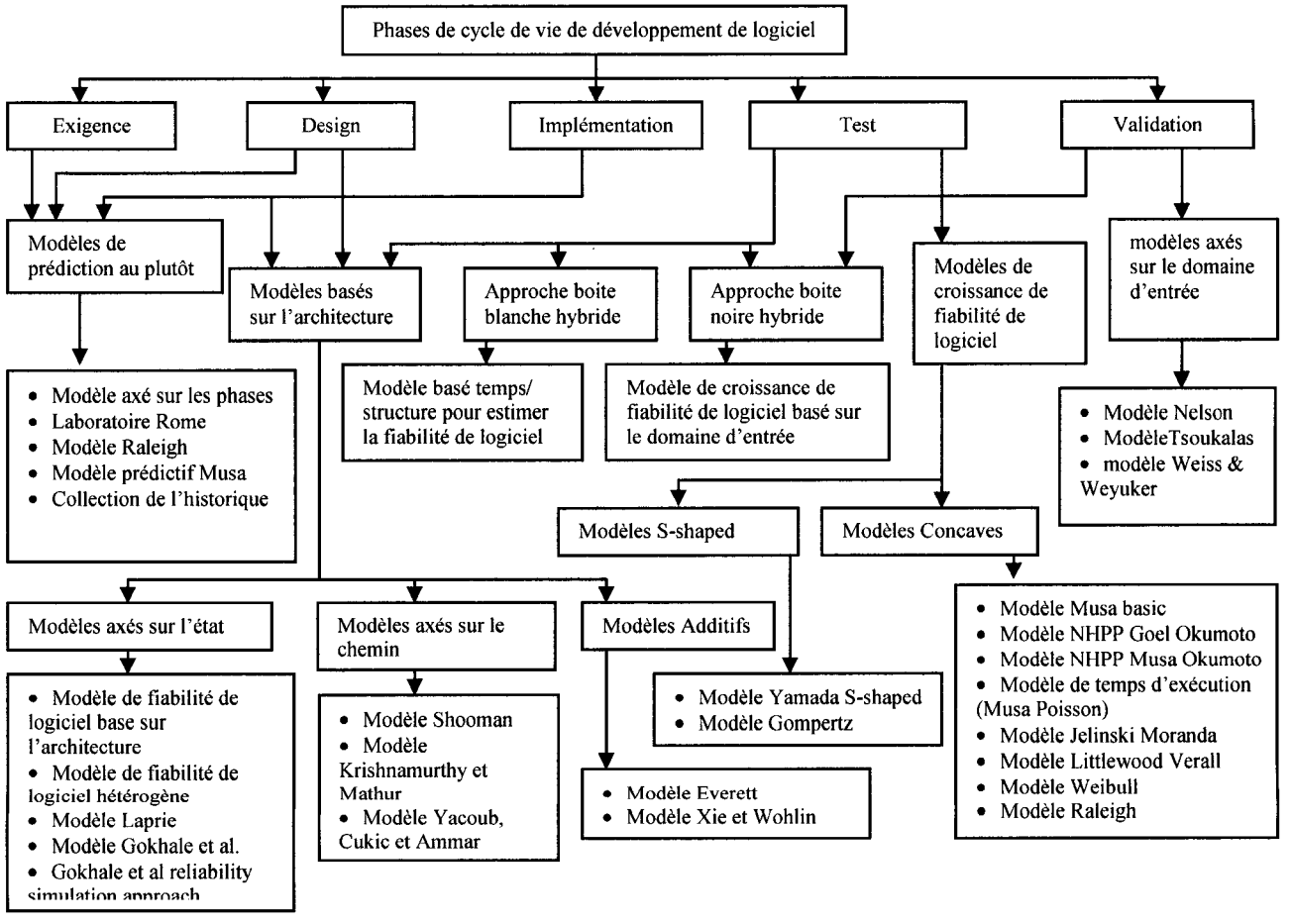

Figure 6 : Classification des modèles de fiabilité de logiciels selon les phases du cycle de vie d'un logiciel [2].

3.3. La modélisation du coût

La qualité d'un logiciel dépend de la durée des tests et des méthodologies utilisées [19]. En effet, d'une part, plus le temps est alloué au test, plus la probabilité d'éliminer les erreurs est élevée, ce qui augmente la fiabilité du logiciel. Cependant, le coût du logiciel augmente. D'autre part, si peu de temps est alloué au test, le coût du logiciel sera réduit, mais le risque de livrer un logiciel non fiable augmente, ce qui peut causer la perte d'une part de marché et des coûts de maintenance élevés lors de la phase opérationnelle.

Plusieurs modèles d'optimisation des coûts et de détermination du temps d'arrêt des tests ont été élaborés [4, 20, 21, 22, 24]. Généralement, ces modèles cherchent à déterminer le temps d'arrêt des tests. Les critères d'évaluation varient d'un modèle à l'autre, notamment, le seuil de fiabilité du logiciel donné, le temps moyen entre les défaillances, le budget donné, le gain en fiabilité ne justifie pas le coût, introduction d'une pénalité sur les délais, etc.

Pham et Zhang ont présenté des modèles de coût permettant de calculer le gain économique net total, ces modèles incluent le coût d'élimination des défauts, le coût de la couverture des tests et le coût de risque dû aux défaillances [19, 20, 22].

Le modèle "Constructive Cost Model II" est le plus connu et réaliste des modèles de planification et d'estimation des coûts d'un projet logiciel [5]. Il utilise une approche "bayesienne" pour combiner les jugements des experts avec les données. Les coûts du modèle COCOMO II sont affectés par les attributs suivants :

- Fiabilité requise
- Taille de la base de données à tester
- Complexité du produit
- La réutilisation
- Documentation pour répondre aux besoins du cycle de vie
- La contrainte du temps d'exécution

- La contrainte de stockage
- Compétence de l'analyste
- Compétence des développeurs
- L'expérience des applications
- Permanence du personnel
- La volatilité de la plate-forme
- L'expérience de la plate-forme
- L'expérience des outils et langage
- Outils utilisés
- Développement multi site
- Délais requis

L'estimation de l'effort en personne-mois est donnée par la formule suivante:

$$\textbf{Effort} = A \times Size^{E} \times M$$

A=2.94 : constante déterminée par la calibration de 161 projets de développement.
M : un multiplicateur obtenu par la combinaison des 15 attributs.
Size : la grandeur du programme donnée en Kilo de Ligne de Code (KSLOC).
E variable dépendant de :

- Flexibilité du développement
- Expérience passée
- Architecture/résolution du risque
- Cohésion de l'équipe
- Maturité du processus

La plupart des modèles et études cités cherchent principalement à estimer et modéliser la fiabilité des logiciels lors de la phase opérationnelle. En effet, ces modèles et ces mesures ne s'attardent pas sur les analyses des défaillances, erreurs et défauts et ne cherchent pas à établir des liens entre eux. D'où la nécessité, dans le cadre de cette étude de développer des outils de prévention de ces derniers à l'origine.

Dans la littérature, la méthode "Analyse des Erreurs et leurs Effets sur le Logiciel" (AEEL) est présentée, mais ne fait pas objet de beaucoup d'études. Généralement, cette méthode est mentionnée brièvement suite aux études AMDEC et AMDE. Il est évident que l'AEEL suit les mêmes démarches de réalisation que l'AMDE et l'AMDEC, cependant les outils et techniques d'analyse doivent être adaptés à l'étude des logiciels, vu les différences intrinsèques des produits matériels et logiciels.

CHAPITRE 4

ANALYSE DES ERREURS ET LEURS EFFETS SUR LE LOGICIEL

Dans ce chapitre, il sera question de décrire la méthode AEEL en présentant la manière de sa préparation et sa démarche générale.

Un programme est désigné pour accomplir des fonctions préspécifiées. Une défaillance survient quand la sortie actuelle dévie de la sortie attendue. Toutefois, la nature de la défaillance diffère d'une application à l'autre, et doit être définie selon les spécifications propres à l'utilisation du logiciel [19]. Par exemple, un temps de réponse de 30 secondes peut être une défaillance catastrophique pour un système de navigation aérien, mais acceptable pour une réservation d'un billet d'avion.

L'"Analyse des Erreurs et leurs effets sur le logiciel" est une méthode de prévention et d'analyse prévisionnelle utilisée généralement lors d'un projet de développement d'un logiciel. Elle permet de mettre en évidence les modules critiques du programme, envisager des hypothèses d'erreurs dans un logiciel, examiner les conséquences de ces erreurs pour décider des actions de validation et de maintenance. L'objectif principal de cette technique est d'identifier les défaillances potentielles et les erreurs de conception et de programmation afin d'analyser leurs effets internes et externes. Il est à noter que cette méthode est dérivée de l'AMDEC, mais adaptée au développement de logiciels.

4.1. Analyse des erreurs, défauts et défaillances

Le but de ces analyses est d'obtenir un historique des informations sur le produit dans le but d'évaluer la fiabilité. Les défaillances, défauts et erreurs sont reliés par des relations de cause à effet. Une défaillance peut être générée par une erreur d'utilisation, un défaut ou un manque dans le produit logiciel ou encore par un défaut matériel. Les erreurs sont introduites dans le code par le concepteur ou le programmeur durant la phase de développement ou la phase de la maintenance. D'autre part, une défaillance du système matériel peut générer des signaux erronés inattendus, ce qui peut causer une défaillance du logiciel. Cet événement, inattendu pour l'opérateur, peut lui causer un stress favorisant l'introduction de valeurs inappropriées qui, à leur tour, activent des erreurs et génèrent des défaillances. La figure 7 représente les relations de cause à effet entre les erreurs, défauts et défaillances [9].

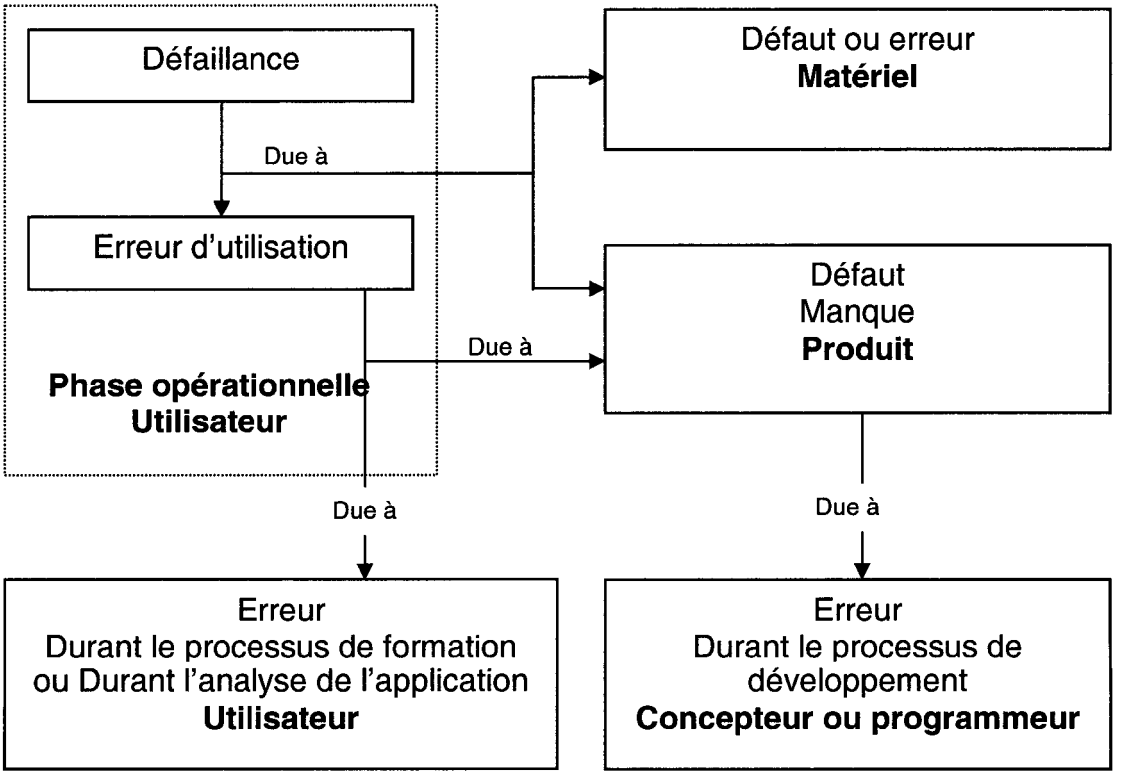

Figure 7 : Dynamique des erreurs, défauts et défaillances.

