**Université Saâd DAHLEB de Blida**

**Faculté des sciences**

**Département d'Informatique**

Mémoire Présenté par :

**TOUAIBIA KHALED**

**BOUABAYA MOHAMED AMINE**

En vue d'obtenir le diplôme de Master

**Domaine : LMD**

**Filière : Informatique**

**Option : Ingénierie Logiciel**

**Organisme d'accueil : Laboratoire de Recherche pour le Développement des Systèmes Informatisés (LRDSI).**

**Sujet :**

# CONSTRUCTION D'UN OUTIL POUR LE CALCUL DE FIABILITE D'UN LOGICIEL

Soutenue le 13 Octobre 2010, devant le jury composé de :

| | | |
|---|---|---|
| Mr M. Ait Akkache | Maître de conférences, USDB | Président |
| Melle L. Toubaline | Maître de conférences, USDB | Examinatrice |
| Melle Y. Chikhi | Maître de conférences, USDB | Examinatrice |
| Mme S. OUKID-KHOUAS | Maître de conférences, USDB | Promotrice |

Promotion : 2009/2010

## Avant propos

*Respectons la tradition et dédions cette page aux remerciements. Bien sûr, ils ne peuvent être que nombreux, hétérogènes et incomplets. Et pourtant, cette page nous servira à fusionner et à combiner toute la gratitude et l'amitié que nous portons à tous ceux grâce à qui ce travail a été possible.*

*Tout d'abord, Nous remercions avant tout Dieu Tout-puissant qui nous a donné la force, le courage et la volonté pour réaliser ce travail. Nous tenons à remercier Mme S. OUKID-KHOUAS, notre directrice de thèse, de nous avoir acceptés durant cette année et pour son suivi, ses orientations et ses précieux conseils, et qui a su nous faire profiter de sa grande expérience.*

*Un grand merci à toutes nos familles pour leur présence, leur préoccupation et le souci qu'ils se sont fait pour nous, leurs encouragements et leur suivi, avec patience, du déroulement de notre projet.*

*Enfin, on doit aussi de remercier tous nos collègues et amis du MASTER 2. Ils sont nombreux et nous risquons d'en oublier. Ils nous ont fait l'honneur de nous accompagner et de nous supporter durant ces cinq longues années. Nous remercions, de tout cœur, tous ceux qui ont contribué de prés ou de loin à la réalisation de ce travail.*

## ملخص

تعقيدات النظم الحاسوبية في ازدياد مستمر والطلب على نوعية البرامج أصبح أكبر من أي وقت مضى , فبات من الضروري اليوم دعم وتطوير البرمجيات من خلال آليات مناسبة للتخطيط والتحليل والتحقق من تصاميم البرامج و مراقبتها, والتي تمكّن من تنمية الوثوق بها وزيادة فعاليتها وتعديلها بسهولة. موثوقية البرمجيات هي احتمال لضمان التشغيل الجيد لبرنامج لفترة معينة و في بيئة معينة من دون أي خلل أو فشل وظيفي . موثوقية البرمجيات تعد أيضا عاملا هاما في موثوقية النظم الاعتمادية. تقنيات نمذجة موثوقية البرامج تسعى لتحقيق الرقي بالمصداقية , ولكن قبل استخدام هذه التقنيات ، يجب علينا اختيار النموذج المناسب. قياسات البرمجيات ليست صحيحة دائما في كل الحالات , لأنه لا توجد أساليب دقيقة والكمية وضعت لقياس موثوقية البرمجيات بدون قيود مفرطة . فبالإمكان استخدام العديد من الأساليب لقياس مدى مصداقية البرنامج , ولكنه , من الصعب أن تأخذ بعين الاعتبار: وقت التطوير والتحكم في الميزانية مع الاعتماد على أساليب موثوقية البرمجيات في نفس الوقت . وهدفنا من هذه المذكرة هو تفصيل عملي لتصميم وبرمجة أداة لقياس مدى موثوقية البرمجيات , واستفادة من مميزاتها. وسيتم تصميم هذه الأداة باعتماد على شبكة بايز "Réseau Bayésien " , و التي ستجعل أيضا التنبؤات للموثوقية أكثر سهولة .


**كلمات البحث :** موثوقية البرمجيات , شبكة بايز , التنبؤ بالموثوقية.

## Résumé

La complexité des systèmes informatiques ne cesse de croître et les exigences à l'égard de la qualité logicielle sont de plus en plus grandes. Il est aujourd'hui essentiel que le développement logiciel puisse être soutenu par des mécanismes pertinents de planification, d'analyse, de conception et de vérification, ce qui permet l'élaboration de systèmes fiables, performants et facilement modifiables. La fiabilité du logiciel c'est la probabilité de son bon fonctionnement sans défaillances pour une période donné et un environnement donné. La fiabilité du logiciel est aussi un facteur important dans la fiabilité des systèmes. Les techniques de modélisation de la fiabilité des logiciels cherchent à atteindre la prospérité, mais avant d'utiliser ces techniques, il faut choisir le bon modèle qui s'appropriées pour notre cas. La mesure des logiciels n'est pas toujours parfaite dans la totalité des cas. Il n'y a pas de méthodes quantitatives exactes qui sont développé pour représenter la fiabilité du logiciel sans limitations excessives. Plusieurs approches peuvent être utilisées pour quantifier la fiabilité du logiciel, mais par contre, il est difficile de prendre en compte le temps de développement et le budget avec la fiabilité du logiciel en même temps. Il s'agit dans ce mémoire de concevoir et réaliser un outil d'aide à la quantification et l'estimation de la fiabilité d'un logiciel. Cette outil sera conçu et réalisé selon une approche basée sur le Réseau Bayésien, et qui permettra aussi la facilité de la prédiction de la fiabilité.


**Mots clés :** fiabilité logiciel, Réseau Bayésien, prédictions de la fiabilité.

## Abstract

The complexity of computer systems continues to grow and demands on software quality are becoming larger. It is essential today as software development can be supported by appropriate mechanisms for planning, analysis, design and verification, which enables the development of reliable, efficient and easily modifiable software. The software reliability is the probability of its operation without failure for a given period and a given environment. The reliability of the software is also an important factor in the reliability of systems. The modelling techniques of software reliability are seeking to achieve prosperity, but before using these techniques, we must choose the right model for our appropriate case. The measurement of software is ever perfect in all cases. There is no accurate quantitative methods are developed for the reliability of the software without excessive restrictions. Several approaches can be used to quantify the reliability of the software, but in reverse, it is difficult to take into account the development time and budget with the reliability of software at the same time. It is in this brief to design and implement a tool for quantifying and to estimate the reliability of the software. This tool will be designed and constructed using an approach based on Bayesian network, and will also make the predictions of reliability simpler.

**Keywords:** software reliability, Bayesian network, reliability predictions.

# Table de matière

## Liste des tableaux

## Liste des figures

# INTRODUCTION GÉNÉRALE

## Introduction générale

La modernisation de notre société durant les dernières décennies a conduit à une intégration massive des techniques informatiques dans la plupart des domaines de la vie courante. Les systèmes informatiques sont devenus des outils indispensables dans tous les domaines d'application civils et militaires : dans les systèmes de gestion d'information, dans les systèmes de transport, les systèmes de communication, ... etc. Compte tenu de la dépendance croissante de la société actuelle des systèmes informatiques, la sûreté de fonctionnement (SdF) de ces systèmes est devenue un enjeu important qui concerne tant les concepteurs, les réalisateurs et les fournisseurs que les utilisateurs des services délivrés par ces systèmes.

La défaillance d'un système informatique peut avoir des impacts catastrophiques sur son environnement par des conséquences de nature humaine ou bien économiques. Par exemple, les défaillances des systèmes informatiques constituent actuellement, la première source de sinistres industriels tels que perçus par les compagnies d'assurance, et leur coût excède, en Europe, les bénéfices de l'ensemble de l'industrie informatique.

Un système informatique défaille à cause de la présence de fautes aussi bien dans le matériel que dans le logiciel. L'introduction massive du logiciel dans les systèmes informatiques s'est accompagnée d'une augmentation de la complexité de ces systèmes et, par suite, des coûts du développement et de la maintenance.

De ce fait et compte tenu des conséquences catastrophiques que peuvent engendrer les fautes activées dans le logiciel, il est indispensable de disposer de techniques et de méthodes permettant d'obtenir et de fournir des systèmes **fiables** (sûrs de fonctionnement) afin de limiter la présence des fautes et surtout leurs impacts sur le service délivré aux utilisateurs.

L'utilisation de techniques évoluées de développement, de validation et de maintenance des systèmes informatiques permet de limiter la présence et les conséquences des fautes introduites dans ces systèmes. Cependant, à cause de la complexité des applications mises en œuvre par ces systèmes et de l'imperfection de toute activité humaine, de telles techniques ne peuvent conduire à un système qui soit exempt de fautes. Il est nécessaire donc de définir des mesures et des méthodes permettant d'une part, d'évaluer l'aptitude du logiciel à délivrer des services conformes aux spécifications, et d'autre part, d'aider les concepteurs à suivre l'évolution de la fiabilité du logiciel au cours de sa validation et pendant la vie opérationnelle.

Ceci fait l'objet des études portant sur la modélisation probabiliste de la fiabilité des systèmes informatiques. Les deux mesures principales qui sont considérées pour évaluer le comportement des systèmes sont la fiabilité et la disponibilité.

La modélisation de la SdF des systèmes informatiques a donné naissance à de nombreux travaux dont une large proportion a porté sur l'étude du comportement du matériel. Les travaux effectués sur le matériel ont été focalisés sur le développement de modèles de comportement d'un système en tenant compte de sa structure. Ces modèles permettent d'étudier la SdF d'un système au cours de sa vie opérationnelle en fonction de celle de ses différents composants en considérant une approche "boîte blanche". Cependant, à quelques rares exceptions, les modèles développés considèrent les fautes physiques comme étant la source principale de défaillance d'un système et ne tiennent pas compte des fautes de conception.

Les études de la SdF du logiciel pour la prédiction de fiabilité sont plus récentes et portent essentiellement sur la modélisation probabiliste, selon une approche "boîte noire", de fiabilité en mettant l'accent plus particulièrement sur les phases de développement et de validation : la fiabilité du logiciel résulte de l'élimination progressive des fautes de conception du logiciel au cours de son cycle de vie. Cependant, en dépit de l'importance majeure de la disponibilité de service pour certains types de logiciels tels que les systèmes de télécommunication et les logiciels de messagerie électronique par exemple, cette mesure n'a pas été considérée, à notre connaissance, dans les modèles d'évaluation de la de fiabilité du logiciel.

Par ailleurs, l'évaluation de la croissance de la fiabilité du logiciel a donné naissance à une multitude de modèles, une quarantaine environ, dont la majorité sont des modèles en temps continu qui caractérisent l'évolution dans le temps du comportement du logiciel dans un environnement donné. Ces modèles ne sont pas bien adaptés pour caractériser la fiabilité de certains systèmes tels que par exemple les systèmes transactionnels, les logiciels de commande de missile, ... etc. Pour ces systèmes, il est plus significatif de représenter l'évolution de la fiabilité en temps discret, c'est-à-dire en fonction du nombre d'exécutions effectuées.

De plus, un logiciel est souvent destiné à être utilisé dans des environnements qui diffèrent par la nature des services sollicités par les utilisateurs et par les caractéristiques matérielles des systèmes sur lesquels le logiciel est mis en œuvre. Or, il est inconcevable de valider un logiciel par rapport à tous les environnements dans lequel il est utilisé. Le problème qui se

Ce mémoire consiste dans une première phase à faire un état de l'art décrivant les fondements théoriques de l'évaluation de la fiabilité du logiciel. Puis pour une deuxième phase, une étude de la problématique recensent les données d'étude comme les métriques logicielles, les tests logicielles et expliquant leurs modes d'utilisation. Et nous essayons d'introduire les Réseaux Bayésiens proposé dans notre solution qui se porte sur les probabilités et la statistique appliquées au calcul de la fiabilité d'un logiciel. Plus précisément, il s'agit d'une part de construire des modèles Bayésiens pour les processus de cycle de vie d'un logiciels au cours de son développement pour maitriser les défaillances et réparations de systèmes divers, et d'autre part de mettre en œuvre des méthodes statistiques (métriques logiciels et tests) pour exploiter les données relevées par les praticiens dans le but de calculer la fiabilité des logiciels. Ensuite, une dernière phase consacrée à la présentation détaillée de la méthode proposée et la conception de l'outil à implémenter. Cet outil et ces méthodes présentées ici ont été implémentés dans un environnement JAVA Eclipse, pour but d'être opérationnel. Et enfin pour conclure, en termine par une conclusion générale et une perspective.

# Chapitre 1 :

# FIABILITÉ DES LOGICIELS

# 1. Fiabilité des logiciels

## 1.1. Introduction

L'informatisation de la société est actuellement indéniable, quelque soit le domaine socio-économique. Dans ce cadre, la fiabilité des logiciels devient un facteur critique : toute panne, tout bug, tout dysfonctionnement, perturbe la bonne marche de nos activités avec parfois des conséquences humaines, financières ou morales importantes.

Contrairement à d'autres domaines industriels plus anciens, la démarche qualité en informatique n'en est encore qu'à ses débuts. Il est couramment convenu comme normal qu'un ordinateur perde un fichier, que l'installation d'un nouveau logiciel en rende un autre inutilisable ou que certaines options d'impression refusent de fonctionner. Si cela peut s'expliquer par la multitude d'équipements existants ainsi que leur évolution très rapide, nous ne tolérerions pas autant de dysfonctionnements sur d'autres appareils.

Cependant, dans des domaines précis (banque, nucléaire, systèmes embarqués, médecine, etc.), dits critiques, on ne peut se permettre d'avoir un système défaillant : une phase importante du développement logiciel est celle de la validation durant laquelle le système est analysé afin de garantir qu'il respecte certaines exigences de fiabilité. Cette phase de validation fait appel à deux approches complémentaires, le test et la vérification :

- Le **test** consiste à faire passer au système informatique un ensemble d'épreuves afin de voir comment il se comporte. Cette phase de validation, antérieurement effectuée à la main selon l'expérience des testeurs, a fait l'objet de très nombreux travaux récents afin d'en améliorer l'efficacité, la qualité et l'automatisation.
- La phase de **vérification** consiste à prouver qu'un système vérifie bien certaines spécifications. Cette phase fournit une preuve mathématique du bon fonctionnement d'un modèle du système. Cependant, de nombreux problèmes de vérification sont indécidables : la vérification se fait donc soit avec l'aide d'un expert (on parle en général de preuve de programme), ou de façon automatique sur des modèles abstraits (on parle alors de model-checking). La preuve est très coûteuse en temps et en ressources humaines et le model-checking est limité dans ses applications.

Dans ce chapitre on va voir qu'il nécessaire de définir un ensemble de notions et terminologies concernant la fiabilité des systèmes informatiques et les logiciels afin de faire comprendre le cadre de notre projet et de cerner la problématique.

## 1.2. Le risque logiciel

Les défaillances des logiciels sont causées par des fautes dans les programmes. Or, d'une part, une étude effectuée aux USA en 1999 [1] a montré qu'un programmeur professionnel fait en moyenne 6 fautes pour 1000 lignes de code (LOC) écrites, et d'autre part, la taille et la complexité des logiciels ne cesse d'augmenter. Par exemple :

• la navette spatiale américaine a besoin pour voler de 500 000 LOC logiciel embarqué et 3 millions et demi de LOC au sol.

• les Réseaux téléphoniques utilisent des millions de LOC pour fonctionner.

• le nombre de LOC est passé de moins de 5 millions pour Windows 95 à plus de 50 millions pour Windows Vista.

• plus généralement, un logiciel commercial standard fait en moyenne 350 000 LOC.

Par conséquent, cela fait environ 2000 fautes potentielles pour un logiciel standard, 24 000 pour la navette spatiale et 300 000 pour Vista !

Evidemment, tout est fait pour éliminer ces fautes, essentiellement par le test du logiciel. Mais il est extrêmement difficile et coûteux de détecter et corriger des fautes dans un logiciel. En effet, une étude de Microsoft [1] a établi qu'il fallait en moyenne 12 heures de travail pour détecter et corriger une faute. Si un logiciel contient 2000 fautes, il faut donc passer 24 000 heures pour le déboguer, soit presque 3 ans de travail cumulé.

C'est pourquoi Microsoft emploie autant de personnel pour tester, vérifier et valider les programmes que pour les créer. Et malgré cela, chacun a pu expérimenter qu'il subsiste des erreurs dans les logiciels de Microsoft. Une étude plus récente [2] évalue à 60% du budget total d'un projet informatique le coût de la détection et correction des erreurs logicielles (ou maintenance du logiciel).

Malgré tous ces efforts, la complexité de la tâche fait qu'il reste toujours des fautes dans un logiciel. Comme partout, et peut-être même moins que partout, le zéro défaut est impossible. Quand ces fautes résiduelles se manifestent, leurs conséquences peuvent aller du minime au franchement catastrophique.

Il est donc impératif de tout faire pour éviter que des pannes informatiques majeures se produisent.

## 1.3. Terminologie spécifique aux logiciels

En première approche, la fiabilité d'un logiciel est la probabilité qu'il fonctionne sans défaillances pendant une durée donnée et dans un environnement spécifié. C'est donc une notion temporelle. Le temps considéré peut être le temps d'exécution CPU (temps effectivement passé par la machine pour exécuter le programme), le temps calendaire, voir également un nombre d'opérations ou de transactions. A terme, seul le temps calendaire est important. Notons que pour certains systèmes (les systèmes réactifs), le temps n'est pas l'élément primordial : ce qui compte, c'est qu'une exécution se déroule correctement. Alors, la fiabilité est définie comme la probabilité qu'une exécution soit correcte. Dans la suite, nous ne nous intéresserons pas à ce type de système et nous conserverons donc la définition temporelle de la fiabilité. Une défaillance se produit quand le résultat fourni par le logiciel n'est pas conforme au résultat prévu par les spécifications. Pour éclaircir cette notion, on peut considérer qu'un logiciel est un système qui, par l'intermédiaire d'un programme, transforme des données d'entrée en résultats ou données de sortie. Un programme est une suite finie d'instructions codées qui exécute une ou plusieurs tâches spécifiées. L'exécution d'un programme peut donc être vue (voir figure 1) comme une application de l'ensemble des données d'entrée dans l'ensemble des données de sortie (appelés espace des entrées et espace des sorties) [3].

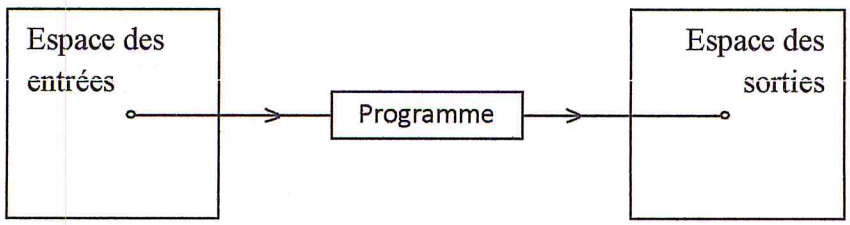



***Figure 1:*** *L'exécution d'un programme [3].*

'Le **logiciel** est l'ensemble des programmes, procédés et règles, et éventuellement de la documentation, relatifs au fonctionnement d'un ensemble de traitement de l'information'. (arrêté du 22 déc 1981). Autrement dit, un **logiciel** est un ensemble de programmes informatiques (du code) mais également un certain nombre de documents se rapportant à ces programmes et nécessaires à leur installation, utilisation, développement et maintenance : spécification, schémas conceptuels, jeux de tests, mode d'emploi …… etc. Notons que l'objectif de la documentation est de permettre la transmission d'information, rendre visible le produit logiciel tout au long du cycle de vie. La documentation est produite au fur et à mesure du développement du projet.

Les **spécifications** définissent quelle doit être la donnée de sortie pour chaque donnée d'entrée possible. Si, pour une donnée d'entrée particulière, la sortie fournie par le programme n'est pas celle prévue par les spécifications, il y a défaillance. On voit ainsi apparaître une relation forte entre donnée d'entrée et défaillance, sur laquelle nous reviendrons.

Une **faute logicielle** ou **bug** est un défaut du programme qui, exécuté dans certaines conditions, entraînera une défaillance. Une faute est un phénomène intrinsèque au programme, elle existe même quand le logiciel n'est pas utilisé. A l'inverse, une défaillance est un phénomène dynamique : le programme doit être exécuté pour qu'elle se manifeste. Une faute est créée suite à une **erreur** humaine de l'analyste, du concepteur ou du programmeur. Aussi, on emploie le terme de **faute de conception**. Les erreurs peuvent être de spécification, de conception ou de codage. Il est également possible qu'une défaillance d'un système informatique soit due à un problème matériel. Ce type de défaillance est en général facilement identifiable. [3]

Si on pouvait tester toutes les entrées possibles d'un logiciel, on détecterait fatalement toutes les fautes. Mais le nombre d'entrées possibles est beaucoup trop grand pour cela. Il faut donc déterminer dans l'espace des entrées un sous-ensemble d'entrées à tester.

Le profil opérationnel définit le choix des entrées et la fréquence de sollicitation du logiciel, en associant à chaque entrée ou groupe d'entrées sa probabilité d'être fournie au programme à un instant donné. Le **profil opérationnel** est en général très différent en phase de test et en vie opérationnelle.

Quand une défaillance survient, on cherche à détecter la faute qui a provoqué cette défaillance et à l'éliminer. On effectue alors une **correction** ou **débogage**. Parfois, un logiciel est amené à changer radicalement certaines de ses fonctionnalités [3]. On procède alors à un changement de spécifications, qui va aboutir à une nouvelle version du logiciel. Les corrections et les changements de spécifications peuvent être interprétés de la même manière comme des modifications du programme. Une modification d'un logiciel est parfois appelée une **maintenance logicielle**. La correction a pour but de réduire l'occurrence d'apparition des défaillances, donc elle devrait augmenter la fiabilité du logiciel.

*Selon l'IEEE* ***La qualité logicielle*** est: Le degré avec lequel un système, un composant ou un processus satisfait à ses exigences spécifiées, d'une part. ET Le degré avec lequel un système, un composant ou un processus satisfait aux besoins ou attentes de ses clients/usagers, d'autre part. [4]

La notion de qualité recouvre deux aspects:

- **conformité avec la définition**, cette notion est contrôlable **en cours de fabrication,**
- **réponse à l'attente de l'utilisateur,** cette notion est contrôlable **à la livraison du produit.**

La qualité est définie de manière générale par « l'aptitude d'un produit ou d'un service à satisfaire les besoins des utilisateurs en termes de fonctionnalités, délais, coûts. ». D'une façon plus générale l'évaluation de la qualité nécessite en plus l'évaluation de la fiabilité d'autres critères afin de se rassurer sur la qualité du logiciel. Dans notre projet la qualité logiciel ne figure pas comme un objectif mais elle est prise en considération lors de l'évaluation de la fiabilité. Pour plus d'information sur la qualité du logiciel réf [4].

## 1.4. Terminologie générale de la sûreté de fonctionnement

Un système est un ensemble de composants en interaction destiné à accomplir une tâche donnée. C'est le cas par exemple des systèmes de production, systèmes de transport, systèmes informatiques, …etc.

La SdF d'un système est la propriété qui permet à ses utilisateurs de placer une confiance justifiée dans le service qu'il leur délivre. On dit aussi que la SdF est la science des défaillances.

Un système subit une défaillance quand il ne peut plus délivrer le service attendu. La panne est l'état du système résultant d'une défaillance.

La sûreté de fonctionnement comprend 5 composantes : la fiabilité, la disponibilité, la maintenabilite, la sécurité-innocuité et la sécurité-confidentialité : [3]

• La fiabilité (reliability) est la caractéristique du système exprimée par la probabilité qu'il délivre le service attendu dans des conditions données et pendant une durée déterminée. La fiabilité exprime l'aptitude à la continuité du service.

• La disponibilité (availability) est exprimée par la probabilité que le système délivre le service attendu dans des conditions données et à un instant donné. La disponibilité caractérise donc l'aptitude du système à fonctionner quand on a besoin de lui.

• La maintenabilité (maintainability) caractérise l'aptitude du système à être réparé quand il est défaillant, ou à évoluer.

• La sécurité-innocuité (safety) caractérise l'aptitude du système à ne pas encourir de défaillances catastrophiques.

• La sécurité-confidentialité (security) caractérise l'aptitude du système à se prémunir contre les accès ou manipulations non autorisées (virus, attaques,...).

Dans les études de SdF, on distingue les approches boîte noire et boîte blanche [3] :

• Approche boîte blanche ou structurelle : on considère qu'un système complexe est constitué de composants et que sa fiabilité dépend à la fois de la fiabilité de ses composants et de la façon dont le bon fonctionnement ou la panne de chaque composant influe sur le bon fonctionnement ou la panne du système tout entier. En particulier, on considère souvent qu'un système réparable est constitué de composants non réparables. Quand un composant tombe en panne, on le remplace par un neuf, mais le système complet, lui, n'est pas remis à neuf.

• Approche boîte noire ou globale : on considère le système comme un tout, qu'on ne cherche pas à décomposer en composants. On s'intéresse alors à la suite des défaillances et réparations successives du système.

## 1.5. Méthodes d'évaluation de la fiabilité des logiciels selon les étapes du cycle de vie

Les méthodes d'évaluation de la fiabilité des logiciels varient suivant la nature des informations disponibles. Celles-ci sont étroitement liées au cycle de vie du logiciel, comme on le voit dans le tableau 1 [1].

***Tableau. 1 :*** *Pourcentages d'erreurs introduites et détectées selon les phases du cycle de vie du logiciel.[1].*

| Phase du cycle de vie | Pourcentage d'erreurs introduites | Pourcentage d'erreurs détectées |
|---|---|---|
| Analyse | 55% | 18% |
| Conception | 30% | 10% |
| Codage et test | 10% | 50% |
| Vie opérationnelle | 5% | 22% |

Les types d'erreurs dans les différentes phases sont les suivantes :

• Analyse : le logiciel ne répond pas à l'attente des utilisateurs.

• Conception : mauvaise traduction des spécifications.

• Codage et test : erreurs de programmation ou de correction.

• Vie opérationnelle : erreur dans les mises à jour du système.

On constate que la majeure partie des erreurs sont introduites dans les premières phases du cycle de vie (85% en analyse et conception) et détectées dans les dernières phases (72% en codage, test et vie opérationnelle).

## 1.6. Utilisation des évaluations de fiabilité des logiciels

Dans un premier temps, les évaluations de fiabilité permettent de quantifier la confiance d'un utilisateur envers un système informatique, c'est-à-dire d'évaluer quantitativement le risque que l'on prend en le faisant fonctionner. Puis elles permettent de s'assurer que le logiciel a atteint un niveau de fiabilité conforme aux objectifs exprimés dans les spécifications. Un objectif de fiabilité est usuellement exprimé en termes de taux de panne ou taux de défaillance. Pour les systèmes faisant l'objet d'une garantie, les évaluations de fiabilité permettent de déterminer la durée et le coût de la garantie. Par exemple, pour le récent métro parisien sans conducteur Meteor, les objectifs annoncés étaient un taux de panne par rame et par heure inférieur à $10^{-9}$ pour le matériel et inférieur à $10^{-11}$ pour le logiciel. Si

les mesures de fiabilité montrent que l'objectif n'est pas atteint, elles peuvent permettre d'évaluer l'effort de test à fournir pour atteindre l'objectif, et en particulier d'estimer le temps nécessaire pour y parvenir.

