الجمهورية الجزائرية الديمقراطية الشعبية

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université de Saad Dahlab Blida

FACULTE DE SCIENCES

DEPARTEMENT D'INFORMATIQUE

Projet de fin d'étude pour l'obtention du diplôme de

Master 2 en Ingénierie des logiciels

Thème :

Validation de l'outil support au modèle de mesure de la satisfaction du décideur.

Présenté par :

M^lle Manel El-kremairi

M^lle Mouna Mahdjoub

Thème dirigé par :

M^me D. Bouaissa

Année : 2018 / 2019

Résumé

La qualité d'un logiciel est obtenue grâce à l'application de techniques de développement et l'utilisation d'un processus de validation tout au long du cycle de vie de logiciel. Un examen attentif d'attributs et d'exigences de validation spécifiques conduit à la sélection d'un ensemble équilibré de techniques d'analyse et de test pouvant être utilisés.

L'objectif de cette étude est d'évaluer la validité d'un logiciel et d'assurer sa qualité. La problématique est par conséquent la suivante : **élaboration d'une approche de validation de logiciel**. Dans un cas particulier, c'est vérifier techniquement et **valider l'outil support au modèle de mesure de la satisfaction du décideur**. Dans ce contexte, la validation de logiciel correspond à la vérification de la conformité entre les exigences et les fonctionnalités spécifiées.

Pour répondre à la problématique, nous nous sommes inspirés du processus de validation du standard [IEEE 2012-1998] pour concevoir notre approche et ensuite l'implémenter en un outil de validation. Notre approche est constituée de deux sous processus : Approvisionnement et Développement. Dont le premier sous processus définit la méthode de validation quant à elle est une succession de trois phases : la validation de documentation, ensuite la validation en boite noire et enfin la validation en boite blanche.

Le deuxième sous processus définit les techniques utilisés dans chaque phase de validation décrite au-dessus, tels que : un questionnaire dans la première phase ensuite, une exécution successive de l'outil en comparant le nombre d'échec à chaque fois dans la deuxième phase, enfin des tests unitaires pour chaque fonction de l'outil, un test de couverture de code et aussi un test de performance dans la troisième phase de validation. A partir des résultats de ces deux sous processus de validation, nous pouvons assurer la validité et la performance de logiciel et il peut être mis en pratique.

Mots clé : validation, logiciel, processus, test, méthode, technique.

Abstract

Software quality is achieved through the application of development techniques and the use of a validation process throughout the software life cycle .A careful examination of specific attributes and validation requirements leads to the selection of a balanced set of analytical and testing techniques that can be used.

The objective of this study is to evaluate validity of software and ensure its quality. The problem is therefore as follows: development of a software validation approach. In particular case, it means technically verifying and validating the tool supporting the decision-maker satisfaction measurement model. In this context, software validation is the verification of compliance between the requirements and the specified functionalities.

To answer the problem, we took inspiration from the validation process of the standard [IEEE 2012-1998] to design our approach and then implement it as a validation tool. Our approach consists of two sub-processes: Procurement and Development. The first sub-process defines the validation method, which is a succession of three phases: documentation's validation then validation in black box and finally validation in white box.

The second sub-process defines the techniques used in each validation phase described above, such as a questionnaire in the first phase then. A successive execution of the tool by comparing the number of failures each time in the second phase. Finally, unit tests for each tool function, a code coverage test and a performance test in the third validation phase. From the results of these two validation sub-processes, we can ensure the validity and performance of the software and it can be putted into practice.

Key words: validation, software, process, test, method, technique.

ملخص

يتم تحقيق جودة البرنامج من خلال تطبيق تقنيات التطوير واستخدام عملية التحقق من الصحة طوال دورة حياة البرنامج

يؤدي الفحص الدقيق لسمات معينة ومتطلبات التحقق من الصحة إلى اختيار مجموعة متوازنة من تقنيات التحليل والاختبار التي يمكن استخدامها.

لهدف من هذه الدراسة هو تقييم صلاحية البرنامج وضمان جودته. المشكلة هي بالتالي: تطوير نهج التحقق من صحة البرمجيات. في حالة معينة، تقوم بالتحقق من الناحية الفنية والتحقق من أداة الدعم وفقًا لنموذج رضا صانع القرار الذي طوره الطالبتان مطاط ودحماني. يعد التحقق من صحة البرنامج هو التحقق من الامتثال بين بين المتطلبات والميزات المحددة

لتصميم نهجنا ثم تنفيذه كأداة .[IEEE 2012-1998]للإجابة على المشكلة، استلهمنا عملية التحقق من الصحة القياسية للتحقق من الصحة. يتكون منهجنا من عمليتين فرعيتين: المشتريات والتطوير. التي تحدد العملية الفرعية الأولى طريقة التحقق من الصحة لأنها سلسلة من ثلاث مراحل: التحقق من الوثائق ثم التحقق من الصحة في الصندوق الأسود وأخيرا التحقق في المربع الأبيض.

تحدد العملية الفرعية الثانية التقنيات المستخدمة في كل مرحلة من مراحل التحقق الموضحة أعله، مثل: استبيان في المرحلة الأولى بعد ذلك، تنفيذ متتابع للأداة بمقارنة عدد حالات الفشل في كل مرة في المرحلة الثانية أخيرًا، تختبر الوحدة لكل وظيفة من وظائف الأداة، واختبار تغطية الرمز وأيضًا اختبار الأداء في مرحلة التحقق الثالثة. من نتائج هذين الأمرين تحت عملية التحقق من الصحة، يمكننا ضمان صحة وأداء البرنامج ويمكن تنفيذه.

الكلمات المفتاحية: التحقق، البرامج، العملية، الاختبار، الطريقة، التقنية

Remerciements

Après avoir rendu grâce à « Allah » le tout puissant et miséricordieux, qui nous a donné la force et la patience d'accomplir ce modeste travail, Nous tenons à remercier vivement tous ceux qui, de près ou de loin ont participé à la rédaction de ce mémoire. Il s'agit plus particulièrement de :

Madame Djamila Bouaissa notre promotrice pour sa disponibilité, sa rigueur scientifique, son précieux conseil et son sens d'écoute et d'échange durant toute la période du travail.

Madame Fatima Boumahdi et aussi Monsieur Rachid Chalal pour leur disponibilité, leurs conseils et leurs sens d'écoute tout au long de la réalisation de ce travail.

Nos vifs remerciements vont également aux membres de jury pour l'intérêt qu'ils ont porté à notre recherche en acceptant d'examiner notre travail et de l'enrichir par leurs propositions.

Enfin, nous tenons également à remercier toutes les personnes qui ont participé à la réalisation de ce modeste travail.

Dédicace

Je dédie ce mémoire
A ma très chère Maman ***Soumaya***

A mon très cher Papa ***Mohamed***

Aucune dédicace, aucun mot ne pourrait exprimer à leur juste valeur la gratitude et l'amour que je vous porte.

Je mets entre vos mais, le fruit de longues années d'études, de longs mois de distance de votre amour et de votre tendresse, de longs jours d'apprentissage.

Chaque ligne de ce mémoire chaque mot et chaque lettre vous exprime la reconnaissance, le respect, l'estime et le merci d'être mes parents.

A mon âme sœur et ma moitié ***Racha*** *pour sa patience et son écoute toute cette période*

A mes sœurs ***Nour ElHouda, Hadia*** *et* ***Rania***

A mon petit frère ***Abderahmmane***

A ma chères tantes ***Hadjira*** *et* ***Atika***

A tous mes oncles

A mon grand-père, que dieu le couvre dans son paradis

A mes très chers grands parents

A ma meilleure amie ***Hadjer*** *pour sa présence et son amitié ces deux année*

A ***Mouna,*** *ma chère amie et mon binôme*

A une personne très chère à mon cœur Imad

Je vous dédie ce travail en témoignage des liens solides et intimes qui nous unissent et pour leurs soutiens, encouragements en vous souhaitant un avenir plein de succès et de bonheur.

Manel

Dédicace

Je dédie ce travail

A mon père Tahar et ma Mère Fatma zohra

Aucune dédicace aussi parfaite et douce soit -elle, ne saurait exprimer toute ma reconnaissance et tout l'amour que je vous porte.

Ce travail présente le fruit de votre soutien, vos sacrifices, et vos encouragements.

Jamais il n'aurait vu le jour sans les conseils que vous avez consentis pour mon éducation.

Que dieu vous protège et vous accorde une longue vie pleine de santé et bonheur.

Madre, Padre : Je vous adore.

A mes chères aimables sœurs Manal et Malak

A mon adorable unique frère Mohamed yazid

C'est le moment de vous réaffirmer toute ma fraternité. Nos liens sanguins religieux et autres ne seront jamais rompus et que ces liens nous conduisent tous au paradis.

À mon grand-père Rabie et ma grand-mère Atika.

Que dieu vous protège tous.

A ma troisième sœur et mon binôme Manal

A tous mes amis Asma, Khouloud, Ikram, Racha, Rania, Fouad

Et spécialement à mes chères Hadjer et Lila

Merci pour les beaux moments qu'on a passé ensemble.

A toute ma famille

A tous mes collègues et Mes amis.

A tous ceux que J'aime.

Table des matières

Résumé ii

Abstract iii

ملخص iv

Remerciements v

Dédicace vi

Dédicace vii

Table des matières viii

Liste des abréviations xii

Liste des tableaux xiii

Liste des figures xiv

Introduction 1

Partie I Etat de l'art 3

Chapitre 1 La validation de logiciel 4

1.1 Définitions 4

1.1.1 Logiciel 4

1.1.2 Validation 7

1.1.3 Validation de logiciel 7

1.2 Le Processus de validation d'un logiciel 10

1.2.1 Objectif du processus de validation 10
1.2.2 Les sous-processus de validation 11
1.3 Méthodes de validation 13
1.3.1 Contrôles basés sur la documentation 14
1.3.2 Contrôles fonctionnels 16
1.3.3 Contrôles basé sur le code source 18
1.4 Limitation de validation 22
Conclusion 23
Chapitre 2 Les tests logiciels 24
2.1 Définitions 24
2.1.1 Test 24
2.1.2 Faute/ Erreur/ Défaillance 24
2.1.3 Cas de test 26
2.1.4 Test logiciel 26
2.2 Les niveaux de test logiciel 28
2.2.1 Les tests unitaires 28
2.2.2 Les tests d'intégration 29
2.2.3 Les tests de validation (système) 30
2.2.4 Les tests d'acceptation 31
2.3 Les types de test logiciel 32
2.3.1 Test de performance 32
2.3.2 Test d'utilisation 33
2.3.3 Test de régression 33
2.4 Les méthodes de test logiciel 35
2.4.1 Les tests fonctionnels 35
2.4.2 Les tests structurels 38
Conclusion 44
Partie II La conception de l'outil de validation 45