4.1.1. Analyse des erreurs

L'analyse des erreurs permet de collecter des informations qui peuvent servir à éliminer les faiblesses durant le processus de développement. Cette analyse est accomplie, principalement, par les mesures processus.

Dans un système logiciel, une erreur peut résulter pour un ensemble de données bien spécifiques (Figure 8). Elle peut être due à l'activation d'un défaut interne ou externe. Un défaut interne peut être un défaut physique ou un défaut de conception, un défaut externe peut être un défaut humain ou dû à l'environnement physique du système.

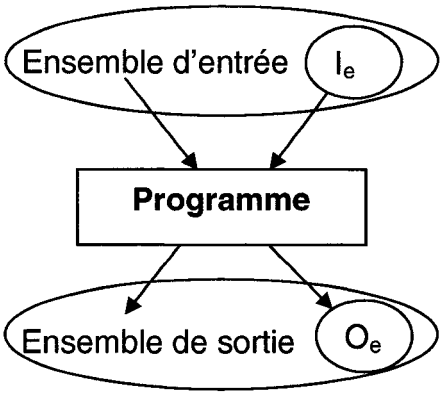

Figure 8 : Un système logiciel du point de vue entrée/sortie [29].

Avec : I_e : ensemble des entrées qui causent une erreur de fonctionnement
O_e : ensemble des sorties erronées.

La figure 9 est une représentation simplifiée de l'introduction et la suppression des défauts durant le cycle de vie d'un logiciel [5].

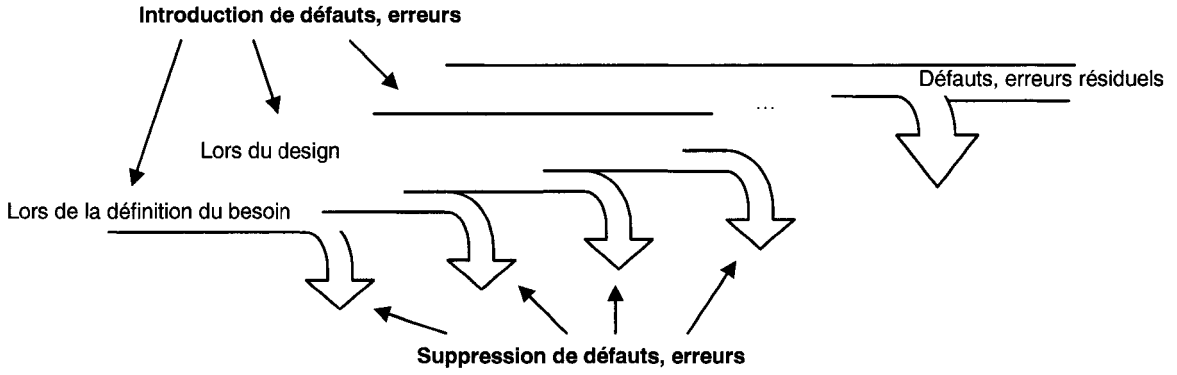

Figure 9 : Introduction et suppression des défauts.

Selon Pham, les erreurs peuvent être spécifiques au langage de programmation, de mémoire, de compilation, d'appels aux bibliothèques ou dans les bibliothèques standards, etc. [19].

4.1.2. Analyse des défauts

L'analyse des défauts est accomplie, principalement, par les mesures produit, vu que la majorité des défaillances dues aux erreurs du développeur est causée par les défauts au niveau du produit logiciel. Les défauts peuvent être permanents ou temporaires. Les sources des défauts sont de nature humaine ou physique [12] :

- Défauts humains :
    - Internes ou de conception : dues aux imperfections accidentelles ou intentionnelles commises soit lors du développement ou la modification du système, soit dans l'établissement des procédures d'utilisation et de maintenance.
    - Externes : dues à la violation accidentelle ou intentionnelle des procédures d'utilisation ou de maintenance.

- Défauts physiques :
    - Internes : dues à des défauts physico-chimiques, par exemple : changements de seuils, courts-circuits, ruptures.
    - Externes : dues aux perturbations de l'environnement, par exemple : perturbations électromagnétiques, rayonnements, température, vibrations.

Pour obtenir un logiciel fiable, on peut tolérer, éliminer, prévoir les défauts ou les éviter par construction [12]. Munson, J.-C. établit un lien entre les défauts et la complexité d'un logiciel [18]. En effet, plus le logiciel est complexe, plus il va y avoir des défauts latents. L'apparition de ces derniers dépend de la fréquence et de la manière d'exécuter le programme.

Le nombre de défauts résiduels peut être considéré comme une variable interne au système et son comportement pourrait être décrit par une équation différentielle de second ordre [10]. Ce modèle incorpore les informations sur le processus du test du système (nombre de testeurs, qualité du processus, complexité du système logiciel) et décrit le comportement de la vie réelle (durée limite, variation du personnel) simultanément.

4.1.3. Analyse des défaillances

L'analyse des défaillances permet d'étudier les modes de défaillances du produit logiciel dans son environnement utilisateur. Une analyse des défaillances peut améliorer les facteurs humains du produit avec une meilleure compréhension des besoins clients et utilisateurs. À cette issue, le développeur peut améliorer la documentation, le support de l'application et du système, notamment par une meilleure détermination des besoins client en matière de formation et d'encadrement.

La figure 10 représente les relations de cause à effet entre défaillances, défauts et erreurs. Les défaillances sont dues, principalement, à une mauvaise

compréhension et interprétation des besoins clients et à leur traduction en spécifications et fonctionnalités. Ceci peut générer des erreurs, des défauts et des changements fréquents dans les phases avancées du cycle de vie du logiciel.

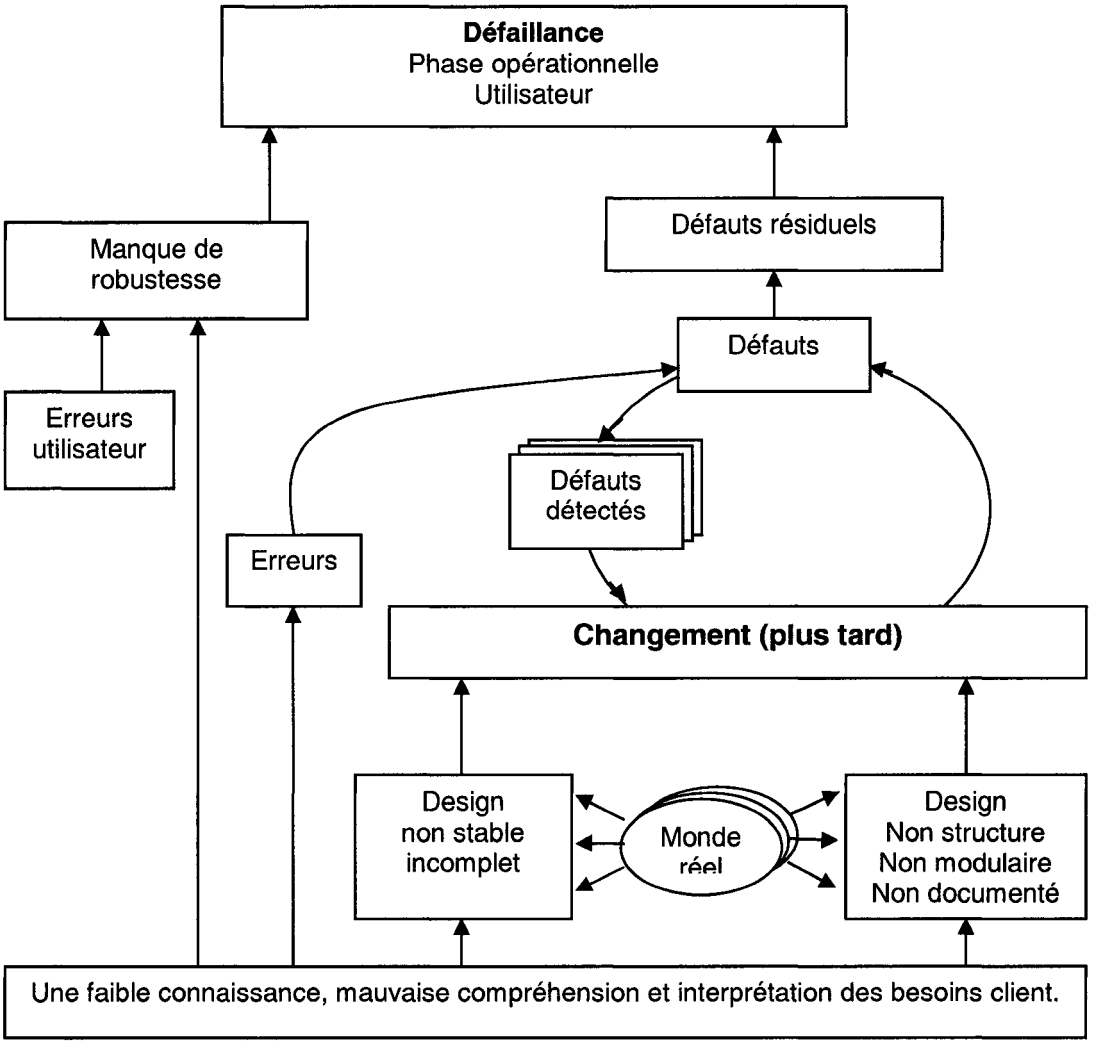

Figure 10 : Causes potentielles d'une défaillance [9].

La détérioration d'un système logiciel n'est pas une fonction du temps, mais plutôt une fonction des changements lors de la phase de maintenance. Les changements peuvent avoir lieu lors de la correction des défauts, de la

considération de nouvelles exigences ou pour l'amélioration des performances du système. Ces changements sont l'une des causes majeures indirectes de la génération des défauts dans un système logiciel. Ils peuvent influencer non seulement la fiabilité, mais aussi les coûts et les délais. Une mauvaise interprétation et compréhension du monde réel présente une grande difficulté et un coût élevé durant l'implémentation et l'exploitation du logiciel.

Les défaillances sont classées en deux grandes catégories [12] :

- Défaillances contrôlées : les modes de défaillances appartiennent à un ensemble bien spécifié de défaillances.
- Défaillances non contrôlées : les modes de défaillances sont quelconques.

Les défaillances peuvent être dues aux défauts du produit ou aux erreurs des utilisateurs [11, 12]. Les défaillances dues aux défauts du produit peuvent résulter d'une incohérence entre les spécifications du produit et l'environnement utilisateur, une configuration matérielle inadéquate, une performance inadéquate du système, défauts résiduels, etc. Les défaillances dues aux erreurs des utilisateurs peuvent être causées par un manque de documentation bien structurée, interfaces utilisateur complexes, manque de formation, manque de support, niveau éducationnel insuffisant de l'utilisateur, etc.

Les défaillances corrélées ou de cause commune (DCC) sont les défaillances de deux composants ou plus due à une seule cause. Les causes communes peuvent être des événements ou des conditions internes dans le système ou externes provenant de son environnement. Il s'avère intéressant de tirer des informations de nature qualitative et quantitative des données sur l'expérience acquise en cours d'exploitation d'un logiciel afin d'éviter les défaillances de cause commune ou d'atténuer leurs conséquences.

4.2. Préparation de l'AEEL

Avant d'entamer la réalisation de l'AEEL, une étude préalable est nécessaire. En effet, il est indispensable de définir l'objectif de l'étude, déterminer le groupe de travail, délimiter le champ et le délai de l'étude, faire une décomposition fonctionnelle et une collecte des données jugées nécessaires. L'objectif de l'AEEL dépend du contexte de l'étude. Dans le cas d'un projet de développement et conception de logiciel, l'objectif principal est, généralement, la fiabilité [8].

Le groupe de travail doit être nécessairement pluridisciplinaire pour avoir une étude plus objective. Le chef de projet choisit les personnes jugées compétentes pour réaliser le projet. Idéalement, une expérience appropriée des développeurs est exigée. Cependant, dans plusieurs cas, le choix est dirigé vers une équipe moins expérimentée pour les raisons suivantes [2] :

- Le budget ne peut pas couvrir un personnel au salaire élevé, donc une équipe moins expérimentée peut être utilisée.
- La non-disponibilité de personnes expérimentées au sein d'une organisation ou même à l'extérieure et il est difficile de recruter de nouvelles personnes. Les bons éléments peuvent être affectés à d'autres projets.
- L'organisation veut améliorer les compétences de son personnel, d'où elle assigne des non expérimentés aux projets pour avoir un gain d'expérience.