Par conséquent, les mesures de fiabilité fournissent un critère d'arrêt des tests : on arrête les tests dès qu'on peut prouver, avec un niveau de confiance raisonnable, qu'un objectif donné de fiabilité est atteint. Une expérience menée à AT&T a montré que la mise en place des mesures de fiabilité a permis une réduction de 15% de la durée de la période de tests, ce qui a entrainé un gain de 4% sur le coût total du projet, alors que le surcoût du aux mesures n'a représenté que 0.2% de ce coût total [1]. D'autres exemples sont mentionnés dans [5]. Par ailleurs, une mesure de fiabilité est un moyen d'évaluer quantitativement la qualité d'une méthode de génie logiciel donnée. Elle peut aussi fournir un indicateur de performance d'un programmeur ou d'un testeur. Cette dimension humaine délicate est parfois un frein à l'utilisation effective des évaluations de fiabilité.

## 1.7. Représentation de la logique d'un système

Le premier problème rencontré par le technicien fiabiliste dans l'étude de la fiabilité d'un système est la description de ce système.

### 1.7.1. Digramme de fiabilité

C'est la représentation la plus naturelle de la logique de fonctionnement d'un système car elle est souvent proche du schéma fonctionnel du système. Le système fonctionne s'il existe un **chemin de succès**. La liste des chemins de succès permet donc de représenter l'ensemble des états de marche du système. Le diagramme logique de fiabilité peut représenter des cas plus complexes : Les blocs du diagramme de fiabilité représentent **éléments**. Il est souvent intéressant de regrouper un certain nombre d'éléments pour constituer un seul bloc avec eux ; un tel bloc sera souvent appelé un **macro- élément**. [6]

Soit par exemple, des chemins de succès : $E_1E_3$, $E_1E_4$, $E_2E_4$, $E_2E_5$.

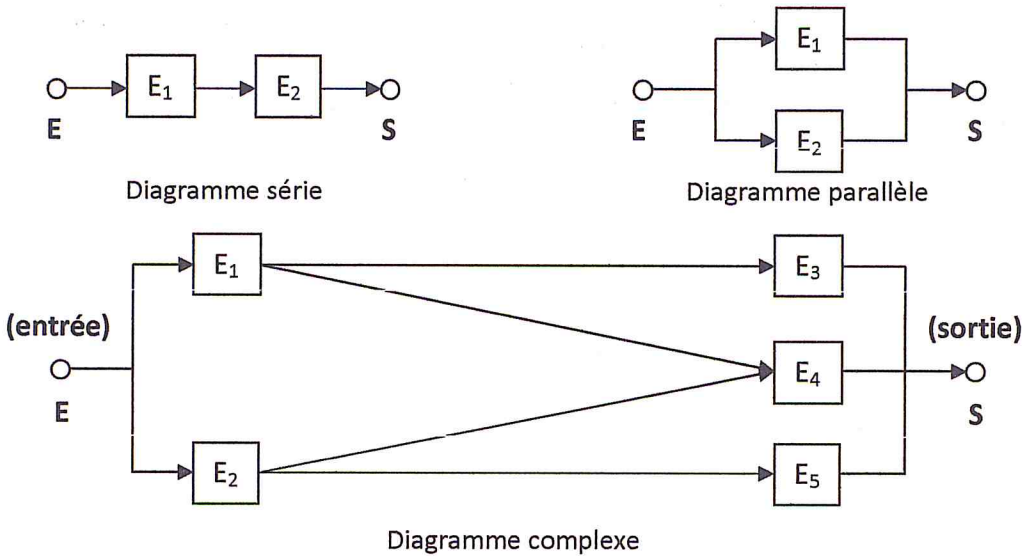
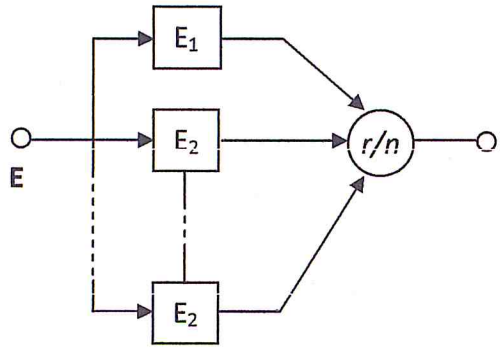
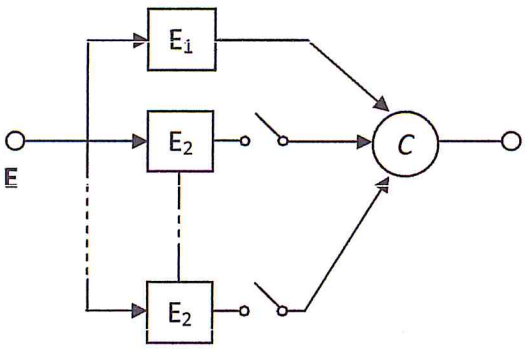



***Figure 2:*** *Symboles de redondance [6].*




redondance r/n

Représente le cas où on exige qu'au moins r éléments parmi les n placés en parallèles soient non défaillants pour que le système fonctionne.




redondance passive (« stand-by ») avec orage de commutation commun.

Représente le ca où il existe des éléments en réserve qui ne démarrent qu'en secours de l'élément principal.

***Figure 3 :*** *Symboles de redondance complexes [6].*

### 1.7.2. L'approche orientée objet

L'approche orientée objet considère un système (logiciel) comme un ensemble d'objets dissociés définis par des propriétés. Une propriété est soit un attribut ou une opération. Un attribut est une entité élémentaire (donnée) de la description de l'état de l'objet. Une opération est une entité élémentaire de la description du comportement de l'objet. Un objet comprend donc à la fois une structure de données (sous forme d'un ensemble d'attributs) et un ensemble d'opérations qui expriment son comportement.

Un objet est caractérisé par

- un identificateur (nom de l'objet) ;
- un état (sous forme d'un ensemble d'attributs) ;
- un comportement (un ensemble d'opérations).

Les objets doivent dans la mesure du possible reposer sur les entités du monde réel et des concepts de l'application ou du domaine concernés

## 1.8. Etude Problématique et Bibliographique:

### 1.8.1. Position du problème

A priori, les défaillances des systèmes informatiques peuvent être soit d'origine matérielle, soit d'origine logicielle. En pratique, plus de 80 % sont d'origine logicielle.
Nous dans notre projet on s'intéressera aux problèmes logiciels. Il y a eu des tentatives de résoudre la fiabilité du logiciel de la même façon que celle des matériels déjà bien métriser. Mais on a constaté qu'il existe des différences fondamentales entre la fiabilité des matériels et celle des logiciels.

- Les défaillances des matériels sont essentiellement dues à l'usure (ou vieillissement) et aux facteurs environnementaux, tandis que celles des logiciels sont dues à des fautes de conception (ou bugs), c'est-à-dire à des erreurs humaines.
- La maintenance des matériels ralentit le vieillissement des systèmes mais ne l'empêche pas, tandis que la correction des logiciels augmente leur fiabilité.
- Un logiciel non utilisé ne tombe pas en panne (le terme de panne est d'ailleurs peu utilisé pour le logiciel). Un matériel non utilisé peut tomber en panne du fait de l'usure ou des facteurs environnementaux.

- En logiciel, une faute bien repérée et corrigée est éliminée définitivement et ne peut plus se manifester. En matériel, on peut observer des défaillances répétables ou chroniques.
- La sensibilité d'un matériel à son environnement est assez forte, mais on peut néanmoins considérer qu'un matériel a une fiabilité en soi : les constructeurs quantifient la fiabilité d'une ampoule électrique quel que soit son environnement. En revanche, la sensibilité d'un logiciel a son environnement, à travers son profil opérationnel, est extrêmement forte. Un logiciel truffé de fautes peut très bien fonctionner sans défaillance si le profil opérationnel n'active pas ces fautes. Un matériel ayant beaucoup de défauts ou très usé tombera fatalement en panne, quelle que soit la manière dont on l'utilise.
- Quand un matériel est en panne, il ne peut pas fonctionner tant qu'on ne l'a pas réparé. Au contraire, un logiciel peut être relancé immédiatement après une défaillance.

On voit que les différences sont nombreuses entre fiabilité des matériels et fiabilité des logiciels. Donc on ne pourra pas traiter les deux aspects de la même manière.

Il y a de nombreuses méthodes dont le but est de produire des logiciels de fonctionnement sûr. On peut classer ces méthodes en 4 catégories:

- La prévention des fautes : ces méthodes ont pour but d'empêcher l'occurrence et l'introduction de fautes dès la conception du logiciel. Par exemple, on a de plus en plus souvent recours à des méthodes formelles pour développer les spécifications.
- L'élimination des fautes : ces méthodes ont pour but de détecter des fautes dans un programme déjà écrit. Elles comprennent les preuves de programmes, les inspections formelles, la vérification et surtout le test du logiciel.
- La tolérance aux fautes : ces méthodes ont pour but de permettre au système de fonctionner correctement même en présence de fautes. On peut par exemple utiliser de la redondance.
- La prévision des fautes : ces méthodes ont pour but d'estimer la présence des fautes et de prévoir les défaillances futures du système

Un bon choix de méthode, c'est la méthode de prévision des fautes, car on pourra remédier fautes le plus tôt possible avant que l'effet des erreurs s'aggrave. Car on peut avec cette méthode avoir une maitrise de la situation.

Malgré tous les efforts de métriser ce domaine, la complexité de la tâche fait qu'il reste toujours des fautes dans un logiciel. Il a été donc impératif de disposé d'un outil pour calculer et se rassurer de la fiabilité d'un logiciel au cours de son développement -cycle de vie- et faire éviter que des pannes informatiques majeures se produisent. Cela nous a motivé à rechercher d'autres méthodes pour résoudre la problématique avec de nouvelles techniques de résolution.

En partant de tous les données cité ci-dessus, nous avons proposé notre sujet de recherche en collaboration avec notre promotrice baptisé la « Construction d'un outil de calcul de la fiabilité d'un logiciel ».

Nous avons pu cerner le cadre de notre recherche sur le calcule de la fiabilité d'un logiciel. En retenant les points suivants comme avant projet:

- ***Le risque d'erreur*** :
  L'humain provoque des erreurs, c'est les chiffres qui dit qu'au 4 Juin 1996 : la fusée Ariane 5 a explosé à cause d'une faute de conception logicielle, et plus précisément un problème de réutilisation. Un programme de contrôle d'un gyroscope de la fusée (en ADA) avait été transféré sans modification d'Ariane 4 à Ariane 5. Or les trajectoires de vol d'Ariane 5 sont sensiblement différentes de celles d'Ariane 4. Cela a provoqué un dépassement de mantisse qui a induit en erreur le calculateur de pilotage et fini par faire exploser le lanceur. Cette erreur a coûté 500 millions de dollars uniquement en matériel, sans parler des retards engendrés et des conséquences sur l'image de marque d'Arianespace. C'est dans ce sens qu'on a considérer l'homme comme 1$^{ére}$ source de défaillance et quand doit l'aider à baisser la probabilité d'avoir une erreur dans la programmation et c'est la première hypothèse

- ***L'effet d'erreur :***
  Dans la fin de test d'un logiciel, s'il y a une erreur nous ne somme pas sure de connaitre où est exactement l'erreur mais on peut voir son effet. Et de mesurer son degré d'affecter sur les composants voisins si on est dans une conception modulaire ceci peut aller de moins grave au catastrophique, donc on considère l'effet d'erreur comme la deuxième hypothèse.

- ***Le Test :***
  Les teste sur le logiciel nous donnent une image sur la continuité du logiciel et la durée de fonctionnement d'un logiciel sans défaillance en plus on a déjà l'information de la durée de vie d'utilisation de logiciel envisagé, donc il nous facilite la tâche d'analyse du fonctionnement du logiciel ainsi nous aide à estimer la durée de vie

## 1.9. Conclusion

L'objectif de notre recherche est la résolution du problème de calcul de la fiabilité d'un logiciel présentant des fautes durant son cycle de vie, au vu de ses défaillances successives et des corrections qui lui sont apportées. Dans ce but, nous proposons une modélisation du comportement du logiciel au cours du temps, où l'on montre que l'évolution de ce système résulte de l'interaction complexe de ces composants, environnement et les mesures de logiciels, cette interaction étant entièrement résumée par la donnée de la fiabilité et/ou la défaillance. Ce résultat permet de construire un outil exploitable. L'approche choisie pour la résolution est les réseaux Bayésiens. Les Réseaux Bayésiens sont actuellement une des techniques les plus intéressantes de l'aide à la décision car ils permettent la représentation de la connaissance par un graphe causal intuitif et compréhensible. De plus, comme ils sont basés sur des probabilités, ils intègrent l'incertitude dans le raisonnement. Qui se rapporte bien avec notre problématiques et le risques de défaillances –l'incertitude- et. Dans le chapitre qui suit, en va introduire des notions sur les différentes données d'étude et ainsi une introduction sur les Réseaux Bayésiens et leurs utilisations dans notre cas d'étude.

# Chapitre 2 :

# ÉTUDE PROBLÉMATIQUE

De ces points la on peut conclure que l'importance d'un logiciel fiable dépend de l'importance de logiciel, on n'applique pas les tests de fiabilité sur un logiciel qui ne sert a rien, et que on a toujours un humain sur la ligne alors la possibilité de commettre une erreur malgré que le développeur utilise des méthodes de génie logiciel, cependant la génie logiciel permet de structurer le travail, comme elle permet de structure les erreurs commises dans chaque étapes de modélisation alors on peut introduire la fiabilité dans chaque étape de modélisation et assurer dans le futur avec un pourcentage la fiabilité du produit final et donc prédire la fiabilité du logiciel. La négociation du développeur avec un client sur un produit déjà existant diffère de celle sur un produit non déjà réaliser, si le logiciel est déjà existant on parle de la capacité de couverture des besoin du client, ceci nous mène a modifier une partie de code mais pas sa totalité pour ajouter certaines nouvelles fonctionnalités, ce qui changera probablement la fiabilité du logiciel, mais, si le logiciel n'est pas encore réalisé la complexité du logiciel augmente et donc on trouve une difficulté dans la réalisation pour atteindre la satisfaction du client et ce qui affect énormément la garante sur la fiabilité vis-à-vis le client.

Les métriques acquis dans la modélisation du logiciel dépendent de tous les points passés, ils représentent un début pour le teste de la fiabilité du logiciel. Le moindre changement du cahier des charges implique un changement total sur les valeurs des métriques fixes. Ensuite, il reste de connaître les défaillances et leurs occurrences par des observations du développeur et des testes.

Mais ces critères et ces métriques ne sont pas établie une seul fois, ils sont collecter et recalculer après chaque itération des tests et correction des fautes qu'ils apparaitront durant les étapes du cycle de vie.

Dans les phases d'analyse, conception et codage, le système n'est pas encore construit, donc il ne peut pas être utilisé et aucune défaillance n'est observée. Les éléments pouvant être utilisés pour des prévisions de fiabilité sont la structure du système et les métriques logicielles (nombre de lignes de code, nombre cyclomatique du graphe de contrôle, mesures d'architecture et de spécifications, etc...). A ce niveau, on peut évaluer la qualité du logiciel, mais pas sa fiabilité. Or on ne sait pas mesurer la corrélation entre qualité et fiabilité d'un logiciel.

En phase de test et en vie opérationnelle, le système fonctionne, des défaillances sont observées et des corrections sont apportées au logiciel pour remédier aux fautes apparues. L'essentiel des méthodes d'évaluation de la fiabilité repose sur l'observation et l'analyse statistique de cette suite de défaillances et corrections successives.

Tout comme les matériels, les logiciels complexes sont constitués de modules unitaires que l'on assemble. Si on est capable d'évaluer la fiabilité de chaque module et d'analyser les liens entre les différents modules, on peut appliquer les méthodes structurelles (boîte blanche) d'évaluation de fiabilité. Ce n'est pas du tout facile en pratique. Aussi considère-t-on en général un logiciel comme un tout et on évalue sa fiabilité par une approche boîte noire. C'est ce qui est fait en générale.

Dans notre travail de recherche, on a considéré le logiciel dans une vision global comme une boîte, avec des entrées et des sorties global, on peut ensuite voir la boîte comme une ensemble de blocs structure avec des entrées et sorties spécifique ; mais lors la fiabilité, l'application des méthodes de fiabilité a besoin des entrées ou des informations sur le logiciel testé, les métriques et bien sur chaque méthode a ces besoin de spécifier le logiciel (la méthode que nous proposons dans notre projet) et aussi comme on le constate le logiciel a besoin de construire un modèle de logiciel contenant tous les informations comme la structure et ces différentes fonctionnalités.

Pou résumer, lorsqu'on termine la construction du modèle logiciel - dans notre cas le modèle logiciel sera basé sur les Réseaux Bayésien -, en commence la méthode de calcule de la fiabilité et ses dérivées.

## 2.2. Les Métriques logicielles

La mesure en génie logiciel consiste à faire correspondre des symboles à des objets, afin de pouvoir quantifier certains attributs de ces objets. Il peut par exemple s'agir de mesurer la taille d'un logiciel, ou d'évaluer des attributs qui ne sont pas directement mesurables, comme les ressources nécessaires au développement d'un projet [7].

Voici les critères qu'une métrique doit respecter pour être valide :

1. Une métrique doit permettre de distinguer différentes entités.
2. Une métrique doit obéir à une condition de représentation.
3. Chaque unité de l'attribut doit contribuer dans la même mesure à la métrique
4. Des entités différentes peuvent avoir des attributs de même valeur.

Bien souvent, les attributs intéressants ne sont pas directement mesurables et il faut employer une mesure indirecte. Une mesure indirecte nécessite l'emploi d'une mesure directe et d'une formule de prédiction. Par exemple, la densité n'est pas directement mesurable : elle est mesurée à partir de la masse et du volume, qui, eux, sont directement mesurables. En informatique, de nombreuses caractéristiques (généralement se terminant en «-ité», comme la

lisibilité, la qualité, la complexité, etc.) ne peuvent pas être mesurées directement et c'est le but de nombreux programmes de métriques de les mesurer indirectement.

Voici les différents critères que doit respecter une métrique indirecte pour être valide:

1. Le modèle doit être défini explicitement.
2. Le modèle doit être homogène au niveau de ses dimensions.
3. II ne doit pas y avoir de discontinuité inattendue.
4. Les unités et les échelles doivent être correctes.

On va lister ci-dessous quelques métriques utilisées dans notre mémoire :

### 2.2.1. Métriques de code

#### 2.2.1.1. Le nombre cyclomatique de McCabe

Le *nombre cyclomatique de McCabe*, introduit en 1976, constitue, après les lignes de code, la métrique la plus employée dans le cadre du développement logiciel. Cette métrique utilise le nombre cyclomatique de la théorie des graphes. L'objectif de McCabe était de mesurer la complexité d'un programme et il est parti de la supposition selon laquelle la complexité est liée au flux de contrôle du programme. La théorie des graphes utilise pour calculer le nombre cyclomatique la formule C=a - n+ 1. McCabe là légèrement modifiée pour obtenir :

La complexité cyclomatique d'un programme structuré est définie par (voir Figure 6):

**C = a - n + 2p** où a = nombre d'arcs

n = nombre de nœuds

p = nombre de composants connexes (normalement égal à 1)

*Pour designer le graphe on utilise*

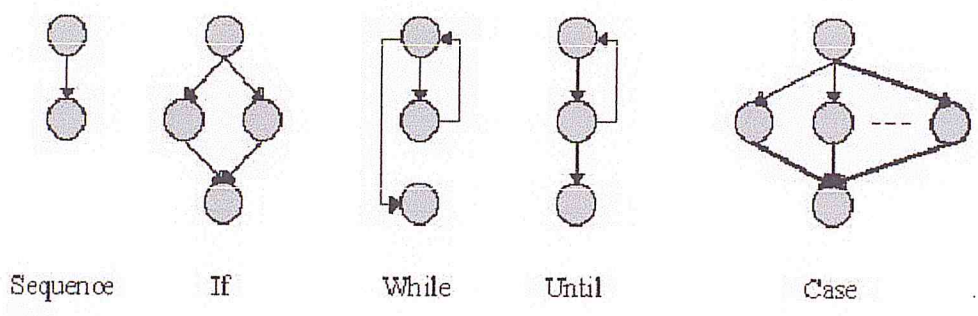



***Figure 4:*** *Composants constituants les graphes de contrôle [7].*

On a besoin de code source alors on spécifier exactement comment le code source est écrit

Example

Begin

Operation1

Operation2

.

.

Operation n

If(operation i) operation j

Else operation k

Operation m

End.

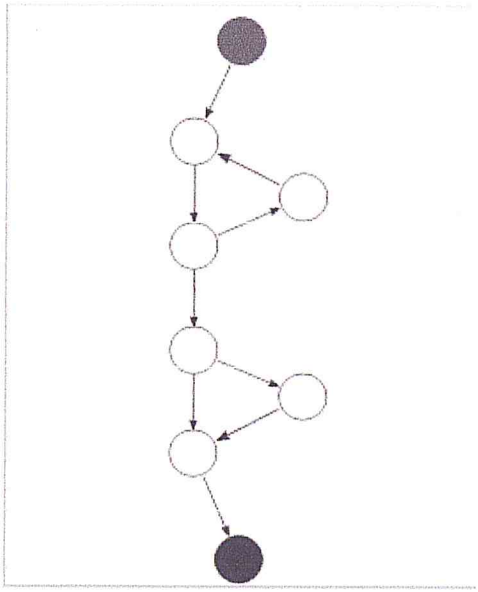


***Figure 5:*** *Graphe de contrôle [7].*

***Calcul direct du nombre de McCabe***

- Produire un graphe de contrôle et l'analyser peut s'avérer long dans le cas de programmes complexes

McCabe a introduit une nouvelle manière de calculer la complexité structurelle

$$C = \pi + 1$$

Avec $\pi$ le nombre de décisions du code

- Une instruction IF compte pour 1 décision
- Une boucle FOR ou WHILE compte pour 1 décision
- Une instruction CASE ou tout autre embranchement multiple compte pour une décision de moins que le nombre d'alternatives

**Valeur de seuil**

Pour chaque métrique, il est bon d'avoir une idée des valeurs qui sont raisonnables et de celles qui ne le sont pas. Après avoir analysé un projet de grande envergure, McCabe a constaté que les modules dont le nombre cyclomatique dépassait 10 avaient connu bien plus de problèmes et posaient bien plus de difficultés de maintenance que ceux dont le nombre cyclomatique était inférieur à cette valeur. Dix est donc considéré comme la valeur seuil pour le nombre cyclomatique d'un module. Si un module présente une valeur supérieure, il est donc souhaitable d'essayer de la réduire ou de décomposer le module [8].

### 2.2.1.2. Métriques de Halstead

***Science informatique de Halstead : [7]***

- Métriques pour la complexité d'un programme
- Fondées empiriquement
- Toujours considérées comme valides, contrairement à ses formules complexes de prédiction
- Entités de base
- Opérandes : jetons qui contiennent une valeur
- Operateurs : tout le reste (virgules, parenthèses, operateurs arithmétiques...)

***Mesures de base $\eta 1$ et $\eta 2$ (êta)***

- $\eta 1$ : nombre d'operateurs distincts
- $\eta 2$ : nombre d'opérandes distincts
- $\eta = \eta 1 + \eta 2$ : nombre total de jetons distincts
- Opérandes potentiels $\eta_2^*$
  - Ensemble de valeurs minimal pour n'importe quelle implémentation
  - Pour comparer différentes implémentations du même algorithme
  - S'obtient généralement en comptant les valeurs qui ne sont pas initialisées à l'intérieur de l'algorithme
    - Valeurs lues par le programme
    - Valeurs passées en paramètres
    - Valeurs globales appelées depuis l'algorithme

Exemple :

```
z = 0;
while x > 0
z = z + y;
x = x − 1;
end-while
print (z);
```

- Opérateurs : – ; while/end-while > + - print () $\eta 1 = 8$
- Opérandes : = z 0 x y 1 $\eta 2 = 5$

### 2.2.2. Métriques de spécification

#### 2.2.2.1. Métriques de Henry-Kafura

***Flux d'informations d'Henry-Kafura [7]***

- Mesurer la complexité des modules d'un programme en fonction des liens qu'ils entretiennent
- On utilise pour chaque module i :
  - Le nombre de flux d'information entrant noté $in_i$
  - Le nombre de flux d'information sortant noté $out_i$
  - Le poids du module noté $poids_i$ calculé en fonction de son nombre de LOC et de sa complexité

- La mesure totale HK correspond à la somme des $HK_i$

Exemple : ***[7]***

A partir des $in_i$ et $out_i$ ci-dessous, calcul des métriques HK en supposant que le poids de chaque module vaut 1.

| Module | a | b | c | d | e | f | g | h |
|---|---|---|---|---|---|---|---|---|
| $in_i$ | 4 | 3 | 1 | 5 | 2 | 5 | 6 | 1 |
| $out_i$ | 3 | 3 | 4 | 3 | 4 | 4 | 2 | 6 |
| $HK_i$ | 144 | 81 | 16 | 225 | 64 | 400 | 144 | 36 |

HK = 1110

### 2.2.3. Métriques d'orienté objets

#### 2.2.3.1. Métriques Objet de Chidamber et Kemerer

- Ensemble de métriques (Metric Suite for Object Oriented Design) [7]
  - Evaluation des classes d'un système
  - La plupart des métriques sont calculées classe par classe
  - Le passage au global n'est pas clair, une moyenne n'étant pas très satisfaisante

***M1** : Méthodes pondérées par classe :*

- WMC : Weighted Methods per Class

$$WMC = \frac{1}{n} * \sum_{i=0}^{n} c_i * M_i$$

Avec C un ensemble de n classes comportant chacune $M_i$ méthodes dont la complexité (Le poids) est noté $c_i$.

*M2 : Profondeur de l'arbre d'héritage*

- DIC : Depth of Inheritance Tree
  - Distance maximale entre un nœud et la racine de l'arbre d'héritage de la classe concerné
  - Calculée pour chaque classe

*M3 : Nombre d'enfants*

- NOC : Number Of Children
  - Nombre de sous-classes dépendant immédiatement d'une classe donnée, par une relation d'héritage
  - Calculée pour chaque classe

*M4 : Couplage entre classes*

- Dans un contexte OO, le couplage est l'utilisation de méthodes ou d'attributs d'une autre classe.
  - Deux classes sont couplées si les méthodes déclarées dans l'une utilisent des méthodes ou instancie des variables définies dans l'autre
  - La relation est symétrique : si la classe A est couplée à B, alors B l'est à A
- CBO : Coupling Between Object classes
  - Pour chaque classe, nombre de classes couplées
  - Calculée pour chaque classe

*M5 : Réponses pour une classe (RFC)*

- {RS} : ensemble des méthodes qui peuvent être exécutées en réponse à un message reçu par un objet de la classe considérée
  - Réunion de toutes les méthodes de la classe avec toutes les méthodes appelées directement par celles-ci
  - Calculée pour chaque classe

$$RFG = |RS|$$

*M6 : Manque de cohésion des méthodes*

- Un module (ou une classe) est « cohésif » lorsque tous ses éléments sont étroitement liés
- LCOM (Lack of COhesion in Methods) tente de mesurer l'absence de ce facteur
- Posons
  - $I_i$ l'ensemble des variables d'instance utilisées par la méthode i
  - P l'ensemble les paires de ( $I_i$; $I_j$) ayant une intersection vide

- Q l'ensemble des paires de $(I_i;\ I_j)$ ayant une intersection non vide

$$LCOM = max(|P| - |Q|, 0)$$

- LCOM peut être visualisée comme un graphe bi-partie
  - Le premier ensemble de nœuds correspond aux n différents attributs et le second aux m différentes méthodes
  - Un attribut est lié à une fonction si elle y accède ou modifie sa valeur
  - L'ensemble des arcs est Q
  - Il y a $n * m$ arcs possibles, et $|P| = n * m - |Q|$

### 2.2.3.2. Métriques MOOD

- Ensemble de métriques pour mesurer les attributs des propriétés suivantes : [7]
  - Encapsulation
  - Héritage
  - Couplage
  - Polymorphisme

*M1: Encapsulation*

- MHF: Method Hiding Factor (10-30%)

$$MHF = \frac{\sum_{i=1}^{TC} M_h(C_i)}{\sum_{i=1}^{TC} M_d(C_i)}$$

Avec :

- $M_d(C_i)$ le nombre de méthodes déclarées dans une classe $C_i$ .
- $M_h(C_i)$le nombre de méthodes cachées
- TC le nombre total de classes.