Chapitre 3 Approche de validation proposée 46
3.1 Processus de validation 46
3.2 Choix des sous-processus de validation 48
3.2.1 Sous-processus d'Approvisionnement 49
3.2.2 Sous-processus Développement 50
Conclusion 54
Chapitre 4 Architecture de l'outil de validation « ValidApp » 55
4.1 Introduction 55
4.2 Choix de la méthode suivie 56
4.3 Etape 1 : La spécification des besoins 57
4.3.1 Besoins fonctionnels 57
4.3.2 Besoins non fonctionnels 61
4.4 Etape 2 : L'analyse 62
4.4.1 Diagramme de classe 62
4.5 Etape 3 : La conception 63
4.5.1 La conception globale (architecturale) 63
4.5.2 La conception détaillée 65
Conclusion 75
Chapitre 5 Implémentation de l'outil de validation « ValidApp » 77
5.1 Choix du langage de programmation 77
5.2 Outils techniques 77
5.2.1 Environnement de développement 77
5.2.2 Outils de tests 78
5.3 Création de la base de données 82
5.3.1 Description des tables 83
5.4 Etape 5 : Test et validation 84
5.4.1 Test 87
5.4.2 Validation 91

Conclusion 93
Conclusion 94
Perspective 95
Bibliographie 96
Annexe A 101

Liste des abréviations

- V&V : Vérification et Validation
- VV&T : vérification, validation et Test
- AMDE : Analyse de Mode de Défaillance et de ses Effets
- ISTQB : International Software Testing Qualification Board
- CSTE : Certified Software Tester

Liste des tableaux

Tableau 1: Comparaison entre les différents cycles de vie d'un logiciel.....7
Tableau 2: Niveau d'intégrité logiciel [IEEE 2012 standard] 9
Tableau 3: Tâches et activités du processus de validation [standard IEEE 2012-1998].....13
Tableau 4 : Questionnaire 1 : Identification du logiciel [Jan et Bengt, 2004] 15
Tableau 5 : Questionnaire 2 : complétude de documentation [Jan et bengt, 2004] 16
Tableau 6: Méthodes et techniques de tests 43
Tableau 7: Questionnaire 1 « validation de documentation proposée » 52
Tableau 8: Les taches associées aux différents acteurs du système.....58
Tableau 9: Messages échangés avec le système 59
Tableau 10 : Les cas d'utilisation associés au système à développer.....60
Tableau 11: Modules, sous modules et fonctionnalités du système proposé.....65
Tableau 12: Description du cas d'utilisation "S'authentifier".....66
Tableau 13: Description du cas d'utilisation "Identification de logiciel".....68
Tableau 14: Description du cas d'utilisation " Menu Principale" 69
Tableau 15: Description du cas d'utilisation " Validation Documentation" 70
Tableau 16: Description du cas d'utilisation "Validation de Documentation".....71
Tableau 17: Description du cas d'utilisation "Validation Boite-Noire" 72
Tableau 18 : Description du cas d'utilisation " validation Boite-Blanche".....73
Tableau 19: Description du cas d'utilisation " Analyser Résultat" 75

Liste des figures

Figure 1: processus de validation [Standard IEEE 2012-1998] 11
Figure 2 : Combinaison de méthodes de validation [Jan et Benget, 2004] 14
Figure 3 : Arbre de Défaillance [I.Olivier, 2014] 22
Figure 4: Méthode de validation - Jan et Brengt, 2004 22
Figure 5: Relation Faute/ Erreur/ Défaillance [S. Kangoye, 2017] 25
Figure 6: Aperçu sur le test logiciel 26
Figure 7 : Niveaux de tests logiciels 28
Figure 8: Positionnement de niveaux de tests [N.Tréves CNAM, VV&T] 32
Figure 9: Types de tests de logiciel 34
Figure 10: Illustration de test Boite-Noire 35
Figure 11: Tests logiciels 44
Figure 12: Déroulement de la Validation de logiciel 46
Figure 13: Sous-processus de validation retenus 47
Figure 14: Vue détaillée du processus de validation 49
Figure 15 : Vue détaillée du processus Approvisionnement 50
Figure 16 : Vue détaillée du sous-processus de développement 51
Figure 17: Cycle de vie de logiciel en cascade 57
Figure 18: Diagramme de cas d'utilisation de l'outil de validation proposé 61
Figure 19: Diagramme de classe de l'outil proposé 62
Figure 20: Conception globale de l'outil proposé 64
Figure 21: Sous Module 64
Figure 22: Module 64
Figure 23: Architecture global de l'outil DMSatisfaction 85
Figure 24: Architecture détaillée de l'outil DMSatisfaction. 86

Introduction

Les logiciels ont aujourd'hui envahi notre quotidien par le lot de services qu'ils apportent dans tous les domaines. En effet, un logiciel est un ensemble de séquences d'instructions et d'un jeu de données nécessaires à ces opérations. Afin d'avoir un logiciel de qualité on procède à sa validation, qui consiste à une confirmation par des preuves tangibles que les exigences pour une utilisation spécifique ou une application prévues ont été satisfaites.

Problématique :

La programmation est une activité de résolution de problème et la détermination de la validité de la solution fait partie du processus. Cette activité traite des techniques de test et d'analyse pouvant être utilisées pour valider un logiciel et instaurer la confiance dans la qualité du produit de programmation. La validation d'un logiciel est la problématique du domaine abordé dans ce travail de recherche, plus précisément la validation de **l'outil support au modèle de mesure de la satisfaction du décideur** suivant un processus de validation inspiré de la littérature y afférente.

Objectif :

C'est dans ce contexte que s'intègre notre projet de fin d'étude, qui a pour objectif d'élaborer un processus de validation qui sert à valider un logiciel afin de diminuer les conséquences liées au dysfonctionnement de logiciel et assurer sa qualité. Ce processus de validation est l'ensemble des activités corrélées qui utilise un élément en entrée qui est l'outil support de mesure de la satisfaction du décideur dans notre cas d'étude, pour produire un résultat escompté.

Plan du mémoire :

Ce mémoire se définit en deux parties :

- Partie 1 : cette partie est une étude de la littérature sur la validation de logiciel et elle se décompose en deux chapitres : le premier chapitre « Validation de logiciel » présente les informations nécessaires et pertinentes sur le processus et la méthode de validation, ensuite dans le deuxième chapitre « les tests logiciels », nous avons détaillé les différents niveaux, types et méthodes de test logiciel afin de trouver les techniques appropriées pour mener à bien notre validation.

- Partie 2 : cette partie est le résultat des études de la littérature de la validation d'un logiciel dont nous avons commencé par proposer d'élaborer un processus de cette dernière dans le troisième chapitre « Approche de validation proposée» qui est une suite de deux sous processus d'Approvisionnement et de Développement, que nous avons suivi pour concevoir et implémenter un logiciel qui valide l'outil support de mesure de la satisfaction du décideur dans le dernier chapitre « Architecture de l'outil de validation ValidApp ».

Nous clôturons ce mémoire par une conclusion générale dans laquelle nous proposons aussi des perspectives dans le but de rendre notre travail meilleur et encore plus général sur des différents logiciels.

Partie I

Etat de l'art

La première partie de notre de étude de recherche contient le résultat de la littérature du domaine de la validation de logiciel ainsi, pour une bonne compréhension du thème, nous y trouvons tous les travaux de recherche pertinents.

Chapitre 1

La validation de logiciel

Dans ce premier chapitre, nous exposons les différents concepts clés du sujet pour une bonne compréhension ainsi que les travaux effectués sur la validation de logiciel.

1.1 Définitions

1.1.1 Logiciel

Un logiciel est un programme qui apporte à l'ordinateur un lot de fonctionnalités supplémentaires qui ne sont pas forcément présentes à l'origine. Un logiciel s'installe sur l'ordinateur via un disque (CD, DVD) ou en téléchargeant sur internet. Il existe des logiciels gratuits et d'autres sont payants. Un logiciel détermine donc les taches qui peuvent être effectuées par la machine, ordonne son fonctionnement et lui procure ainsi son utilité fonctionnelle. [Dictionnaire informatique : cour-informatique]

Deux types de logiciels sont distingués, les logiciels systèmes (Windows, MacOs, Linux…) qui jouent le rôle d'interface entre matériel (hardware) et les logiciels d'applications qui sont dédiés à des tâches spécifiques.

1.1.1.1 La qualité logicielle

[ISO] « Ensemble des traits et des caractéristiques d'un produit logiciel portant sur son aptitude à satisfaire des besoins exprimés ou implicites. »

[IEEE 2012, 1998] « La qualité du logiciel correspond au degré selon lequel un logiciel possède une combinaison d'attributs désirés. »

1.1.1.2 L'assurance qualité logicielle

[IEEE 2012, 1998] définit l'assurance qualité logiciel comme suit :

« Un modèle planifié et systématique de toutes les actions nécessaires pour fournir une confiance adéquate qu'un produit est conforme à ses exigences techniques établies. Et aussi comme étant un ensemble d'activités conçues pour évaluer le processus par lequel les produits sont développés ou fabriqués. »

Les objectifs liés à l'assurance qualité logiciel sont :

- L'aptitude de l'organisation à satisfaire le niveau de qualité désiré.
- Une méthode de conduite de projet et une démarche qui vise à appréhender, évaluer et améliorer les activités des entreprises d'ingénierie.

1.1.1.3 Le cycle de vie du logiciel

La norme [ISO/IEC 12207:2008 and ISO/IEC 15288:2008] définit le cycle de vie d'un logiciel comme suit : « Le cycle de vie de logiciel est évolution d'un système, produit, service, projet ou autre entité d'origine humaine de la conception à la retraite. »

Le cycle de vie d'un logiciel indique les étapes par lesquelles doivent passer un logiciel de sa conception jusqu'à sa mise en marche. Il permet de détecter les erreurs plus tôt tout au long du processus de réalisation et ainsi les corriger pour produire un logiciel de qualité. La validation logicielle se fait à la fin du développement de celui-ci.

Les étapes sont les suivantes :

- **Pré-étude** : Définition des objectifs du produit et le domaine d'activité.
- **Analyse** : recueillir et formaliser les besoins client
- **Conception** : élaborer la structure générale du système et aussi celle du chaque sous système
- **Développement** : coder et programmer les fonctionnalités définit auparavant
- **Tests** : tester le logiciel conformément aux spécifications (test unitaire, test d'intégration, test fonctionnel et test de validation)
- **Réception :** vérifier la conformité aux spécifications par le client
- **Maintenance :** prendre en charge les actions collectives du système

1.1.3.3 Modèles de cycle de vie d'un logiciel

➢ **Modèle en Cascade**

Le principe de ce modèle définit les étapes du développement de logiciel en phases séquentielles, à la fin de chaque phase un document est créé pour vérifier sa conformité avec les besoins exigés. Si la conformité est validée on passe à la phase suivante sinon on retourne à la précédente.