Les gestionnaires des projets de développement doivent anticiper les problèmes et préparer les alternatives pour les projets. Au début de la planification, une évaluation des contraintes est nécessaire : la date de livraison, le personnel disponible, le budget, etc. Les plans initialement élaborés doivent être utilisés comme guide pour le projet.

Une phase critique de l'analyse des erreurs est la décomposition fonctionnelle ou modulaire du logiciel. Les fonctions d'un système logiciel peuvent être classées selon leurs fonctionnalités ou bien leur fréquence d'exécution [25].

Les classes selon les fonctionnalités sont les suivantes :

Fonctions essentielles : ce sont les fonctions requises pour que le système réalise son principal objectif. Ce sont les raisons de création du système.

Fonctions auxiliaires : ce sont le support des fonctions essentielles. Dans plusieurs cas, une défaillance de ces fonctions peut être plus critique que la défaillance des fonctions essentielles.

Fonctions informatives : ce sont des fonctions qui retournent des informations, telles que les fonctions de contrôle, alarmes, etc.

Fonctions interfaces : ce sont les fonctions qui font le lien entre les différents modules, avec le matériel ou avec l'utilisateur.

Fonctions superflues : ce sont des modules qui sont superflus. Ceci peut être dû aux changements fréquents sur une longue période ou bien le système est conçu d'une façon superflue pour tolérer les défauts.

Les classes de fonctions selon la fréquence d'utilisation sont les suivantes :

Fonctions en ligne : ce sont les fonctions utilisées d'une façon continue

Fonctions hors ligne : ce sont les fonctions utilisées en intermittence ou non fréquemment.

Une fois les fonctions déterminées, il est nécessaire d'identifier et constituer les liens et les correspondances entre les fonctions logicielles et les différents modules du logiciel. Ceci est suivi par un affinage du classement des modules vis-à-vis des contraintes et de l'identification des différentes interfaces.

4.3. Démarche générale de l'AEEL

Une fois les fonctions logicielles définies et la préparation faite pour entamer l'AEEL, la démarche à suivre comporte quatre étapes (figure 11). Cette démarche a été adaptée de l'AMDE et de l'AMDEC.

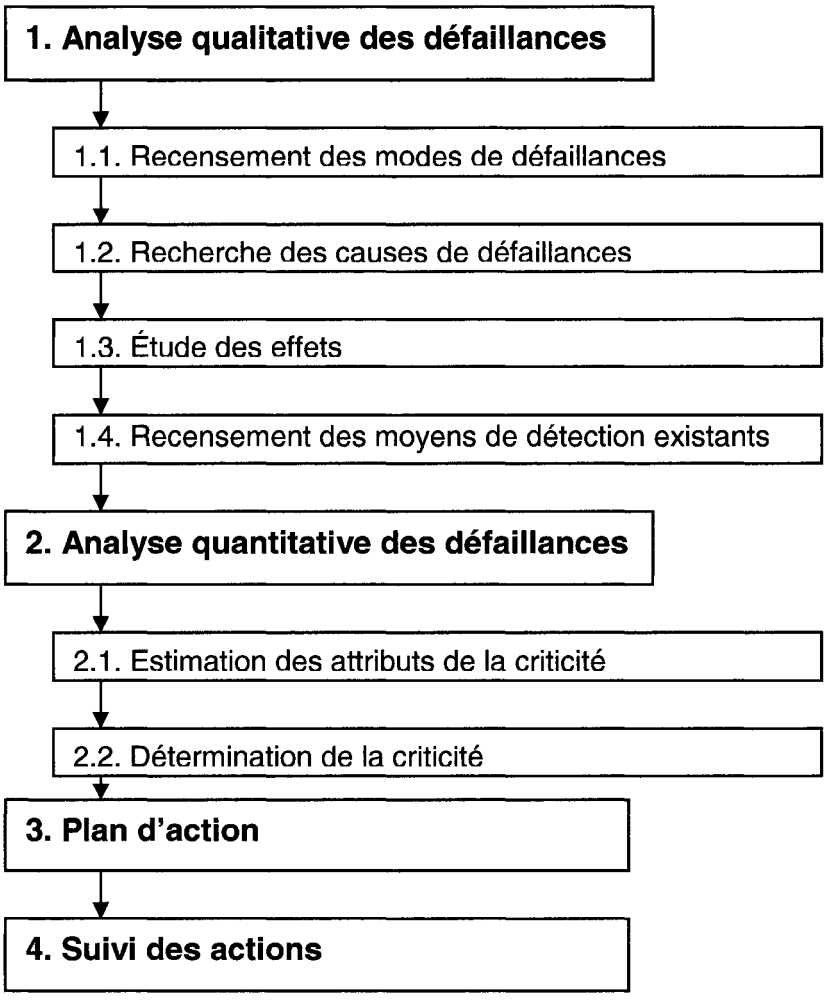

Figure 11 : Démarche de l'AEEL [8].

4.3.1. Analyse qualitative des défaillances

4.3.1.1. Recensement des modes de défaillances

Il s'agit d'identifier et recenser tous les modes et les types de défaillances potentiels qui peuvent survenir pour chaque sous-système, fonction ou module analysé. Chaque mode de défaillance correspond à une fonction.

4.3.1.2. Recherche des causes de défaillances

Il s'agit de rechercher les causes potentielles des défaillances tout au long du cycle de vie du logiciel et de déterminer les types d'erreurs : calculs, algorithmiques, interfaces, etc. Les principaux outils de travail pour mener à bien cette étape sont : diagramme d'Ishikawa, diagramme de bloc fonctionnel, diagramme de bloc fiabilité et arbre de défaillance. Ces outils découlent de l'analyse des matériels et doivent être adaptés pour le cas de logiciels.

4.3.1.3. Étude des effets

L'étude des effets consiste en la détermination des effets des dites défaillances au niveau local, mais également au niveau global et ce, en respect à l'analyse de leurs conséquences pour chaque mode de défaillance envisageable. Par exemple, une fourniture de résultat incorrecte, absence de résultat sur l'écran, etc.

4.3.1.4. Recensement des moyens de détection existants

Il s'agit de déterminer les moyens de détection d'une défaillance. Par exemple : vérification, message d'erreur, indicateur, etc.

Un exemple de support pour mener l'étude qualitative consiste en les 9 premières colonnes du tableau 6, à savoir : Fonction analysée, Défaut/erreur envisagé, Effet sur le module analysé, Effet sur les autres modules, Effet sur le

système, Risque qualitatif, Causes possibles, Moyens de détection et Moyen de prévention.

4.3.2. Analyse quantitative des défaillances

L'analyse quantitative des défaillances consiste en la quantification des modes de défaillances recensés lors de l'étude quantitative.

4.3.2.1. Estimation des attributs de la criticité

En se basant sur une grille d'analyse, les défaillances sont classées selon des groupes appropriés. Pour chaque classe identifiée, il est important de définir les conditions de la fiabilité en utilisant la métrique appropriée, mais également définir les critères de quantification afin d'accorder une note à chacun de ces critères. Généralement, les grilles d'analyse sont préétablies.

En général, trois critères sont utilisés pour déterminer la criticité d'une défaillance [8] :

- Sévérité ou gravité (G) : l'incidence provoquée par l'effet.
- Risque de la non-détection (ND) : la probabilité que la cause et le mode surviennent et que la défaillance atteigne l'utilisateur.
- Fréquence d'apparition : la probabilité P pour que la cause se produise et qu'elle génère le mode de défaillance concerné : $P = P_1 \times P_2$

  avec : $\mathbf{P_1}$: probabilité que la cause de défaillance se produise,
  $\mathbf{P_2}$: probabilité que la défaillance survienne avec la présence de la cause.

Généralement, la cote F est donnée par une grille de cotation préétablie. Le tableau 5 donne un exemple d'une grille de cotation de la fréquence F par intervalle de l'indice de la fréquence.

Tableau 6 : Exemple d'une grille de cotation de la fréquence

F	**Probabilité d'apparition : P_1xP_2**
1	[0 à 3 / 100 000[
2	[3 / 100 000 à 10 / 100 000[
3	[10 / 100 000 à 3 / 10 000[
4	[3 / 10 000 à 10 / 10 000[
5	[1 / 1000 à 3 / 1000[
6	[3 / 1000 à 10 / 1000[
7	[1 / 100 à 3 / 100[
8	[3 / 100 à 10 / 100[
9	[10 / 100 à 30 / 100[
10	[30 / 100 à 100 / 100[

La fréquence peut être estimée aussi par le choix d'un modèle approprié d'estimation de la fiabilité du logiciel (figure 6).

Durant ces étapes, l'objectif est de chercher les liens entre les causes potentielles des défaillances et les indicateurs de la fiabilité obtenus durant le processus et la fiabilité opérationnelle du produit fini.

4.3.2.2. Détermination de la criticité

La détermination d'un indice de criticité permet de classer les modes de défaillances selon la sévérité de leurs effets. Plus l'indice est élevé, plus la défaillance est sévère et il est nécessaire d'agir pour corriger ces défaillances.

La criticité (C) est déterminée comme suit :

$$\mathbf{C = F \times G \times ND}$$

Si l'échelle de chaque indice est de 1 à 10, la criticité C varie entre 1 et 1000. Une évaluation des seuils et des criticités fixés permet de mieux atteindre les objectifs. En effet, toutes les valeurs qui sont supérieures à une valeur de criticité

préétablie feront l'objet d'une autre étude plus approfondie et spécifique émanant à agir afin de minimiser la sévérité de la défaillance. Ceci pourrait être effectué par la prévision des redondances ou autres solutions permettant de diminuer les indices F, G et ND. Cette diminution peut concerner un, deux ou les trois indices à fois.

Typiquement, la sévérité des défaillances est classée selon les quatre grandes catégories suivantes [19] :

- *Catastrophique :* effets désastreux, comme la perte d'une vie humaine.
- *Critique :* effets désastreux, mais réparables, comme des dommages des équipements sans perte de vie humaine.
- *Majeur :* une défaillance sérieuse du système logiciel sans dommages physiques aux individus ou autres systèmes.
- *Mineur :* les défaillances qui mènent aux dérangements marginaux à un système logiciel ou à ses utilisateurs.

Un exemple de support pour mener l'étude quantitative est donné dans le tableau 7.

4.3.3. Plan d'action

Le plan d'action découle essentiellement de l'étude quantitative des défaillances. À ce niveau, le responsable d'AEEL recommande des actions pour pouvoir détecter et éliminer les erreurs ou encore, éviter leur propagation. Les actions peuvent être préventives ou correctives. Elles visent à éliminer les causes de défaillances et à minimiser la criticité.

4.3.4. Suivi des actions

Une fois l'AEEL terminé, les responsables du projet seront chargés de suivre la mise en œuvre des actions et les résultats trouvés. Une réévaluation des indices F, G, ND et C est faite à une date ultérieure pour mesurer l'atteinte des objectifs et ainsi recommander, s'il y a lieu, de nouvelles actions. Un responsable de suivi est désigné pour mener cette phase. Un exemple de support pour mener le suivi est donné dans le tableau 7.

Tableau 7 : Outil d'analyse des erreurs et de leurs effets sur le logiciel (adapté de [8, 25, 27]).

Analyses des Erreurs et de leurs Effets sur le Logiciel (AEEL)			**Page : .... de ...** **Fait par :**
Nom du programme :	**Calculateur :** **Version :**	**Langage :**	**Date AEEL (origine) :** **(dernière révision) :**

Fonction/ module analysée	Défaut/ erreur envisagé	Effet sur le module analysé	Effet sur les autres modules	Effet sur le système	Risque qualitatif	Causes possibles	Moyens de détection	Moyen de prévention	**État actuel** F	G	N D	C	Responsable du suivi	Suivi/résultat F	G	N D	C	Observations

Étude qualitative

Étude quantitative

Suivi

CHAPITRE 5

ÉTUDE DE CAS

Ce chapitre décrit l'étude de cas effectuée. Il s'agit de présenter le contexte de l'étude empirique, d'identifier l'application et le modèle étudié et d'expliquer la méthodologie d'analyse.