- AHF : Attribute Hiding Factor (70-100%)

$$AHF = \frac{\sum_{i=1}^{TC} A_h(C_i)}{\sum_{i=1}^{TC} A_d(C_i)}$$

Avec :

- $A_d(C_i)$ le nombre d'attributs déclarés dans une classe $C_i$ .
- $A_h(C_i)$le nombre d'attributs cachés

*M2 : Facteurs d'héritage*

- MIF : Method Inheritance Factor (65-80%)

$$MIF = \frac{\sum_{i=1}^{TC} M_i(C_i)}{\sum_{i=1}^{TC} M_a(C_i)}$$

Avec :

- $M_i(C_i)$ le nombre de méthodes héritées (et non surchargées) de $C_i$
- $M_a(C_i)$ le nombre de méthodes qui peuvent être appelées depuis la classe i

- AIF : Attribute Inheritance Factor (50-60%)

$$AIF = \frac{\sum_{i=1}^{TC} A_i(C_i)}{\sum_{i=1}^{TC} A_a(C_i)}$$

Avec :

- $A_i(C_i)$ le nombre d'attributs héritées (et non surchargées) de $C_i$
- $A_a(C_i)$ le nombre d'attributs qui peuvent être appelées depuis la classe i

*M3 : Facteur de couplage*

- CF : Coupling Factor (5-30%)
- Mesure le couplage entre les classes sans prendre en compte celui dû à l'héritage

$$CF = \frac{\sum_{i=1}^{TC} \sum_{j=1}^{TC} client(C_i, C_j)}{TC^2 - TC}$$

Avec :

- $client(C_i, C_j) = 1$ si la classe i a une relation avec la classe j, et 0 sinon
- Les relations d'héritage ne sont pas prises en compte dans les relations

*M4 : Facteur de polymorphisme*

- PF : Polymorphism Factor (3-10%)
- Mesure le potentiel de polymorphisme d'un système

$$PF = \frac{\sum_{i=1}^{TC} M_0(C_i)}{\sum_{i=1}^{TC} M_n(C_i) * DC(C_i)}$$

Avec :

- $M_0(C_i)$ le nombre de méthodes surchargées dans la classe i
- $M_n(C_i)$ le nombre de nouvelles méthodes dans la classe i
- $DC(C_i)$ le nombre de descendants de la classe i

## 2.3. Les Test logiciels

### 2.3.1. Fondement du test

Le test est une recherche d'anomalie dans le comportement de logiciel. C'est une activité paradoxale : il vaut mieux que ce ne soit pas la même personne qui développe et qui teste le soft. D'où le fait qu'un bon test est celui qui met à jour une erreur (non encore rencontrée).

Remarque (difficulté) : il faut arriver à gérer une suite de test la plus complète possible à un coup minimal.

Un test ne peut pas dire « il n'y a pas d'erreur » car il teste le logiciel de façon poussive, plus que dans l'utilisation réelle.

#### 2.3.1.1. Cycle de développement de test

Lorsqu'une erreur est détectée alors que commence le débogage, la correction d'une erreur dont la différence avec résultat en du juif est de l'ordre de 0,01% peut prendre… En fait, ce n'est pas fonction de l'importance de l'erreur. Ce qui induit une difficulté concernant la planification du débogage [7].

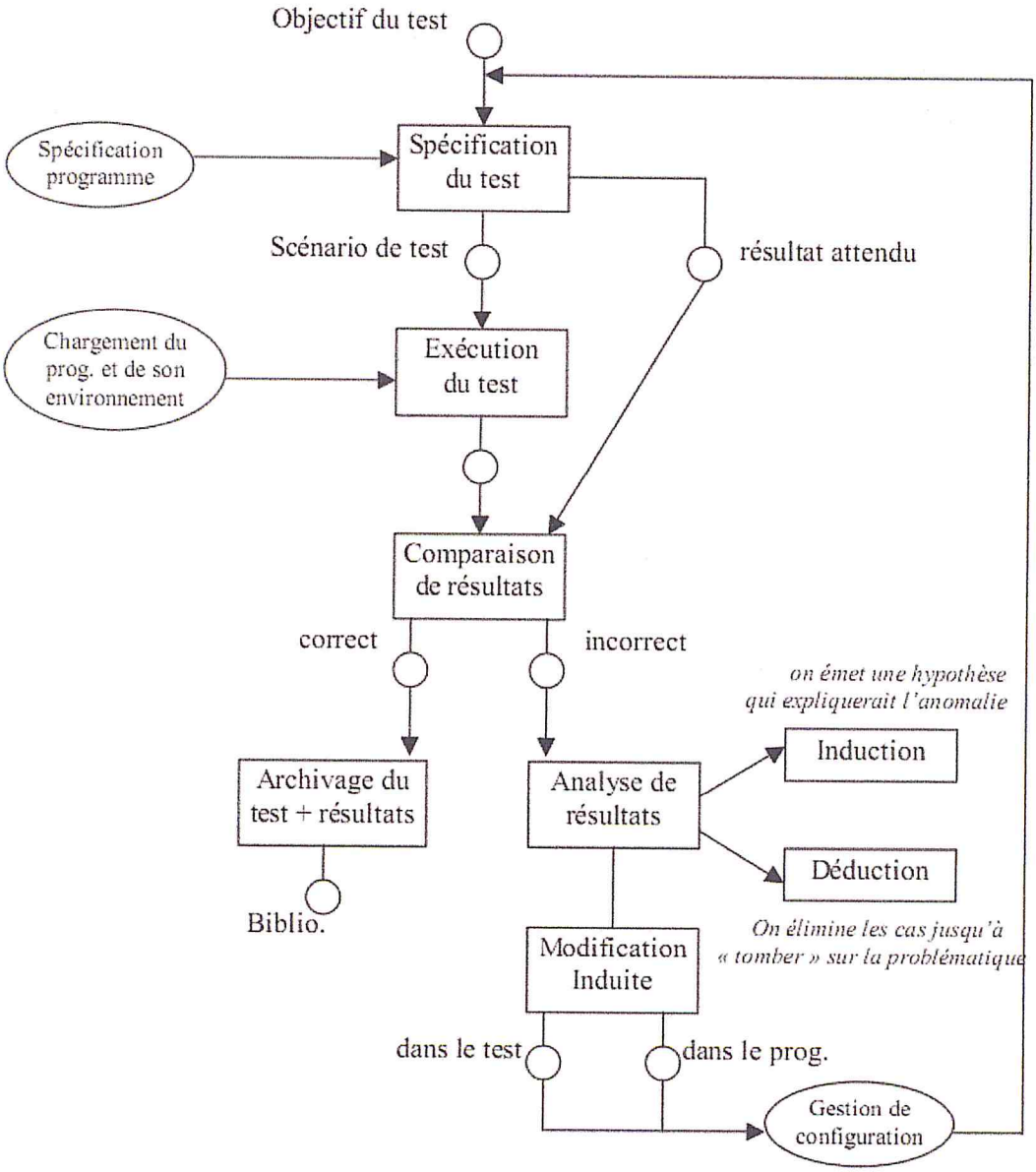



***Figure 6 :*** *Cycle de développement de Test [7].*

#### 2.3.1.2. Mise au point Inductive

On met une hypothèse sur l'ensemble.

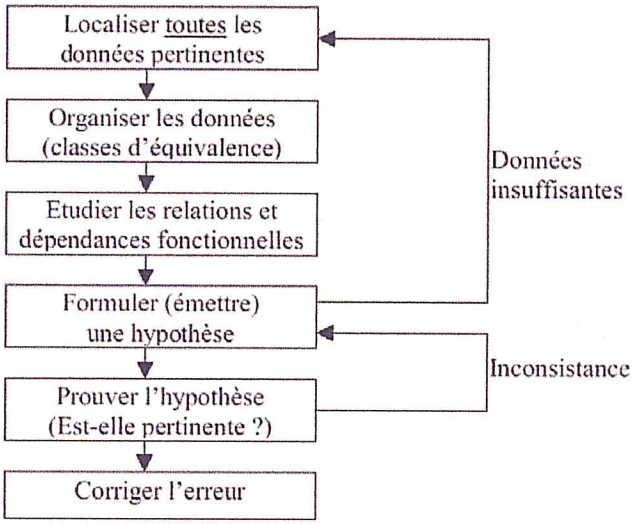



***Figure 7 :*** *Mise au point Inductive [7].*

#### 2.3.1.3. Mise au point Déductive

On traite chaque cause séparément.

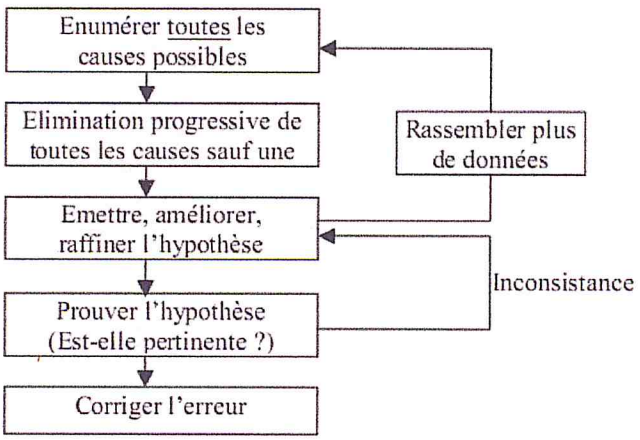
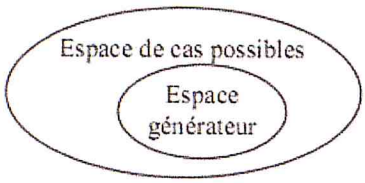



***Figure 8 :*** *Mise au point Déductive [7].*

### 2.3.2. Techniques de test

Plusieurs techniques qui dépendent de l'objectif du test. Mais aucune technique ne sera jamais complète. Le problème est de savoir quelle technique nous assure la complétude, car en fait, aucune ne peut le garantir [7].




*Cela revient à échantillonner de façon représentative.*

***Figure 9 :*** *Techniques de test [7].*

Propriétés recherchées : Si l'espace générateur est couvert alors la probabilité d'une défaillance dans l'espace de cas possible est très faible (inférieure à une limite fixée à l'avance). La difficulté et de faire que l'espace générateur soit consistant et complet.

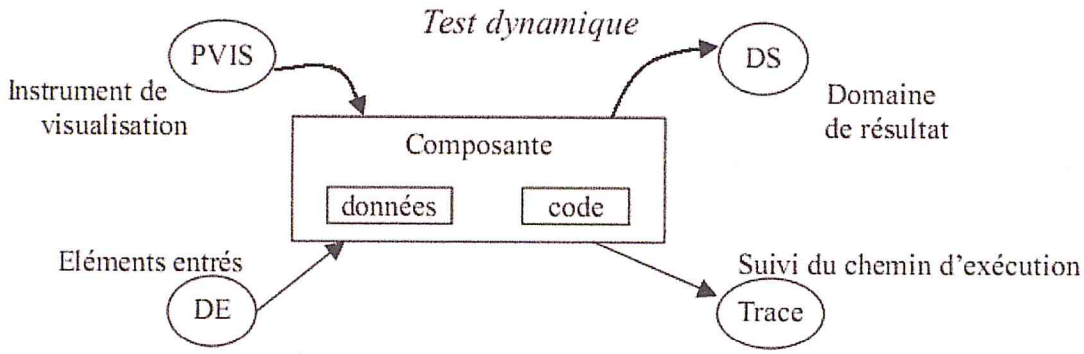



***Figure 10 :*** *Les Différentes techniques de test [7].*

### 2.3.2.1. Tests logiciels [7]

- Tester un logiciel : Exécuter le logiciel avec un ensemble de données réelles

« Un programme sans spécifications est toujours correct »

- Il faut confronter résultats de l'exécution et résultats attendus
- Impossibilité de faire des tests exhaustifs
  - Ex : 2 entiers sur 32 bits en entrée : 264 possibilités. Si 1ms par test, plus de 108 années nécessaires pour tout tester
- Choix des cas de tests :
  - Il faut couvrir au mieux l'espace d'entrées avec un nombre réduit d'exemples
  - Les zones sujettes à erreurs nécessitent une attention particulière

### 2.3.2.2. Tests fonctionnels

***Tests fonctionnels :*** [7]

- Identification, à partir des spécifications, des sous-domaines à tester
  - Produire des cas de test pour chaque type de sortie du programme
  - Produire des cas de test déclenchant les différents messages d'erreur
- Objectif : Disposer d'un ensemble de cas de tests pour tester le programme complètement lors de son implémentation
- Test « boîte noire » parce qu'on ne préjuge pas de l'implémentation
  - Identification possible des cas de test pertinents avant l'implémentation
  - Utile pour guider l'implémentation

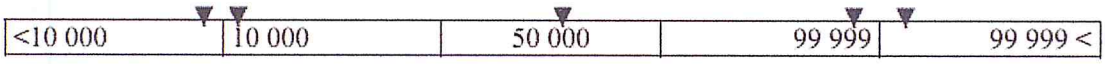
**Exemple a:**

Soit P un programme. Supposons que les données de P soient des nombre de cinq chiffres. Alors les classes de nombre à cinq chiffres s'obtiennent de la manière suivante:

1. $x < 10\,000$ 2. $10000 \leq x \leq 99\,999$ 3. $X \geq 100\,000$

Les cas de test aux limites de classes sont donc 00 000 et 09 999 pour la première classe, 10 000 et 99 999 pour la deuxième classe et 100 000 pour la troisième.

On a donc à tester les nombres suivants :




***Matrices de test***

- Les matrices de test permettent de
  - Formaliser l'identification des sous-domaines à partir des conditions des spécifications
  - Indiquer systématiquement si les combinaisons de conditions sont vraies ou fausses
  - Toutes les combinaisons possibles de V et de F seront examinées

**Exemple b :**

A partir des spécifications, on identité des conditions d'exécution du programme qui permettront de produire les sorties voulues :

1. a = b ou a = c ou b = c
2. a = b et b = c
3. a < b + c et b < a + c et c < a + b
4. a > 0 et b > 0 et c > 0

**Tableau 2 :** Table de vérité (Solution du l'exemple b) [7].

| Condition | Cas 1 | Cas 2 | Cas 3 | Cas 4 | Cas 5 | Cas 6 | Cas 7 | Cas 8 |
|---|---|---|---|---|---|---|---|---|
| (1) | V | V | V | V | V | F | F | F |
| (2) | V | V | F | F | F | F | F | F |
| (3) | F | V | F | F | V | F | F | V |
| (4) | F | V | F | V | V | F | V | V |
| Entrée<br>Sortie | (0,0,0)<br>Erron. | (3,3,3)<br>Equi. | (0,4,0)<br>Erron. | (3,8,3)<br>Pas Tri. | (5,8,5)<br>Iso. | (0,5,6)<br>Erron. | (3,4,8)<br>Pas Tri. | (3,4,5)<br>Scal. |

Il est impossible d'avoir (3)=V et (4)=F, ou encore (2)=V et (1)=F

#### 2.3.2.3. Tests structurels [7]

- Tests « boîte blanche »: détermination des cas de test en fonction du code
- Critère de couverture : Règle pour sélectionner les tests et déterminer quand les arrêter
  - Au pire, on peut sélectionner des cas de test au hasard jusqu'à ce que le critère choisi soit satisfait
- Oracle : Permet de déterminer la sortie attendue associée aux cas sélectionnés
  - Difficile à mettre en place si on veut automatiser complètement le processus de test

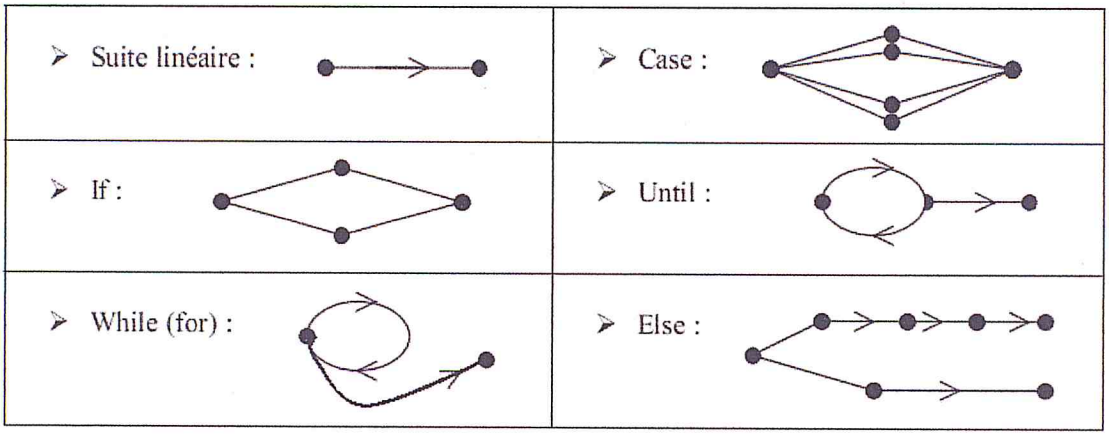
***Test « Boite blanche »***

Ce test consiste à analyser la structure interne du programme en déterminant les chemins minimaux. Afin d'assurer que:

- Toutes les conditions d'arrêt de boucle ont été vérifiées.
- Toutes les branches d'une instruction conditionnelle ont été testées.
- Les structures de donne interne ont été testées (pour assurer la validité).

**Structures de la représentation de la boîte blanche.**

La structures de contrôle se présente sous la forme d'un graphe dit graphe de flot. On représente les instructions comme cela :




Remarque: If A & B & C ⇔ If A then if B then if C then…

***Figure 11 :*** *Mesure de complexité de McCabe [7].*

Cette mesure donne le nombre de chemins minimaux. Elle est donnée par la formule suivante qui correspond au nombre de régions du graphe de flot :

Nb.Arcs – Nb.Nœuds + 2 Nombre cyclomatique

Exemple : Supposons un programme représenté par l'organigramme suivant:

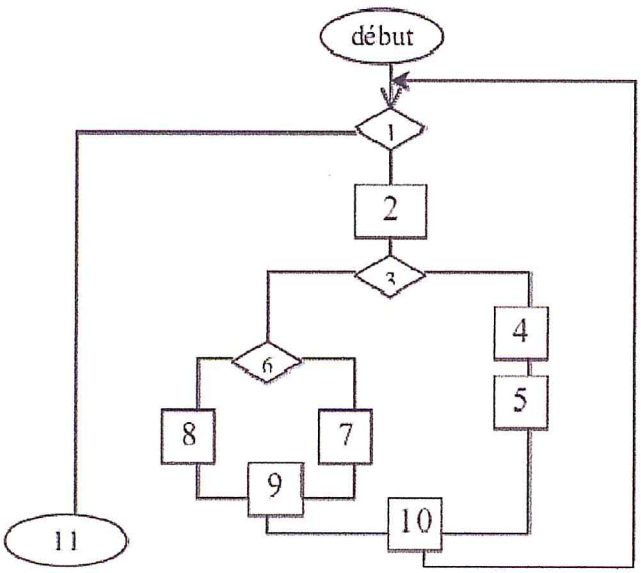



***Figure 12 :*** *Exemple de mesure de McCabe [7].*

On en déduit le graphe de flot suivant :

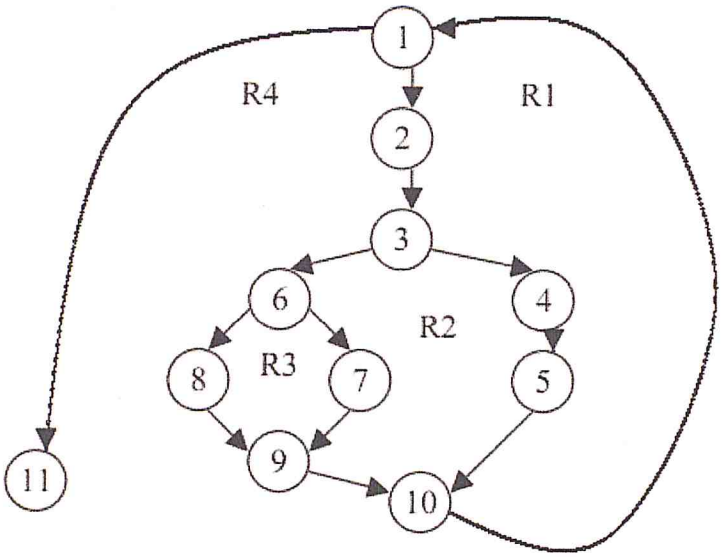



***Figure 13 :*** *Graphe de flot exemple[7].*

Donc le nombre cyclomatique est : Nb.Arcs – Nb.Nœuds + 2 = 13 – 11 + 2 = 4

Pour vérifier, on regarde les chemins minimaux (un test par chemin pour tester toutes les possibilités du programme) :

1. 1.11
2. 1.2.3.4.5.10.1.11
3. 1.2.3.6.7.9.10.1.11
4. 1.2.3.6.8.9.10.1.11

***Couverture de chaque instruction (C0)***

- Selon ce critère, chaque instruction doit être exécutée avec des données de test
- Sélectionner des cas de test jusqu'au ce qu'un outil de couverture indique que toutes les Instructions du code ont été exécutées

**Exemple**

***Tableau 3 :*** *Exemple de couverture des chemins du graphe [7].*

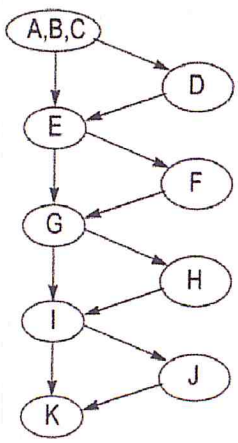
| Noeud | Ligne de code | (3,4,5) | (3,5,3) | (0,1,0) | (4,4,4) |
|---|---|---|---|---|---|
| A | `read a,b,c` | × | × | × | × |
| B | `type="scalène"` | × | × | × | × |
| C | `if (a==b\|\|b==c\|\|a==c)` | × | × | × | × |
| D | `type="isocèle"` | | × | × | × |
| E | `if (a==b&&b==c)` | × | × | × | × |
| F | `type="équilatéral"` | | | | × |
| G | `if (a>=b+c\|\|b>=a+c\|\|c>=a+b)` | × | × | × | × |
| H | `type="pas un triangle"` | | | × | |
| I | `if (a<=0\|\|b<=0\|\|c<=0)` | × | × | × | × |
| J | `type="données erronées"` | | | × | |
| K | `print type` | × | × | × | × |

- Après le quatrième test, toutes les instructions sont exécutées
- Il est rare que le jeu minimal soit bon d'un point de vue fonctionnel

***Test de toutes les branches (C1)***

- Plus complet que C0
- Selon ce critère, il faut emprunter les deux directions possibles au niveau de chaque décision
- Nécessite la création d'un graphe de contrôle et de couvrir chaque arc du graphe

**Exemple**



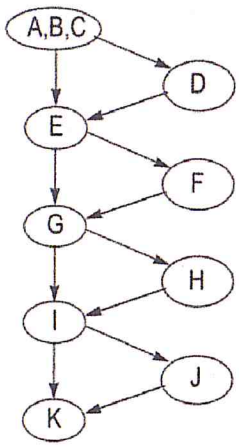

| Arcs | (3,4,5) | (3,5,3) | (0,1,0) | (4,4,4) |
|---|---|---|---|---|
| ABC-D | | x | x | x |
| ABC-E | x | | | |
| D-E | | x | x | x |
| E-F | | | | x |
| E-G | x | x | x | |
| F-G | | | | x |
| G-H | | | x | |
| G-I | x | x | | x |
| H-I | | | x | |
| I-J | | | x | |
| I-K | x | x | | x |
| J-K | | | x | |

***Figure 14 :*** *Test de toutes les branches (C1)[7].*

***Test de tous les chemins***

- Encore plus complet
- Selon ce critère, il faut emprunter tous les chemins possibles dans le graphe
- Chemin : suite unique de nœuds du programme exécutes par un jeu spécifique de données de test
- Peu adaptée au code contenant des boucles, le nombre de chemins possible étant souvent infini

**Exemple**




| Chemin | V/F | Données | Sortie |
|---|---|---|---|
| ABCEGIK | FFFF | (3,4,5) | Scalène |
| ABCEGHIK | FFVF | (3,4,8) | Pas un triangle |
| ABCEGHIJK | FFVV | (0,5,6) | Données erronées |
| ABCDEGIK | VFFF | (5,8,5) | Isocèle |
| ABCDEGHIK | VFVF | (3,8,3) | Pas un triangle |
| ABCDEGHIJK | VFVF | (0,4,0) | Données erronées |
| ABCDEFGIK | VFVV | (3,3,3) | Equilatéral |
| ABCDEFGHIJK | VVVV | (0,0,0) | Données erronées |

***Figure 15 :*** *Test de tous les chemins [7].*

***Couverture des conditions multiples***

- Un critère de test de conditions multiples impose que :
- Chaque condition primitive doit être évaluée à la fois comme vraie et comme fausse
- Toutes les combinaisons V/F entre primitives d'une condition multiple doivent être testées
- Très complet, ne pose pas de problème en cas d'itérations

**Exemple : le cas** Pour if (a==b||b==c||a==c)

***Tableau 4 :*** *Exemple de Couverture des conditions multiples [7].*

| Combinaison | Données de test | Branche |
|---|---|---|
| V?? | (3,3,4) | ABC-D |
| FV? | (4,3,3) | ABC-D |
| FFV | (3,4,3) | ABC-D |
| FFF | (3,4,5) | ABC-E |

## 2.4. Introduction aux Réseaux Bayésiens

L'homme est curieux et c'est sans doute ce qui explique le mieux son cheminement depuis le début de l'humanité jusqu'a nos jours. Ce besoin de comprendre les phénomènes observés et le désir de les anticiper est au cœur de ses préoccupations. C'est ce qui explique l'émergence et le succès de la statistique, une discipline scientifique en plein essor. Les Réseaux Bayésiens ont pour objectif d'acquérir, représenter et utiliser la connaissance.

La représentation des connaissances et le raisonnement à partir de ces représentations a donné naissance à de nombreux modèles. Les modèles graphiques probabilistes, et plus précisément les Réseaux Bayésiens, initiés par Pearl Judea dans les années 1980, se sont révélé des outils très pratiques pour la représentation de connaissances incertaines, et le raisonnement à partir d'informations incomplètes.

Un Réseau Bayésien $B = (G, \theta)$ est défini par [9]

- $G = (X,E)$, graphe orienté sans circuit dont les sommets sont associés à un ensemble de variables aléatoires $X = \{X_1, \cdots, X_n\}$,
- $\theta = \{P(X_i \mid Pa(X_i))\}$, ensemble des probabilités de chaque nœud $X_i$ conditionnellement à l'état de ses parents $Pa(X_i)$ dans G.

Ainsi, la partie graphique du Réseau Bayésien indique les dépendances (ou indépendances) entre les variables et donne un outil visuel de représentation des connaissances, outil plus facilement appréhendable par ses utilisateurs. De plus, l'utilisation de probabilités permet de prendre en compte l'incertain, en quantifiant les dépendances entre les variables. Ces deux propriétés ont ainsi été à l'origine des premières dénominations des Réseaux Bayésiens, "systèmes experts probabilistes", où le graphe était comparé à l'ensemble de règles d'un système expert classique, et les probabilités conditionnelles présentées comme une quantification de l'incertitude sur ces règles [9].