➢ **Modèle en V**

Ce modèle s'agit d'un modèle en cascade dans lequel le développement des tests et du logiciel sont effectués de manière synchrone. Son principe est qu'avec toute description d'un composant est accompagnée de tests qui permettent de s'assurer qu'il correspond à sa description.

➢ **Modèle par incrément**

Le modèle par incrément découpe le système en domaines qui sont traités individuellement sur le modèle en cascade. Par contre aux modèles précédents, un logiciel est décomposé en composants développés séparément et intégrés à la fin du processus. Dans ce modèle, un seul ensemble de composants est développés à la fois.

Tableau comparatif des différents cycles de vie d'un logiciel

Cycle de vie	Utilisation	Validation
Cascade	• La phase de spécification est bien définit • La définition du produit est stable • Une nouvelle version du produit • Implantation d'un produit existant sur une nouvelle plate-forme	• A la dernière étape de développement de logiciel.
En V	• Spécification des besoins bien définit • Excellent pour les systèmes	• Après chaque étape de développement de logiciel.

	requérant une grande sureté • Les changements doivent être faits ava l'analyse	
Par incrément	• Les produits utilisant de nouvelles technologies • Les produis ayant une durée de développement longue	• En parallèle avec le développement de logiciel.

Tableau 1: Comparaison entre les différents cycles de vie d'un logiciel

En termes de conclusion, la validation a pour objectif de vérifier que toutes les étapes de développement aboutiront à un produit conforme aux exigences de l'autorisation de mise sur le marché de manière stable et reproductible.

1.1.2 Validation

La validation est une opération destinée à démontrer qu'une procédure ou une activité conduit effectivement aux résultats escomptés. Elle comprend la qualification des systèmes et des équipements.

Selon la norme [ISO 9000, 2005], la validation est : « la confirmation par des preuves tangibles que les exigences pour une utilisation spécifique ou une application prévue ont été satisfaites. »

Et la norme [IEEE Standard 1012 – 2004] définit la validation comme : « Une confirmation par examen et fourniture de preuves objectives que les exigences particulières applicables à un usage spécifique sont remplies. »

En d'autres termes la validation a pour but de répondre à la question suivante : « est-ce que le système fait bien ce qu'il est censé faire (comme spécifié par les exigences) ? ».

1.1.3 Validation de logiciel

[FDA, clause 3.1.2] définit la validation de logiciel comme étant « une confirmation par examen et fourniture de preuves objectives que les spécifications du logiciel sont conformes aux besoins de l'utilisateur et aux utilisations auxquelles il est destiné, et que les exigences particulières mises en œuvre par le logiciel peuvent être systématiquement satisfaites. »

1.1.3.1 L'importance de la validation de logiciel

« La validation logicielle constitue une activité qui consiste à fournir l'assurance qu'un système logiciel fonctionnera conformément à un ensemble d'exigences ou de spécifications. Elle s'inscrit dans le cycle de vie du développement logiciel et permet de réduire les écarts entre les spécifications et le produit logiciel. En effet, cette activité minimise le nombre de défauts ainsi que leur coût de correction en les détectant au plus tôt après leur introduction dans le cycle de vie du logiciel car, plus le défaut est introduit en amont, plus son impact est grand et sa correction coûteuse » [Trousse de déploiement Vérification et validation].

L'activité de validation peut être utilisée pour appuyer les objectifs liés à l'assurance de la qualité du logiciel, à la gestion de projet et à la gestion de la configuration. Ces objectifs sont valides durant tout le cycle de vie du logiciel, du début du cycle de développement et pendant toute la période de maintenance/vie du logiciel. Ainsi pour la planification du processus de validation d'un logiciel un niveau d'intégrité logicielle est attribué pour identifier les taches de validation.

Selon le standard [IEEE 2012-1998], Les tâches inhérentes à l'activité de validation incluent :

- Évaluation
- Confirmation de la conformité
- Inspection du code source
- Revues
- Tests

L'objectif de la **validation** est de confirmer que le produit logiciel satisfasse aux besoins du client [ISO 12207]

1.1.3.4 Niveaux d'intégrité logicielle

Les logiciels se diffèrent en fonction de l'utilisation prévue et de l'application du système à des utilisations critiques ou non critiques. Certains systèmes logiciels affectent des systèmes essentiels au maintien de la vie, tandis que d'autres sont des outils de recherche autonomes non critiques. La criticité des logiciels est une description de l'utilisation prévue et application d'un système, donc l'approche de niveau d'intégrité logicielle est pour quantifier la criticité logicielle.

Les niveaux d'intégrité logicielle désignent une plage de valeurs de criticité logicielle nécessaires pour maintenir les risques dans des limites acceptables. Ces propriétés logicielles peuvent inclure la sécurité, la complexité du logiciel, les performances, la fiabilité ou d'autres caractéristiques.

Un logiciel critique à haute intégrité requiert généralement un ensemble plus vaste et une application plus rigoureuse des tâches minimales de validation à exécuter.

Le standard IEEE définit un schéma de quatre niveaux d'intégrité logicielle comme une méthode pour définir les taches minimales de validation associé à chaque niveau d'intégrité.

Criticité	Description	Niveau
Haute	Le logiciel doit être exécuté correctement sinon des conséquences très graves se produiront (Fonctions sélectionnées affectent les performances critiques du système et une perte de système). • Aucune atténuation n'est possible	4
Majeur	Le logiciel doit être exécuté correctement sinon l'utilisation prévue du système ne sera pas réaliser, avec des conséquences graves (Fonctions sélectionnées affectent les performances du système). • Une atténuation partielle est possible	3
Modérée	Le logiciel doit être exécuté correctement sinon une fonction prévue ne sera pas réaliser, avec des conséquences mineures (Fonctions sélectionnées affectent les performances du système). • Atténuation complète possible	2
Faible	Le logiciel doit être exécuté correctement sinon la fonction prévue ne sera pas réaliser, avec des conséquences négligeables. (Fonctions sélectionnées affectées ont un effet notable sur les performances du système). • Aucune mesure d'atténuation	1

Tableau 2: Niveau d'intégrité logiciel [IEEE 2012 standard]

Les taches minimales de la validation :

Le niveau d'intégrité logicielle permet d'identifier les taches minimales du processus de validation suivantes :

- Planification de l'interface entre l'effort de la validation et le fournisseur.
- Evaluation du changement proposé.
- Analyse de risque.
- Portée de l'effort de la validation.
- Evaluation de la conception du logiciel

1.2 Le Processus de validation d'un logiciel

Le processus de validation de logiciel détermine si les produits de développement d'une activité donnée sont conformes aux exigences de cette activité et si le logiciel satisfait à l'usage auquel il est destiné et aux besoins des utilisateurs [IEEE 2012, 1998].
Cette détermination peut inclure l'analyse, l'évaluation, la révision, l'inspection, l'évaluation et le test de produits et de processus logiciels. Les processus de validation évaluent le logiciel dans le contexte du système, y compris l'environnement opérationnel, le matériel, les logiciels d'interface, les opérateurs et les utilisateurs.

Le processus de validation doit décrire l'organisation de l'effort de validation, y compris le degré d'indépendance requis, Ainsi que la relation de processus de validation avec d'autres processus tels que le développement, la gestion de projet, l'assurance qualité et la gestion de la configuration et les lignes de communication au sein de l'effort de la vérification et la validation, l'autorité de résolution des problèmes soulevés par les tâches de validation et l'autorité d'approbation des produits de validation.

1.2.1 Objectif du processus de validation

Les processus de validation fournissent une évaluation objective des produits logiciels tout au long du cycle de vie du logiciel. Cette évaluation montre si les exigences logicielles sont correctes, complètes, précises et testables.

Le standard IEEE définit les objectifs du processus de validation suivants :

- Montrer que le système n'a pas de défauts.
- Découvrir des défauts et des erreurs dans le système plus rapidement.
- Démontre que le système et les logiciels requis sont corrects, achevés, précis, cohérents et testables.
- Montrer que les charges sont bien implémentées.
- Améliorer la compréhension de la gestion dans le risque de processus et le risque produit

Donc le processus de validation fournit la preuve que le logiciel répond aux exigences du système attribué au logiciel et résout le problème.

1.2.2 Les sous-processus de validation

Selon la norme [ISO/IEC 12207] l'application du processus de validation (Figure 1) se diffère d'un cycle de développement d'un logiciel à un autre et l'exécution des activités de ce processus de validation se fait soit parallèlement au développement du logiciel, soit à sa conclusion.

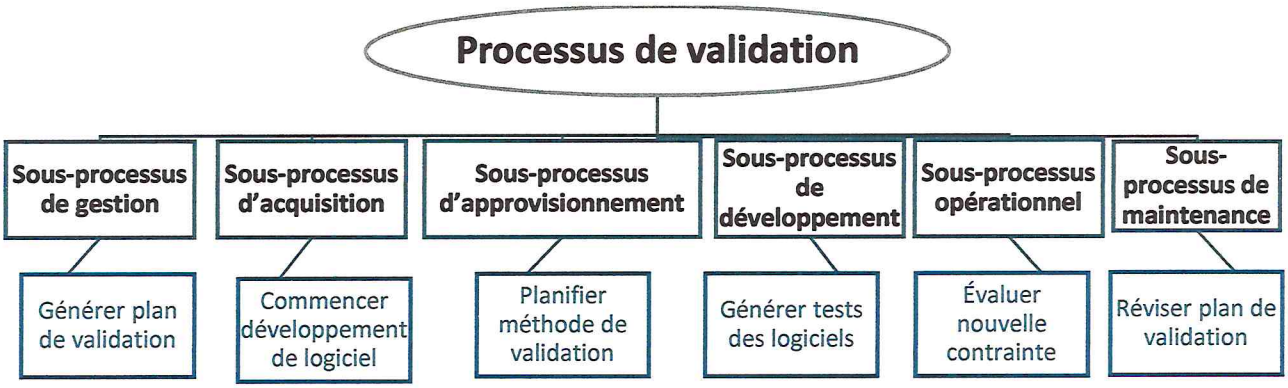


Figure 1: processus de validation [Standard IEEE 2012-1998]

Le processus de validation prend en charge les principaux sous-processus suivants :

- **Sous-processus de gestion** : Le processus de gestion est effectué dans tous les processus et les activités du cycle de vie du logiciel. Ce processus contient les activités et les taches génériques qui examinent en performance l'effort de validation du logiciel en fonction des calendriers de projet mis à jour et de l'état de développement, et coordonne les résultats de la validation avec le développeur et les autres processus tels que l'assurance qualité.

- **Sous-processus d'acquisition :** ce processus commence par la définition du besoin d'acquérir un système ou un logiciel, puis continue avec la préparation et l'émission d'une demande de proposition pour enfin adresser le lancement du projet et la préparation du contrat.