5.1. Mise en contexte

L'étude de cas réalisée dans ce travail porte sur un logiciel de gestion intégrée (ERP : Entreprise Resource Planning). Le choix de ce type de logiciel pour cette étude est dû, d'une part, à sa relation étroite avec le domaine du génie industriel, et d'autre part, à sa complexité, sa fiabilité et son coût élevés. La complexité est due à la multitude des modules et fonctionnalités intégrés (finances, qualité, comptabilité, ordonnancement, etc.). Le coût est dû aux licences d'utilisation et au processus d'implémentation qui nécessite des développements spécifiques, formation et assistance technique.

Les projets de mise en place d'un système de gestion intégrée sont très complexes et coûteux. La complexité vient de l'envergure du projet et de l'implication de plusieurs ressources telles que financières, humaines, matérielles, etc. Généralement, lors de ces projets, des développements spécifiques sont faits pour satisfaire les besoins du client.

En général, se procurer un système de gestion intégrée est un enjeu majeur qui fait partie des décisions stratégiques de l'entreprise. De ce fait, les objectifs, les délais et coût sont fixés et, en général, fermes. Ce qui limite la marge de manœuvre du fournisseur du logiciel. Pour faire face à ce défi, le fournisseur doit

respecter les attentes client et ce, pourrait être en se dotant d'une équipe compétente, des techniques fiables et faire preuve d'efficacité.

Les systèmes ERP sont développés, généralement, autour d'un Système de Gestion de Base de Données (SGBD). Une présentation du fonctionnement de ces derniers est jugée utile pour la compréhension de la partie qui suivra.

Un système de gestion de bases de données est un logiciel qui permet de gérer efficacement une base de données. Il fournit les moyens de définir, contrôler, mémoriser, manipuler et traiter les données tout en assurant sécurité, intégrité et confidentialité, éléments indispensables à considérer lorsqu'un grand nombre d'utilisateurs veulent interagir simultanément avec les données de la base [1].

Le SGBD permet trois opérations de mise à jour des données à savoir insertion (ajout), suppression et modification. Lorsqu'une contrainte est violée lors d'une opération d'insertion, la valeur insérée est rejetée par le système. L'opération de suppression ne peut alors violer que l'intégrité référentielle. Si c'est le cas, la suppression est soit rejetée, soit supprimée en cascade tous les tuples référencés par le tuple à supprimer. Il est également possible que l'attribut de référence soit modifié, ce qui cause la violation de l'intégrité. En général, la modification des valeurs d'un attribut qui n'est ni clé primaire ni clé étrangère ne pose aucun problème.

Le SGBD présente 3 principaux niveaux d'abstraction. Le niveau externe consiste en le contrôle d'identité, le contrôle de syntaxe, le contrôle des autorisations ainsi que les droits d'accès. Quant au niveau interne, il englobe la poursuite de l'exécution de la requête ainsi que la localisation des données nécessaires pour répondre à cette requête. Finalement, la couche de gestion couvre l'accès au support physique des données et l'extraction des données pour répondre à la requête.

5.2. Identification de l'application à étudier

Le logiciel analysé dans le cadre de cette étude est intitulé "Orchestra 2000", produit de "Concepts Industriels 2000". C'est un système de gestion intégrée développé avec Visual Basic, Asp.net et utilise SQL Server comme SGBD relationnel et est formé par plusieurs sous-systèmes, à savoir :

- Orchestra pour PME
- Le Système de Paie pour PME
- Configurateur pour PME
- Prévisionniste pour PME
- Maintenance pour PME
- Orchestra version Web
- Gestion des Rapports pour PME
- Le comptable version Web.

Le noyau du logiciel est "Orchestra pour PME". Les autres sous-systèmes peuvent être greffés sur le noyau selon les besoins des entreprises pour ajouter des fonctionnalités supplémentaires.

Le logiciel est composé d'une base standard, à laquelle les développeurs apportent les modifications nécessaires tout en l'adaptant aux besoins spécifiques du client.

Le logiciel est conçu pour être utilisé par des cadres, programmeurs d'application, administrateurs de bases de données, gestionnaires, etc. L'accès aux données se fait via des interfaces, formulaires spécifiques et des générateurs de rapports.

La commercialisation du logiciel se fait suite à l'achat direct du client ou à une prospection. Un cahier de charge fonctionnel est rédigé à cette issue et est transféré au département de développement. Une fois l'accord établit entre le client et le fournisseur, ce dernier identifie les besoins et exigences client, modélise les différents processus de gestion de l'entreprise et traduit les exigences en fonctionnalités et en modules d'une part, et établit les liens

fonctionnels d'autre part. Une fois les modules spécifiques développés, ils sont testés séparément, puis intégrés et greffés, et le système est compilé et testé en totalité.

"Concepts Industriels 2000" est doté un système informatisé intitulé "Mantis Bug Tracker System" (Mantis). Le but de Mantis est, d'une part, de faciliter la soumission et le suivi de bogues et d'autre part, d'assurer la traçabilité de toute anomalie rencontrée dans un logiciel durant la phase de développement, de validation et d'exploitation. Un utilisateur ou testeur peut soumettre un bogue pour une catégorie donnée du logiciel. Ce bogue est ensuite confirmé par le responsable du logiciel, et est affecté à un ou plusieurs développeurs. Il est également possible de soumettre des améliorations au logiciel. En général, une criticité et une priorité sont associées à un bogue. Une fois soumis, les états que peut avoir un bogue sont : "nouveau", "commentaire", "accepté", "confirmé", "affecté", "résolu" ou "fermé". Les attributs d'un bogue sont donnés à l'annexe A.

5.3. Modélisation du système à étudier

Pour des fins d'analyse, le système à étudier a été modélisé dans son environnement opérationnel (figure 12). Le système logiciel interagit avec l'utilisateur et le programmeur. Le programmeur et l'utilisateur communiquent via Mantis ou directement. Les utilisateurs soumettent des requêtes soit en cas de défaillance ou pour un soutien technique. Comme suite à la soumission d'une requête, le programmeur intervient pour régler le problème.

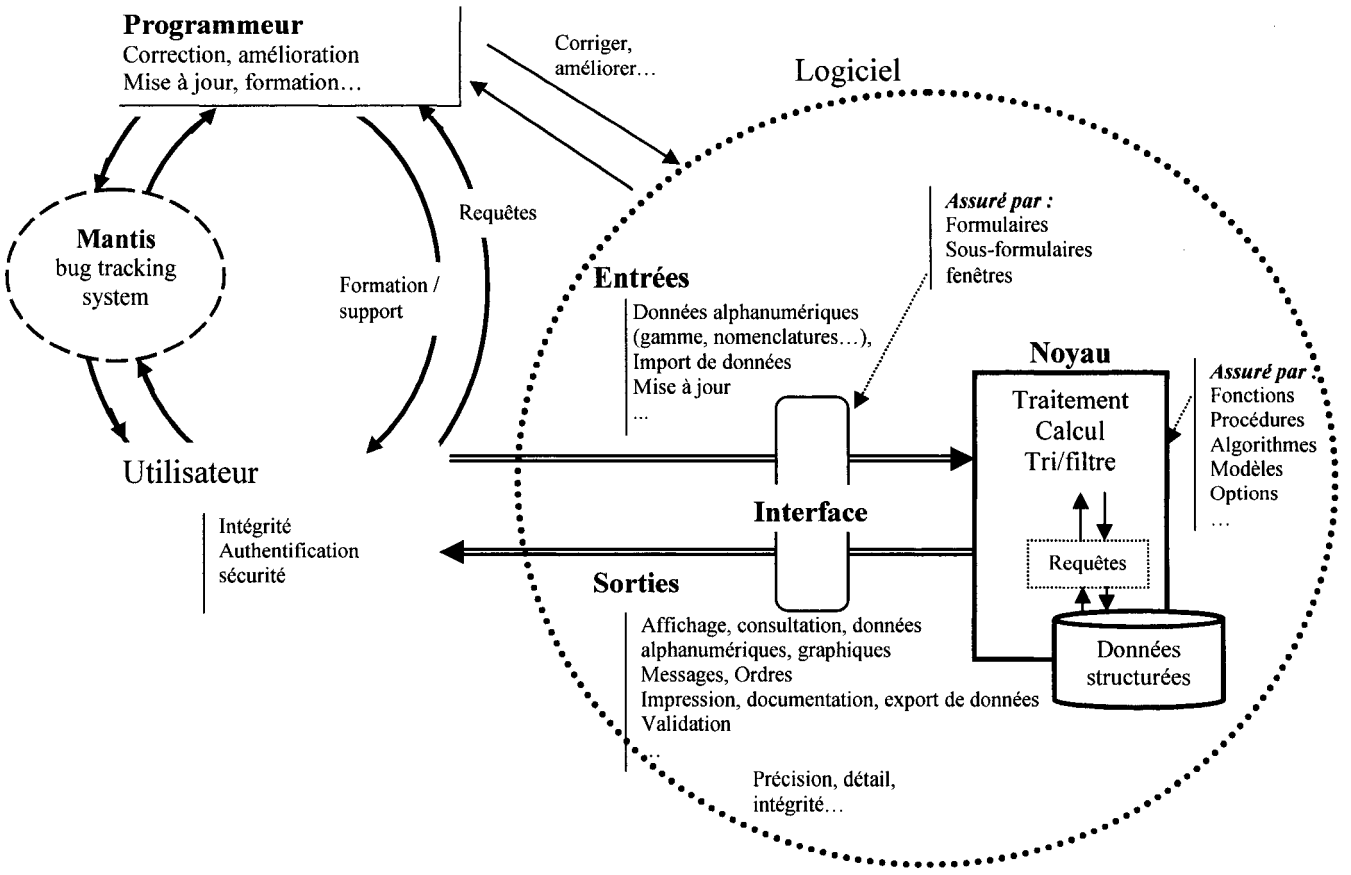

Figure 12 : Modélisation du système étudié.

5.4. Méthodologie de l'analyse

Cette étude découle de l'analyse des données historiques de suivi de bogues et d'irrégularités de fonctionnement d'Orchestra pour PME. Les données s'étalent sur la période de 2004-2006. Pratiquement, une période de 2 ans est supérieure à la durée moyenne d'implémentation d'un ERP.

En se basant sur les informations disponibles sur "Mantis", les modes de défaillances ont été décelés ainsi que leurs causes et solutions, et ce, sous forme de AEEL. Cette étude n'est pas prévisionnelle, elle cherche à déterminer les modes de défaillance qui sont survenus lors de l'exploitation du logiciel et se

limite seulement à l'étude qualitative. À ce titre, n'ont été retenus que les modes, la criticité, les causes de défaillances et les actions recommandées (Tableau 8).

Tableau 8 : Outil utilisé pour l'analyse des bogues.

ID Bogue	Mode de défaillance	Criticité de la défaillance	Cause(s)/ mécanisme(s) de défaillance	Actions recommandées	Mode de défaillance générique

L'analyse s'est basée sur 850 bogues, mais seulement 765 bogues ont été retenus. Les 85 bogues supprimés sont de simples notes, demande de vérification, message pas clair, références non disponibles. Un bogue peut comporter une ou plusieurs défaillances.

Les étapes suivies pour mener cette étude sont les suivantes :

1. Classer les bogues par sous-système,
2. Analyser des bogues en utilisant le tableau 7. Pour chaque bogue, les modes de défaillance et leurs criticités sont déterminés et, s'il y a lieu, les causes et les actions recommandées. Par la suite, chaque mode de défaillance est reformulé pour le généraliser,
3. Raffiner la liste des modes de défaillance génériques obtenus par la compilation des similarités et déterminer l'occurrence de ces derniers,
4. Classer des modes de défaillance selon des familles,
5. Extraire les modes de défaillance critiques et déterminer les causes possibles par mode.

CHAPITRE 6

RÉSULTATS DE L'ÉTUDE

Ce chapitre présente les résultats de l'étude de cas. Dans un premier temps, les modes de défaillances seront classés et les modes de défaillances génériques seront listés. Dans un deuxième temps, ces résultats seront interprétés et les causes de défaillances seront analysées.

6.1. Classement des modes de défaillances

En se basant sur la figure 12 et la liste des modes de défaillance décelés, ces derniers peuvent être classés selon les familles suivantes :

- Entrée
- Sortie
- Fonctionnalité
- Paramétrage
- Ergonomie
- Informations
- Données
- Règles de gestion
- Performances
- Traitement des données
- Divers

La liste finale obtenue (tableau 9) comporte 82 modes de défaillance. Les modes de défaillance obtenus ainsi que leurs occurrences sont classés selon les familles citées ci-haut.