Pearl et al.[1] Ont aussi montré que les Réseaux Bayésiens permettaient de représenter de manière compacte la distribution de probabilité conjointe sur l'ensemble des variables :

$$P(X_1, X_2, \cdots, X_n) = \pi_{i=1}^{n} P(X_i \mid Pa(X_i)) \qquad (2.2)[9].$$

*1 pour plus d'informations réf. Pearl et al, « Causality: Models, reasoning en inference», MIT PRESS, 2000.*

Cette décomposition d'une fonction globale en un produit de termes locaux dépendant uniquement du nœud considéré et de ses parents dans le graphe, est une propriété fondamentale des Réseaux Bayésiens. Elle est à la base des premiers travaux portant sur le développement d'algorithmes d'inférence, qui calculent la probabilité de n'importe quelle variable du modèle à partir de l'observation même partielle des autres variables. Ce problème a été prouvé NP-complet, mais a abouti à différents algorithmes qui peuvent être assimilés à des méthodes de propagation d'information dans un graphe. Ces méthodes utilisent évidemment la notion de probabilité conditionnelle, i.e. quelle est la probabilité de $X_i$ sachant que j'ai observé $X_j$ , mais aussi le théorème de Bayes, qui permet de calculer, inversement, la probabilité de $X_j$ sachant $X_i$, lorsque $P(X_i \mid X_j)$ est connu. Ce sont des modèles graphiques probabilistes utilisant le théorème de Bayes pour "raisonner" [9].

Les Réseaux Bayésiens modélisant efficacement la loi de probabilité conjointe de l'ensemble des variables, sont un formalisme privilégié pour l'utilisation de méthodes d'échantillonnage stochastique. Celles-ci permettent de générer à volonté des données simulées. Les Réseaux Bayésiens sont alors des outils de simulation qui permettent à l'expert d'observer le comportement de son système dans des contextes qu'il n'est pas forcément capable de tester lui même.

**Pourquoi les Réseaux Bayésiens ?**

Les Réseaux Bayésiens présentent pour certaines utilisations des avantages notables sur d'autres modèles. Ils sont également un moyen de combiner :

- une connaissance préalable avec une information nouvelle ;
- des variables catégorielles, discrets ou continues ;
- des données empiriques et des jugements d'experts.

Les responsables et les décideurs apprécient souvent, dans une approche par Réseau Bayésien, le fait que les résultats apparaissent sous forme de lois de probabilité qui mettent en évidence les incertitudes. Ces représentations sont adaptées aux contextes d'analyse de risques et de gestion de risques. La combinaison de ces caractéristiques – dont certaines peuvent être assurées par d'autres techniques – rend les Réseaux Bayésiens particulièrement intéressants aussi bien pour les spécialistes que pour les responsables. D'autres approches de modélisation peuvent compléter l'utilisation de Réseaux Bayésiens : les techniques statistiques traditionnelles, les méthodes d'ordination et de corrélation, et aussi les autres modes de

représentation d'avis d'experts tels que les modèles de logique floue, les Réseaux neuronaux ou les systèmes experts.

### 2.4.1. Réseau Bayésien causaux :

Un Réseau causal (RBc) est un Réseau Bayésien $\langle V, G, P(Vi|Pa(Vi))\rangle$ mais avec quelques propriétés supplémentaires[2]. La propriété principale réside dans le fait que chaque ensemble d'arcs $Pa(V_i) \rightarrow V_i$ ne représente plus seulement une dépendance probabiliste, mais aussi une relation causale [14].

Dans un RBc, chaque distributions de probabilité conditionnelle (CPD) $P(V_i|Pa(V_i))$ représente un processus stochastique dans lequel les valeurs de $V_i$ sont choisies en fonction des valeurs de $Pa(V_i)$ (mais pas l'inverse). Seul le Réseau du bas est un Réseau Bayésien causal puisque il y a un vrai lien de cause à effet entre X et Y.

En plus de leur aspect plus intuitif, les RBc permettent de faire des calculs d'inférence probabiliste classique (Comme les RB usuels), mais aussi d'inférence causale, i.e. de calculer l'effet d'une intervention sur une certaine variable sur d'autres variables [14].

***Exemple :***
La connaissance de X implique / influe sur la connaissance de Y
(X est une cause directe de Y, Y une conséquence directe de X)

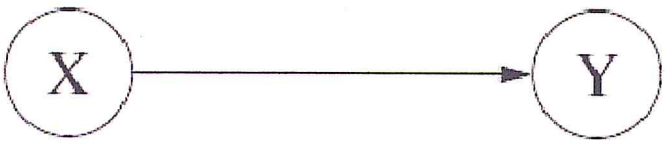



***Figure 17:*** *Réseau Bayésien causaux [14].*

S'il existe une information causale de X vers Y, toute information sur X peut modifier la connaissance qu'on a de Y, et, réciproquement, toute information sur Y peut modifier la connaissance qu'on a de X.

---

*2 Selon la définition de Pearl Judea dans son livre, réf. Pearl et al, « Causality: Models, reasoning en inference», MIT PRESS, 2000.*

### 2.4.2. Fonctionnement des Réseaux Bayésiens

Tout comme pour les Réseaux de neurones, il convient de construire un Réseau Bayésien spécifiquement dédié au problème que l'on souhaite traiter. On va distinguer deux phases [10]:

- La phase de construction du Réseau qui peut solliciter des experts et des techniques d'apprentissage ;

- La phase d'utilisation qui fait appel à une capacité très intéressante des Réseaux Bayésiens, l'inférence.

#### 2.4.2.1. Construction des Réseaux Bayésiens

Les Réseaux Bayésiens permettent de combiner la connaissance d'experts avec la connaissance extraite à partir des données. Les experts peuvent par exemple déterminer les dépendances entre les variables alors qu'un apprentissage automatique permettra de déterminer la distribution de probabilité associée à chaque variable.

Il ne s'agit ici que d'un exemple car il est tout à fait envisageable que des experts définissent entièrement un Réseau Bayésien, graphe et distribution de probabilité. Au contraire, ces deux éléments peuvent être construits automatiquement par apprentissage du système [10].

#### 2.4.2.2. Utilisation des Réseaux Bayésiens

L'utilisation des Réseaux Bayésiens repose sur la propagation de l'information au sein du Réseau, c'est à dire des calculs de probabilités, c'est ce que l'on nomme l'inférence. Après avoir fait une observation sur une variable comment cette information va t-elle se répercuter sur l'état des autres variables ? [10].

### 2.4.3. Applications des Réseaux Bayésiens

La première application des Réseaux Bayésiens est le diagnostic. Connaissant la panne, un système basé sur des Réseaux Bayésiens pourra déterminer les causes les plus probables ayant entraînées le problème. Toutefois, les Réseaux Bayésiens sont aussi utilisés pour faire de la classification. Ils vont alors se baser sur un certain nombre de caractéristiques des textes pour pouvoir les classer dans une catégorie [10].

## 2.5. Etude de deux méthodes basées sur RB et deux populations de donnée

### 2.5.1. Le cas du projet « SERENE » : SafEty and Risk Evaluation using Bayesian Nets:

#### 2.5.1.1. Utilisations possibles du modèle

Un R.B. permettra de donner des estimations d'autant plus précises que l'on aura plus de données sur le système évalué. Il pourra servir dès les premières phases (spécifications) pour donner des premières tendances. On pourra simuler l'impact de différents processus de développements et en choisir un qui satisfaire les objectifs que l'on se fixe. En fin de développement, il aide à l'élaboration d'un argumentaire de sûreté, et un outil de dialogue avec les autorités de sûreté (i.e. outil d'aide à la décision).

#### 2.5.1.2. Interprétation d'un Réseau Bayésien

Un RB est une représentation concise de la loi conjointe sur l'ensemble des variables aléatoires du Réseau. Cette loi est l'information la plus complète que l'on puisse avoir sur l'ensemble des variables aléatoires considérées, et leurs interrelations.

**Loi conjointe** = $\{ P ( X_1 = V_1 , X_2 = V_2 ,... \ X_n = V_n )\}$ où $V_1, V_2,...V_n$ prennent toutes les combinaisons de valeurs possibles pour les $X_i$ ($X_i$ = V.A. associée au nœud i du RB).

• *une* : la relation entre les faits "pair" et "égal à 6" pour le résultat d'un jet de dé peut se représenter par : pair -> égal à 6, P(pair)=1/2, P(égal à 6|pair) = 1/3, P(égal à 6|impair) = 0 ou bien par: égal à 6 -> pair, P(égal à 6) = 1/6, P(pair |égal à 6) = 1, P(pair |non égal à 6) = 2/5

• *concise* : Ex d'un RB dans lequel tout nœud est binaire, et a 2 parents, sauf P nœuds initiaux : on représente la loi conjointe par 4(N - P) + P nombres, au lieu de $2^N$. [12].

#### 2.5.1.3. Argumentaire de sûreté utilisant un RB

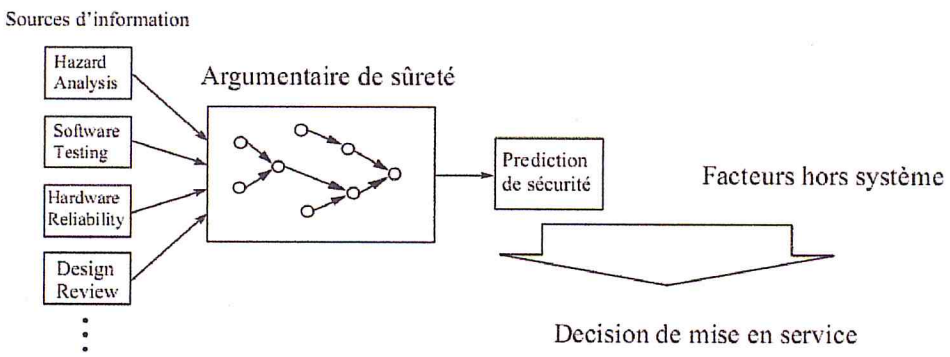



***Figure 18:*** *Argumentaire de sûreté utilisant un RB [11].*

**Les Information disponible vers la fin du cycle de vie sont :**

- Diagnostics sur processus
- Spécification
- Design & Développement
- Activité de validation pour chaque phase
- Documentation technique
- Résultats de tests ...

## Vue historique sur le RB d'EDF

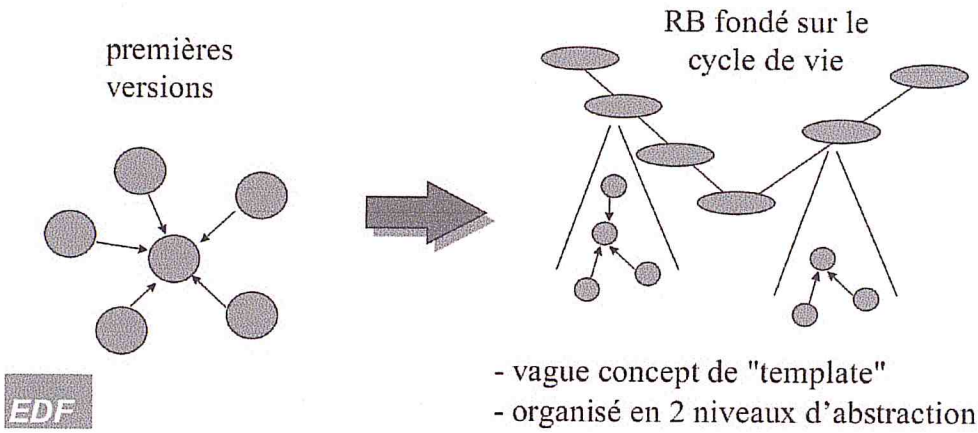



***Figure 19 :*** *Vue historique sur le RB d'EDF [11].*

**Identification de «motifs» répétitifs**

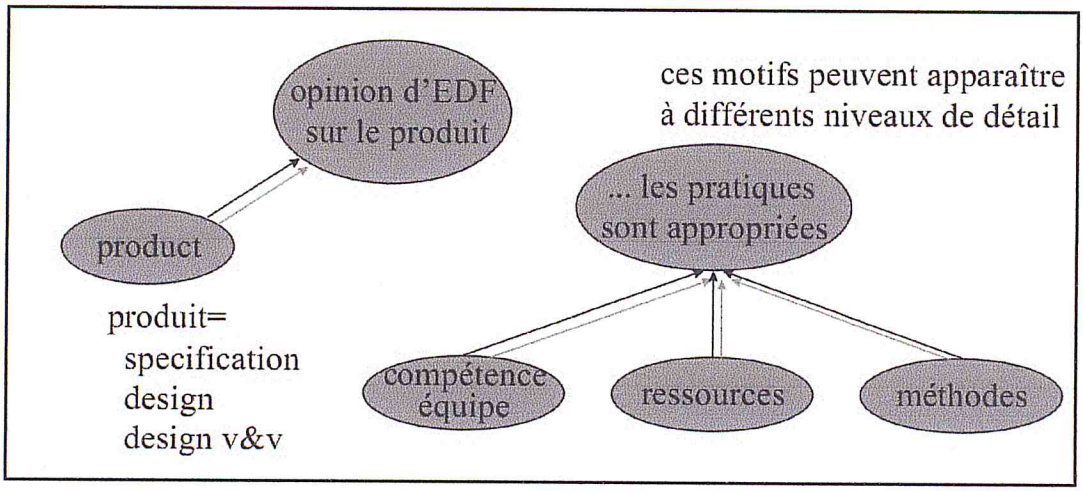



***Figure 20:*** *La méthode SERENE introduit le concept d' « idiome » [11].*

**Qu'est ce qu'un idiome ?**

Un Idiome résume une classe de raisonnements typiques dans un argumentaire de sûreté. Il est considérer comme un concept abstrait, il n'a pas de réalisation informatique directe.

1 idiome ←→ **beaucoup** d'instances, qui peuvent être sauvegardées sous forme de «Template» avec l'outil SERENE.

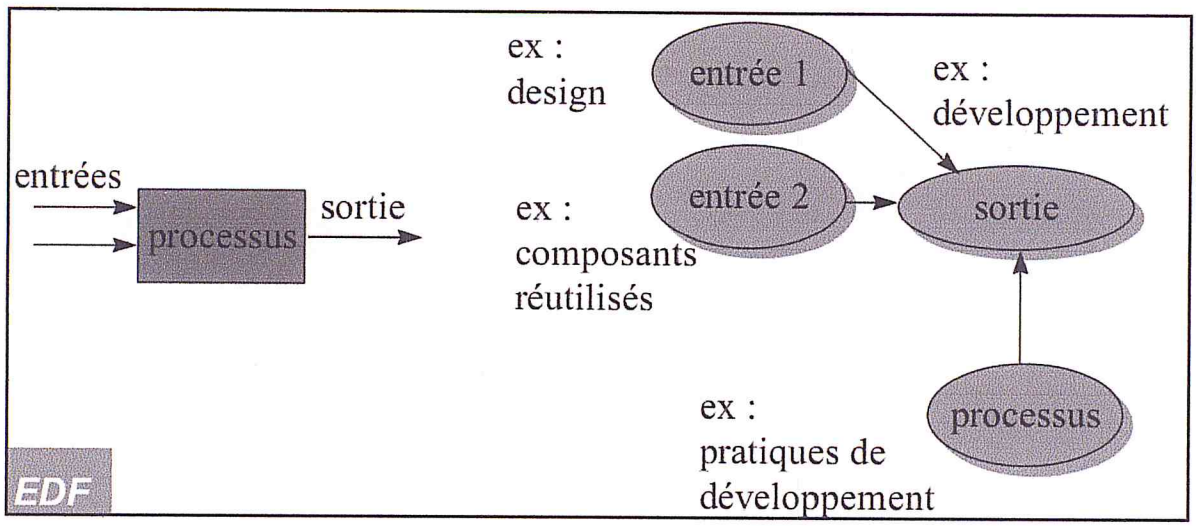



***Figure 21**: L'idiome processus-produit [11].*

L'idiome peut être vu comme une «mesure», **Exemple :**

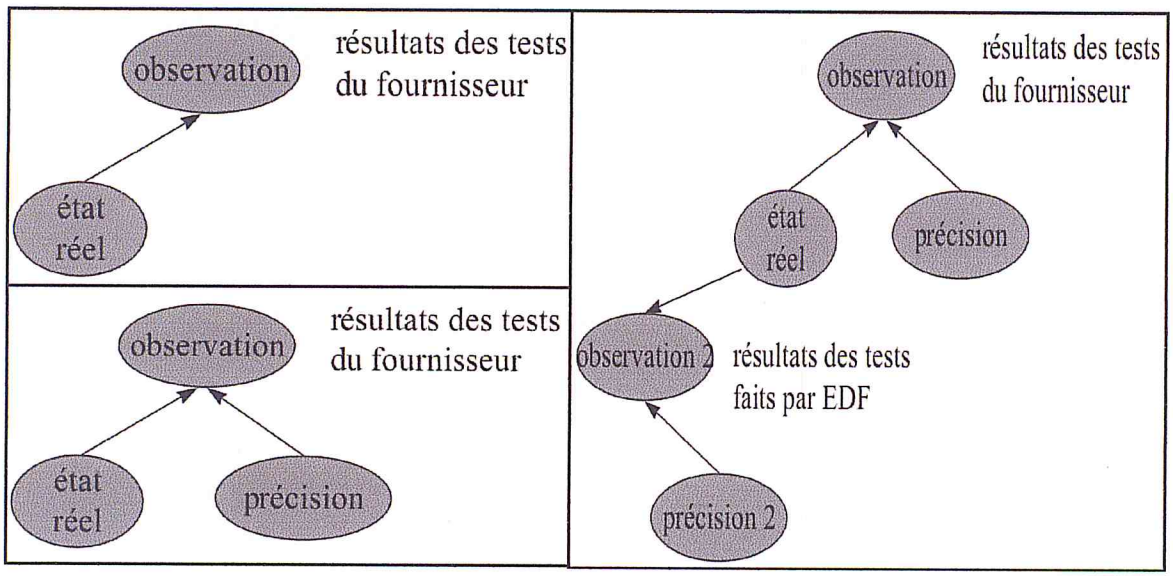



***Figure 22 :** Exemple d'idiome comme « mesure » [11].*

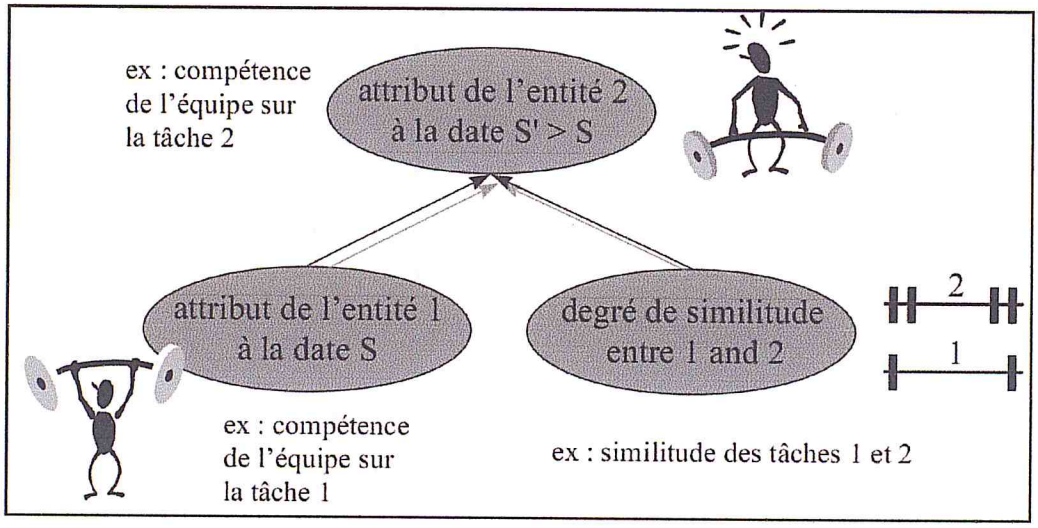



***Figure 23:*** *L'idiome comme expérience historique [11].*

Comme notre projet est la fiabilité de logiciel on peut construire le Réseau Bayésien d'une façon très facile car l'ensemble des objets a étudie sont fixe qui ne change pas, ils acceptent d'autres objets.

#### 2.5.1.4. Structure hiérarchique d'un argumentaire de sûreté en RB

Concept de nœuds abstraits, possible grâce à la jonction automatisée. Ex :

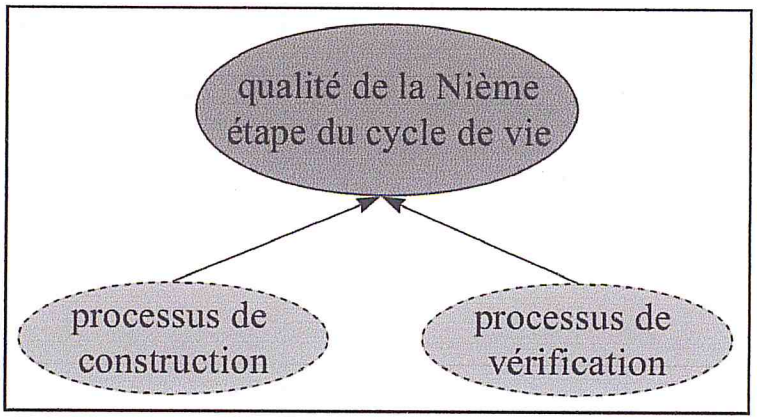



***Figure 24 :*** *Structure hiérarchique d'un argumentaire de sûreté en RB [11].*

#### 2.5.1.5. Récapitulation

La méthode SERENE est fondée sur deux mécanismes fondamentaux : une décomposition top-down : des nœuds abstraits (qui représentent des sous-arguments) aux nœuds concrets, et un mécanisme d'assemblage entre idiomes. Ces deux mécanismes SERENE seront supportés par notre outil qu'on va le proposer dans le prochain chapitre [12].

### 2.5.1.6. Élicitassions des tables de probabilités des nœuds (NPT)

Quelques choix pragmatiques pour réduire le temps d'élicitassions des NPTs

- 2 états pour chaque nœud : oui et non
- Probabilités: 0 - 0.25 - 0.5 - 0.75 – 1
- Valeurs 0 et 1 modélisent des relations déterministes => signifient en fait "proche de 0/1"

***Tableau 5:*** *Exemple de table de probabilités pour un nœud [11].*

| Les méthodes de vérification de la spéc. sont appropriées. | O | | | | N | | | |
|---|---|---|---|---|---|---|---|---|
| Les experts du fournisseur sont compétents pour cette tâche | O | | N | | O | | N | |
| Cette tâche bénéficie de ressources suffisantes | O | N | O | N | O | N | O | N |
| La vérification de la spéc. par le fournisseur est correcte | 1 | .75 | .25 | 0 | .75 | .5 | 0 | 0 |

### 2.5.2. Le cas des fonctions élémentaires et « Réseau Bayésien oriente objet »

La modélisation par RB de systèmes comportant un nombre élevé de variables aboutit généralement à un modèle complexe. Afin d'éviter ce phénomène, Koller[3] a défini une classe particulière de RB, les Réseaux Bayésiens Orientés Objet (RBOO). Leur modélisation est ainsi basée sur la décomposition du réseau global en niveaux hiérarchiques. Ce mode de représentation permet de décentraliser et de structurer l'information en faisant communiquer des RB de taille réduite. Les RBOO, de par leur structure, sont donc adaptés à la modélisation de systèmes industriels [13].

D'après les travaux de Koller, Nous présentons un modèle d'estimation de la fiabilité de logiciel basé sur la théorie des Réseaux Bayésiens (RB). La construction de ce modèle est réalisée en intégrant les informations apportées par diverses méthodes d'analyse fonctionnelle et dysfonctionnelle traduisant les connaissances classiquement disponibles dans le milieu d'étude. La méthode de modélisation proposée permet la prise en compte de mode commun et de plusieurs modes de défaillances par composant.

---

*3 Koller D et A. Pfeffer. Object Oriented Bayesian Networks. In Proceeding of the Thirteenth Annual Conference on Uncertainty in Artificial Intelligence (AI-97). Rhode Island, USA, Juillet, 1997.*

L'apport de l'expertise au travers de diverses méthodes d'analyse est non négligeable car il permet l'introduction de *variables non physiques* et *informatives* qui décomposent le système de façon compréhensible. Les schémas sont alors plus facilement interprétables, car les étapes importantes y sont décomposées. La quantification des relations mises en évidence est également facilitée par la sémantique du modèle obtenu. En contrepartie, la qualité du modèle dépend de la qualité des connaissances de l'expert.

Les méthodes d'analyse que nous utilisons pour construire le modèle permettent de représenter le fonctionnement et les dysfonctionnements d'un système. La construction d'un modèle de fiabilité, à partir de ces analyses.

### *1. Détermination de la structure d'un Réseau Bayésien à partir d'analyses fonctionnelles*

La structure des RBOO permet une décomposition arborescente du fonctionnement du système. Les fonctions mises en évidence par l'analyse fonctionnelle sont alors représentées par des RBOO [13].

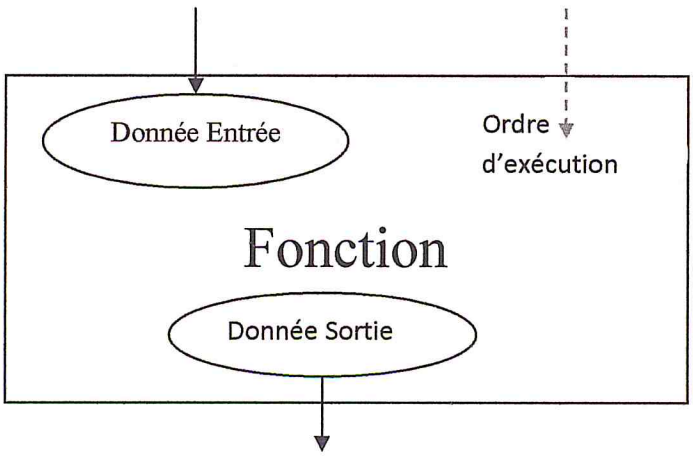



***Figure 25:*** *Transcription générique d'une fonction en RBOO [13].*

Qui sont ensuite assemblés pour établir le modèle global du fonctionnement du système.

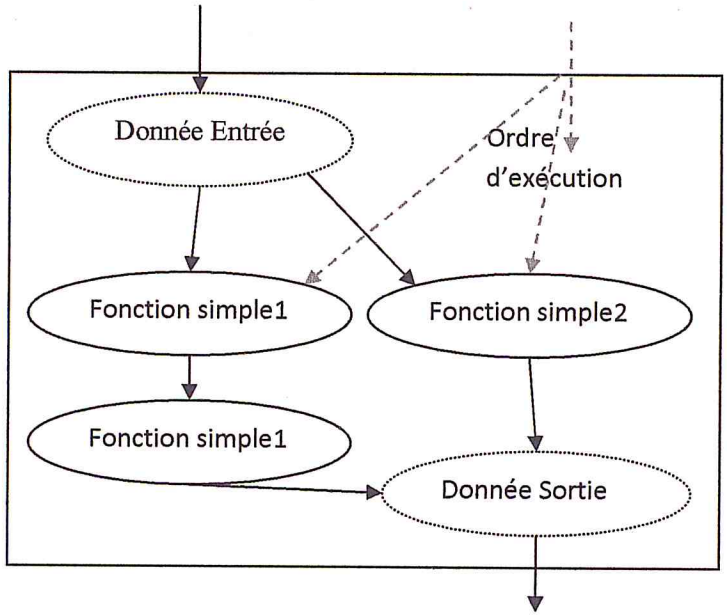



***Figure 26:*** *Décomposition d'une fonction [13].*

Les nœuds en pointillés et en trait plein indiquent respectivement les nœuds d'entrée et de sortie de la fonction étudiée. Le Support ou Composant n'est présent qu'au niveau des fonctions élémentaires (FE)

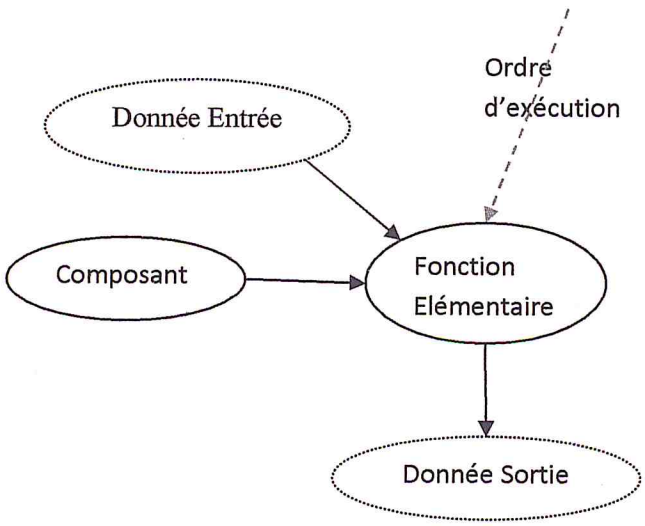



***Figure 27 :*** *Fonction élémentaire [13].*

Lors de la modélisation des fonctions supportant directement les composants (CMP). Cette représentation sous forme générique permet de construire des modèles de composants indépendants de la structure du système modélisé.