- **Sous-processus d'approvisionnement :** ce processus est lancé après la prise de décision de préparer une proposition, il consiste à adresser l'initialisation et la planification du projet, y compris l'élaboration et l'exécution des plans jusqu'à la livraison du système.

- **Sous-processus de développement** : ce processus contient les activités et les taches du développeur. Il contient les activités d'analyse, de conception, de codage, d'intégration, de test, d'installation et d'acceptation des exigences relatives aux produits logiciels.

- **Sous-processus opérationnel :** un processus d'exploitation qui implique l'utilisation du logiciel par l'utilisateur final dans un environnement opérationnel pour évaluer l'impact des changements dans l'environnement du fonctionnement.

- **Sous-processus de maintenance :** ce processus est activé lorsque le logiciel ou la documentation associée doivent être modifiés pour répondre à un besoin de modification ou d'amélioration du système.

1.2.2.1 Taches et activités des différents sous-processus de validation

Processus	Activité	Tâches
Processus de gestion	Gestion de l'effort de validation : effectuer en fonction du niveau d'intégrité logicielle sélectionné.	• génération de plan de la validation logiciel. • revue de direction de l'effort de la validation. • évaluation des changements de base. • assistance à la révision technique. • identifier les opportunités d'amélioration des processus dans la conduite de validation.
Processus d'acquisition	Support aux acquisitions de validation : concerne le lancement du projet et la préparation du contrat.	• portée de l'effort de validation • planification de l'interface entre l'effort de validation et le fournisseur • examen des exigences du système.
Processus d'approvisionnement	planifier la validation	• planification de l'interface entre les efforts de validation et les fournisseurs • vérification du contrat • validation du code source
Processus de développement	Conception, exigence, design, mise en œuvre, test et installation et les exigences relatives aux produits logiciels	• analyse de traçabilité • évaluation d'exigences logicielles • analyse de criticité • génération de test • test d'acceptation • gestion de la configuration et d'évaluation

Processus opérationnel	Opération de validation	• évaluation de nouvelles contraintes • évaluation des procédures opérationnelles • analyse de la sécurité • analyse des risques
Processus de maintenance	Maintenance de validation	• révision du plan de la validation • évaluation des anomalies • analyse de criticité • itération de tache

Tableau 3: Tâches et activités du processus de validation [standard IEEE 2012-1998]

1.3 Méthodes de validation

La validation du logiciel peut être effectuée de différentes manières, par l'*analyse* qui signifie l'examen de la conception d'un outil sans le faire fonctionner et aussi par le *test* qui signifie l'examen de cet outil en observant son comportement.

Certaines des instructions de validation décrites par la suite sont assez rapides à utiliser, tandis que d'autres nécessitent des efforts considérables. Aussi une bonne procédure de validation combine plusieurs types de méthodes de validation qui sont présentées en trois groupes :

- Méthodes de validation des contrôles basés sur la documentation
- Méthodes de validation pour les contrôles fonctionnels
- Méthodes de validation des contrôles basés sur le code source.

La validation doit aboutir à un rapport de validation final incluant l'identification du logiciel, la méthode de validation effectuée, les différents types de test choisis et les résultats attendus et obtenus après chaque test afin de comprendre cette méthode de validation de manière détaillées et obtenir un résultat performant [Carl E, 2003].

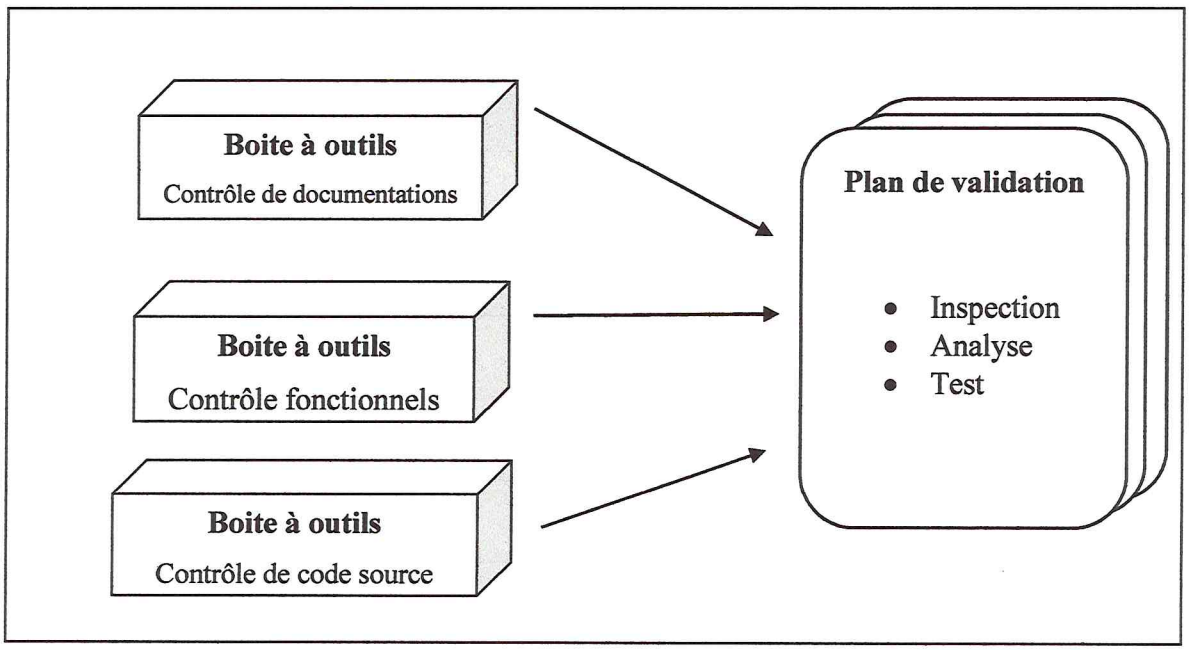


Figure 2 : Combinaison de méthodes de validation [Jan et Benget, 2004]

1.3.1.1 Identification du logiciel

Le logiciel peut être décrit comme un ensemble de données, de paramètres et de code exécutable qui est l'ensemble des instructions permettant son fonctionnement. [Jan et Bengt, 2004]

L'identification de logiciel est pratiquement impossible par inspection visuelle, donc son identification devrait inclure les informations sur toutes les parties du logiciel, son objectif et toutes ses fonctionnalités et aussi le système d'exploitation dans lequel ce logiciel s'exécute afin de bien comprendre la structure et le comportement du logiciel.

Une liste de questions à répondre est établie dans le guide des exigences logicielles [Jan et Bengt, 2004] extraite de la norme [ISO/ICE 17025], pour l'identification de logiciel est la suivante :

Questions	Oui	Non	Commentaire
Les parties physiques de l'instrument de mesure sont-elles été identifiées ?			
Des systèmes d'exploitation sont-ils utilisés ?			

Existe-t-il une identification unique de la version du logiciel ?			
Est-il possible de télécharger une nouvelle version de ce logiciel ?			
Les outils de développement logiciel ont-ils été identifiés et documentés ?			

Tableau 4 : Questionnaire 1 : Identification du logiciel [Jan et Bengt, 2004]

La réponse à ces questions se fait par un *oui/non* et ajouter des commentaires si nécessaire, puis le résultat obtenue (taux *Oui* > taux *Non*) définit la validité de logiciel.

1.3.1.2 Complétude de documentation

La mise en œuvre logicielle devrait être documentée [WELMEC, 1999], l'utilisateur doit avoir un aperçu et une description de la structure générale du logiciel.

Ce type de méthode sert à vérifier l'exhaustivité et la complétude de la documentation décrivant le logiciel et pour assurer que les conditions requises ont été remplies.

Une liste de questions extraites du guide [WELMEC 7.1, clause 4.2] est utilisée pour appuyer ce contrôle :

Question	Oui	Non	Commentaire
Existe-t-il un manuel d'utilisation ?			
Existe-t-il un manuel d'installation ?			
Existe-t-il des descriptions de la conception du logiciel ?			
Existe-t-il une documentation sur les fonctionnalités du logiciel ?			
Existe-t-il une description des menus et des boites de dialogues ?			
Existe-t-il une description sur les algorithmes utilisés ?			

Existe-t-il un aperçu de l'aspect de sécurité, par exemple : les comptes des utilisateurs ?			

Tableau 5 : Questionnaire 2 : complétude de documentation [Jan et bengt, 2004]

La réponse à ces questions se fait par un *oui/non* et ajouter des commentaires si nécessaire, puis le résultat obtenue (taux *Oui* > taux *Non*) définit la validité de logiciel.

1.3.1.3 Analyse de la documentation du logiciel

L'analyse de la documentation du logiciel a pour objectif de consulter la documentation de la manière dont une fonction est implémentée dans le logiciel. [Jan et Bengt, 2004]

La documentation du logiciel peut être en texte brut ou également une représentation graphique telle que des organigrammes ou des diagrammes UML.

Cette méthode de validation est faite par un examinateur sans le soutien des ingénieurs de développement dont le but est d'acquérir une connaissance détaillée des fonctions pertinentes et aussi la partie du code source de ces fonctions. Le résultat obtenu est documenté.

1.3.1.4 Inspection du cahier des charges

L'inspection du cahier des charges est une technique générale dans laquelle les différentes qualités d'un document de spécification autrement dit « un cahier des charges », sont évaluées par une équipe indépendante n'ayant pas participé au développement du logiciel. Cette dernière pose des questions au développeur du logiciel, qui doit y répondre de manière satisfaisante afin d'examiner les objectifs de conception des fonctions pertinentes et s'assurer que toutes les exigences ont été prises en comptes. Ils doivent également vérifier que tous les aspects techniques opérationnels et organisationnels sont couverts [IEC 61508-7].

1.3.2 Contrôles fonctionnels

1.3.2.1 Test fonctionnel

« Les tests basés sur des définitions ou sur des spécifications sont appelés tests fonctionnels ou test 'boite-noire'. Il identifie le cas de test en fonction de la définition de ce que le logiciel est censé faire,

ce cas de test remet en cause l'utilisation prévue d'un programme, ainsi que les interfaces internes et externes du programme.» [FDA, 2002, clause 5.2.5]

Le test fonctionnel a pour objectif de révéler les défauts dans les fonctions pertinentes du logiciel et déterminer si l'unité fonctionnelle est performante conformément aux exigences de ce logiciel.

Lors des tests fonctionnels, des examens sont effectués pour voir si les caractéristiques et les exigences spécifiées du système ont été atteintes. Dans cette phase de validation, aucune connaissance de la structure interne du système n'est utilisée pour guider les tests. Le système reçoit des données d'entrée qui caractérisent de manière adéquate le fonctionnement attendu. Les résultats sont observés et leurs réponses sont comparées à celles données par la spécification puis les écarts par rapport à la spécification sont documentés.