Tableau 9 : Liste des modes de défaillances et leurs occurrences par famille.

Famille	Mode de défaillance	Occurrence
Entrée	Problème d'insertion	26
	Saisie impossible/difficile	13
	Application ne démarre pas, run-error	7
	Modification simultanée non permise	1
	Changement non pris en compte	1
Sortie	Manque d'un rapport	26
	Problème d'impression	16
	Rapport incorrect	14
	Résultat erroné	14
	Manque de précision (détail)	12
	Consultation impossible, difficile	11
	Manque, difficulté de validation	10
	Rapport non fonctionnel, ne s'ouvre pas	6
	Sortie erronée/absente	5
	Manque d'un champ dans un rapport	4
	Ne s'affiche pas	3
	Manque de spécification	2
Fonctionnalités	Manque d'option	29
	Tri/filtre non fonctionnel	29
	Bouton non fonctionnel	27
	Manque de filtre/tri	16
	Manque de bouton	13
	Perte de fonctionnalité	12
	Option non fonctionnelle	5
	Composant ne fonctionne pas	4
	Manque de fonctionnalité	3
Divers	MAJ version non faite	7
	Problème de sécurité	5
	Manque d'un formulaire/interface	5
	Manque/ absence d'un lien	5
	Transfert de données impossible	1
	Tooltip erroné (inadéquat)	1
	Opérations non concaténées	1
Paramétrage	Valeur par défaut erronée	11
	Fonction non activée	4
	Fonction activée	1
	Recherche mal configurée	1
Information	Manque d'information	66
	Perte d'information	12
	Information superflue	14
	Information erronée	3

Tableau 9 : Liste des modes de défaillances et leurs occurrences par famille *(suite).*

Famille	Mode de défaillance	Occurrence
Données	Perte de données	39
	Incohérence de données	22
	Redondance de données	15
	Manque de données	16
	Données erronées	7
	Perte d'historique	4
Performances	Système lent, se plante, runtime	32
	MAJ données ne se fait pas instantanément	9
	Multifenêtre impossible	3
Traitement des données	Calcul erroné	34
	Traitement erroné	26
	Traitement mal fait	17
	Données non ajustées	7
	Irrégularité dans le traitement/données	9
	Difficulté de traitement	8
	Traitement non fait	8
	Opération superflue/inutile	8
	Traitement impossible	7
	Opération non complétée	6
	Opération non faite	5
	Traitement double	4
	Unité inappropriée/ incorrecte	2
	Calcul mal fait	2
Ergonomie	Mauvaise apparence	27
	Affichage mal fait	29
	Message d'erreur superflu (inutile) fréquent	18
	Format non standard, incorrect, inapproprié	14
	Espace insuffisant	8
	Information tronquée	7
	Manque d'un message	4
	Bouton superflu	3
	Formulaire ne se ferme pas	3
	Redondance d'endroits de saisie	2
	Application ne se ferme pas toute seule	2
	Menu ne se déploie pas	2
Règles de gestion	Règle de gestion violée	19
	Intégrité de données violée	19
	Condition non considérée	11
	Exception, cas particulier non considéré	9
	Problème d'accès aux données	2
	Condition superflue	1

L'occurrence élevée de certains modes de défaillance peut être expliquée par la fréquence de sollicitation d'un module contenant des erreurs. En effet, l'appel de plusieurs fonctions à un module pourrait activer une erreur résiduelle qui à son tour génère une défaillance. Dans ce cas, le développeur est amené à agir pour diminuer la complexité d'exécution.

L'attention des équipes de développement doit être portée sur l'amélioration et l'augmentation du niveau de fiabilité des modules les plus fréquemment appelés par les fonctions logicielles. Cela est naturellement le cas en phase de développement, en se basant sur la batterie de tests unitaires. Le croisement des tests unitaires et des tests d'intégration dans l'environnement de travail final devrait permettre de déceler le maximum de défaillances et de les résoudre.

6.2. Liste des modes de défaillance génériques

Il est possible d'extraire une liste des modes de défaillances génériques à partir des modes de défaillances déterminés dans cette étude. Cette liste est formée des modes de défaillance les plus fréquents et représente 80% de la totalité des modes de défaillances recensés (tableau 10).

Tableau 10 : Liste des modes de défaillance génériques.

1	Manque d'information
2	Perte d'information
3	Données incohérentes
4	Perte de données
5	Intégrité de données violée
6	Redondance de données
7	Condition non considérée
8	Règle de gestion violée
9	Calcul erroné
10	Traitement erroné
11	Traitement mal fait
12	Traitement irrégulier
13	Traitement difficile
14	Traitement non fait
15	Ne s'imprime pas
16	Manque de validation

Tableau 10 : Liste des modes de défaillance génériques *(suite)*

17	Perte de sortie
18	Sortie erronée
19	Consultation impossible
20	Difficulté d'insertion
21	Manque d'option
22	Ne fonctionne pas
23	Perte de fonctionnalité
24	Manque de fonction
25	Mauvaise apparence
26	Affichage mal fait
27	Message d'erreur inutile
28	Information superflue
29	Système se plante
30	Système ralenti
31	Manque de spécification
32	Version n'est pas à jour
33	Format incorrect
34	Exception non considérée
35	Manque de données

Cette liste servira comme support pour le responsable de la maintenance, le développeur et le groupe de l'AEEL. En effet, tenir compte de ces modes de défaillances permettra, premièrement, de prévenir et minimiser les erreurs lors du développement. Deuxièmement, de faciliter l'analyse prévisionnelle des défaillances et troisièmement, d'aider à identifier facilement les défaillances en cas de la maintenance. Il est évident que dans ces trois situations, cet outil permet de prévenir et minimiser les erreurs dans le programme. Ce qui, d'une part, augmentera la fiabilité du logiciel et d'autre part, permettra de minimiser l'effort de la maintenance et de l'analyse des défaillances, ce qui diminue les délais et les coûts de développement et de la maintenance.

En cas d'élaboration de l'AEEL, cette liste sera utilisée lors de la phase du recensement des modes de défaillances.

6.3. Interprétation des résultats

Le tableau 11 présente le nombre de défaillances et leur pourcentage par famille.

Tableau 11 : Occurrence et pourcentage des modes de défaillances par famille

	Nombre de défaillances	Défaillances (%)
Traitement des données	143	15,61
Fonctionnalités	138	15,07
Sortie	123	13,43
Ergonomie	119	12,99
Données	103	11,24
Information	95	10,37
Règles de gestion	61	6,66
Entrée	48	5,24
Performance	44	4,80
Divers	25	2,73
Paramétrage	17	1,86
Total	**916**	

Les défaillances les plus fréquentes sont celles correspondantes aux classes traitement de données, fonctionnalité, sortie, ergonomie, données et information. Ceci peut être expliqué par la raison d'être du logiciel, soit le calcul des besoins nets. En effet, les fonctions essentielles sont basées sur des algorithmes et des formules de calculs mathématiques. Ces fonctions sont les plus sollicitées par les utilisateurs. Les défaillances sont concentrées, à hauteur de 80%, sur les fonctions essentielles du logiciel. La sollicitation de ces dernières ainsi que leur complexité pourrait donner une explication raisonnable à cette grande fréquence de défaillance.

La criticité de la défaillance varie d'un système à l'autre, d'une application à l'autre et est subjective. La subjectivité réside dans le fait que la criticité accordée dépend de la perception du rapporteur de la défaillance. En effet, plusieurs variables peuvent influencer le jugement, notamment l'environnement, la situation, la contrainte temps, etc.

Dans le cas de cette étude, les modes de défaillances jugés les plus critiques sont ceux avec les criticités "majeur", "crash" et "bloquant", dans le sens où ces défaillances affectent des fonctions essentielles ainsi que la disponibilité du système.

Les données portant sur les défaillances critiques par familles sont recensées dans le tableau 12.

Tableau 12 : Occurrence et pourcentage des défaillances critiques par famille.

	Nombre de défaillances critiques	**Défaillance critique par famille (%)**	**Défaillance critique totale (%)**
Données	23	**22,33**	21,90
Ergonomie	17	**14,29**	16,19
Traitement des données	13	**9,09**	12,38
Fonctionnalité	11	**7,97**	10,48
Information	11	**11,58**	10,48
Sortie	7	**5,69**	6,67
Règles de gestion	7	**11,48**	6,67
Entrée	6	**12,50**	5,71
Performance	6	**13,64**	5,71
Paramétrage	3	**17,65**	2,86
Divers problèmes	1	**4,00**	0,95
	105		

Les familles ayant le nombre de défaillances élevé sont : données, paramétrage, ergonomie, performance et entrée. Les modes de défaillance critiques sont concentrés dans les familles Données, Ergonomie, Traitement de données, Fonctionnalité, Information, Sortie et Règles de gestion. La figure 13 donne une classification du nombre de modes de défaillance critiques et le nombre total des modes de défaillance par famille.

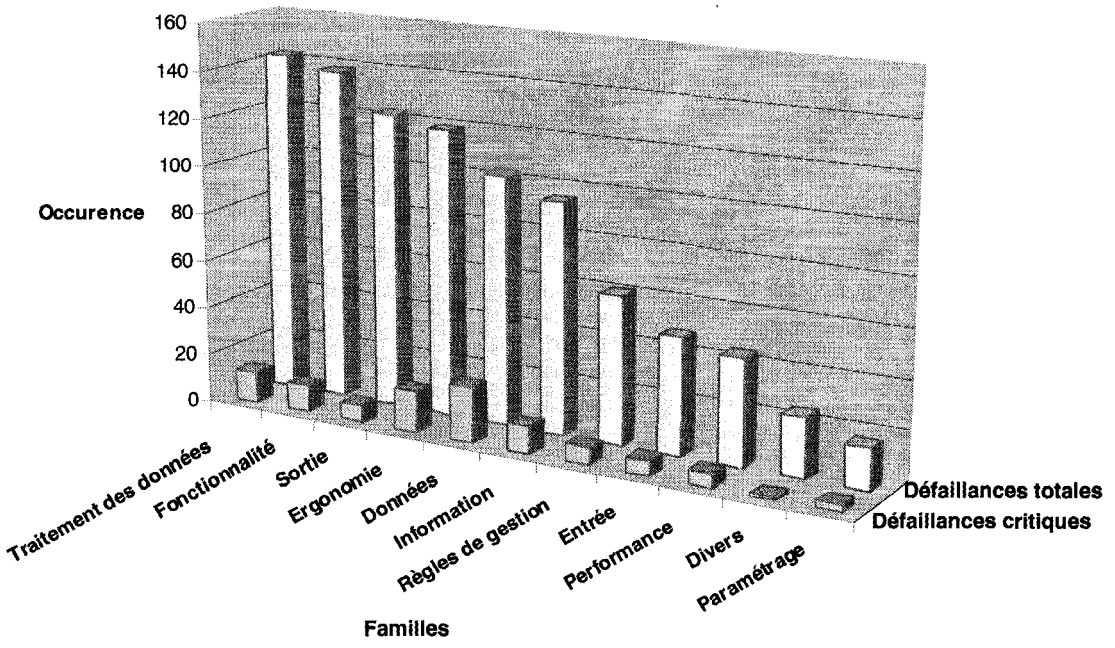

Figure 13 : Nombre de modes de défaillance critiques par rapport au nombre total des modes de défaillance classées par famille.

Bien que les modes de défaillances les plus fréquents et les plus critiques appartiennent aux mêmes familles, le nombre de défaillances critiques n'est pas proportionnel au nombre total de défaillances par famille.

Le tableau 13 met en évidence les modes de défaillances critiques selon leur occurrence.

Tableau 13 : Défaillances critiques vs défaillances totales par famille.