De cette façon, une "boîte à outils" de composants facilite les mises à jour et les modifications de paramètres.

### 2. *Détermination de la structure d'un Réseau Bayésien à partir d'analyses dysfonctionnelles*

Les méthodes d'analyse fonctionnelle n'apportent pas d'informations sur les effets des défaillances. Il est donc nécessaire de compléter le Réseau à l'aide de méthodes d'analyse dysfonctionnelle qui permettent de spécifier les états des variables [13].

Exemple : étude d'une partie de logiciel qui fait la fonctionnalité X.

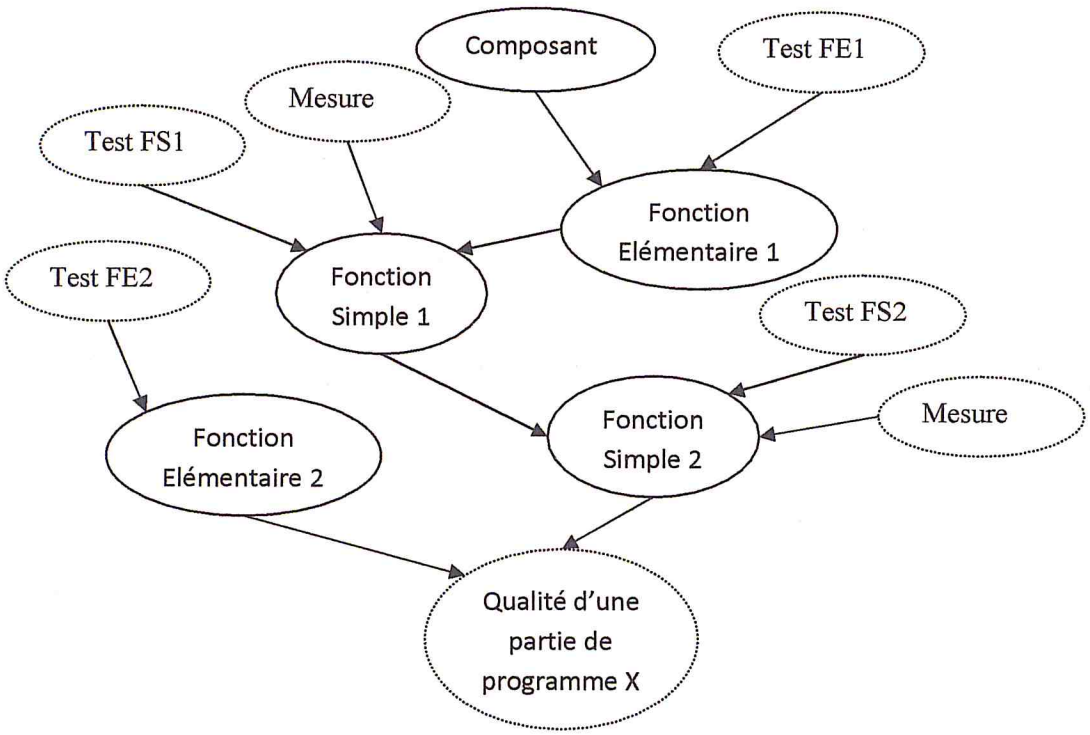



***Figure 28:*** *Structure RBOO à partir d'analyses dysfonctionnelles.*

Les tests représentent les données entrant car on ne peut pas les essayer tous alors on fait un ensemble de tests pour prendre une image sur la qualité de variable. Les mesures représentent les propriétés d'une variable et ses effets sur la qualité de variable.

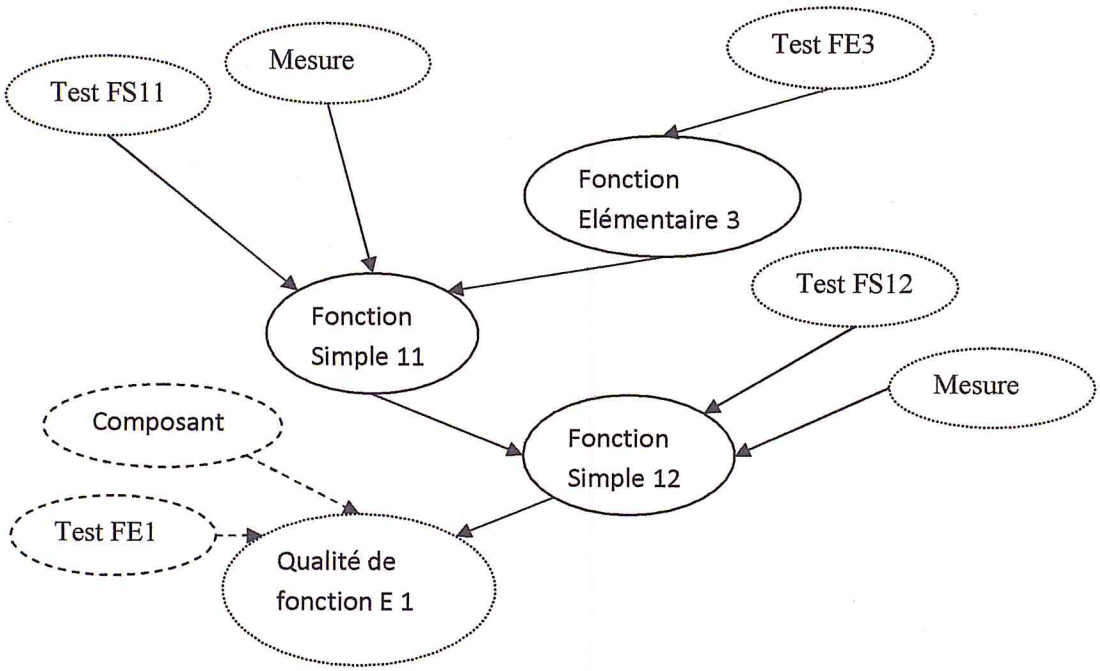



***Figure 29 :*** *Ajout des variables externe dans la structure RBOO.*

Un mode de défaillance représente l'effet par lequel une défaillance est observée. Les analyses dysfonctionnelles telles que l'Analyse des Modes de Défaillances et de leurs Effets recensent ces modes associés aux variables du logiciel. Elles permettent ainsi de définir tous les états des nœuds du RB.

On peut facilement construire un Réseaux Bayésien oriente objet avec les différent schéma d'exécution de différent option de logiciel on traite un schéma d'exécution comme une fonction élémentaire, et les fonctions principal des fonction de base indécomposable et pour les relation d'indépendance l'effet d'une fonction utilise les sorties d'une fonction ou appel une fonction et de représente une fonction plusieurs fois dans des différentes situations et alors étudie une fonction dans différent intervalle de données

Exemple : Schéma d'exécution d'une partie PX d'un logiciel

Si on définie la flèche rouge comme un appel d'utilisation et le noir pour le passage des donnée

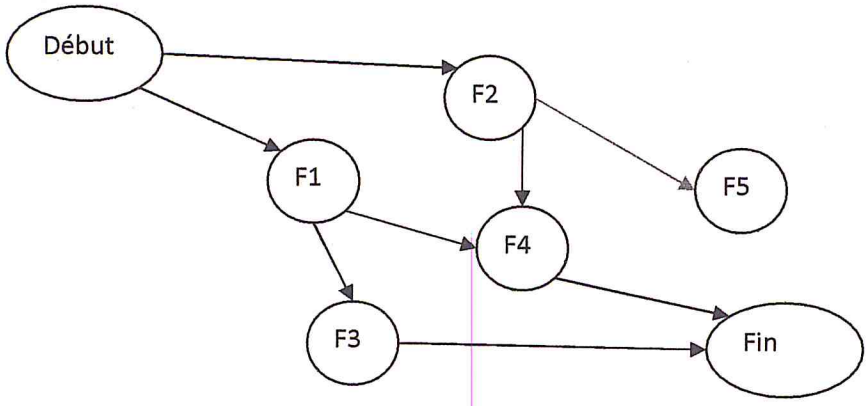



***Figure 30 :*** *Schéma d'exécution d'une partie PX d'un logiciel.*

On obtient le Réseau Bayésien suivant :

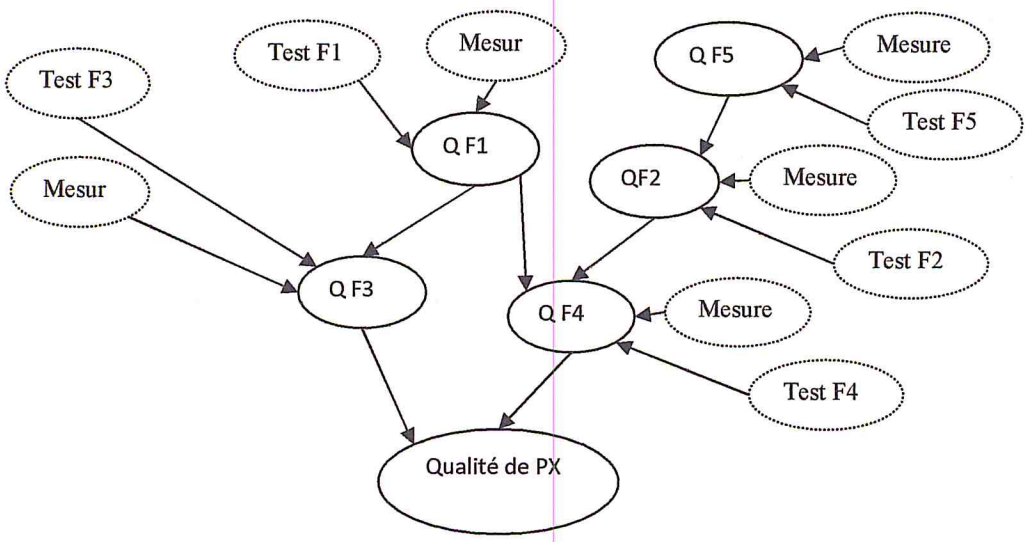



***Figure 31 :*** *Schéma d'exécution d'une partie PX d'un logiciel avancée.*

Elles permettent ainsi de définir tous les états des nœuds du RB. La TPC est considérée comme une matrice de passage permettant la transition entre deux nœuds qui représentent des composants, des fonctions élémentaires, des fonctions intermédiaires ou principales. Ainsi, les principaux éléments contenus dans les analyses dysfonctionnelles apparaissent dans la TPC et à chaque variable sont associés un mode de fonctionnement normal et *n* modes de défaillances [13].

Lors de l'élaboration d'une analyse dysfonctionnelle, les effets d'une défaillance d'un sous-système deviennent généralement des modes de défaillance du système. Les états des variables du RBOO représentent alors :

- au niveau Composants, les modes de défaillance et de fonctionnement ;

- au niveau Fonctions Elémentaires, les effets des modes de défaillance ou de fonctionnement des composants ;
- au niveau des Fonctions principales, les effets des modes de défaillance ou de fonctionnement des composants ou des fonctions élémentaires.

## 2.6. Conclusion

La modélisation qu'on va proposer repose sur la dualité des analyses fonctionnelles et dysfonctionnelles d'un système dont les performances doivent être évaluées et/ou améliorées. C'est grâce aux RBOO que le modèle est décrit dans un cadre statistique formel. Les analyses fonctionnelles et dysfonctionnelles sont généralement menées pendant les phases de cycle de vie et permettent de construire facilement la structure du RBOO en le décomposant en plusieurs niveaux d'abstraction.

Les travaux en relation avec le projet SERENE ne décrivent pas les fondements théoriques de l'inférence des réseaux Bayésiens « hiérarchisés ». Mais les travaux de Koller *et al.*, (1997) explique les principes d'inférence des Réseaux Bayésiens Orientés Objet (RBOO). Cette nouvelle structure permet une modélisation par des réseaux Bayésiens communiquant au sein d'une arborescence hiérarchisée. Néanmoins, pour les systèmes complexes, un RB de grande taille est nécessaire. Il est, dans ce cas, difficile aussi bien de concevoir le modèle que de le faire évoluer en tenant compte du cycle de vie du processus. C'est pour cette raison que la méthode adoptée par le projet SERENE est intéressante.

Le premier intérêt de l'utilisation des RBOO est la formalisation générique de chaque niveau fonctionnel. Le modèle, basé sur l'évaluation probabiliste des RB, est simple et facile à construire. Les composants sont définis d'une manière indépendante du reste du modèle. Ils peuvent être issus de bibliothèques de composants [vu comme une prévision] et leurs caractéristiques (métriques, tests) sont mises à jour facilement.

Le second intérêt des RBOO réside dans le calcul des probabilités associées aux états d'un nœud, connaissant l'état d'une (ou de plusieurs) variable(s) du système. Il est ainsi possible d'estimer l'impact de l'état d'une variable aléatoire sur le processus. Les calculs prennent en compte les événements incertains et les propagent par des relations de causes à effets. Les RBOO constituent donc un outil puissant pour l'aide à la décision et l'estimation -le calcule- de la fiabilité des logiciels.

# Chapitre 3 :

## MÉTHODE ET CONCEPTION

# 3. Méthode et Conception

## 3.1. Introduction :

Ce chapitre à pour objectif de présenter notre méthode proposé qui est facile à mettre en œuvre qui combine entre une vue graphique du logiciel et le calcul probabiliste –estimation de la fiabilité-, valide et applicable à n'importe quel logiciels, basé sur les Réseaux Bayésiens OO qui ont pour objectif d'acquérir, représenter et utiliser les connaissances. Plus précisément, il s'agit d'une part de construire des modèles Bayésiens pour chaque processus de cycle de vie d'un logiciels au cours de son développement et un autre globale pour pouvoir bien maitriser les défaillances et réparations des systèmes divers, et d'autre part de mettre en œuvre des méthodes statistiques (avec des métriques logiciels et des tests) pour exploiter les données relevées par les praticiens dans le but de calculer la fiabilité et ses dérivées.

## 3.2. Démarche adapté pour le calcul de la fiabilité du logiciel

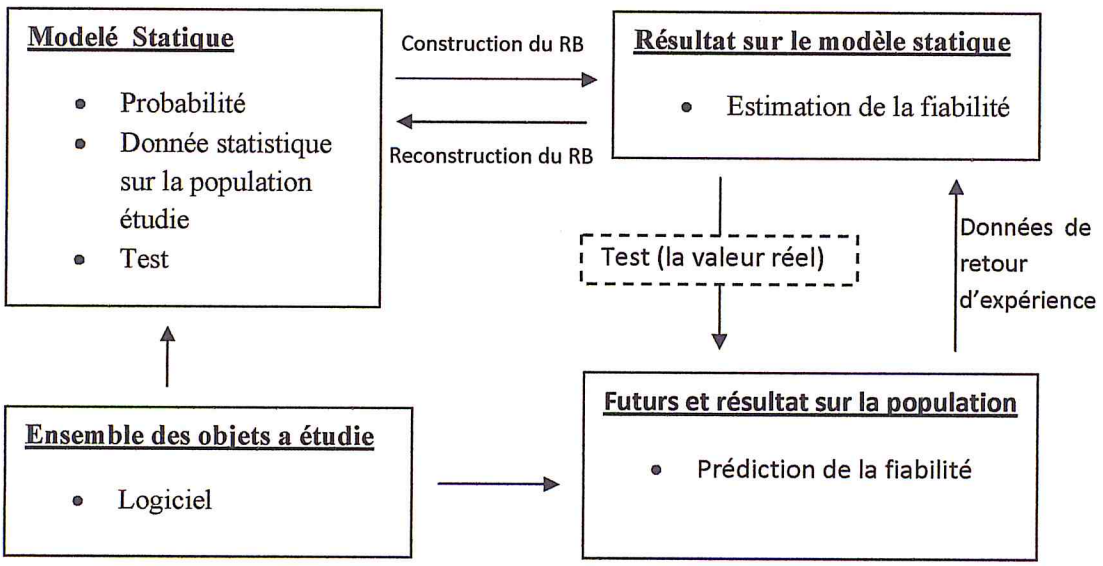



***Figure 32:*** *Démarche adapté pour le calcul de la fiabilité.*

On utilise comme approche de calcule de la fiabilité les Réseaux Bayésiens construit à partir du modèle statique:

- Probabilité : rapport entre le nombre de cas favorables et le nombre de cas possibles
- Statistiques : définir des hypothèses, effectuer des tests, faire des estimations, fonction de données (moyenne, écart-type, médian, …)
- Estimateur : opérations statistiques échantillons prélevés

Alors on démarre depuis une population d'objet pour construire un modèle statique sous forme d'un Réseau Bayésien causaux afin d'estimer le calcule de la fiabilité, ensuite en

exploite les résultats obtenus avec des données des tests pour prédire et calculer la fiabilité future du logiciel.

## 3.3. Le modèle statique

Depuis SERENE pour calcule la fiabilité il faut calculer ou estimer la qualité du processus d'une phase de développement plus les résultats des testes. Et construire le Réseau Bayésien hiérarchisée selon les phases de cycle de vie, et qu'on peut exprimer le processus de chaque étape par un Réseau Bayésien spécial pour chaque étape dans le cycle de vie.

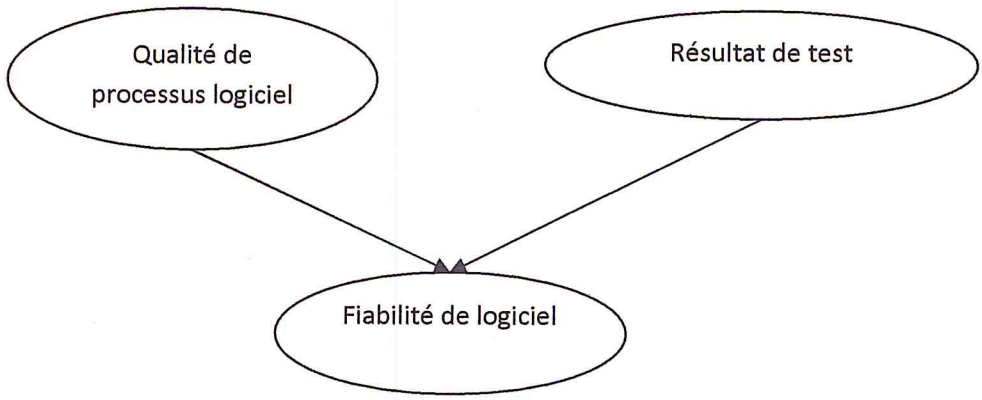



***Figure 33 :*** *RB pour le calcule de fiabilité selon SERENE.*

### 3.3.1. Cycle de vie

Le « **cycle de vie d'un logiciel** » désigne toutes les étapes du développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la **validation** du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la **vérification** du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre.

L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend généralement à minima les activités suivantes : Définition des objectifs, Analyse des besoins et faisabilité, Conception générale, Conception détaillée, Conception détaillée, Codage, Tests unitaires, Tests Intégration, Qualification, Documentation, Maintenance.

Depuis SERENE dans chaque étape on collecte des informations pour estimer la défaillance ou la qualité du processus.

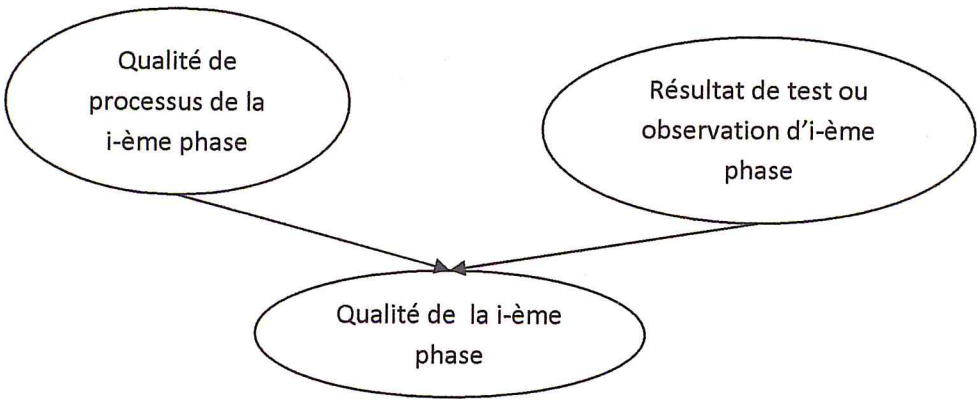



***Figure 34 :*** *Structure du RB pour chaque processus.*

### 3.3.1.1. Définition des objectifs :

Consistant à définir la finalité du projet et son inscription dans une stratégie globale, Le management étudie la stratégie et décide de la nécessité de fabriquer ou acheter un nouveau produit. On s'intéresse aux produits contenant du logiciel.

C'est pendant cette phase qu'est défini un schéma directeur dans le cas de la création ou de la rénovation d'un système d'information complet d'une entreprise prenant en compte la stratégie de l'entreprise.

On peut prendre ces informations l'estimer par une observation ou un jugement de réussit de l'étape de cycle de vie et son effet sur les autres étapes de cycle de vie

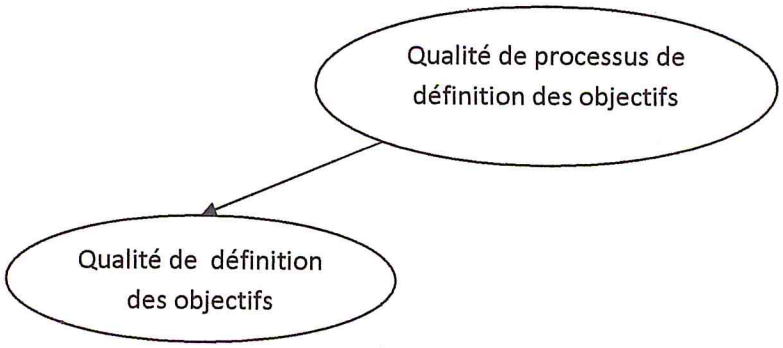



***Figure 35** : Qualité de définition des objectifs.*

Avec le nœud de « Qualité de processus de définition des objectifs » on estime le processus avec le travaille fait dans l'étape, la qualité des développeurs sur le logiciel a construit même des expériences passé, pour a la fin, on déduire la réussite de l'étape de définition des objectifs.

#### 3.3.1.2. Analyse des besoins et faisabilité :

C'est-à-dire l'expression, le recueil et la formalisation des besoins du demandeur (le client) et de l'ensemble des contraintes, Un cahier des charges est établi par le client après consultation des divers intervenants du projet (utilisateurs, encadrement...), un appel d'offres est éventuellement lancé.

Le cahier des charges décrit, en langage naturel, les fonctionnalités attendues du produit ainsi que les contraintes non fonctionnelles (temps de réponse, contraintes mémoire...). Dans le cas de la refonte d'un système complet on peut avoir un cahier des charges par sous domaine. Le produit intermédiaire obtenu à l'issue de cette phase est le **cahier des charges.**
On peut décrire le produit à partir de différents scénario d'utilisation (Use Case)

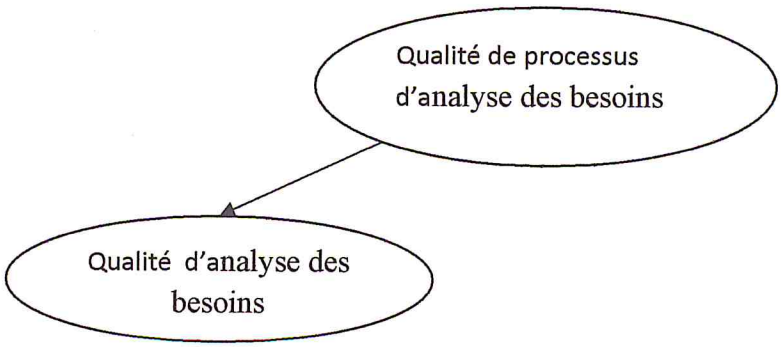



***Figure 36 :** Qualité d'analyse des besoins.*

Avec le nœud de « Qualité de processus d'analyse des besoins» on estime le processus avec le travaille fait dans l'étape, la qualité des informations disponible sur le logiciel a construit, la possibilité d'arriver a satisfait les besoins, même des expériences passe pour a la fin déduire la réussite de l'étape de d'analyse des besoins.

### 3.3.1.3. Conception générale :

Il s'agit de l'élaboration des spécifications de l'architecture générale du logiciel, dans cette phase l'architecture du logiciel est définie ainsi que les interfaces entre les différents modules. On veillera tout particulièrement à rendre les différents constituants du produit sont aussi indépendant que possible de manière à faciliter à la fois le développement parallèle et la maintenance future.

A l'issue de cette phase les produits intermédiaires sont

- **le dossier de conception**
- **le plan d'intégration**
- **les plans de tests**
- **le planning mis à jour**.

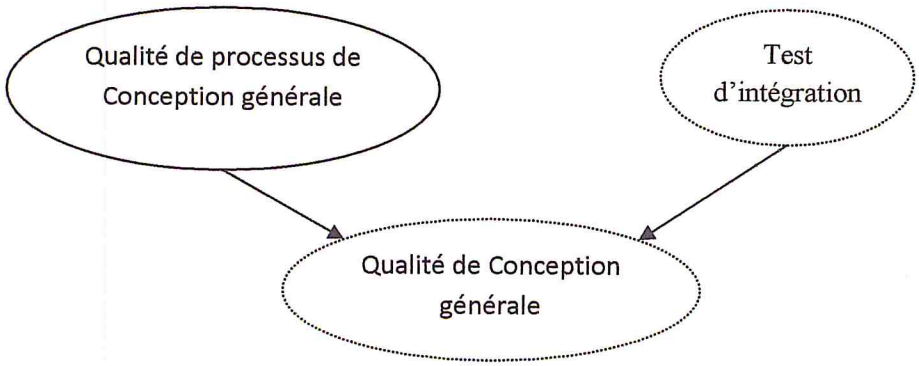



***Figure 37 :*** *Qualité de Conception générale.*

Le test d'intégration est un test de structure

- Pour les modèles est un test de flot de donnée
- Pour l'objet est test de couverture

La qualité des processus est défini par l'observation ou par l'ensemble d'objet qui le constitue et donc construire un Réseau Bayésien qui estime cette étape on étudie le passage

des donnée entre les interfaces, qu'ils sont eux aussi des modèles ou des classes dans le cas de l'oriente objet alors on peut appliquer des métriques sur chaque interface :

- Pour les modèles on utilise le métrique Flux d'informations d'Henry-Kafura.
- Pour l'objet on utilise les Métriques Objet de Chidamber et Kemerer.