Selon la norme [ICE, 61508] plusieurs techniques sont utilisé pour le test fonctionnel que nous allons les détailler dans le chapitre suivant tels que : Test classe d'équivalence, Test All-pairs, Test aux limites, Graphe de cause à effet, Transition d'état, Test de cas d'utilisation, Test exploratoire et Analyse de domaine.

1.3.2.2 Simulation des signaux d'entrée

La notion de simulation

« La simulation en informatique est une série de calculs effectués sur un ordinateur et reproduisant un phénomène physique. Elle aboutit à la description du résultat de ce phénomène, comme s'il s'était réellement déroulé. Cette représentation peut être une série de données, une image ou même un film vidéo. » [futura-sciences, 2019]

L'objectif de la simulation des signaux d'entrée est de détecter les erreurs des fonctions pertinentes du logiciel par une série de tests fonctionnels qui sont effectués conformément à la description des exigences initiales de ce logiciel. [Jan ET Bengt, 2004]

Un instrument de mesure ou un logiciel ne peut pas être toujours testé dans l'environnement dans lequel il sera utilisé, certains signaux d'entrée ne sont disponibles que lors d'une installation réelle, c'est pour cela que la simulation de certains de ces signaux est nécessaire.

La simulation peut être réalisée avec des générateurs d'impulsion « modèles apportent des fonctionnalités de contrôles énormes sur les signaux d'entrés. [Todd, 2004] », Qui donne la possibilité d'aller au-delà des paramétrages de base permettra de tester le logiciel et améliorer les résultats de ces test, ainsi de mieux satisfaire les besoins du client.

Il existe plusieurs générateur de simulation tel que : PowerFlow, HyperWorks, AnyLogic, Solver SDK…

Après terminer la deuxième phase de validation d'un logiciel qui ne nécessite pas une connaissance approfondie sur le logiciel ni son déroulement interne, la troisième et la dernière phase commence. Dans cette phase la connaissance du comportement interne et du code source est importante pour effectuer les cas de tests nécessaires que nous allons détailler dans le chapitre suivant.

1.3.3 Contrôles basé sur le code source

Le contrôle de code source est la pratique consistant à suivre et à gérer les modifications apportées au code et est aussi un élément essentiel du processus de développement. Les systèmes de gestion du contrôle de code source fournissent un historique continu de développement de code et aident à résoudre les conflits lors de la fusion des contributions de plusieurs sources. [Jan & Bengt, 2004]

Pour la validation de logiciel, plusieurs méthodes basées sur la vérification du code source sont utilisées telles que : l'analyse de défaillance et des effets, l'analyse d'arborescence des évènements et l'analyse de la détection des pannes. Elles peuvent également être utilisées si le code source n'est pas disponible. Dans ce cas, l'analyse du comportement du logiciel sera basée sur la documentation ou le contrôle fonctionnel.

1.3.3.1 Revue de la conception du logiciel

La notion de la revue

« Une revue est un ensemble de méthodes (inspection, revue de projet 'walkthroughs' et revue personnelles de produits) qui aide à réviser la conception du logiciel, appelé aussi une *inspection* » [GSI]

Revue de la conception du logiciel

La revue de conception se définit comme suit selon la norme [ISO, 2000] ; « Examen d'une conception, menée de façon complète et systématique à l'aide de documents en vue d'évaluer sa capacité à satisfaire aux exigences de qualité, d'identifier les problèmes et éventuellement de proposer le développement de solutions. »

Selon [FDA, 2002], la revue de la conception est une méthode de validation basée sur la présentation des ingénieurs de développement de logiciel, qui a pour objectif de détecter et de réparer les fautes le plus tôt possible dans le cycle de développement des logiciels avec une économie de temps et d'argents, aussi pour garantir la conformité des exigences spécifiées dans le cahier des charges.

Principe de revue de la conception

- Produire une conception révisable avec un contexte définie et une structure claire et consistante.
- Suivre une stratégie de revue précise qui spécifie l'ordre de la révision des éléments de la conception par étape.
- Revoir la conception par étape et s'assurer que tous les éléments du logiciel ont été conçus et que les méthodes et la procédure sont correctement utilisées et vérifier la structure et le flux d'ensembles du programme.
- Vérifier que la logique implémente correctement les besoins des utilisateurs et s'assurer que chaque fonction qui est exigée apparait dans la conception.

La revue de la conception améliore la qualité du programme et permet d'économiser le temps de développement.

1.3.3.2 Test structurel (test boite blanche)

Le test structurel est le type de test réalisé pour tester la structure du code source, il est également appelé test de la boite blanche. Le test structurel est plus préoccupé par la façon dont le système fonctionne que par la fonctionnalité du système. Il fournit plus de couverture pour les tests, dont le principal but est de révéler les erreurs dans les fonctions logicielles par des tests fonctionnels. [URL : www.softwaretestingcalass.com]

Le test structurel peut être utilisé à différents niveaux, tels que les tests unitaires, les tests d'intégrations, les tests fonctionnels et les tests de validation que nous allons exploiter dans le chapitre suivant et donner les différents techniques de ce type de test.

Selon [Jan & Bengt, 2004], Le test boite blanche signifie que le fonctionnement du logiciel est testé sur la base de connaissances acquises lors de sa conception et des cas de test seront sélectionnées en fonction de ses connaissances. Ces cas de test remettent en cause les décisions de contrôles prises par le programme, ainsi que les structures de données.

Le test structurel peut également signifie le test de modules logiciels individuels dans un environnement artificiel selon. [FDA, clause 5.2.5]

Test des modules logiciels

Selon [P.Muller, 2000] « un module consiste en un seul bloc de code pouvant être appelé de la même manière qu'une procédure, une fonction ou une méthode, il encapsule les données pour implémenter une fonctionnalité particulière. Aussi, un module a une interface qui permet aux clients d'accéder à ses fonctionnalités de manière uniforme. »

L'objectif principal du test des modules logiciels est de vérifier le comportement d'entrée et de sortie de l'objet à tester et garantir la conformité entre les fonctionnalités implémenter et les spécifications définis.

Principe de test des modules logiciels

Un module de code source est identifié comme important pour une fonction pertinente [Jan et Bengt, 2004]. Ce module logiciel peut ensuite être testé en tant que module autonome dans un environnement de développement logiciel.

En fonction des variables d'entrée du module, des cas de test seront sélectionné pour couvrir différentes parties du code. Les modules peuvent être testés soit comme une « boite noire » sans tenir compte de la logique du code source, soit comme une « boite blanche » dans le cas de connaissance du code source. Le chapitre suivant définit toutes les techniques de ces deux types de tests. Les résultats seront observés et automatiquement enregistrés pour chaque série de test [Jan et Bengt, 2004].

1.3.3.3 Analyse de l'arbre de défaillances

La notion de défaillance

Selon [Aliani, 2006], une défaillance est : « une altération ou cessation de l'aptitude d'un système, à accomplir es fonctions requises avec les performances définies dans les spécifications techniques. »

L'arbre de défaillance représente de façon synthétique l'ensemble des combinaisons d'évènements qui peuvent conduire à une défaillance, cette recherche de combinaisons de causes pouvant provoquer une défaillance se poursuit par une recherche des coupes minimales (ensembles d'évènement de base, ou de conditions nécessaires et suffisants à produire la défaillance). [Aurelie, SD]

D'après le travail effectué par [Jan & Bengt, 2004], l'analyse de l'arbre de défaillance a pour objectif d'examiner l'influence des défauts sur les fonctions pertinentes du logiciel.

Le principe de l'analyse de l'arbre de défaillances

L'analyse par arbre de défaillances est une méthode de type déductif. En effet, il s'agit, à partir d'un événement redouté défini a priori, de déterminer les enchainements d'événements ou combinaisons d'événement pouvant finalement conduire à cet événement. Cette analyse permet de remonter de causes en causes jusqu'aux événements de base susceptibles d'être à l'origine de l'événement redouté (un événement final qui fait l'objet de l'analyse [Olivier, 2014]).

Les événements de base correspondent généralement à des :

- Evénements élémentaires qui sont suffisamment connus et écrits par ailleurs pour qu'il ne soit pas utile d'en rechercher les causes.
- Evénements dont les causes seront développés ultérieurement au gré d'une nouvelle analyse.
- Evénement survenant normalement et de manière récurrente dans le fonctionnement de l'installation

Ainsi, l'analyse par arbre de défaillances permet d'identifier les successions et les combinaisons d'événement décrites avec des opérateurs logiques (AND, OR) [Jan & Bengt, 2004] qui conduisent des événements de base jusqu'à l'événement indésirable retenu. Cette méthode utilise une symbolique graphique particulière qui permet de présenter les résultats dans une structure arborescente proposées dans la norme [CEI 61025 :1990]

La figure ci-dessous montre l'illustration d'un arbre de défaillances :

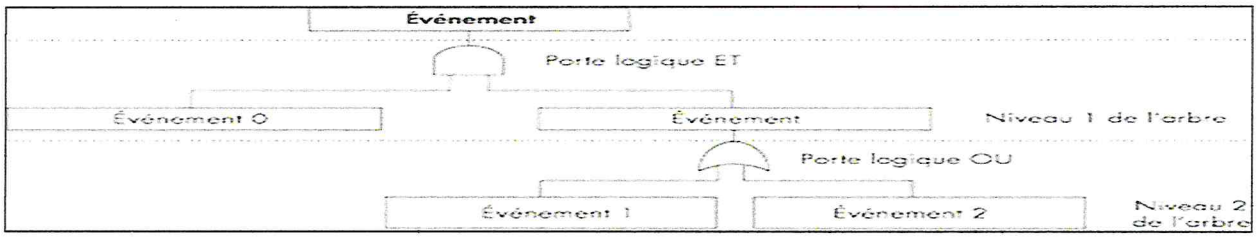


Figure 3 : Arbre de Défaillance [I.Olivier, 2014]

Synthèse sur les méthodes de validation de logiciel

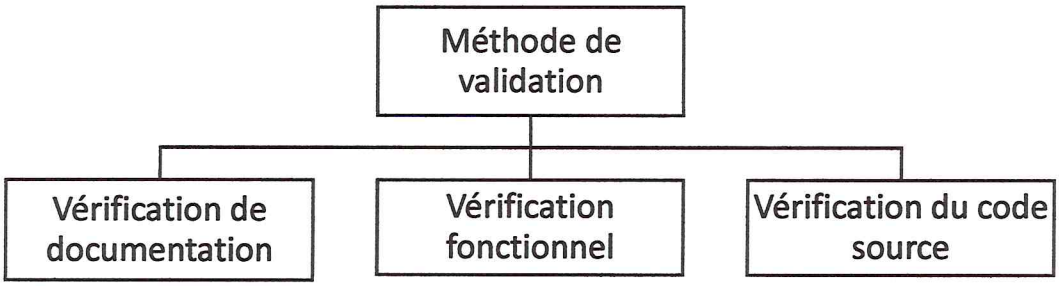


Figure 4: Méthode de validation - Jan et Brengt, 2004

1.4 Limitation de validation

La validation de logiciel est le processus d'évaluation d'un système. Elle permet de détecter les anomalies, mais cela est difficile pour la plupart des programmes, ce qui cause :

- **Impraticabilité des tests de toutes les données :** il est peu pratique d'essayer de tester tous les programmes avec toutes les entrées possibles, grâce à une combinatoire d'explosion.
- **Impraticabilité des tests sur tous les chemins :** il est peu pratique d'essayer de tester tous les chemins d'exécution à travers le produit logiciel.
- **Pas de preuve absolue d'exactitude :** À moins qu'une spécification formelle puisse s'avère correcte et reflète en effet exactement les attentes de l'utilisateur, aucune revendication d'exactitude du produit logiciel peut être faite. « Howden : affirme qu'il n'existe pas d'absolu de preuve d'exactitude »

Conclusion

Nous avons présenté dans ce chapitre une introduction sur le processus et la méthode de validation et les techniques utilisées pour cette dernière. Ainsi nous avons passé en revue quelques concepts fondamentaux de la littérature de la validation de logiciels qui est le processus d'évaluation d'un système suivi pour assurer sa conformité à sa spécification

Cette étude nous a permis de poser des bases pour nos futurs travaux en situant progressivement le périmètre de ceux-ci. Dans le prochain chapitre nous traiterons les tests logiciels utilisés dans le processus de validation.