Famille	Mode de défaillance	Occurrence des défaillances	Occurrence des défaillances critiques
Entrée	Problème d'insertion	17	4
	Saisie impossible/difficile	13	1
	Application ne démarre pas, run-error	7	1
Sortie	Rapport incorrect	14	1
	Consultation impossible, difficile	11	2
	Impression impossible	10	2
	Rapport non fonctionnel, ne s'ouvre pas	6	1
	Manque de spécification	2	1
Fonctionnalité	Manque d'option	29	1
	Tri/filtre non fonctionnel	23	7
	Perte de fonctionnalité	12	1
	Composant ne fonctionne pas	4	2
Divers	MAJ version non faite	7	1
Paramétrage	Valeur par défaut erronée	11	2
	Recherche mal configurée	1	1
Information	Manque d'information	66	6
	Information superflue	14	3
	Perte d'information	12	2
Performance	Système lent, se plante, runtime	32	6
Données	Perte de données	39	14
	Incohérence de données	22	3
	Redondance de données	15	3
	Manque de données	7	1
	Manque de valeur (liste)	9	2
Traitement des données	Calcul erroné	34	10
	Traitement non fait	8	1
	Opération superflue/inutile	8	1
	Unité inappropriée/ incorrecte	2	1
Ergonomie	Mauvaise apparence	27	4
	Affichage mal fait	25	3
	Format non standard, incorrect, inapproprié	14	5
	Espace insuffisant	8	1
	Manque d'un message d'erreur	4	1
	Redondance d'endroits de saisie	2	1
	Application ne se ferme pas	2	1
	Menu ne se déploie pas	2	1
Règles de gestion	Règle de gestion violée	19	2
	Intégrité de données violée	19	2
	Condition non considérée	11	1
	Exception, cas particulier non considéré	9	2

La subjectivité dans le jugement de la criticité des défaillances est compréhensible dès lors, qu'il s'agit de défaillance ayant un impact direct sur les résultats attendus par l'utilisateur final et que ces résultats sont importants pour l'utilisateur. Plus le résultat attendu habituellement présente une « erreur », plus son niveau de criticité augmente. En d'autres termes, l'utilisateur ne supporte plus qu'il soit bloqué par un logiciel dans l'avancement de son travail.

6.4. Analyse des causes des défaillances

L'analyse des causes de défaillances permettra de relier les défaillances aux causes et les dites causes aux phases du cycle de vie où l'erreur à été introduite.

Une meilleure étude des modes de défaillance critiques et les plus fréquents permet d'améliorer la robustesse du système logiciel. En effet, d'une part, les défaillances peuvent être éliminées par construction, ce qui permet de diminuer les événements qui les causent, notamment le temps de la maintenance, et ainsi assure une meilleure disponibilité du système. D'autre part, la prise en considération des défaillances critiques permet de diminuer la probabilité de corruption des données.

6.4.1. Liste des causes de défaillance

Ce paragraphe est consacré à l'étude des modes de défaillance critiques. En se basant sur l'étude faite dans le chapitre 5, les causes des défaillances critiques sont décelées et regroupées par mode de défaillance. L'étape suivante consiste à raffiner la liste des causes de défaillance obtenue par la compilation des similarités (annexe B.). La liste finale des causes ainsi que leur occurrence est donnée dans le tableau 14.

Tableau 14 : Liste des causes des défaillances critiques et leurs occurrences.

Causes	Occurrence
Requête/ formule erronée	30
Critères, paramètres erronés	17
Lien non fait, perdu	22
Condition, exception non considérée	15
Manque, perte de fonctionnalité	12
Procédure erronée/ inadéquate	12
Mise à jour non faite, n'est pas automatique	11
Demande de changement, correction, ajout	10
Manque de support/formation	10
Lien Excel/configurateur	7
Filtre non fonctionnel	7
Erreur utilisateur	7
Erreur de programmation	7
Données non indexées	6
Format inadéquat, non spécifié, non standard	6
Changement de pratiques	5
Anomalie	5
Traitement non optimisé	5
Entrée non permise, erronée	5
Manque de sortie (rapport, champ...)	5
Message d'erreur inadéquat	4
Mauvaise apparence	4
Données perdues	3
Violation de règles de gestion	3
Base de données non restaurée	3
Manque d'option	3
Fonction erronée	3
Procédure complexe	2
Unité incorrecte	2
Manque de robustesse	2
Poste client (mémoire)	2
Perte d'informations	2
Problème d'affichage de données	1
Nom du rapport très long	1
Dates non considérées	1
Libellé incorrect	1

6.4.2. Causes de défaillance lors du cycle de vie d'un logiciel

L'application de l'AEEL permet de déterminer les erreurs et les défauts susceptibles de générer des défaillances. En analysant ces derniers, il est possible de déterminer au cours de quelle phase de cycle de vie du logiciel les erreurs peuvent être introduites (Tableau 15). Ces liens sont établis sur la base des informations contenues dans les bogues et sur les processus de différentes phases de cycle de vie d'un logiciel.

6.4.3. Erreurs, défauts versus les défaillances

Les défaillances, défauts et erreurs sont reliés par des relations de cause à effet. L'analyse détaillée de ces derniers permet de relier les défaillances à leurs causes. L'élaboration des liens est établie sur la base des informations contenues dans les bogues. Le résultat est présenté sous forme d'une liste de vérification (Tableau 16).

Ces deux listes de vérification présentées dans les tableaux 15 et 16 permettent de prévenir les défaillances potentielles. Le fait de disposer de telles listes de vérification diminue les risques d'oubli et d'omission, et aide à déterminer les causes potentielles lors de la correction d'une défaillance.

Tableau 15 : Causes de défaillance lors du cycle de vie d'un logiciel.

	Entrée non permise, erronée	Manque de sortie (rapport, champ...)	Manque, perte de fonctionnalité	Fonction erronée	Filtre non fonctionnel	Critères, paramètres erronés	Format inadéquat, non spécifié, non standard	Message d'erreur inadéquat	Mauvaise apparence	Perte d'informations	Lien non fait, perdu	Lien Excel/configurateur	Données perdues	Violation de règles de gestion	Condition, exception non considérée	Poste client (mémoire)	Données non indexées	Traitement non optimisé	Manque de robustesse	Procédure erronée/ inadéquate	Requête/ formule erronée	Procédure complexe	Erreur de programmation	Mise à jour non faite, n'est pas automatique	Manque de support/formation	Demande de changement, correction, ajout	Erreur utilisateur	Anomalie	Unité incorrecte	Manque d'option	Base de données non restaurée	Nom très long du rapport	Dates non considérées	Problème d'affichage de données	Libellé incorrect	Changement des pratiques
Maintenance				✓	✓	✓	✓				✓	✓	✓						✓	✓	✓		✓	✓	✓			✓		✓	✓		✓			✓
Exploitation										✓			✓			✓			✓	✓				✓	✓	✓	✓	✓			✓	✓		✓		
Intégration et installation sur site											✓	✓								✓					✓			✓							
Tests			✓	✓							✓	✓								✓	✓														
Codage, traduction des données				✓	✓		✓	✓	✓		✓	✓		✓	✓		✓	✓	✓	✓	✓		✓	✓				✓		✓		✓	✓	✓	
Conception détaillée						✓		✓	✓									✓	✓			✓	✓					✓		✓					
Conception préliminaire			✓																			✓							✓						
Élaboration des spécifications		✓	✓			✓	✓							✓	✓							✓							✓	✓					
Élaboration d'un CdCF	✓	✓	✓											✓		✓												✓		✓					

Tableau 16 : Erreurs, défauts versus les défaillances.

	Entrée non permise, erronée	Manque de sortie (rapport, champ...)	Manque, perte de fonctionnalité	Fonction erronée	Filtre non fonctionnel	Critères, paramètres erronés	Format inadéquat, non spécifié, non standard	Message d'erreur inadéquat	Mauvaise apparence	Perte d'informations	Lien non fait, perdu	Lien Excel/configurateur	Données perdues	Violation de règles de gestion	Condition, exception non considérée	Poste client (mémoire)	Données non indexées	Traitement non optimisé	Manque de robustesse	Procédure erronée/ inadéquate	Requête/ formule erronée	Procédure complexe	Erreur de programmation	Mise à jour non faite, n'est pas automatique	Manque de support/formation	Demande de changement, correction, ajout	Erreur utilisateur	Anomalie	Unité incorrecte	Manque d'option	Base de données non restaurée	Nom très long du rapport	Dates non considérées	Problème d'affichage de données	Besoin non spécifié, non exprimé	Changement des pratiques
Rapport incorrect					✓																															
Consultation impossible, difficile			✓								✓										✓															
Problème d'impression	✓		✓									✓				✓				✓	✓											✓				
Rapport non fonctionnel											✓										✓															
Manque de spécification																			✓							✓									✓	
Manque d'option			✓												✓				✓																	
Tri/filtre non fonctionnel			✓	✓	✓	✓	✓														✓			✓	✓	✓									✓	
Perte de fonctionnalité								✓																			✓									
Composant ne fonctionne pas																																				
MAJ version non faite																								✓												
Valeur par défaut erronée						✓																							✓							
Recherche mal configurée						✓															✓															
Manque d'information		✓									✓										✓			✓		✓					✓				✓	
Perte d'information						✓					✓							✓			✓		✓				✓									
Information superflue					✓				✓															✓		✓		✓								
Incohérence de données						✓					✓				✓						✓						✓									
Perte de données	✓		✓	✓			✓				✓	✓					✓			✓	✓					✓	✓								✓	
Redondance de données											✓			✓	✓						✓	✓	✓				✓				✓					
Manque de données			✓							✓	✓		✓								✓						✓			✓						
Système lent, se plante			✓		✓		✓				✓	✓				✓	✓	✓		✓				✓												

Tableau 16 : Erreurs, défauts versus les défaillances *(suite)*

	Entrée non permise, erronée	Manque de sortie (rapport, champ...)	Manque, perte de fonctionnalité	Fonction erronée	Filtre non fonctionnel	Critères, paramètres erronés	Format inadéquat, non spécifié, non standard	Message d'erreur inadéquat	Mauvaise apparence	Perte d'informations	Lien non fait, perdu	Lien Excel/configurateur	Données perdues	Violation de règles de gestion	Condition, exception non considérée	Poste client (mémoire)	Données non indexées	Traitement non optimisé	Manque de robustesse	Procédure erronée/ inadéquate	Requête/ formule erronée	Procédure complexe	Erreur de programmation	Mise à jour non faite, n'est pas automatique	Manque de support/formation	Demande de changement, correction, ajout	Erreur utilisateur	Anomalie	Unité incorrecte	Manque d'option	Base de données non restaurée	Nom très long du rapport	Dates non considérées	Problème d'affichage de données	Besoin non spécifié, non exprimé	Changement des pratiques
Calcul erroné						✓					✓	✓			✓					✓	✓		✓		✓		✓	✓	✓					✓			
Traitement non fait																		✓							✓											
Opération superflue/inutile																				✓			✓						✓	✓					✓	
Unité inappropriée/ incorrecte															✓																					
Mauvaise apparence		✓	✓				✓		✓		✓															✓									✓	
Affichage mal fait					✓		✓			✓	✓				✓						✓														✓	
Format non standard, incorrect,						✓	✓				✓	✓									✓		✓			✓		✓						✓	✓	
Espace insuffisant																																				
Manque d'un message d'erreur																						✓														
Redondance d'endroits de saisie																							✓		✓											
Application ne se ferme pas toute seule																								✓				✓								
Menu ne se déploie pas																							✓		✓											
Règle de gestion violée											✓			✓	✓						✓		✓			✓				✓						
Intégrité de données violée			✓								✓		✓		✓									✓												
Condition non considérée															✓													✓								
Exception non considéré																				✓			✓				✓	✓							
Problème d'insertion	✓		✓								✓				✓			✓		✓								✓							
Saisie impossible/difficile																				✓				✓	✓										
Application ne démarre pas															✓																					

6.5. Recommandations

Il paraît évident que la fiabilité d'un logiciel est étroitement liée au design et au développement. Elle est caractérisée par la fiabilité des différents modules qui forment le logiciel. La probabilité de défaillance d'un module est reliée à sa complexité [18]. La compétence du développeur peut, d'une certaine façon, atténuer la complexité des différents modules. La réutilisation de modules ou l'acquisition des modules standards (composants) peut aussi diminuer la complexité.

Vu la corrélation des défauts et de la complexité d'un logiciel, il s'avère nécessaire de faire un bon choix des jeux de tests. En effet, une bonne couverture du test et un bon balayage de la totalité des fonctions, séquences d'exécution, modules, etc. permettent de découvrir plus de défauts et erreurs latents. Ceci est dû au fait que l'apparition de ces derniers dépend de la fréquence et de la manière d'exécuter le programme. Cependant, le temps de test doit être optimisé pour ne pas en consommer beaucoup et générer des coûts élevés.