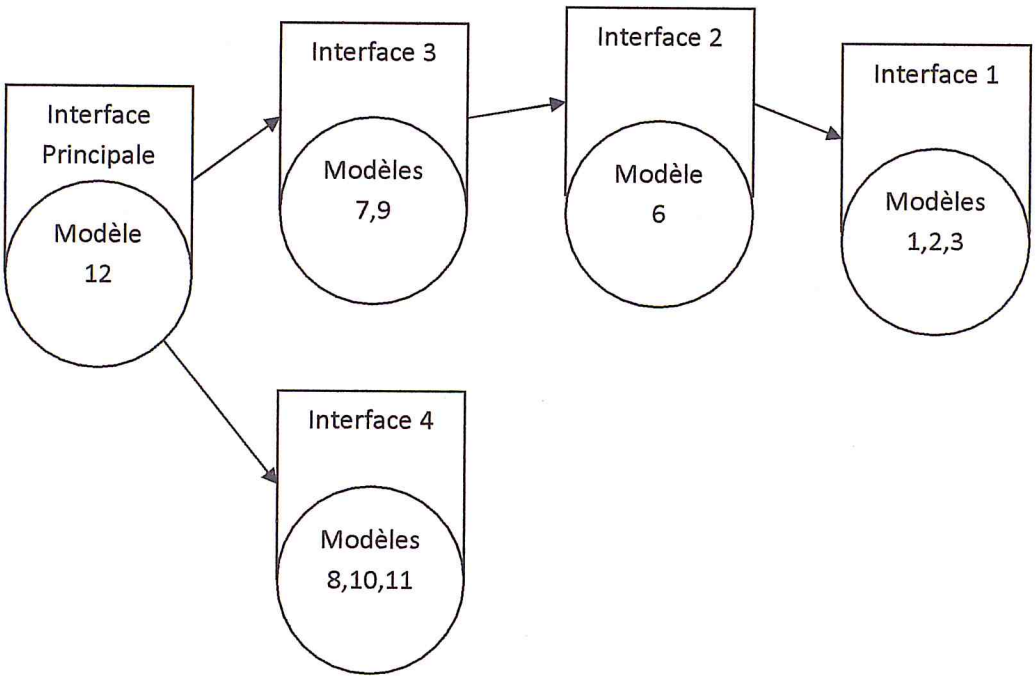



***Figure 38 :*** *Un ensemble d'interface modulaire.*

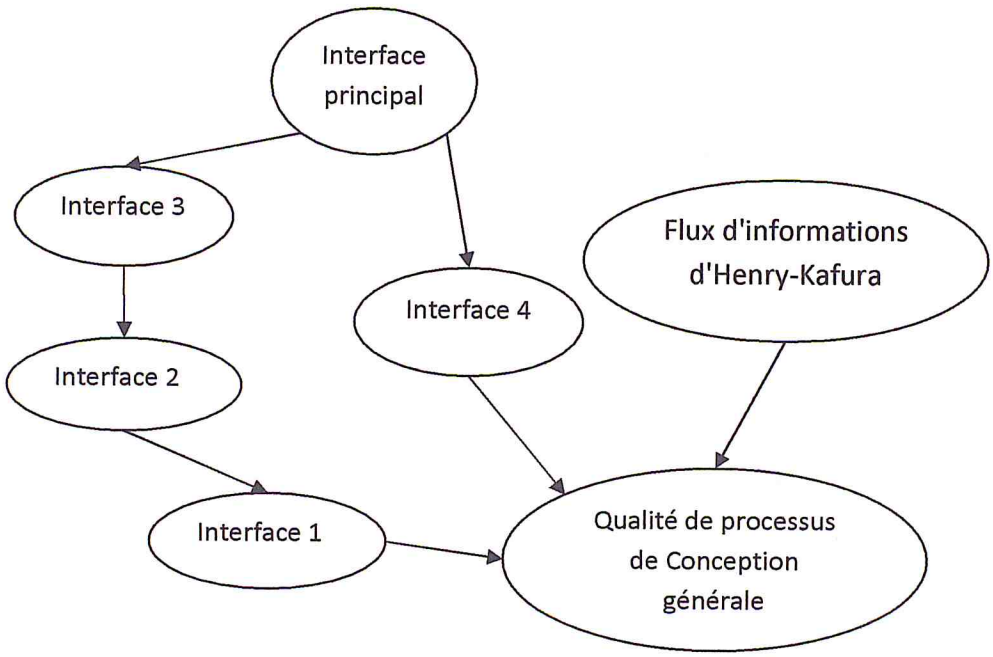



***Figure 39 :*** *Réseau Bayésien proposé pour l'interface modulaire.*

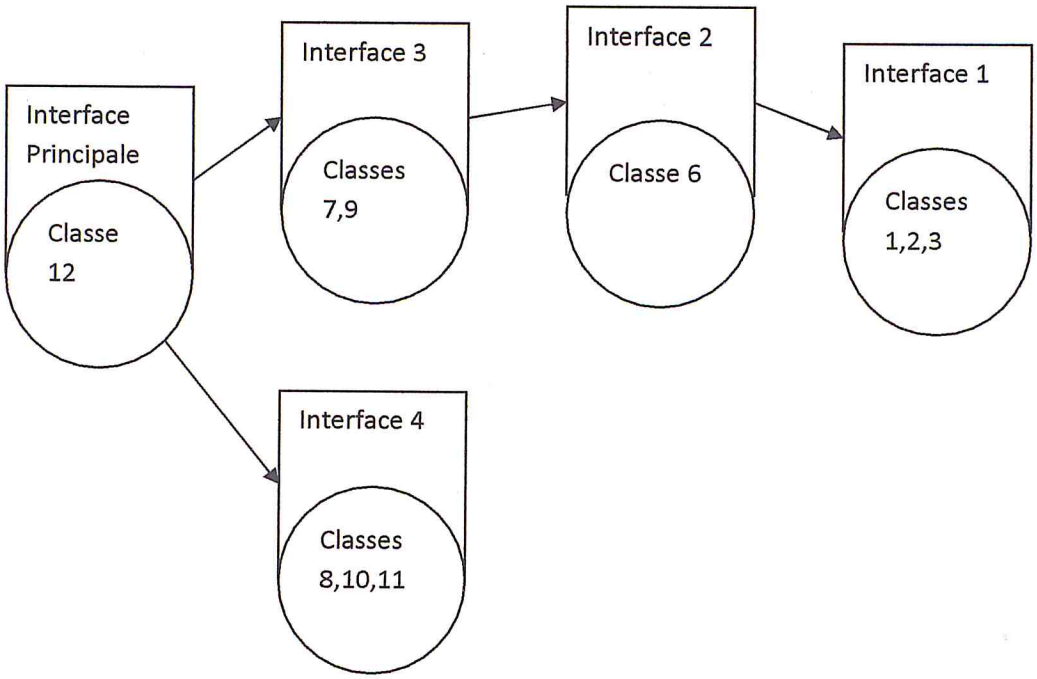



***Figure 40 :*** *Un ensemble d'interface objet.*

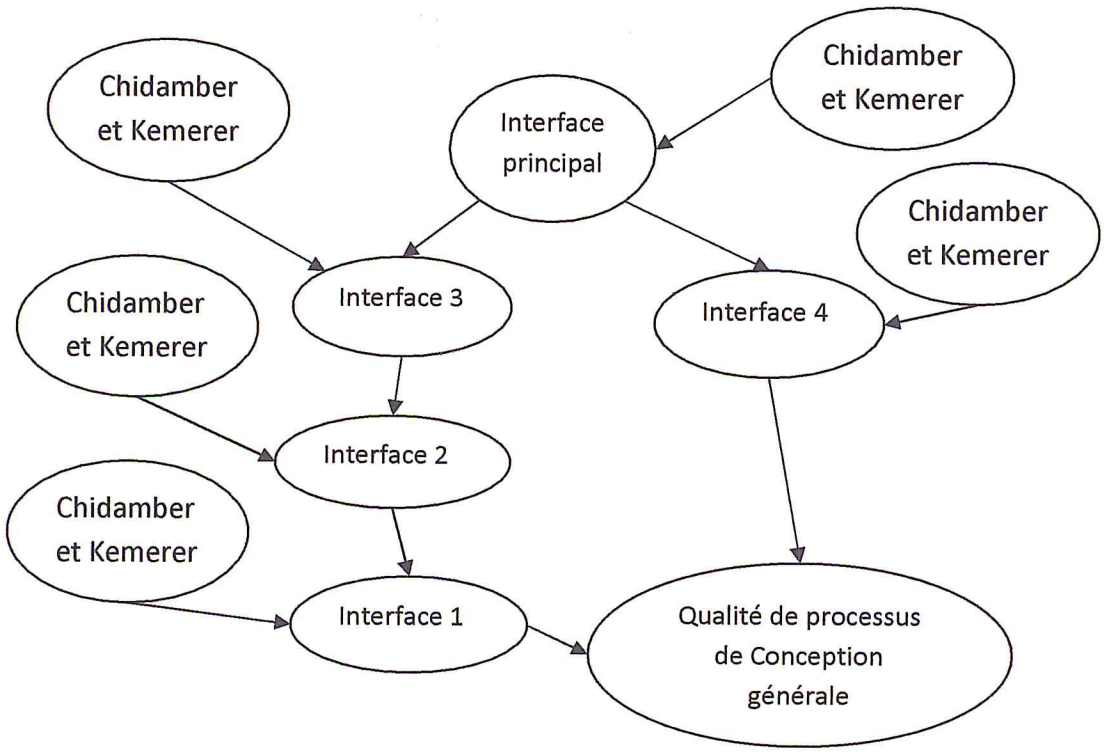



***Figure 41 :*** *Réseau Bayésien proposé pour l'interface objet.*

### 3.3.1.4. Conception détaillée :

La conception détaillée consiste à définir précisément chaque sous-ensemble du logiciel.

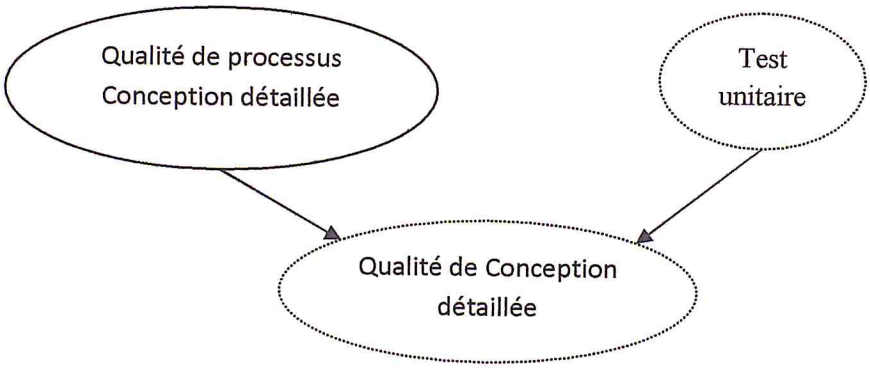



***Figure 42:*** *Qualité de Conception détaillée*

Le test unitaire est un test de structure

- Pour les modèles est un test de flot de donnée
- Pour l'objet est test de couverture

La qualité de processus est défini par l'observation ou par l'ensemble d'objet qui construit et donc de construit un Réseau Bayésien qui estime l'étape on étudie le passage des donnée entre les modèle ou les objets, Donc on peut appliquer les métriques :

- Pour les modèles on utilise le métrique Flux d'informations d'Henry-Kafura
- Pour l'objet on utilise les Métriques Objet de Chidamber et Kemerer est on peut aussi ajouter les métrique de MOOD

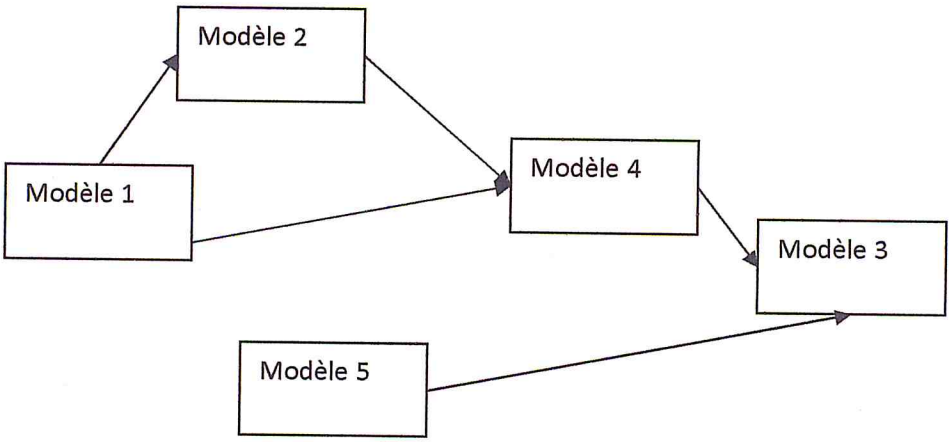



***Figure 43 :*** *Un ensemble d'interface modulaire.*

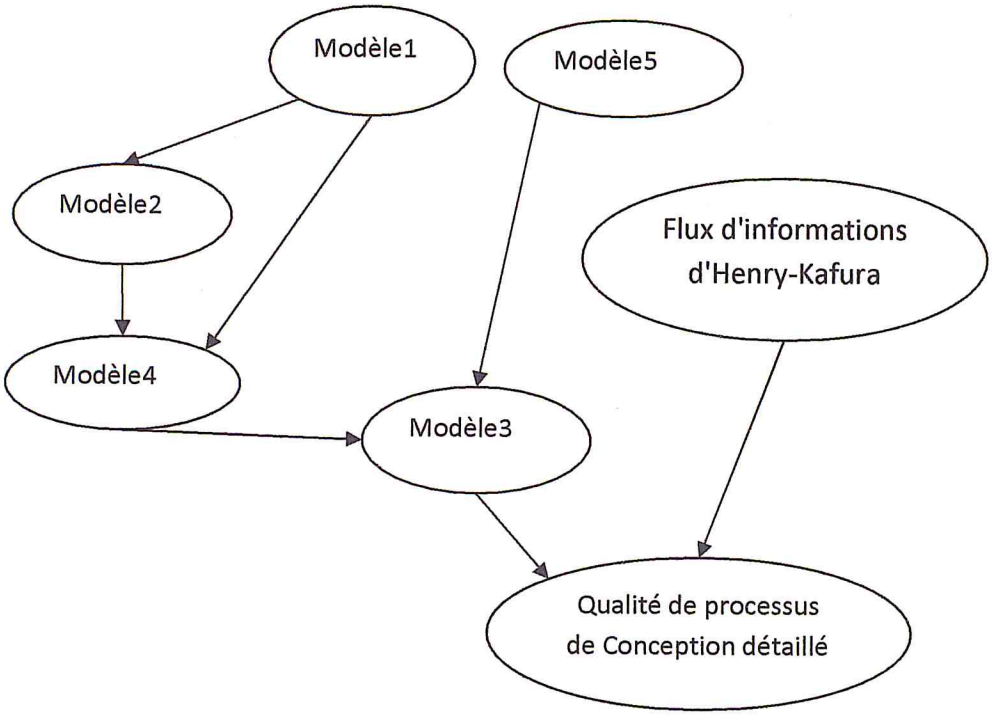



***Figure 44 :*** *Réseau Bayésien proposé pour l'interface modulaire.*

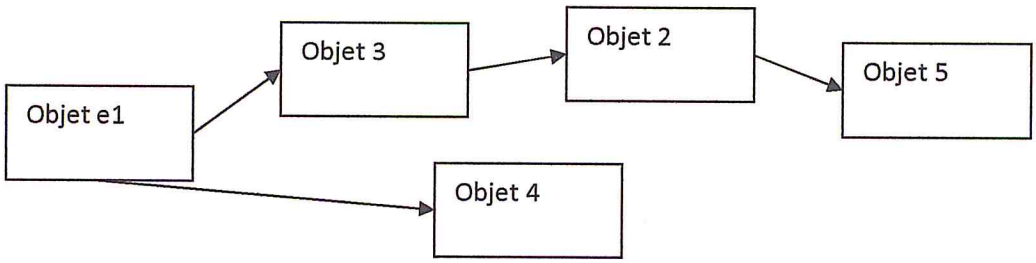



***Figure 45 :*** *Un ensemble d'interface objet.*

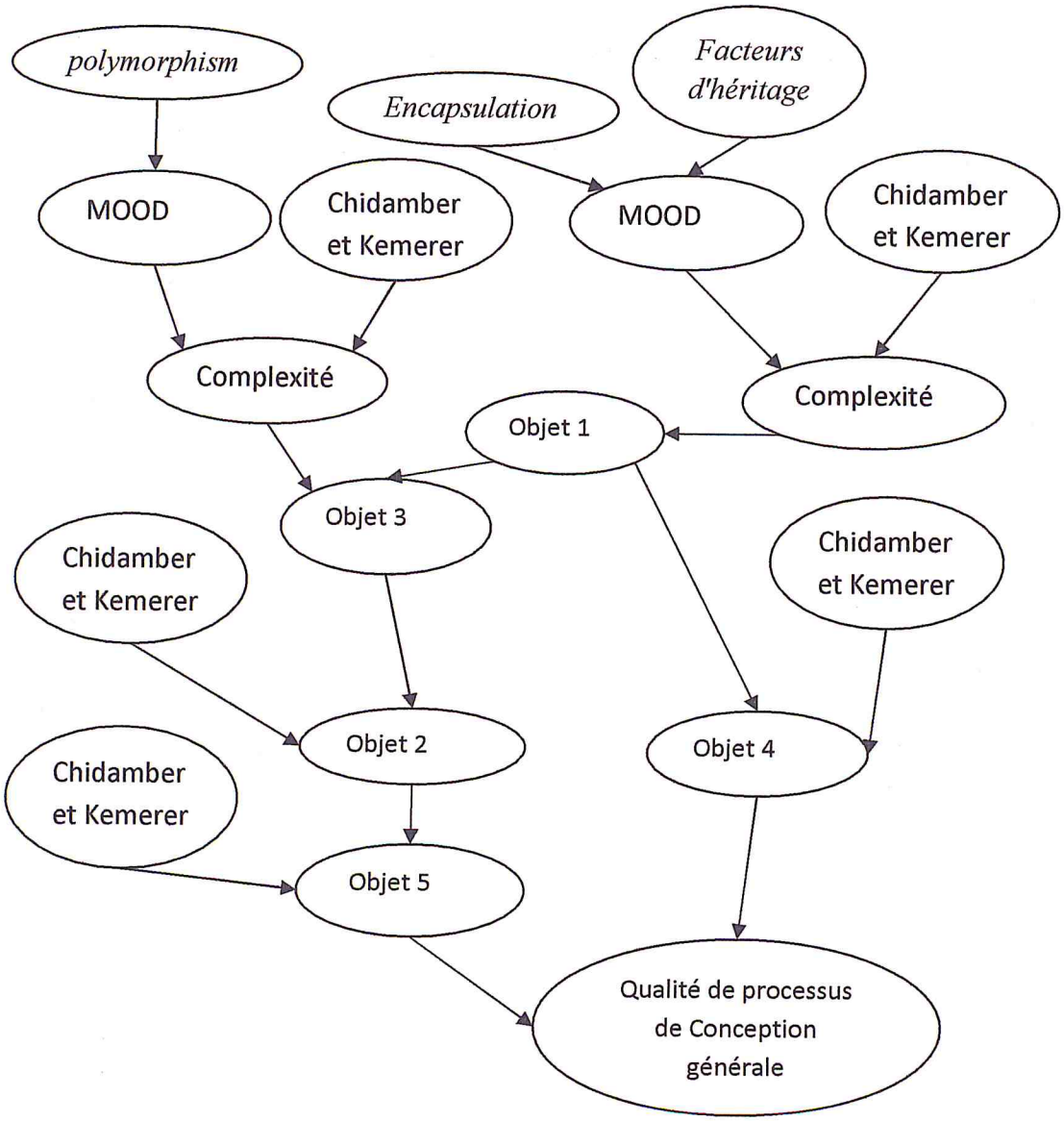



***Figure 46 :*** *Réseau Bayésien proposé pour l'interface objet.*

### 3.3.1.5. Codage (Implémentation ou programmation)

Le Codage : soit la traduction dans un langage de programmation des fonctionnalités définies lors de phases de conception.

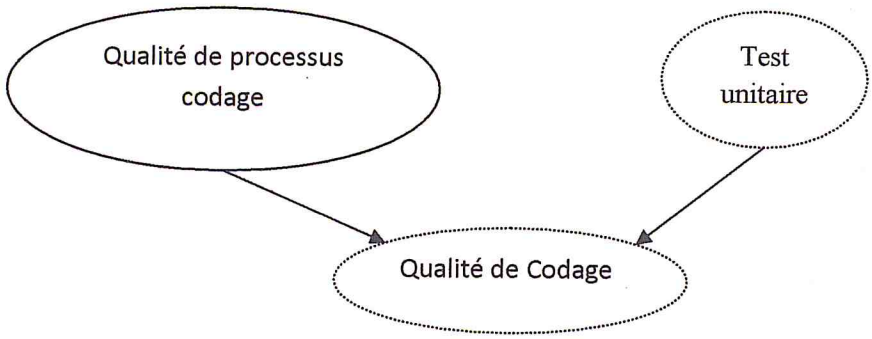



***Figure 47 :*** *Qualité de Codage.*

Le test unitaire est un test de structure

- Pour les modèles est un test de flot de donnée
- Pour l'objet est test de couverture

La qualité de processus est défini par l'observation ou par l'ensemble d'objet qui construit et donc de construit un Réseau Bayésien qui estime l'étape on étudie le passage des donnée entre les fonctions

Alors on peut appliquer les métriques :

- par calcule la complexité structurel de McCabe
- par la complexité de science informatique de Halstead

Et on utilise de test fonctionnel ou une matrice de test pour chaque fonction

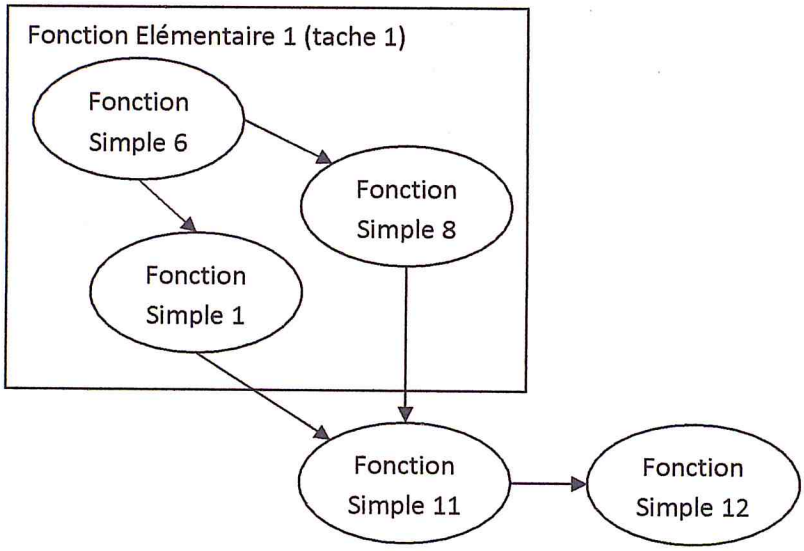



***Figure 48 :*** *Schéma d'exécution d'une tache.*

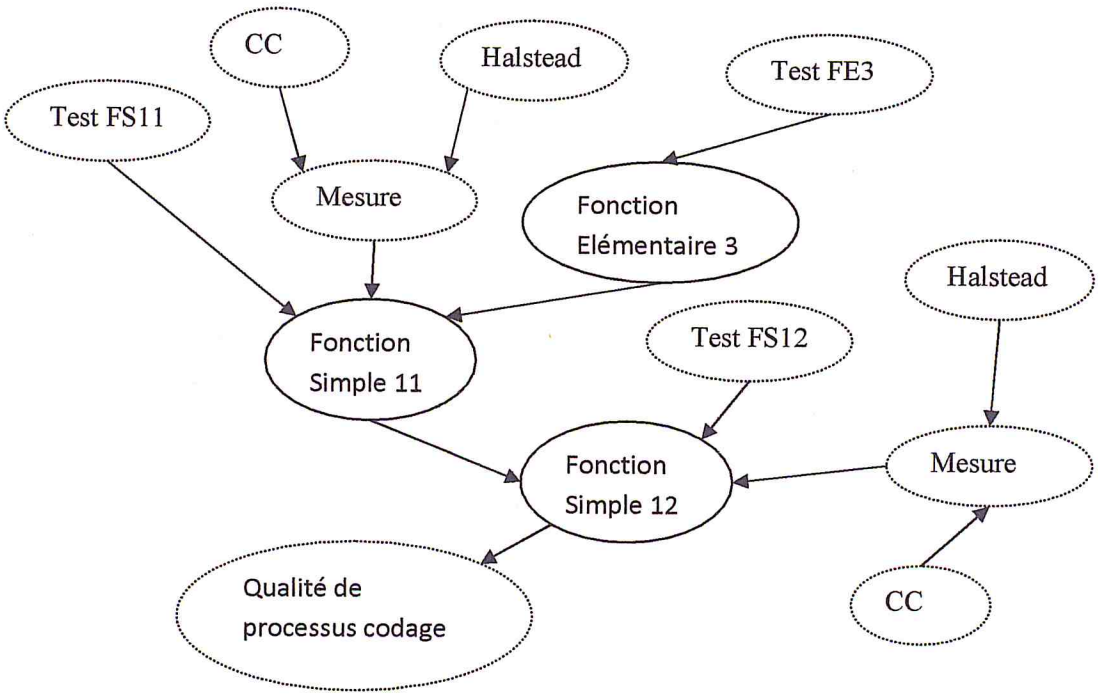



***Figure 49 :*** *Réseau Bayésien de codage.*

### 3.3.1.6. Tests unitaires :

Permettant de vérifier individuellement que chaque sous-ensemble du logiciel est implémenté conformément aux spécifications.

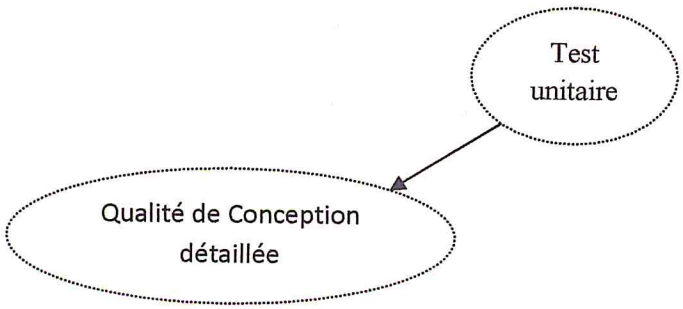



***Figure 50 :*** *Réseau Bayésien de Test unitaire.*

### 3.3.1.7. Tests Intégration

Dont l'objectif est de s'assurer de l'interfaçage des différents éléments (modules) du logiciel. Elle fait l'objet de *tests d'intégration* consignés dans un document.

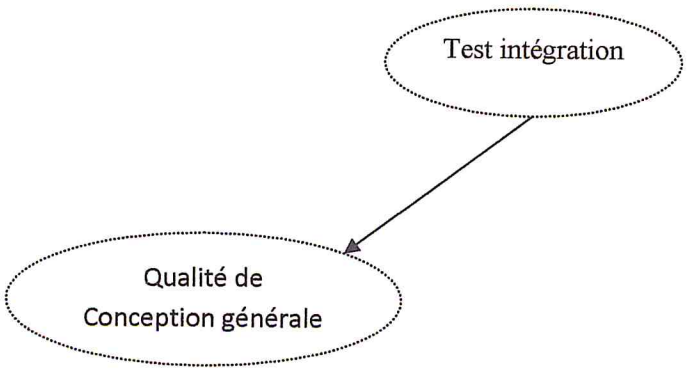



***Figure 51 :*** *Réseau Bayésien de Test intégration.*

### 3.3.1.8. Qualification :

Lorsque le logiciel est terminé et les phases d'intégration matériel/logiciel achevées, le produit est qualifié, c'est à dire testé en vraie grandeur dans des conditions normales d'utilisation. Cette phase termine le développement. à l'issue de cette phase le logiciel est prêt à la mise en exploitation.