Chapitre 2

Les tests logiciels

Dans ce chapitre, nous exposons les différents types, niveaux et méthodes de tests logiciels permettant de valider un logiciel.

2.1 Définitions

Nous présentons dans cette section les définitions de concepts clés.

2.1.1 Test

«Le test est l'exécution ou l'évaluation d'un système ou d'un composant par des moyens automatiques ou manuels, pour vérifier qu'il réponde à ses spécifications ou identifier les différences entre les résultats attendus et les résultats obtenus.» [IEEE 729, 2002]

« Il est bien connu que le test représente une part très importante du cout d'un projet de développement, les évaluations se trouvant généralement dans une fourchette de 30 à 50% du cout total. Ce cout comprend non seulement le temps nécessaire pour exécuter les plans de test en fonction des exigences initiales de l'application, mais aussi le temps nécessaire à l'analyse des résultats de ces tests et le temps pris à corriger les erreurs détectées » [Arnaud, 2005]

2.1.2 Faute/ Erreur/ Défaillance

Dans le monde du logiciel il est assez courant de parler de « bugs » ou d'anomalie pour désigner un comportement inattendu résultant de l'exécution d'un logiciel. Cependant, ces

termes ne permettent pas d'identifier et d'établir des relations entre les causes de cette exécution inattendue.

Faute. Une faute dans un logiciel ou un système peut être vu comme un élément ou un évènement ayant entraîné une erreur dans celui-ci. Par exemple, une faute dans un logiciel est le plus souvent d'origine humaine : erreur d'interprétation d'une exigence, oubli d'une condition dans une expression booléenne, etc. Cependant, une faute peut exister sans pour autant perturber le fonctionnement du logiciel, dans ce cas elle est dite *dormante* ou *passive*. Dans le cas contraire, si son activation produit une erreur, elle est dite *active* [L. Morell, 1983].

Erreur. Une erreur dans un système est un état non spécifié, atteint par celui-ci. Elle pourrait par exemple être une "mauvaise" valeur d'un paramètre d'une fonction, mauvaise initialisation d'une variable due à une faute, etc. Une erreur qui se propage dans le logiciel peut entraîner une défaillance de celui-ci [A. Offutt, 1988].

Défaillance. Une défaillance d'un système est un état dans lequel le service délivré par celui-ci n'est pas conforme au service attendu. En d'autres termes, une différence est observée entre le comportement fourni par le système et le comportement attendu (vis à vis des spécifications). Une défaillance peut avoir des conséquences plus ou moins graves dépendant de sa criticité. En général, on les classe en fonction de leur niveau de criticité [B. Beizer, 1990].

En somme, il est possible d'établir une implication potentielle à partir des termes Faute, Erreur et Défaillance, comme illustrée par la Figure suivante :

Figure 5: Relation Faute/ Erreur/ Défaillance [S. Kangoye, 2017]

2.1.3 Cas de test

« un ensemble de valeurs d'entrée (données de tests), de pré conditions d'exécution, de résultats attendus et de post conditions d'exécution, développées pour un objectif ou une condition de tests particulière, tel qu'exécuter un chemin particulier d'un programme ou vérifier le respect d'une exigence spécifique» [IEEE STANDARD 2012-1998]

2.1.4 Test logiciel

« Le test de logiciel est une enquête technique empirique menée pour fournir des informations sur la qualité du produit logiciel et sur sa validité. » [Talal Iqbal, 2017].

« Le test logiciel est un Processus d'analyse d'un programme avec l'intention de détecter des anomalies dans le but de valider sa conformité. » [F. X. Fournari, 2011]

Les tests peuvent être effectués soit par une analyse statique (par examiner des documents tels que les exigences et les spécifications prédéfinit) ou une analyse dynamique (par l'exécution du code et l'exploitation de l'application).

La figure ci-dessous montre le mécanisme d'un test de logiciel

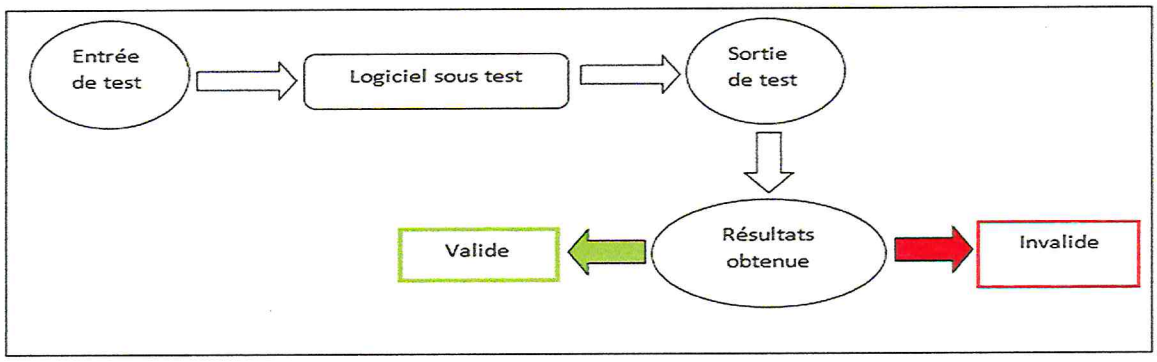


Figure 6: Aperçu sur le test logiciel

2.1.4.1 L'objectif des tests logiciels

Les principaux objectifs du test logiciel sont de :

- répondre correctement à toutes les sortes d'entrées (traiter correctement les entrées valides et invalides)
- fournir un programme pour vérifier que le logiciel produit satisfait les spécifications et la conception du logiciel.
- Commettre toutes les erreurs possibles au plutôt, car certaines erreurs sont couteuse, voir dangereuses et catastrophiques si leurs détection est tardive.

2.1.4.2 Le plan de test logiciel

Le plan de test est un document obligatoire déterminant de déroulement des tests durant le projet. Son objectif est de vérifier que le logiciel produit satisfait la spécification et la conception du logiciel.

Un plan de test doit :

- Définir les éléments à tester et l'ordre dans lequel ils doivent être testés
- Décrire l'environnement de tests
- Définir la façon dont les tests vont être menés
- Décrire et constituer les fiches de tests (les actions à réaliser et les jeux de tests à utiliser)
- Fixer les critères d'arrêt des tests

2.1.4.3 L'importance de test logiciel

Des tests logiciels sont réellement nécessaires pour signaler les défauts et les erreurs commises au cours des différentes phases de développement du logiciel. C'est essentiel car les tests permettent de s'assurer de la fiabilité du client et de sa satisfaction dans l'application. Il est très important d'assurer la qualité du produit. Un produit de qualité livré aux clients aide à gagner leur confiance et en savoir plus sur le produit reçu. Des tests sont nécessaires pour

fournir un produit ou un logiciel de haute qualité qui nécessite un coût de maintenance inférieur et donc un résultat plus précis, cohérent et fiable.

2.2 Les niveaux de test logiciel

Les tests sont regroupés par leur ajout au cours du processus de développement dans le cycle de vie de logiciel et par niveau de spécification. [Software engineering body of knowledge, 1999]

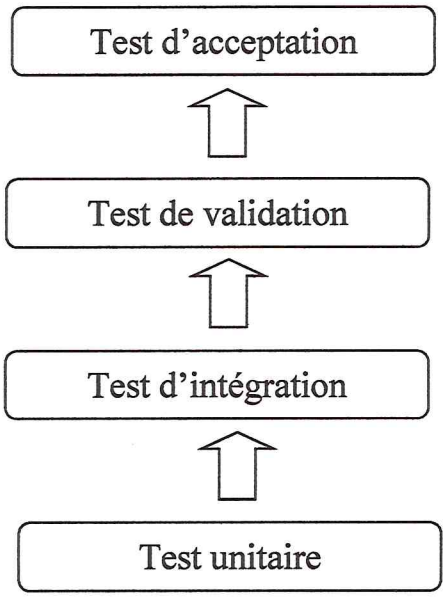


Figure 7 : Niveaux de tests logiciels

Les tests logiciels sont divisés et classés comme suit :

2.2.1 Les tests unitaires

Les tests unitaires sont des tests structurels et sont essentiellement écrits et exécutés par les développeurs de logiciels pour s'assurer que le code est conforme à sa conception et à ses exigences et permettent aussi de détecter le maximum de défaillances. [Y, Boukhouchi, 2017]. Il s'agit pour le programmeur de tester un module, indépendamment du reste du

programme, ceci afin de s'assurer qu'il réponde aux spécifications fonctionnelles et qu'il fonctionne correctement en toutes circonstances.

Les tests unitaires ont deux objectifs :

- La couverture de code : stipule de tester chaque ligne de code écrit (appel de fonctions, boucle et décision)
- La couverture de données : oriente les tests vers les données (données valides, données invalides et trop de données)

2.2.1.1 Le fonctionnement

L'exécution d'un test unitaire est faite en quatre phases :

- Initialisation : (fonction « setUp »), définition d'un environnement de test complétement reproductible
- Exercice : le module à tester est exécuté
- Vérification : (fonction « assert, success, failure »), la comparaison des résultats obtenus avec les résultats attendus.
- Désactivation : (fonction « treamDown »), désinstallation des fonctions initiales pour retrouver l'état initial du système, dans le but de ne pas polluer les tests suivants.

2.2.2 Les tests d'intégration

Le test d'intégration est une phase dans les tests logiciels, qui est précédée des tests unitaires et également suivie par les tests de validation. Dans cette phase de test, chacun des modules indépendants du logiciel est assemblé et testé dans l'ensemble (se base sur l'architecture de conception). Le test d'intégration permet également de vérifier l'aspect fonctionnel, les performances et la fiabilité du logiciel.