L'optimisation d'un programme, que ce soit avec le choix de modèles simples ou par l'indexation de données, permet d'accélérer le traitement et d'augmenter le temps de réponse du système logiciel. De la phase test d'intégration jusqu'à la qualification sur le site d'utilisation, il est nécessaire de s'assurer de la présence de toutes les spécifications dans le produit, vérifier et valider le produit final et estimer la fiabilité opérationnelle.

La fréquence des défaillances critiques observées est plus élevée pour les fonctions essentielles du logiciel. La complexité de ces fonctions y est pour quelque chose. Un plan d'action ciblé sur ces fonctions doit alors être élaboré par l'éditeur du logiciel. Ce travail concentré sur les fonctions essentielles ne devrait pas être consommateur de ressources puisque la compétence de l'éditeur est forcément dans ces dites fonctions. A l'issue du plan d'action et à

chaque fois qu'une défaillance est corrigée, l'impact sur l'amélioration de la fiabilité est important et très visible. Cette action est donc optimisée.

Un nombre important de bogues est d'origine ergonomique ou est lié à une mauvaise interprétation des données d'entrées/sorties par les utilisateurs du produit. L'implication de l'utilisateur final du produit dès la phase en amont des tests permettrait de s'assurer au mieux de la compatibilité du produit avec les besoins utilisateurs, de s'assurer de l'adéquation du produit avec la manière de procéder de l'utilisateur et éviterait aux développeurs une mauvaise interprétation des messages utilisateurs. Un concepteur peut interpréter les besoins clients d'une manière incorrecte tout en les programmant correctement.

CHAPITRE 7

CONCLUSION GÉNÉRALE

Cette étude développe un cadre d'analyse de défaillances, des défauts et des erreurs tout au long du cycle de vie d'un logiciel. Une compréhension de la dynamique de ces derniers a permis de développer des outils pour améliorer la fiabilité des logiciels. Ces outils consistent en des listes et des listes de vérification qui sont déterminées par l'application de la méthode AEEL sur un cas réel.

La méthodologie adoptée était celle propre à la recherche-action. Le processus diagnostic consiste à faire un recensement des écrits dans le domaine de la fiabilité des logiciels et une étude du monde réel. Plus spécifiquement, rechercher tout ce qui est en relation étroite avec les analyses de défaillances, erreurs et défauts durant le cycle de vie d'un logiciel. Quant au processus de planification, il consiste à proposer un support d'aide à la prévention des défaillances potentielles dès la phase de gestation, de sorte à augmenter la fiabilité tout en respectant un budget et des délais donnés. Les processus réalisation et évaluation visent à appliquer sur un cas réel ce qui a été planifié, et de mesurer les écarts avec les objectifs prédéterminés. Enfin, le processus apprentissage consiste à ajuster les résultats et à élaborer une expérience spécifique dans le cas d'étude.

L'application de la méthode AEEL sur l'étude de cas a donné de bons résultats dans l'ensemble. D'abord, le classement des modes de défaillances selon plusieurs familles, à savoir : Entrée, Sortie, Fonctionnalité, Paramétrage, Ergonomie, Informations, Données, Règles de gestion, Performances, Traitement des données et Divers. Ensuite, la détermination des modes de défaillances les plus fréquents, les plus critiques et une liste des modes de défaillance génériques. Enfin, la présentation sous forme de listes de vérification, d'une part, les liens entre les défaillances et leurs causes, et d'autre part entre les causes et le moment (phase de cycle de vie du logiciel) de leur introduction dans le programme.

Cette étude permettra sûrement d'aider les développeurs à prévenir des défaillances susceptibles de survenir lors de la phase d'exploitation d'un logiciel. Ce qui permettra d'améliorer la fiabilité des produits et des services associés à savoir, une meilleure disponibilité, une réduction des coûts de maintenance ainsi que le gain de confiance des clients. Ceci par un meilleur pilotage des projets de développement, une augmentation de la qualité, une augmentation de la productivité et une réduction du coût du projet.

Il existe plusieurs limites à cette étude. Premièrement, malgré que l'échantillon soit significatif, cette étude s'est limitée à l'analyse des défaillances d'un seul logiciel. Donc, ces résultats sont plus valides et applicables pour les logiciels de gestion intégrée. Deuxièmement, les données sur "Mantis" s'étalent sur 2 années, ce qui fait qu'il n'a pas une grande variation du personnel sur cette période, d'où l'effet de la compétence des développeurs et leur nombre n'est pas considéré dans le cadre de cette étude. Troisièmement, dans cette étude, les langages de programmation utilisés sont les mêmes, ce qui neutralise l'effet du langage de programmation sur la fréquence de défaillances.

Finalement, cette étude conduit à plusieurs nouvelles pistes de recherche éventuelles. Premièrement, il serait intéressant de reprendre cette étude dans le cadre de différents types de logiciels afin de mieux généraliser les résultats de

cette étude. Deuxièmement, il serait utile d'étudier le besoin des entreprises à optimiser les processus de développement et de la maintenance et de quantifier l'impact de l'introduction des modes de défaillance génériques dans ces processus. Troisièmement, la conception d'un progiciel de gestion des défaillances permettra aux différents utilisateurs de résoudre des problèmes avant de s'adresser aux développeurs et ainsi proposer des programmes de soutien, de formation et de nouveaux outils de gestion.

RÉFÉRENCES BIBLIOGRAPHIQUES

[1] Amel, G. (2000). Notes de cours : Systèmes d'information et d'aide à la décision. Tunis : École nationale d'Ingénieurs de Tunis.

[2] Asad Ch-A., Irfan Ullah M. et Jaffar-Ur Rehman M. (2004). An Approach for Software Reliability Model Selection. Proceedings of the 28th Annual International Computer Software and Applications Conference (COMPSAC'04). Islamabad, Pakistan : Mohammad Ali Jinnah University.

[3] Banker R, Davis G et Slaughter S. (1998). Software development practices, software complexity and software maintenance performance: a field study. *Management science*, Vol. 44, No. 4 p. 433-450.

[4] Berman O et Cutler M. (2004). Resource allocation during tests for optimally reliable software. *Computers & Operations Research* 31 p. 1847–1865.

[5] Boehm, B.-W., Horowitz E, Madachy R., Reifer Donald, Clark B.-K., Steece B., Brown A.-W., Chulani S. et Abts C. (2000). Software cost estimation with COCOMO II. New Jersey : Pertinence Hall PTR.

[6] Favre, J.-M. (1997). Outils pour la maintenance et la rétro ingénierie des logiciels : l'état de l'art. Software Logistics'97 (LOGISTICS'97), Paris (France).

[7] Fournier, J.-P. (1993). *Fiabilité du logiciel : concepts, modélisation, perspectives*. Éditions : Hermes, Paris.

[8] Garin, H. (1994). AMDEC/AMDE/AEEL : l'essentiel de la méthode. Paris : Association française de normalisation (AFNOR).

[9] IEEE Std 982.2-1988. (1989). IEEE Guide for the Use of IEEE Standard Dictionary of Measures to Produce Reliable Software. IEEE : New York.

[10] Kan, S. (2003). Metrics and models in software quality engineering. Boston: Addison-Wesley

[11] Laprie, J.-C. (1995). *Guide de sûreté de fonctionnement*. Toulouse, France : Cépaduès-Éditions, 2ème édition.

[12] Laprie, J.-C. Courtois, B., Gaudel, M-C. et Powell, D. (1989). Sûreté de fonctionnement des systèmes informatiques. Paris, France : Bordas.

[13] Lee, M., Pham H. et Zhang, X,. (1999). A methodology for priority setting with application to software development process. *European Journal of Operational Research* 118 p. 375-389.

[14] Littlewood B. (1980). The Littlewood-Verall Model for software reliability compared with some rivals. *The journal of systems and software* 1, p. 251-258.

[15] Lyu M.-R. (1996). *Handbook of software reliability engineering.* McGraw-Hill New York ; Washington, D.C. : IEEE Computer Society Press.

[16] McCabe, Thomas J. (1976). A complexity measure. *IEEE transactions on software engineering*, Vol. SE-2, No 4, Avril 1976.

[17] Menzies, T., Di Stefano, J-S. et Chapman, M. (2003). Learning Early Lifecycle IV&V Quality Indicators. Lane Department of computer Science, West Virginia University.

[18] Munson J -C. (1996). Software faults, software failures and software reliability modeling. *Information and software technology* 38 p. 687-699.

[19] Pham H. (2003). Software reliability and cost models: Perspectives, comparison, and practice. *European Journal of Operational Research*, Volume 149, Issue 3, 16 September 2003, Pages 475-489.

[20] Pham H et Zhang X. (2003). NHPP software reliability and cost models with testing coverage. *European Journal of Operational Research*, Volume 145, Issue 2, 1 March 2003, Pages 443-454

[21] Pham, H. et Zhang, X. (1999). A software cost model with warranty and risk costs. *IEEE Transactions on Computers* 48 (1) p. 71–75.

[22] Pham, H et Zhang X. (1999). Software release policies with gain in reliability justifying the costs. *Annals of software engineering,* 8 p. 147-166.

[23] Pham, H et Zhang, X. (2000). An analysis of factors affecting software reliability. The journal of systems and software 50 p. 43-56.

[24] Pham, L. et Pham, H. (2001). A bayesian predictive software reliability model with pseudo-failures. *IEEE transactions on reliability*, Vol 31, No. 3.

[25 Rausand, M. (1998). Reliability centered maintenance. *Reliability Engineering and System Safety* 60 (1998) p.121-132.

[26] Saint-Amant, G. (1995). Le management est-il la pratique qui précède la pratique? Ou quelle est la place de la recherche-action au sein de la formation en gestion?. *Collection histoire, gestion. Organisations,* 4, 1995.

[27] Séminaire de formation (1998). Maintenance basée sur la fiabilité. Centre de veille en équipement de transport terrestre.

[28] Shao J et Wang Y. (2003). A new measure of software complexity based on cognitive weights. *Can. J. Elect. Comput. Eng.*, Vol. 2.

[29] Sommerville, I. (1996). Software engineering. 5th ed. Wokingham, Angleterre; Don Mills, Ont. : Addison-Wesley.

[30] Xie M et Pham H. (2005). Modeling the reliability of threshold weighted voting systems. *Reliability Engineering and System Safety* 87 p. 53–63.

[31] Zhang X, Shin M -Y et Pham H. (2001). Exploratory analysis of environmental factors enhancing the software reliability assessment. *The journal of systems and software* 57 p. 73-78.

ANNEXE A : ATTRIBUTS D'UN BOGUE

ID : numéro (entier) de réclamation

Catégorie : le module du système de gestion auquel se rapporte le bogue.

Sévérité : la criticité accordée au bogue par le rapporteur. L'échelle considérée par ordre de criticité croissant : fonctionnalité, simple, texte, cosmétique, mineur, majeur, crash, bloquant.

Reproductibilité : tâches pour reproduire le bogue.

Date de soumission : Le jour et l'heure de la soumission d'un bogue.

Rapporteur : personne rapportant le bogue (utilisateur, client, testeur, administrateur...)

Assigné à : personne qui se charge de réparer le bogue (développeur...)

Priorité : l'ordre de priorité de l'événement (aucune, basse, normale, élevée, urgente, immédiate)

État : Public ou privé

Plate-forme : plate forme utilisée

Build : compilation

Résumé : un résumé de l'événement survenu.

Description : description détaillée de l'événement survenu.