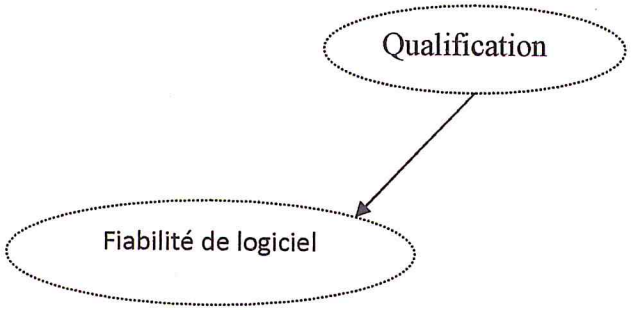



***Figure 52 :*** *Réseau Bayésien de Qualification.*

On utilise pour la qualification le test du logiciel jusqu'à atteindre un niveau de satisfaction, ce test initialise a nous la durée périodique de la valeur estimer de fiabilité de logiciel

#### 3.3.1.9. Documentation

La documentation visant à produire les informations nécessaires pour l'utilisation du logiciel et pour des développements ultérieurs elle possède même tous les informations de maintenance et de test et donc une référence pour la construction du logiciel.

#### 3.3.1.10. Maintenance

La maintenance comprenant toutes les actions correctives (maintenance corrective) et évolutives (maintenance évolutive) sur le logiciel.

##### 3.3.1.10.1. Maintenance corrective :

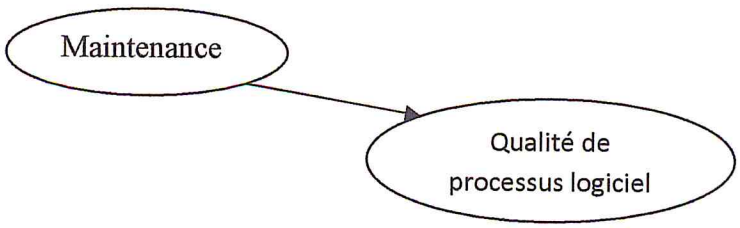



***Figure 53 :*** *Réseau Bayésien de Maintenance corrective.*

### 3.3.1.10.2. Maintenance évolutive :

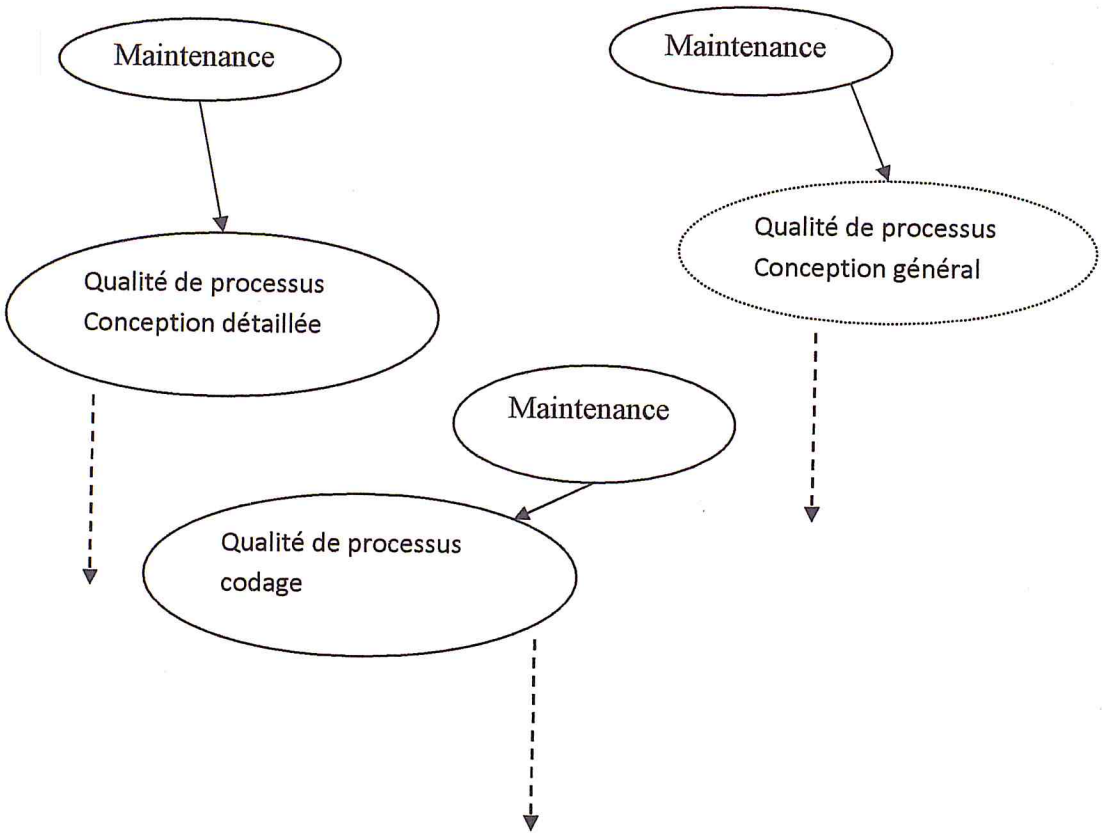



***Figure 54 :*** *Réseau Bayésien de Maintenance évolutive.*

*Remarque :*

Il y a autres phases mais nous n'intéressons pas à eux, car ils n'ont pas d'informations calculables.

### 3.3.2. Modèle de cycle de vie

La séquence et la présence de chacune de ces activités dans le cycle de vie dépend du choix d'un modèle de cycle de vie entre le client et l'équipe de développement.

On va faire rappel des modèles le plus utilise :

#### 3.3.2.1. Modèle en cascade

Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante [7]:

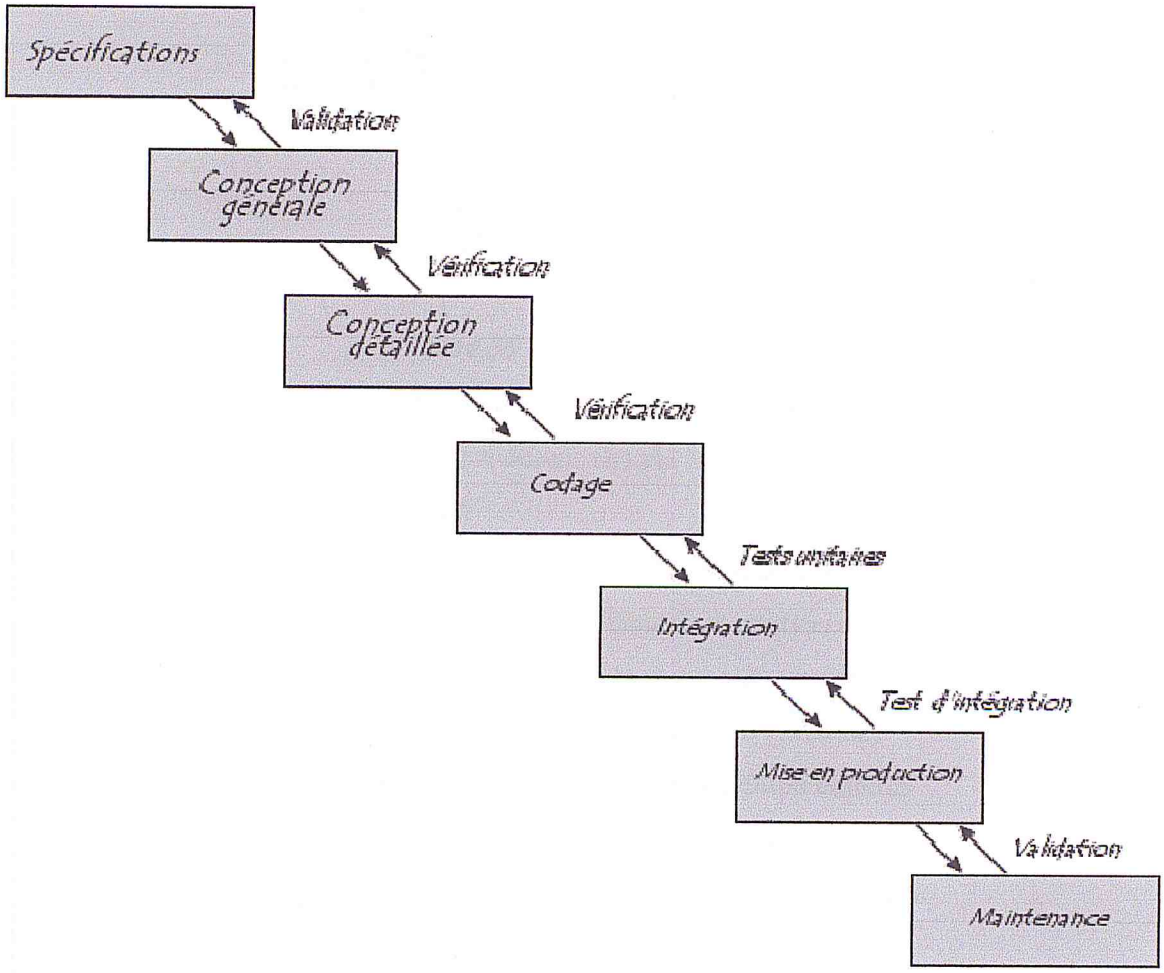



***Figure 55 :*** *Modèle en cascade [7].*

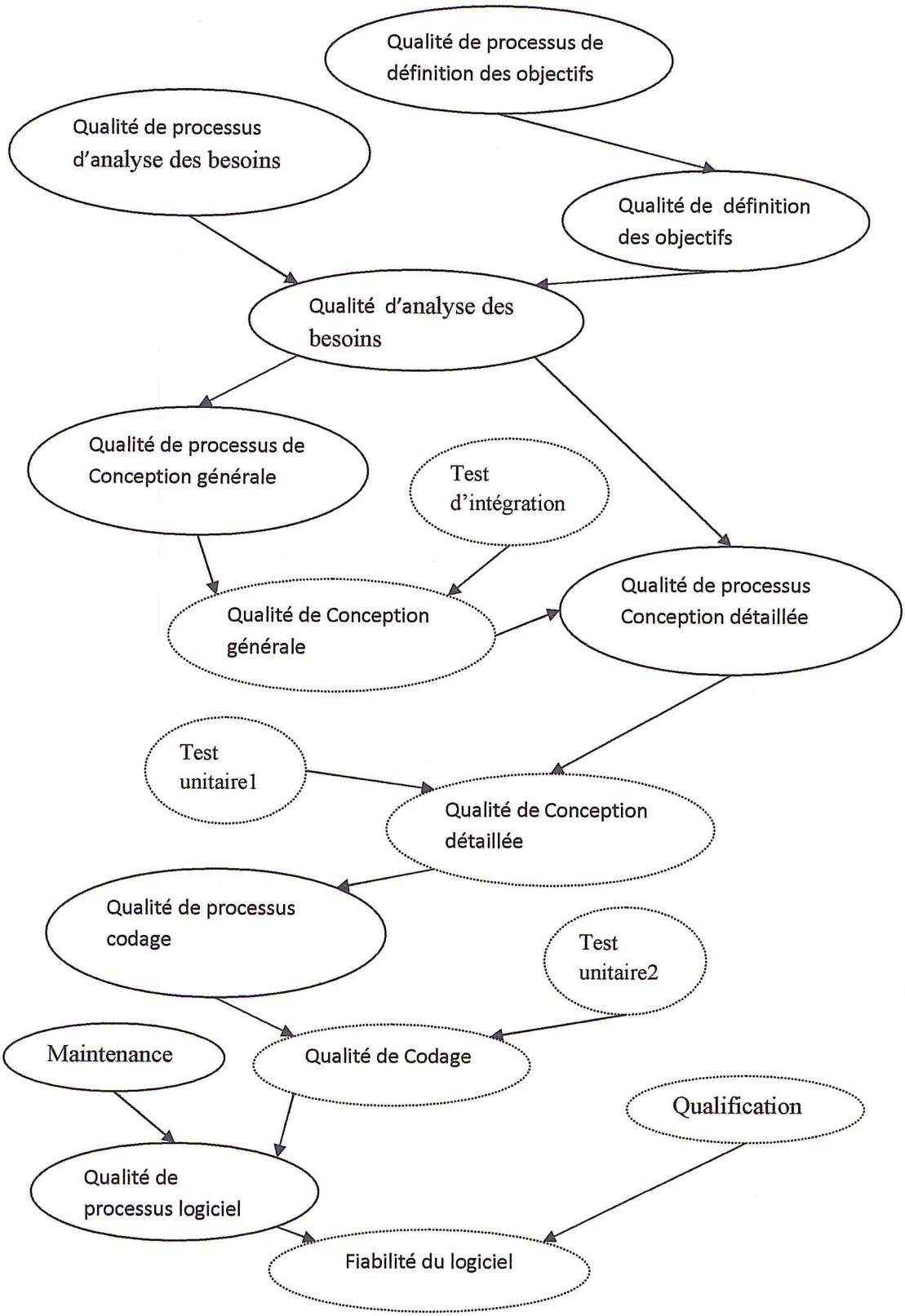



***Figure 57 :*** *Un Réseau Bayésien global.*

Des Réseaux Bayésiens sont dits équivalents au sens de Markov s'ils représentent la même série d'indépendances conditionnelles.

***Apprentissage d'un Réseau Bayésien***

Dans notre cas nous présentons le fonctionnement élémentaire d'une méthode basée sur des tests d'indépendance conditionnelle, qui s'appelle « Algorithme PC ».

*Cet algorithme se décompose en trois phases [14]:*

1. Recherche d'indépendances conditionnelles dans les données en utilisant des tests statistiques.
2. Utilisation des indépendances découvertes pour former un graphe sans circuit partiellement dirigé (PDAG), en deux étapes :
   - les arêtes X–Y d'un graphe non dirigé complètement connecté sont supprimées pour chaque indépendance découverte ($X \perp\!\!\!\perp Y$).
   - le graphe non dirigé obtenu est ensuite partiellement dirigé grâce aux indépendances conditionnelles découvertes (elles correspondent à des V-structures $X \rightarrow Y \leftarrow Z$).
3. le PDAG obtenu est ensuite "complété" (CPDAG) grâce aux règles suivantes. Ce CPDAG est le représentant de tous les graphes équivalents au sens de Markov. Les arcs ajoutés lors de cette phase seront appelés par la suite arcs inférés
   - si $X \rightarrow Y$, avec Z adjacent à Y mais pas à X, et que Y –Z n'est pas orienté, alors il faut l'orienter en $Y \rightarrow Z$.
   - s'il existe un chemin dirigé de X vers Y mais que X–Y n'est pas orienté, alors il faut l'orienter en $X \rightarrow Y$ (pour ne pas créer de cycle).

Appelons $\in$ l'espace des CPDAG, i.e. l'espace des représentants des classes d'équivalence de Markov. A l'issue des trois phases, les arcs d'un CPDAG de $\in$ sont issus, soit de la découverte de V-structures (phase2), soit "inférés" lors du passage du PDAG au CPDAG (phase 3) [14].

Afin d'obtenir un RB, il faut finir d'orienter le CPDAG obtenu, sans créer de circuit ou de nouvelle V-structure . Cela revient à choisir un des DAG de la classe d'équivalence dont nous venons de trouver le représentant à partir de données.

Dans le cas des Réseaux Bayésiens causaux, le choix du DAG dans la classe d'équivalence représentée par le CPDAG précédent ne peut plus être effectué de la même manière. En effet, un seul DAG de cette classe représentera le mécanisme causal sous-jacent. Le choix de ce DAG ne peut plus se faire de façon arbitraire comme dans l'algorithme proposé par

Dor and Tarsi[4], en 1992. Nous ne pouvons plus d'ailleurs utiliser les données d'observation ayant servi à l'apprentissage du CPDAG puisque toute l'information contenue dans ces données a déjà été extraite pour découvrir le plus d'arcs possibles et arriver à ce CPDAG [14].

Nous proposons donc de rajouter une nouvelle phase consistant en une série d'expériences à effectuer sur le système. Ces expériences permettront d'obtenir de nouvelles informations afin de finir d'orienter notre graphe [14].

Supposons que nous ayons trouvé que X et Y sont reliés. Dans ce cas il est impossible de décider si X est une cause de Y ou l'inverse. Afin d'obtenir la structure causale correcte, nous devons pouvoir manipuler une des variables et puis observer les répercussions éventuelles sur les autres variables. Par exemple, si nous manipulons X et que nous observons un changement de Y, nous savons que la structure causale est X → Y. De plus, l'ordre dans lequel ces expériences sont exécutées peut influencer la rapidité de l'algorithme d'apprentissage [14].

D'autre part, il est également possible que les résultats de l'exécution d'une expérience sur une variable puissent se "propager" en orientant d'autres arcs "inférés", toujours dans le but de ne pas créer de V-structure qui aurait du être découverte lors de la construction du CPDAG.

***Du CPDAG au Réseau Bayésien causal :***

Il y a plusieurs algorithmes dont le but est d'indiquer quelle est la série d'expériences à réaliser pour obtenir un Réseau Bayésien causal à partir du CPDAG généré par un algorithme d'apprentissage de structure classique [14].

1. ***Ordre basé sur la connectivité :***

   Une première méthode consiste à assigner à chaque nœud du graphe une importance correspondant à la quantité d'information que l'on obtiendra en le manipulant.

   Notre but étant d'orienter toutes les arêtes X–Y, il semble logique de commencer par manipuler la variable X qui possède le plus d'arêtes de ce type, et donc le plus de voisins "non dirigés" dans le graphe (i.e. qui ne sont ni ses parents ni ses enfants).

   La fonction d'évaluation peut donc s'écrire :

   $Eval_1(x) = \#(N_{eU}(x))$

4 *Dor, D. and Tarsi, M. (1992). A simple algorithm to construct a consistent extension of a partially oriented graph. Technical report, Cognitive Systems Laboratory, UCLA Computer Science Department.*

2. ***Ordre basé sur le coût d'intervention***

Toutes les expériences n'ont pas forcément le même coût, certaines sont même impossibles à réaliser. Pour cette raison, il est judicieux d'associer un coût à chaque expérience (une expérience impossible ayant un coût infini (∞)).

Cette nouvelle fonction d'évaluation est donc :

$$Eval_2(x) = \frac{\#(N_{eU}(x))}{Cost(x)}$$

***Algorithme : Apprentissage de la structure d'un Réseau Bayésien causal [14]***

- ***Entrée***: un ensemble de données d'observation, une table Cost(X) indiquant le coût de manipulation de chaque variable X.
- ***Sortie***: le CPDAG représentant de la classe d'équivalence, et une séquence d'expériences à exécuter pour finir de déterminer la structure du Réseau Bayésien causal

1. Appliquer l'algorithme PC avec les données d'observation pour obtenir le CPDAG G.
2. Calculer $Eval(x)$ pour chaque nœud X tel que

$$\#(N_{eU}(x)) > 0 \text{ Dans G}$$

$$\text{Et } Eval(x) = \frac{\#(N_{eU}(x))}{Cost(x)}$$

3. Ajouter à la séquence des expériences le nœud $X_{High}$ qui maximise $Eval(x)$ .
4. Marquer toutes les arêtes connectées à $X_{High}$ comme *traitées*.
5. Recalculer $Eval(x)$ pour chaque nœud X tel que

$$\#(N_{eU}(x)) > 0 \text{ Dans G}$$

$$\text{Et } Eval(x) = \frac{\#(N_{eU}(x))}{Cost(x)}$$

On ne considérant plus les arêtes traitées dans le calcul du voisinage.

6. Retourner en (3) tant qu'il existe des arêtes non traitées.
7. Rendre le CPDAG G obtenu en (1) et la séquence d'expériences à réaliser.

Bien sur dans notre projet on appliquer cette algorithme dans le Réseau Bayésien global qui représente le cycle de vie du logiciel, les effets ou les relations de causalité entre les différentes étapes de cycle de vie est partialement connue –avis d'experts et observations-.

Exemple : Application à notre cas d'étude :

Dans l'étape de codage et conception, le test unitaire est l'utile pour savoir le taux de défaillance alors le test unitaire est une cause pour la conséquence de défaillance du codage.

Même chose dans les sous-Réseaux Bayésiens dans l'étape de codage en trouve des fonctions comme ensemble d'objet a étudie, dans notre projet on collecte tous les schéma

d'exécution chacun représentera une tache particulier dans le logiciel, puis déterminer les fonctions qui exécute dans une tache on sort par un Réseau Bayésien qui donne le taux de défaillance de cette tache mais comment on le construit ?, tous simplement ; car la relation de casualité entre une fonction et une autre définie par si la fonction 1 utilise les sorties des fonctions 2 et si il y a un erreur dans la fonction 2, il passe automatiquement à la fonction 1 donc la cause de défaillance de fonction 1 est la fonction 2 .

Alors on peut utilise l'algorithme ou simplement construire le Réseau Bayésien de façon logique depuis un expert.

## 3.5. Estimation de la fiabilité d'un logiciel

Après la construction du Réseau Bayésien on commence par entrer les évidences comme les tests et les mesures calculé ou l'estimation d'un nœud à partir du Réseau Bayésien est caché dans la TPC. Les RB sont généralement employés pour modéliser des connaissances incertaines car ils permettent de représenter et de quantifier les relations causales entre les variables d'un système par un formalisme combinant la théorie des probabilités et une représentation par graphes.

Un RB est représenté par un graphe orienté acyclique dans lequel chaque nœud modélise une variable du système et chaque arc une relation de dépendance ou de cause à effets liant ces variables est une TPC.

Une variable aléatoire discrète modélisée par un nœud $n$ est décrite par un nombre fini d'état $S_n$: $\{S_1^n, \dots, S_k^n, \dots, S_K^n\}$ mutuellement exclusifs. A chaque nœud du Réseau est associée une TPC Celle-ci détermine les dépendances entre les différents états des variables aléatoires par des probabilités conditionnelles. Pour chaque nœud, ces probabilités peuvent être définies soit a priori, soit par apprentissage. La distribution associée à un nœud sans parent peut être décrite par le vecteur [13]

$$Dist(n) = [\Pr(n = S_1^n), \dots, \Pr(n = S_K^n)]$$

La TPC associée à un nœud $n$ est obtenue à partir de la distribution des probabilités sur chacun de ses états connaissant les états de ses nœuds parents, notés *Pa(n)*.

$$\begin{matrix} \Pr\left(n = S_1^n | Pa(n) = S_1^{Pa(n)}\right) & \dots & \Pr\left(n = S_1^n | Pa(n) = S_K^{Pa(n)}\right) \\ \dots & \dots & \dots \\ \Pr\left(n = S_L^n | Pa(n) = S_1^{Pa(n)}\right) & \dots & \Pr\left(n = S_L^n | Pa(n) = S_K^{Pa(n)}\right) \end{matrix}$$

Tout calcul de probabilités dans un RB relève de l'inférence. Dans ce mémoire, nous ne décrirons pas les différents algorithmes existants, aussi nous invitons le lecteur à se référer, pour la description des mécanismes d'inférence des RB. De manière simple, il est possible de calculer par le théorème des probabilités totales, les probabilités marginales de $n$ à partir de la distribution de probabilités de ses parents [13].

$$\Pr(n = S_1^n) = \Pr\left(n = S_1^n | Pa(n) = S_1^{Pa(n)}\right) * \Pr\left(Pa(n) = S_1^{Pa(n)}\right) + \cdots$$

$$\ldots + \Pr\left(n = S_1^n | Pa(n) = S_K^{Pa(n)}\right) * \Pr\left(Pa(n) = S_K^{Pa(n)}\right).$$

Il est également possible de réaliser l'opération inverse grâce au théorème de Bayes, on peut alors calculer :

$$\Pr\left(Pa(n) = S_1^{Pa(n)} | n = S_1^n\right) = \frac{\Pr\left(n = S_1^n | Pa(n) = S_1^{Pa(n)}\right) * \Pr\left(Pa(n) = S_1^{Pa(n)}\right)}{\Pr(n = S_1^n)}$$

Les RB permettent l'estimation de la distribution de probabilités sur l'ensemble des états d'une variable en fonction de l'observation des états (évidence) d'une ou de plusieurs autres variables. De cette façon, les probabilités associées aux états d'un nœud sont calculées par propagation des évidences. Un nœud possède plusieurs statuts S [13].

Exemple :

Pr (Somking=yes)=12% ;

Pr (Somking=no)=88% ;

***Tableau 6 :*** *Exemple de TPC.*

| | Smoking=yes | Smoking=no |
|---|---|---|
| Cancer=yes | 0.88 | 0.12 |
| Cancer=no | 0.09 | 0.91 |

On a la Somme de

Pr(Cancer=yes|Smoking=yes)+ Pr(Cancer=yes|Smoking=no)=1

Pr(Cancer=no|Smoking=yes)+ Pr(Cancer=no|Smoking=no)=1

Exemple :

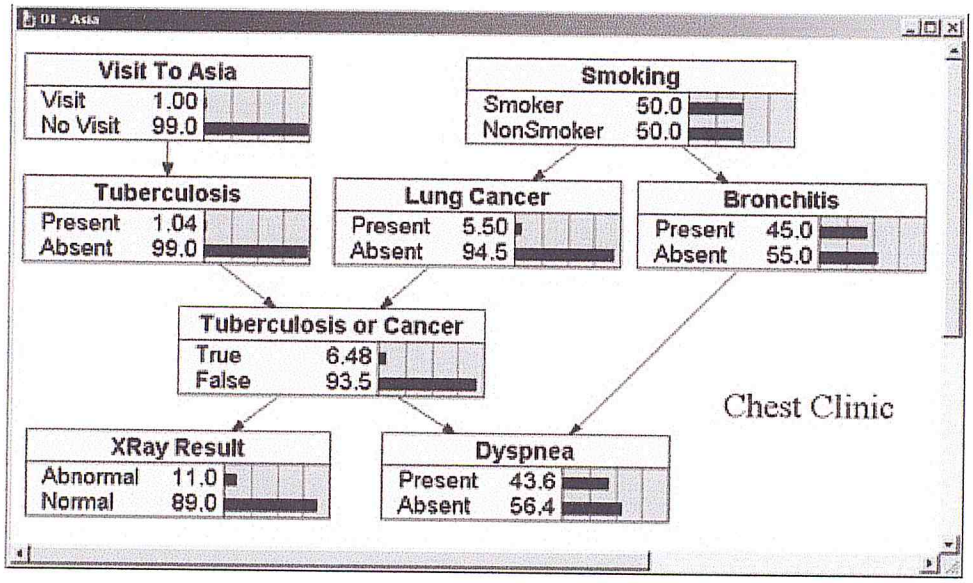



***Figure 58 :*** *Réseau Bayésien pour l'exemple Smoking.*

## 3.6. Prédiction de la fiabilité d'un logiciel

Parfois la seule raison de croire un peu à la fiabilité du logiciel. Quelle est sa valeur prédictive ? Est-ce qu'il prouve vraiment quelque chose quant à la fiabilité du système ?

Une façon ironique de poser le problème consiste à énoncer la règle des cinq minutes. On pourrait aussi énoncer la règle de la première fois : si un logiciel fonctionne bien la première fois, c'est qu'il fonctionnera bien toujours.

Il est évident que ces règles ne fournissent pas toujours de bonnes prédictions : certains édifices s'effondrent quelque temps après avoir été construits, des machines qui fonctionnaient bien cessent de fonctionner. Quel sens alors faut-il donner à la règle de l'essai préalable ? Doit-on conclure qu'un essai ne sert à rien ? Bien sûr que non. Pourquoi ?

Si on veut des prédictions infaillibles, alors oui, les essais ne servent en général à rien. Ils ne permettent que rarement de faire des prédictions infaillibles, et encore seulement si on est indulgent sur la nature de l'infaillibilité qu'ils produisent.

Quand des vies humaines sont en jeu on pourrait croire que l'infaillibilité est le problème majeur des études de fiabilité. Dans certains cas particuliers oui mais pas en général. Ce serait nous demander beaucoup trop. Même quand des vies humaines sont en jeu, on ne peut que rarement faire des prédictions infaillibles. Tout le monde le sait bien. On prend un risque à chaque fois qu'on fait un pas sur un trottoir. Il serait absurde d'exiger l'infaillibilité des prédictions. Ce serait nous condamner à ne plus rien faire ou presque.