Les difficultés principales de l'intégration sont :

- L'implantation non conforme au modèle architectural ;

- Une interface floue ;
- Risque de réutilisation de composants hors domaine ;

2.2.2.1 L'organisation d'un test d'intégration

Le but de l'organisation d'un test d'intégration est de définir la stratégie de son activité en termes d'ordre d'intégration, de test à réaliser, du matériel sur lequel seront lancés les tests et les outils suivant la procédure suivante :

- Introduction et organisation : présentation du document.
- Stratégie : identifier le produit fini du test d'intégration, les spécifications et l'ordre dans lequel les tests doivent être effectués.
- Contenue des spécifications du test d'intégration : définir l'assemblage inhérent aux tests et les attentes du design qui sont à vérifier.

2.2.3 Les tests de validation (système)

Le test de validation permet d'assurer que le système complet, matériel et logiciel, correspond bien à la définition des besoins tels qu'ils avaient été exprimés, et de vérifier si toutes les exigences client, décrites dans le document de spécification du logiciel sont respectées.

Les tests de validation se décomposent généralement en plusieurs phases :

- Validation fonctionnelle : les tests fonctionnels assurent que les différents modules ou composants implémentent correctement les exigences client. Ces tests peuvent être de type valide, invalide, inopportuns, etc. ;
- Validation solution : les tests solution assurent que les exigences client sont respectées d'un point de vue cas d'utilisation. Généralement, ce sont des tests en volume. Chaque grand cas d'utilisation est validé isolément, puis tous les cas d'utilisation sont validés ensemble. L'intérêt de ces tests est de valider la stabilité d'une solution par rapport aux

différents modules qui la composent, en soumettant cette solution à un ensemble d'actions représentatif de ce qui sera fait en production ;

- Validation performance et robustesse : les tests de performance vérifient la conformité de la solution par rapport à ses exigences de performance, alors que les tests de robustesse vont essayer de mettre en évidence des éventuels problèmes de stabilité et de fiabilité dans le temps

2.2.4 Les tests d'acceptation

ISTQB définir le test d'acceptation comme suit : « test formel concernant les besoins des utilisateurs, les exigences et les processus métier conduits pour déterminer si un système satisfait ou non aux critères d'acceptation et pour permettre à l'utilisateur de décider de l'accepter ou non. »

Le test d'acceptation est un niveau de test logiciel où l'acceptabilité d'u système est testée.L'objectif de ce test est d'évaluer la conformité du système aux exigences de l'utilisateur et de déterminer s'il est acceptable pour la livraison.

Ce test est le dernier niveau de test logiciel effectué après de le test de validation et avant de rendre le système disponible pour une utilisation réelle.

La figure ci-dessous illustre le positionnement des niveaux de tests :

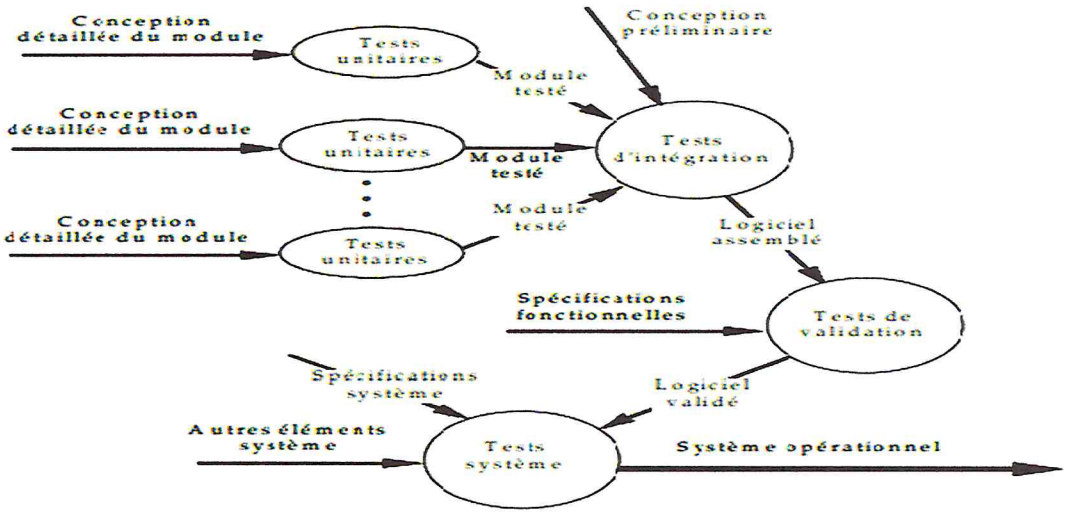


Figure 8: Positionnement de niveaux de tests [N.Tréves CNAM, VV&T]

2.3 Les types de test logiciel

Les types de tests de logiciels listés ci-dessous ne sont que quelques-uns des centaines de types de tests de logiciel :

2.3.1 Test de performance

« Le test de performance détermine la bonne exécution d'un système informatique en mesurant ses temps de réponse sans contexte particulier. Son objectif est de fournir des métriques sur la rapidité de l'application» [Nathalie, 2019].

Le test de performance répond à un besoin des utilisateurs en matière de vitesse et déterminer le comportement d'un système en termes de réactivité et de stabilité sous une certaine charge dont l'objectif est de permettre :

- De connaitre la capacité du système et ses limites.
- De détecter et surveiller ses points faibles.

- D'optimiser ses coûts en infrastructure et en exécution.
- De s'assurer qu'il fonctionne sans erreurs.
- D'optimiser le temps de réponse pour améliorer l'expérience utilisateur.

Il existe essentiellement quatre types de tests de performance :

- **Le test de charge** est un type de test de performance réalisé pour évaluer le comportement d'un système face à une charge de travail croissante.
- **Le test de résistance** est un type de test de performance réalisé pour évaluer le comportement d'un système à la limite de sa charge de travail ou au-delà.
- **Les tests d'endurance** sont un type de tests de performance conduits pour évaluer le comportement d'un système lorsqu'une charge de travail importante est donnée en permanence.

2.3.2 Test d'utilisation

ISTQB définit le test d'utilisation comme suit : « test permettant de déterminer dans quelle mesure le produit logiciel est compris, facile à apprendre, facile à utiliser et attrayant pour les dans des conditions spécifiées. »

Ce test a pour objectif de passer en revue l'interface utilisateur de l'application avec les personnes qui l'utiliseront, cela permet de garantir que la conception (mise en page) permet aux fonctions commerciales d'être exécutées aussi facilement et intuitivement que possible [CSTE CBOX, SD].

Les tests d'utilisations sont effectués aux niveaux de test de validation (système) et le test d'acceptation.

2.3.3 Test de régression

Une régression : Un défaut introduit dans un logiciel à l'occasion de correction de bogues ou un changement établi dans un logiciel. [E, Dijkstra, 1972]

Un test de régression est un sous-ensemble de test d'un programme préalablement testé pour s'assurer que les défauts n'ont pas été introduits ou découverts dans des parties non modifiées du logiciel, et qui vérifie qu'une modification n'a pas eu d'effets de bord sur le logiciel.

L'intérêt principal de ce test est de limiter les anomalies relevées lors du déploiement de l'application, ce test est mis en place pour compléter les tests unitaire et les tests d'intégration.

La difficulté des tests de régression, c'est qu'ils sont :

- des tests longs à exécuter même si le changement sur le système est mineur et les scénarios de tests sont fastidieux à rejouer.
- Des tests qui ne garantissent pas l'absence de défauts, « Le test de programmes peut montrer la présence de bugs, mais ne permet pas de garantir leur absence » [E, Dijkstra, 1972]
- Des tests très couteux.

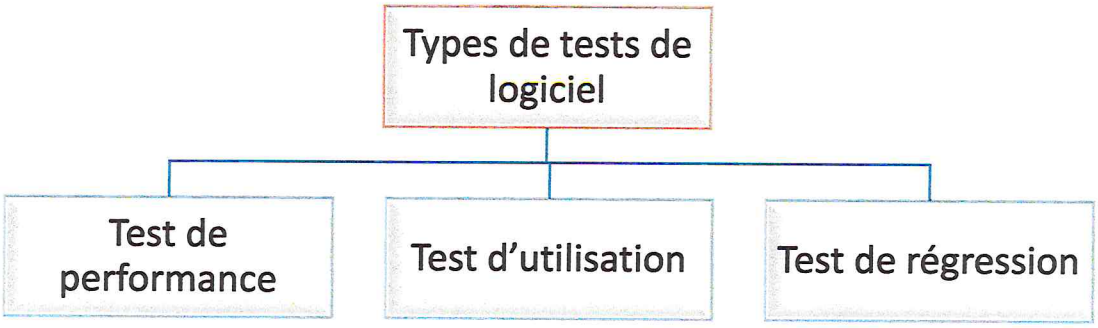


Figure 9: Types de tests de logiciel

2.4 Les méthodes de test logiciel

Les méthodes de test de logiciel répertoriées ci-dessous sont les principales méthodes utilisées lors de la réalisation de différents types de test de logiciel à différents niveaux de test de logiciel.

2.4.1 Les tests fonctionnels

Le test fonctionnel est un service d'assurance qualité qui consiste à s'assurer qu'un système (ou un de ses composants) fonctionne adéquatement. Autrement est un test destiné à vérifier, pour une spécification donnée son comportement fonctionnel.

Le test fonctionnel est appelé aussi test boite-noire « Black box test », une méthode de test logiciel effectuée durant la phase de développement dans laquelle la structure interne, la conception et l'implémentation du système testé n'est pas connue par le testeur.

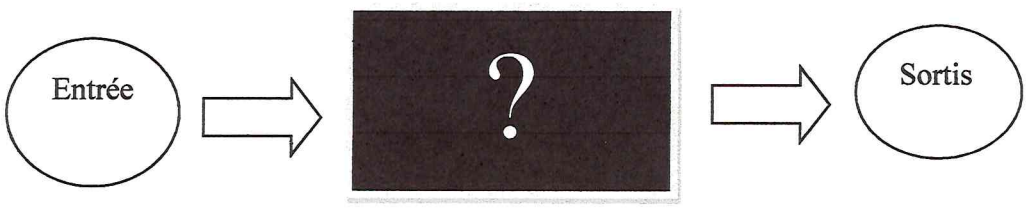


Figure 10: Illustration de test Boite-Noire

Son objectif

L'objectif principal de ce type de test est de :

- Analyser le produit final et déterminer s'il répond bien aux attentes définit sur le plan technique ;

- Vérifier le comportement du logiciel par rapport aux spécifications ;
- Respecter les contraintes (performance, mémoire et délai) et la qualité (portabilité, maintenabilité et documentation).