Étapes pour reproduire : étapes pour reproduire la défaillance

Informations complémentaires : informations jugées utiles pour aider à résoudre le bogue

Fichiers attachés : fichiers attachés au bogue

Notes : commentaires ou notes ajoutés pour détailler le bogue

Dernière mise à jour

Version

Version du produit

Résolution

Projection

ANNEXE B : REGROUPEMENT DES CAUSES DE DÉFAILLANCE

ID Bogue	Causes possibles	Mode de défaillance
Données		
2454 1991 1618 1567 1582 1574 1583 1435 1600 2093 1506 1944 2045 2284	(4) changement de saisie de gamme, changement des pratiques de lancement d'un OF - OF se lance au moment de l'ordonnancement, changement des manières de saisie de gamme et/ou lacement de OF - pointeur erroné - fonction erronée - cases vides - entrée de valeurs non permises - entrée manuelle de données - manque de support/ formation - pointeur erroné - fonction erronée - lien Excel/configurateur - pointeur erroné - enregistrement des données ne se fait pas des 2 côtés - pièces recréer dans Excel - erreur introduite lors de la maintenance - interface config/Excel - anciens paramètres non restaurer - mauvaise procédure d'insertion au milieu d'un job - données perdues - valeurs entrées ne pointent pas sur le bon champ -champ n'accepte pas d'entrée - procédure erronée - grand nombre des OF (36000)- perte de fonctionnalité dans le programme - lien erroné - OF pointe sur un article inexistant	perte de données
643 1336 1576	données historiques non déployées - manque d'option - manque outils de suivi, perte d'informations - requête erronée - liens perdus - perte de fonctionnalité - erreur utilisateur	manque de données
2180 1616 1328	erreur utilisateur - coefficients non pris en compte - cartes de temps non punchées à la fin du shift - condition non prise en compte - formule erronée - pointeur erroné- condition n'est pas prise en compte - condition/requête erronée	incohérence de données
1787 2355 1689	lien erroné avec les lignes de commandes - requêtes non fonctionnelles - BDD non restaurée - faute de programmation - violation de règles de gestion- liens entre tables - requête erronée (phase maintenance) - erreur utilisation - manque de contrainte sur la saisie des données - complexité de la procédure de saisie des crates	Redondance de données
Règles de gestion		
2355 1465	Base de données (BDD) non restaurée - faute de programmation - violation de règles de gestion- liens entre tables - requête erronée (phase maintenance) - erreur utilisation - règle de priorité non respectée - message manquant- lien non fait	Règle de gestion violée
1517	OF lancé 2 fois - procédure de lancement instable - autorisation du lancement OF 2fois	Condition non considérée
1619 2314	fichier perdu- fonctionnalité perdue – Mise à jour non faite - lien non fait - condition non considérée	Intégrité de données violée
2313 520	anomalie - procédure inadéquate - erreur utilisateur/ programmation	Exception, cas particulier non considéré

Information		
2095 1931 1685 2440 2350 2201	demande de changement et correction - manque de champ sur le rapport - besoin non spécifié lors de la conception - manque de rapport- manque de support - manque de champ nom sur bon de livraison - nouvelle exigence (besoin) non exprimée à la phase conception et développement - (ajout d'une colonne) - pointeur erroné - requête erronée - Mise à jour dans les champs n'est pas automatique - BDD non restaurée - champ manquant dans le rapport - libellé incorrect - demande de modification temporaire	manque d'information
2287 1878	formule erronée - erreur dans modèle choisi - OF se pointe sur d'autres champs - notification non fondée - erreur dans MRP - lien/requête erroné - Changement de valeur - mauvais paramètres - lien/pointeur erroné	Perte d'information
879 1939 2441	(2) demande de changement (amélioration/ajout) - filtre non fonctionnel - Mise à jour non faite - mauvais emplacement des informations	Information superflue
Fonctionnalités		
1780 1931 1685 1939 1682 1940 1906	nom des pièces non standardisé - fonctionnalité désactivée - filtre non fonctionnel - demande d'amélioration (ordre tri) - manque de support - critère de tri erroné - nouvelle exigence (besoin) non exprimés à la phase conception et développement - critère de tri erroné - filtre non fonctionnel – Mise à jour non faite - (demande d'amélioration/ajout) - filtre non fonctionnel, perte de fonctionnalité, - requête erronée - type et format du champ non pris en compte - tiret dans le noms des champs (ignoré par fonction de tri) - fonction de tri erronée	Filtre/tri non fonctionnel
2099 2098	désactivation de l'ancien formulaire - Mise à jour ne tient pas compte du formulaire	Composant ne fonctionne pas
2356	erreur utilisateur (erreur de saisie) - manque d'un message d'erreur	Perte de fonctionnalité
1565	priorité des OFL sur OFF - perte de fonctionnalité - exception non prise en compte -(phase conception)	Manque d'option
Sortie		
1787 2030	lien erroné avec les lignes de commandes - requêtes non fonctionnelles, perte de lien/requête - erreur introduite lors de la maintenance	Consultation difficile, impossible
2444	manque de la robustesse du rapport - amélioration - nouvelles exigences - mauvaise traduction du monde réel	Manque de spécification
1518 1914	poste client - perte de fonctionnalité - nom très long du rapport - existence de division par 0 - problème avec Excel/configurateur - entrée erronée - procédure d'impression erronée	Problème d'impression
1599	filtre non fonctionnel	Rapport incorrect
1669	les liens des champs avec les données - requête erronée - erreur introduite lors de la maintenance	Rapport non fonctionnel, ne s'ouvre pas

Divers		
1947	Mise à jour n'est pas faite	Mise à jour non faite
Entrée		
2284 1711 2440 1521	valeurs entrées ne pointent pas sur le bon champ - champ n'accepte pas d'entrée - procédure erronée - roulette de la souris créée les crates vides - procédure inadéquate - absence de liens automatique entre les tables - condition n'est pas prise en compte - manque de fonctionnalité - modèle inapproprié	Problème d'insertion
2183	manque d'opération automatique (Mise à jour) - manque de support (formation) -procédure inadéquate	Saisie impossible/difficile
1649	manque de condition (création de crates vides/bon de livraison)	Application ne démarre pas, run-error
Traitement des données		
2209 1970 2021 1804 1633 1501 1705 1517 2045 1680	(8) requête/formule/pointeur/erroné - dates non considérées - paramètres de la planification incorrects - lien manquant - erreur d'utilisateur - manque de communication de la manière du fonctionnement de la nouvelle politique - erreur de programmation - (3) erreur introduite lors de la maintenance - unité erronée (pouce) - OF lancé 2 fois - procédure de lancement instable - autorisation du lancement OF 2 fois - manque de contrainte (condition) - pièces recréer dans Excel - anciens paramètres du rapport non restaurer - calcul n'est pas bon - paramètres erronés - oubli de changement d'unité (division par 144)	calcul erroné
2283	Manque de support - formation utilisateur -requête non optimisée	traitement non fait
1907	condition non prise en compte - n'est pas spécifiée dans les besoins client - erreur de programmation - unité incorrecte	unité inappropriée
1894	formule de calcul erroné - manque de support/formation	opération non faite
1257	procédure actuelle inadéquate - manque d'option	opération superflue/ inutile
Ergonomie		
198 1878 1935	nouvelle exigence (besoin) non exprimée à la phase conception et développement, perte d'informations- filtre non fonctionnel - lien erroné - mauvais paramètres - lien/pointeur erroné - requête mal faite (manque de précision) - manque de condition - besoin non spécifié	affichage mal fait
1669	les lieux des champs avec les données - requête erronée - erreur introduite lors de la maintenance	marge superflu
2095 879 1681	demande de changement et correction - besoin non spécifié lors de la conception - manque de rapport - labors n'apparaissent pas sur la même ligne - demande de changement (amélioration) – un format inadéquat - mauvaise mise en page - perte de fonctionnalité/lien	mauvaise apparence
1684 2237 2094	lien configurateur/rapport mal fait/erroné - format non spécifié - erreur de programmation - anomalie - demande de changement et correction - besoin non spécifié lors de la	Format non standard, incorrect, inapproprié

1680 1744	conception - Paramètres erronés - problème d'affichage de données - erreur dans le format du code pièces- requête erronée	
2177	mauvaise configuration du champ texte	Espace insuffisant
1686	erreur de conception/développement- manque de formation (utilisateur)	redondance d'endroits de saisie
1689	maque de message de notification - complexité de la procédure de saisie des crates	manque de message de validation
2207	patch non installé - désinstallé - n'est pas à jour	Application ne se ferme pas toute seule
1837	erreur dans le programme- support indisponible- manque de formation	menu ne se déploie pas
Paramétrage		
2440 1928	valeur par défaut absente - mauvais paramétrage - option désactivée	valeur par défaut erronée
2181	requête erronée - manque de paramètres filtre	recherche mal configurée
Performance		
1910 1898 1792 1780 1944 1991	(3) données non indexées - poste client (mémoire) - manque de patch- rapports non optimisé - Manque de filtre - Perte de fonctionnalité - Mise à jour non faite - beaucoup d'information dans les tables - procédure non optimisée - nom des pièces non standardiser - fonctionnalité désactivée - filtre non fonctionnel - lien Excel/configurateur - pointeur erroné - enregistrement des données ne se fait pas des 2 côtés - mauvaise procédure d'insertion au milieu d'un job	Système lent, se plante, runtime

Changement de pratiques

- Changement de saisie de gamme, changement des pratiques de lancement d'un OF

Entrée non permise, erronée

- Entrée erronée
- Entrée de valeurs non permises
- Entrée manuelle de données
- Champ n'accepte pas d'entrée

Traitement non optimisé

- Procédure non optimisée
- Rapports non optimisé
- Requête non optimisée
- Modèle inapproprié
- Erreur du modèle choisi

Données perdues

- Données historiques non déployées
- Fichier perdu

Manque, perte de fonctionnalité
- Perte de fonctionnalité
- Manque de fonctionnalité
- Manque outils de suivi
- Fonctionnalité désactivée

Manque de sortie (rapport, champ...)
- Manque de rapport
- Manque de champ sur le rapport
- Manque du champ nom sur bon de livraison (ajout d'une colonne)
- Champ manquant dans le rapport

Erreur de programmation
- Erreur dans MRP

Lien non fait, perdu
- Pointeur erroné
- Perte de lien/requête
- Perte de fonctionnalité/lien
- Liens perdus
- Absence de liens

Mise à jour non faite, n'est pas automatique
- Mise à jour dans les champs n'est pas automatique
- Mise à jour ne tient pas compte du formulaire
- Manque de patch
- Patch non installé - désinstallé - n'est pas à jour

Format inadéquat, non spécifié
- Erreur dans le format du code pièces
- Type et format du champ non pris en compte
- Tiret dans les noms des champs (ignoré par fonction de tri)
- Nom des pièces non standardisé

Manque d'option
- Option désactivée

manque de support/formation
- Manque de communication de la manière du fonctionnement de la nouvelle politique

Critères, paramètres erronés
- Mauvaise configuration du champ texte
- Critère de tri erroné
- Paramètres erronés
- Valeur par défaut absente
- Désactivation de l'ancien formulaire
- Cases vides
- Coefficients non pris en compte
- Mauvais paramétrage
- Mauvais paramètres
- Paramètres de la planification incorrects
- Anciens paramètres non restaurés

- Manque de paramètres

Procédure erronée/ inadéquate
- Procédure erronée
- Mauvaise procédure d'insertion au milieu d'un job
- Procédure inadéquate
- Procédure impression erronée
- Enregistrement des données ne se fait pas des 2 côtés
- OF se lance au moment de l'ordonnancement

Lien Excel/configurateur
- Problème avec Excel/configurateur
- Lien configurateur/rapport mal fait/erroné
- Pièces recréer dans Excel

Lien non fait, perdu
- Absence de liens
- Lien manquant
- Pointeur erroné
- OF pointe sur un article inexistant
- Valeurs entrées ne pointent pas sur le bon champ

Anomalie, Instable
- Changement de valeur
- Anomalie
- Roulette de la souris créée les crates vides
- Procédure de lancement instable

Requête/ formule erronée
- Calcul erroné
- Oubli de changement d'unité (division par 144)
- Requête erronée
- Formule erronée
- Existence de division par 0
- Requête mal faite (manque de précision)
- Requêtes non fonctionnelles

Manque d'indexation
- Beaucoup d'information dans les tables
- Grand nombre des OF (36000)

Procédure complexe
- Complexité de la procédure de saisie des crates

Mauvaise apparence
- Mauvais emplacement des informations
- Labors n'apparaissent pas sur la même ligne
- Mauvaise mise en pages
- Les lieux des champs avec les données

Condition, exception non considérée
- Condition non prise en compte
- Condition non considérée
- Exception non prise en compte
- Manque de contrainte sur la saisie des données
- Manque de condition
- OF lancé 2 fois
- Autorisation du lancement OF 2 fois
- Création de crates vides/bon de livraison
- Cartes de temps non punchées à la fin du shift
- Manque de contrainte

Manque de robustesse
- Priorité des OFL sur OFF

Message d'erreur inadéquat
- Notification non fondée
- Message (notification, erreur) manquant

Manque de robustesse
- Rapport non robuste