Le problème général des études de fiabilité ce n'est pas l'infaillibilité, c'est de réduire les risques. On sait qu'il y a des risques mais on veut faire les bons choix : parmi toutes les solutions réalisables quelles sont les plus risquées, les plus dangereuses, les moins fiables ?

Du point de vue de l'infaillibilité la règle de l'essai préalable est insensée. Mais du point de vue de la réduction des risques, son intérêt est évident : on prend beaucoup moins de risques si on fait un essai au préalable que si on n'en fait pas.

Bien que fausse la règle de la première fois est une bonne règle de réduction des risques. Cet intérêt de la fausseté est bien connu des scientifiques. Même quand elles sont très fausses les théories peuvent être très importantes parce qu'elles fournissent quelques bonnes prédictions et parce qu'elles sont des points de départ pour construire d'autres théories qui elles sont vraies, ou moins fausses. Une théorie peut être fausse et fournir des prédictions vraies parce qu'il suffit en toute rigueur d'une seule prédiction fausse pour que toute la théorie soit considérée comme fausse (mais ce principe n'est jamais appliqué sans précautions).

Une très ancienne tradition philosophique fait de l'infaillibilité la marque de la science. La géométrie euclidienne était citée en exemple. Pour les positivistes conséquents, les choses sont très différentes. Le problème c'est d'avoir des connaissances qui permettent de réduire les risques. Même les théories fausses, même les théories d'ordre zéro, sont des connaissances, si elles permettent de réduire les risques, si en moyenne quelqu'un réussit mieux avec elles que sans elles. Pour un positiviste, même la règle de la première fois est déjà un peu vraie. Elle est même très vraie si on considère tous les services qu'elle rend.

### 3.6.1. Avantages

La prédiction de l'évolution de la fiabilité logicielle est utile à plusieurs titres.

- **Informer** le public et les professionnels de la fiabilité de l'évolution future d'un logiciel ;
- **Aider** les décideurs à prévoir des modifications nécessaires à l'amélioration et à la prise en charge diagnostique des besoins des clients.
- Servir de base de **comparaison** avec l'évolution réelle future afin d'évaluer les traitements et les actions de prévention contre éventuelle défaillances.

### 3.6.2. Objectif

– Estimer l'incidence et le nombre de cas incidents futurs pour les défaillances dans le logiciel

– Tester la validité d'une méthode d'analyse fondée sur une approche Bayésienne.

Quand on veut savoir si un logiciel est fiable, la première chose à faire, quand on peut, c'est de l'essayer au préalable. On veut voir que ça marche. Cet essai préliminaire est même

## 3.7. Conception

### 3.7.1. Identification des acteurs

Nous allons maintenant énumérer les acteurs susceptibles d'interagir avec le système, Les acteurs du système identifiés dans un premier temps sont :

- Les développeurs.

### 3.7.2. Identification des messages

On va détailler les différents messages échangés entre le système et l'extérieur.

**Le système émis les messages suivants :**

- Afficher Réseau Bayésien de logiciel.

**Le système reçoit les messages suivants** :

- Création d'un logiciel.
- Création d'un cycle de vie.
- Création d'une phase.
- Affectation des différentes interfaces.
- Affectation des différentes modèles / objets.
- Affectation des différentes fonctions.
- Création de dépendances conditionnelles.
- Affectation des métriques.
- Affectation des tests.
- Affectation des tables de probabilité conditionnelles.

### 3.7.3. Capture des Besoins Fonctionnels :

Cette phase représente un point de vue « fonctionnel » de l'architecture système. Par le biais des cas d'utilisation, nous serons en contact permanent avec les acteurs du système en vue de définir les limites de celui-ci, et ainsi éviter de trop s'éloigner des besoins réels de l'utilisateur final.

### 3.7.3.1. Déterminer les cas d'utilisations

L'identification des cas d'utilisation une première fois établie, nous aurons un aperçu des fonctionnalités futures que doit implémenter le système.

Pour constituer les cas d'utilisation, il faut considérer l'intention fonctionnelle de l'acteur par rapport au système dans le cadre de l'émission ou de la réception de chaque message. En regroupant les intentions fonctionnelles en unités cohérentes, on obtient les cas d'utilisations.

Ce tableau (***Tableau 8)*** défini les cas d'utilisation, ses acteurs et les messages qu'on a besoin pour exécuter ou valider la tâche

***Tableau 7 :*** *Identification des cas d'utilisation, ses acteurs et leurs messages.*

| Cas d'utilisation | Les acteurs | Les messages de système et de l'acteur |
|---|---|---|
| Crée un logiciel | **Développeur** | Vers le Système :<br>• Création d'un logiciel.<br>• Création d'un cycle de vie. |
| Crée la population des donnée a étudie. | **Développeur** | Vers le Système :<br>• Création d'une phase.<br>• Affectation des différentes interfaces.<br>• Affectation des différentes modèles \ objets.<br>• Affectation des différentes fonctions. |
| Construire le modèle | **Développeur** | Vers le Système :<br>• Création de dépendances conditionnelles.<br>• Affectation des métriques.<br>• Affectation des tests.<br>• Affectation des tables de probabilité conditionnelles |
| Animer le Réseau Bayésien | **Développeur** | Vers l'acteur :<br>• Afficher le Réseau Bayésien de logiciel. |

**Diagramme de cas d'utilisation :**

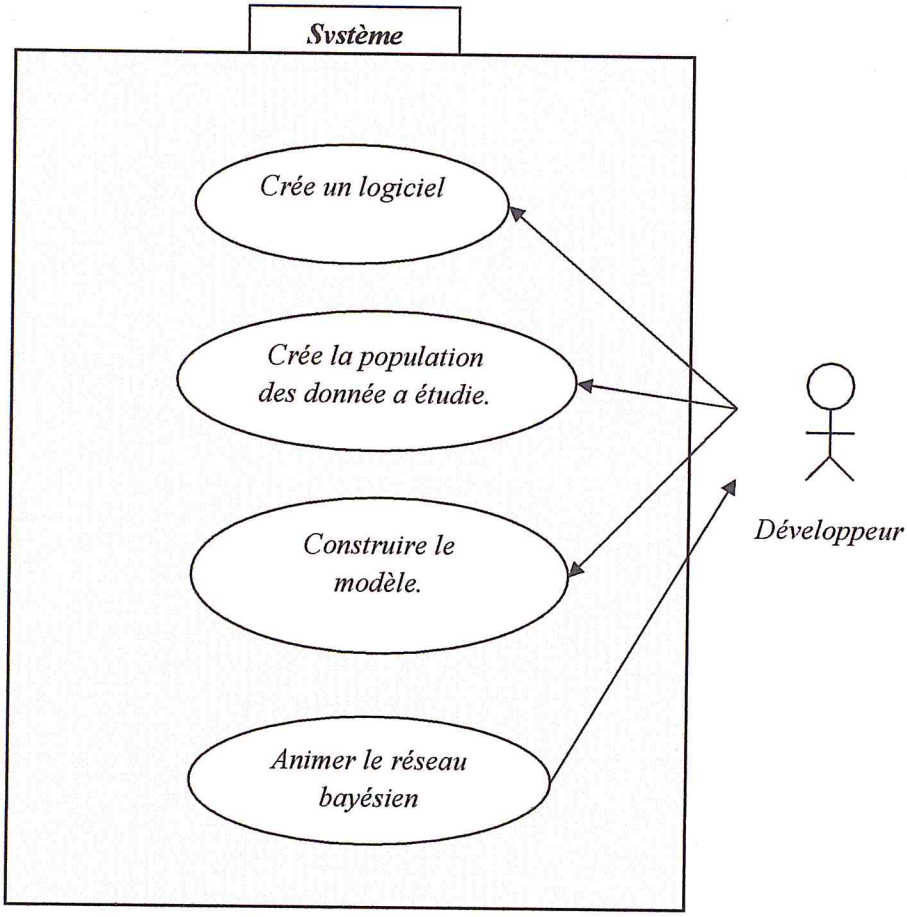



***Figure 59 :*** *Diagramme de cas d'utilisation global (notation UML 2.0).*

### 3.7.4. Description détaillée des cas d'utilisations

Nous allons maintenant détailler chaque cas d'utilisation qui doit faire l'objet d'une définition a priori qui décrit l'intention de l'acteur lorsqu'il utilise le système et les séquences d'actions principales.

#### 3.7.4.1. Crée un logiciel :

***Description de scénario :***

**Pré Conditions :**

Aucun

**Scénario nominal :**

Ce cas d'utilisation commence lorsque le développeur lance un logiciel.

*Enchaînement (a) :* Créer un logiciel

- Le développeur crée un nouveau logiciel avec un nouveau nom.
- Pour l'initialisation il faut qu'il définie le cycle de vie à utiliser.

- Ce cas d'utilisation se termine lorsque le développeur valider la création.

**Diagramme d'activité de crée un logiciel:**

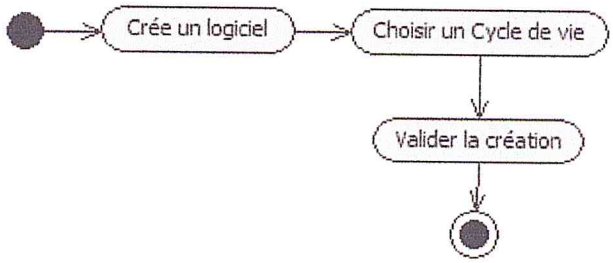



***Figure 60:*** *Diagramme d'activité de crée un logiciel (notation UML 2.0).*

### 3.7.4.2. Crée la population des donnée a étudie:

***Description de scénario :***

**Pré Conditions :**

- Au moins un logiciel.

**Scénario nominal :**

Ce cas d'utilisation commence lorsque le développeur demande au système de ajouter des donnée a étudie

*Enchaînement (a) :* ajouter un phase

- Le développeur crée une nouvelle phase de cycle de vie.

*Enchaînement (b) :* Ajouter une interface

- Le développeur crée une nouvelle interface

*Enchaînement (c) :* Ajouter un modèle / objet

- Le développeur crée un nouvel modèle / objet

*Enchaînement (d) :* Ajouter une fonction

- Le développeur crée une nouvelle fonction

**Enchaînements alternatifs :**

*Enchaînement (e) :* modifier une phase

- Le développeur modifier le contenue de phase (interface, modèle, fonction)

*Enchaînement (f) :* supprimé une phase

- Le développeur supprimer le contenue de phase

- Ce cas d'utilisation se termine lorsque le développeur a terminé.

**Diagramme d'activité de la création de population des donnée a étudie:**

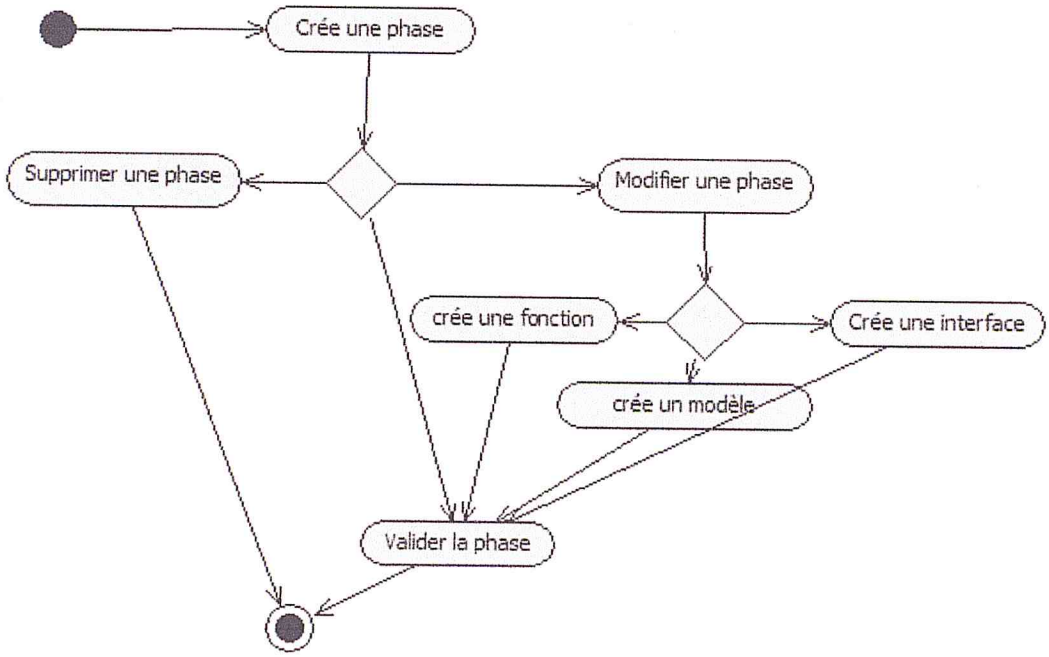



***Figure 61:*** *Diagramme d'activité de la création de population des donnée a étudie (notation UML 2.0).*

### 3.7.4.3. Construire le modèle.

***Description de scénario :***

**Pré Conditions :**

- Au moins un logiciel.
- Plusieurs phases
- Plusieurs populations des données

**Scénario nominal :**

Ce cas d'utilisation commence lorsque le développeur demande au système de construire le modèle.

*Enchaînement (a) :* Affecter une métrique

- Le développeur choisir un objet a étudie et il lui affecte un calcul.

*Enchaînement (b) :* Affecter un test

- Le développeur choisir un objet a étudie et il lui affecte un calcul.

*Enchaînement (c) :* définie les dépendances conditionnelles :

- Le développeur choisir deux objets a étudie et il lui affecte une relation.

*Enchaînement (d) :* définie les TPC :

- Le développeur choisir une dépendance entre deux objets a étudie et il lui affecte une table.

Ce cas d'utilisation se termine lorsque le développeur a terminé le travaille.

**Diagramme d'activité de construction du modèle:**

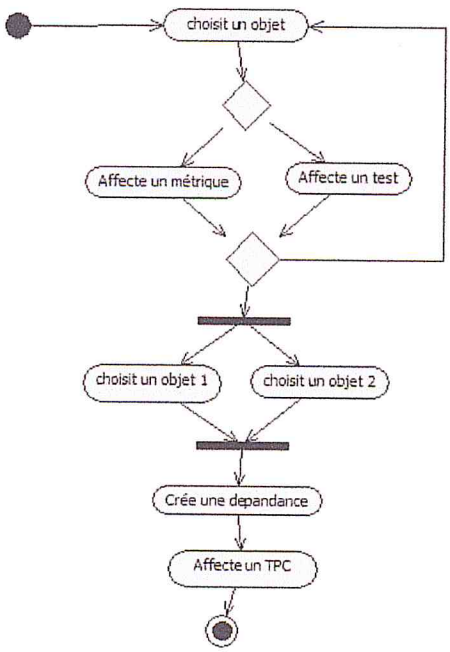



***Figure 62:*** *Diagramme d'activité de Construction du modèle (notation UML 2.0).*

#### 3.7.4.4. Animer le Réseau Bayésien

***Description de scénario :***

**Pré Conditions :**

- Au moins un logiciel.
- Toutes les phases
- Toutes les populations des données
- Toutes les métriques effectuées
- Tous les tests effectués.
- Toutes les dépendances effectuées.
- Tous les TPC effectués.

**Scénario nominal :**

Ce cas d'utilisation commence lorsque le développeur demande au système Animer le Réseau Bayésien.

*Enchaînement (a)* Animer le Réseau Bayésien.

- Le développeur anime le Réseau Bayésien

- Ce cas d'utilisation se termine lorsque le développeur a terminé le travaille.

### 3.7.5. Structuration des cas d'utilisations dans des packages

Cette phase va permettre de structurer les cas d'utilisations en groupes fortement cohérents, ceci afin de préparer le terrain pour la prochaine phase qui est le « découpage en catégories ».

La structuration des cas d'utilisations se fait par domaine d'expertise métier c.à.d. les éléments contenus dans un *package* doivent représenter un ensemble fortement cohérent et sont généralement de même nature et de même niveau sémantique.

***Tableau 8 :*** *Structuration des cas d'utilisations dans des packages.*

| Cas d'utilisation | Acteurs | Package |
|---|---|---|
| Crée un logiciel | **Développeur** | Gestion des données statique |
| Crée la population des donnée a étudie. | **Développeur** | |
| Construire le modèle. | **Développeur** | Gestion de calcul |
| Animer le Réseau Bayésien | **Développeur** | |

### 3.7.6. Identification des classes

Définition des objectifs, Analyse des besoins et faisabilité, Conception générale, Conception détaillée, Conception détaillée, Codage, Tests unitaires, Intégration, Qualification, Maintenance, Logiciel,

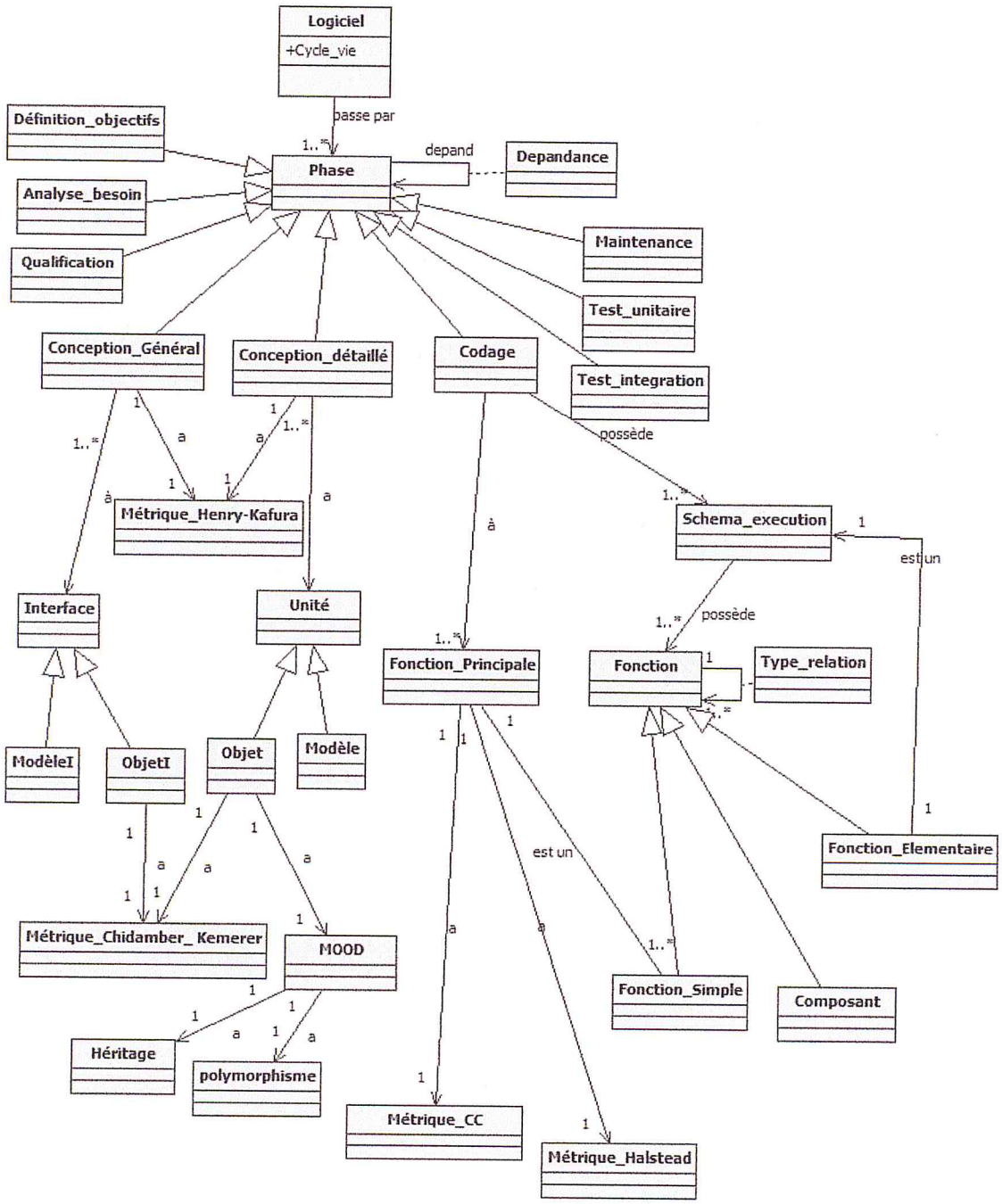



***Figure 63:*** *Diagramme de classe de l'outil de calcule de la fiabilité (notation UML 2.0).*

## 3.8. Conclusion

Ce projet est volontairement concentré sur les aspects de modélisation et les résultats qu'ils peuvent engendrer pour pouvoir aboutir à la construction d'un outil de quantification du niveau de fiabilité d'un logiciel, conçu selon une approche basé sur les Réseaux Bayésien : combinant des informations connues et des informations issues des tests, et des métriques durant les différentes processus de cycles de vie d'un logiciel.

Nous avons proposé une modélisation des phénomènes de défaillances de systèmes explicitant des fautes, qui touche le logiciel le plus large possible. Elle prend en compte un maximum de facteurs régissent le comportement de ces systèmes vis-à-vis les défaillances selon les différents phases de cycles de vie d'un logiciel. Enfin, cette approche nous a permet de construire une nouvelle classe de modèle basée sur des hypothèses réalistes, qui conduisent à des problèmes de statistique mathématique très intéressantes et il reste l'application de cette méthode à des données réelles qui va donnée des résultats satisfaisantes théoriquement, et qui permet ainsi de faire des prédictions de fiabilité. Mais malheureusement, il s'agit d'un domaine de recherche récent et l'offre logicielle est encore pauvre et incomplète pour la validation totale des résultats issus de cet outil.

# CONCLUSION GÉNÉRALE

# Conclusion générale

Actuellement, les modèles de fiabilité du logiciel sont encore très peu utilisés par les développeurs et restent même un sujet assez controversé. Les ingénieurs suivent des méthodes anarchiques de réalisation. Cette situation rend difficile les opérations d'amélioration des logiciels et leurs maintenance. Les modèles de fiabilité des logiciels dont l'objectif est d'aider le développeur à quantifier l'évolution de son logiciel durant son processus de déploiement (cycle de vie), depuis la phase de conception et même après la phase de test, afin de capter et analyser les effets des changements sur le fonctionnement globale du logiciel, pour des raisons de future amélioration et maintenance du code. Mais malheureusement ce qui n'est pas exploité. Les développeurs sont les seuls à fournir une évaluation quantifiée du niveau de fiabilité obtenu en fin de développement, ce point n'est jamais explicitement traité, ou est abordé de manière subjective, ou encore, il est remplacé par des exigences sur le processus de réalisation. Cette situation est due principalement à la difficulté de maitrise de ces modèles, pour cette raison, ils n'ont pas encore été conçus ou mal conçus et réalisé convenablement. Dans tous les cas, les techniques de substitution utilisées ne permettent pas d'avoir une idée précise du risque que l'on prend en mettant un logiciel en opération, ce qui est par essence, le but de la quantification de la fiabilité, que nous essayions de l'aborder par notre projet.

Ce travail nous a permis d'essayer de résoudre le problème de l'évaluation de la fiabilité d'un logiciel présentant des fautes à un instant donné de son cycle de vie, au vu de ses défaillances successives et des corrections qui lui sont apportées. Dans ce but, nous avons proposer une modélisation du comportement du logiciel au cours du temps, où l'on a montrer que l'évolution de ce système résulte de l'interaction complexe de ces composants, environnement et les mesures de logiciels, cette interaction étant entièrement résumée par la donnée de la fiabilité.

Ce résultat nous a permis de construire des graphiques et des statistiques exploitables. Les modèles proportionnels utilisé les RBs, qui donnent lieu à d'intéressants problèmes de statistique appliquée. Ce projet a été volontairement concentré sur les aspects de modélisation et les résultats qu'ils peuvent engendrer. Nous avons proposé une modélisation des phénomènes de défaillances de systèmes explicitant des fautes de conception, qui touche le logiciel le plus large possible. Elle prend en compte un maximum de facteurs régissent le comportement de ces systèmes vis-à-vis les défaillances.

# CONCLUSION GÉNÉRALE

Les travaux que nous présentons dans ce mémoire s'inscrivent dans le cadre d'initier des travaux de recherche plus approfondie dans ce domaine par le Laboratoire de Recherche et de Développement des Systèmes d'information, pour la continuité des études sur la croissance de la sûreté de fonctionnement. Nos travaux s'articulent autour de d'un modèle de calcul de la fiabilité d'un logiciel : basé sur le modèle RB et fondé sur les processus de cycles de vie.

Enfin, en fiabilité comme dans beaucoup d'autres branches, les situations rencontrées ont souvent plusieurs facettes complémentaires. Citons les dualités : systèmes réparables ou non réparables, fiabilité des logiciels ou des matériels, temps continu ou temps discret, système vu comme une entité (« boîte noire ») ou comme un ensemble de composants (« boîte blanche »), données complètes ou censurées, durées de bon fonctionnement ou de réparation, etc… Quand on a traité une facette d'un problème, une perspective naturelle est de traiter les autres facettes. On peut ainsi construire facilement d'autres directions de recherche à partir des problèmes que nous avons traités jusqu'à maintenant.

Pour conclure, nous espérons que notre contribution dans ce domaine de sûreté de fonctionnement des systèmes informatiques, et la fiabilité des logiciels, soit bénéfique pour les programmeurs, les développeurs et pour quoi pas pour un but pédagogique ou de recherche ; et à permet de mettre l'accent sur les contributions de ce projet à l'avancement des connaissances et au développement des technologies, tout en identifiant ses limites et ses contraintes, et de permettre également d'identifier de nouvelles voies de recherche.

# PERSPECTIVE

## Références bibliographiques

[1] Pham H., Software reliability, Springer, 2000.

[2] Printz J., Productivité des programmeurs, Hermès-Lavoisier, 2001.

[3] Olivier GAUDOIN, «FIABILITE DES SYSTEMES ET DES LOGICIELS», note de cours, Grenoble INP - ENSIMAG – 3ème année, 2004.

[4] Saoudi LEILA, « cours génie logiciel », Maître de conférences en université de Tlemcen, 2007/2008.

[5] Vallée F. et Vernos D., Comment utiliser la fiabilité du logiciel comme critère d'arrêt du test, 13ème Colloque Européen de Sûreté de Fonctionnement (λμ13 - ESREL 2002), 164-167, 2002.

[6] Vallée F., « Fiabilité des logiciels », Techniques de l'Ingénieur, P 1-10, 2004.

[7] David GUSTAFSON, « Engineering Software », Schaum's, EdiScience, 2002.

[8] Klaus LAMBERTZ, Article sur « Complexité et qualité : Comment mesurer la complexité d'un logiciel», Société Verifysoft Technology, 2007.

[9] Philippe LERAY, sujet de la thèse de doctorat: « Réseaux Bayésiens : apprentissage et modélisation de systèmes complexes », Université de Rouen, 2006.

[10] Gilles BALMISSE, livre « Réseaux Bayésiens », ellipeses, septembre 2002.

[11] William MARSH, Project Manager, The SERENE Method Manual Version 1.0 (F), ERA Technology Ltd, 14 May 1999.

[12] Marc BOUISSOU, cours : « Argumentaires de sûreté : comment formaliser les évaluations des experts avec la méthode SERENE », EDF Electricité de France, 26/01/1999.

[13] Philippe WEBER, Lionel CERISIER, « Modèle de type Réseau Bayésien Pour l'estimation de la fiabilité des Procèdes», Centre de Recherche en Automatique de Nancy (CRAN), 2002.

[14] Philippe LERAY et al, Article : «Learning causal Bayesian networks from data and manipulation», collaboration labo. Université Rouen et Vrije Universiteit Brussel, versions 2004.