2.4.1.1 La préparation efficace d'un test fonctionnel

Pour effectuer un test fonctionnel de manière efficace, il est important de détailler de manière claire et précise le produit ou les sections de votre produit que vous souhaitez tester.

Il est recommandé de transmettre aux testeurs une liste d'objectifs fonctionnels pour chaque section à tester. Ces objectifs doivent indiquer les résultats attendus pour chaque section.

2.4.1.2 Les techniques de test Boite-noire

La validation d'un logiciel par le test boite-noire est faite avec plusieurs techniques, sont :

- **Test classe d'équivalence :** Le partitionnement par équivalence est une technique de test logiciel qui divise les données d'entrée d'une unité logicielle en partitions de données équivalentes à partir desquelles des cas de test peuvent être dérivés. En principe, les scénarios de test sont conçus pour couvrir chaque partition au moins une fois.

  Cette technique tente de définir des scénarios de test qui mettent à jour des classes d'erreurs, réduisant ainsi le nombre total de scénarios de test à développer. Le partitionnement par équivalence est généralement appliqué aux entrées d'un composant testé, mais peut être appliqué aux sorties dans de rares cas

  L'avantage de cette technique est la réduction du temps requis pour tester un logiciel en raison du nombre réduit de cas de test.

- **Test All-pairs :** Le test de toutes les paires ou le test par paire est une méthode combinatoire de test de logiciel qui, pour chaque paire de paramètres d'entrée dans un

système (typiquement un algorithme logiciel), teste toutes les combinaisons discrètes possibles de ces paramètres. En utilisant des vecteurs de test choisis, cela peut être fait beaucoup plus rapidement qu'une recherche exhaustive de toutes les combinaisons de tous les paramètres, en "parallélisant" les tests des paires de paramètres Equivalence partitionné.

- **Analyse de valeur limite (test aux limites) :** L'analyse de la valeur limite est basée sur des tests aux limites entre les partitions. Avec une possibilité d'avoir à la fois des limites valides (dans les partitions valides) et des limites non valides (dans les partitions non valides).

- **Graphe de cause à effet :** Le graphique de cause à effet est un graphique orienté qui mappe un ensemble de causes sur un ensemble d'effets. Les causes peuvent être considérées comme une entrée dans le programme et les effets comme une sortie. le graphique montre les nœuds représentant les causes et d'autres représentants les effets, il peut y avoir des nœuds intermédiaires entre ceux-ci qui combinent les entrées à l'aide d'opérateurs logiques tels que AND et OR.

- **Test de transition d'état :** c'est le test des sortis, il est déclenché par des modifications des conditions d'entrée ou des modifications de l'état du système. En d'autres termes, les tests sont conçus pour exécuter des transitions d'état valides et non valides.

- **Test de cas d'utilisation :** Le test de cas d'utilisation consiste à identifier des scénarios de test qui exercent l'ensemble du système, transaction par transaction, du début à la fin.

- **Analyse de domaine :** L'analyse de domaine est le processus par lequel un ingénieur logiciel acquiert des informations d'arrière-plan. Il doit apprendre suffisamment d'informations pour pouvoir comprendre le problème et prendre les bonnes décisions lors de l'analyse des besoins et des autres étapes du processus de génie logiciel.

- **Combinaison de techniques :** Tester deux variables ou plus en combinaison les unes avec les autres.

2.4.1.3 Les avantages de tests fonctionnels

La boîte noire est plus efficace lorsqu'elle est appliquée sur de grands systèmes. Le testeur n'est pas forcément une personne plus technique. Puisque le testeur et le développeur sont indépendants l'un de l'autre dans le test de la boîte noire, il est donc équilibré et sans préjugés.

Les testeurs ne sont pas nécessairement tenus d'avoir une connaissance fonctionnelle approfondie du système. Le test doit être effectué du point de vue de l'utilisateur final et des propriétaires de l'entreprise, l'utilisateur final devant accepter et utiliser le système / le logiciel.

Les tests de la boîte noire contribuent également à identifier le flou et les contradictions dans la conception fonctionnelle et les spécifications au fur et à mesure du déroulement des tests,

Les scénarios de test sont conçus dès que la conception fonctionnelle et les spécifications sont terminées et que le testeur n'a pas besoin d'attendre pour commencer le test.

2.4.1.4 Les inconvénients de test fonctionnel

Les cas de test peuvent être difficiles à concevoir sans spécifications fonctionnelles claires. De plus, il est difficile d'identifier des entrées difficiles et non valides si les scénarios de test ne sont pas développés sur la base de spécifications.

En raison de cette contrainte, il est difficile d'identifier les entrées possibles dans un temps limité, ce qui peut entraîner un retard dans la rédaction des scénarios de test et poser un risque pour le logiciel.

2.4.2 Les tests structurels

Le test structurel, ou test 'boite blanche' permet de définir les données de test à partir de l'analyse du code source selon divers critères :

- Critère "tous les chemins",
- Critère "toutes les branches",
- Critère "toutes les instructions"

Le test de la boîte blanche « White Box testing » est une méthode de test dans laquelle la structure, conception et l'implémentation interne du logiciel testé sont connue par le testeur, ce dernier choisit les entrées pour parcourir le code et détermine les sorties appropriées. [T, Iqbal, 2017]

Le savoir-faire en matière de programmation et la connaissance de la mise en œuvre sont essentiels. Les tests de boîte blanche testent au-delà de l'interface utilisateur et dans les moindres détails d'un système.
Cette méthode de test est nommée ainsi car elle permet de sélectionner des cas de test sur la base d'analyse de la structure interne d'un composant ou de tout le système.

2.4.2.1 Les techniques de tests Boite-Blanche

Les techniques de conception de test structurel « Boite blanche » incluent les critères de couverture de code suivants :

- **Test de flux de contrôle :** cette technique repose sur la sélection judicieuse d'un ensemble de chemins de test dans le programme. L'ensemble de chemins choisis est utilisé pour obtenir une certaine mesure de minutie de test, par exemple, choisissez suffisamment de chemins pour vous assurer que chaque instruction source est exécutée au moins une fois.

- **Test de flux de données :** Le test de flux de données est issu de test basé sur la sélection de chemins dans le flux de contrôle du programme afin d'explorer des séquences d'événements liées au statut de variables ou d'objets de données. Cette technique est centrée sur les points auxquels les variables reçoivent des valeurs et sur les points où ces valeurs sont utilisées.

- **Test de branche :** Le test de branche est utilisé pour s'assurer que chacune des branches possibles à partir de chaque point de décision est exécutée au moins une fois, assurant ainsi que tout le code accessible est exécuté.

- **Couverture de décision :** La couverture de décision ou couverture de branche est une méthode de test qui vise à garantir que chaque branche possible à partir de chaque point de décision est exécutée au moins une fois, garantissant ainsi que tout le code accessible est exécuté. Autrement dit, chaque décision est prise dans chaque sens, vrai et faux. Cela aide à valider toutes les branches du code en s'assurant qu'aucune branche ne conduit à un comportement anormal de l'application.

- **Couverture de relevé de compte :** La couverture des instructions est une technique de test de la boîte blanche, qui implique l'exécution de toutes les instructions au moins une fois dans le code source. C'est une métrique, qui est utilisée pour calculer et mesurer le nombre d'instructions dans le code source qui ont été exécutées. En utilisant cette technique, nous pouvons vérifier ce que le code source est censé faire et ce qu'il ne devrait pas faire. Il peut également être utilisé pour vérifier la qualité du code et le flux de différents chemins dans le programme. Le principal inconvénient de cette technique est que nous ne pouvons pas tester la fausse condition décrite dans la formule ci-dessous :

  Couverture de relevé de compte = (nombre d'instructions exécutés / nombre total d'instructions dans le code source * 100)

- **Couverture de condition ou décision modifiée :** Dans les tests de logiciels, la couverture de condition ou décision modifiée est un critère de couverture de code qui requiert tous les éléments lors des tests.

  Chaque point d'entrée et de sortie est invoqué, chaque décision prend toutes les conséquences possibles, chaque condition d'une décision prend toutes les

conséquences possibles et chaque condition d'une décision affecte indépendamment l'issue de la décision.

- **Test de chemin et chemin principal :** La méthode analyse le graphe de flux de contrôle d'un programme pour trouver un ensemble de chemins d'exécution linéairement indépendants, comme indiqué par la formule ci-dessous :

Test de branche = (nombre de décision prises / nombre total de résultats de décision)*100%

2.4.2.2 Les avantages de tests structurels

Le test en boîte blanche est l'une des deux plus grandes méthodologies de test utilisées aujourd'hui. Le test de la boîte blanche présente plusieurs avantages majeurs qui sont résumés comme suit :

- Les effets secondaires liés à la connaissance du code source sont bénéfiques pour des tests approfondis.
- Optimisation du code en révélant les erreurs cachées et en éliminant ces éventuels défauts.
- Donne au programmeur l'introspection, car les développeurs décrivent soigneusement toute nouvelle implémentation, ce qui permet une traçabilité des tests depuis la source
- saisir facilement les modifications futures du logiciel lors des modifications apportées aux tests.
- Anticipation en effectuant des tests au cours du développement d'un programme et cela permet de révéler les point bloquants qui pourraient se transformer en erreurs ou problèmes dans le futur.
- Exhaustivité en permettant au testeur l'accès à son code pour une l'optimiser et d'apporter de meilleurs performances au système étudié.

Les tests de la boîte blanche sont faciles à automatiser. Ils fournissent des règles claires, basées sur l'ingénierie, et aussi la possibilité de déterminer des arrêts aux tests.

2.4.2.3 Les inconvénients de tests structurels

Bien que le test en boîte blanche présente de grands avantages, il n'est pas parfait et présente certains inconvénients qui sont résumés ci-dessous :

- La complexité des tests en boîte blanche, car le testeur doit avoir une connaissance du programme dans lequel le code est écrit.
- La possibilité de tester certaines conditions et non pas toutes les conditions.
- Les tests se concentrent sur le logiciel actuel et les fonctionnalités manquantes peuvent ne pas être découvertes.

Par conséquent, en raison de la résection temporelle, la zone critique du code / des fonctionnalités n'est pas testée de manière approfondie et le temps nécessaire aux tests est perdu dans une partie triviale du code.

Le tableau ci-dessous résume toutes les techniques de chaque méthode de test :

Méthode de test logiciel	**Techniques de test logiciel**
Test fonctionnel	• Test classe d'équivalence • Test All-pairs • Test aux limites • Graphe de cause/effet (table de décision) • Transition d'état • Test de cas d'utilisation • Analyse de domaine

Test structurel	• Test de flux de contrôle • Test de flux de données • Test de branche • Couverture de décision • Couverture de code • Couverture de condition • Couverture de relevés de comptes • Test de chemins principaux

Tableau 6: Méthodes et techniques de tests

La figure ci-dessous représente un schéma récapitulatif des différents tests effectué sur un logiciel pour tester sa validité :

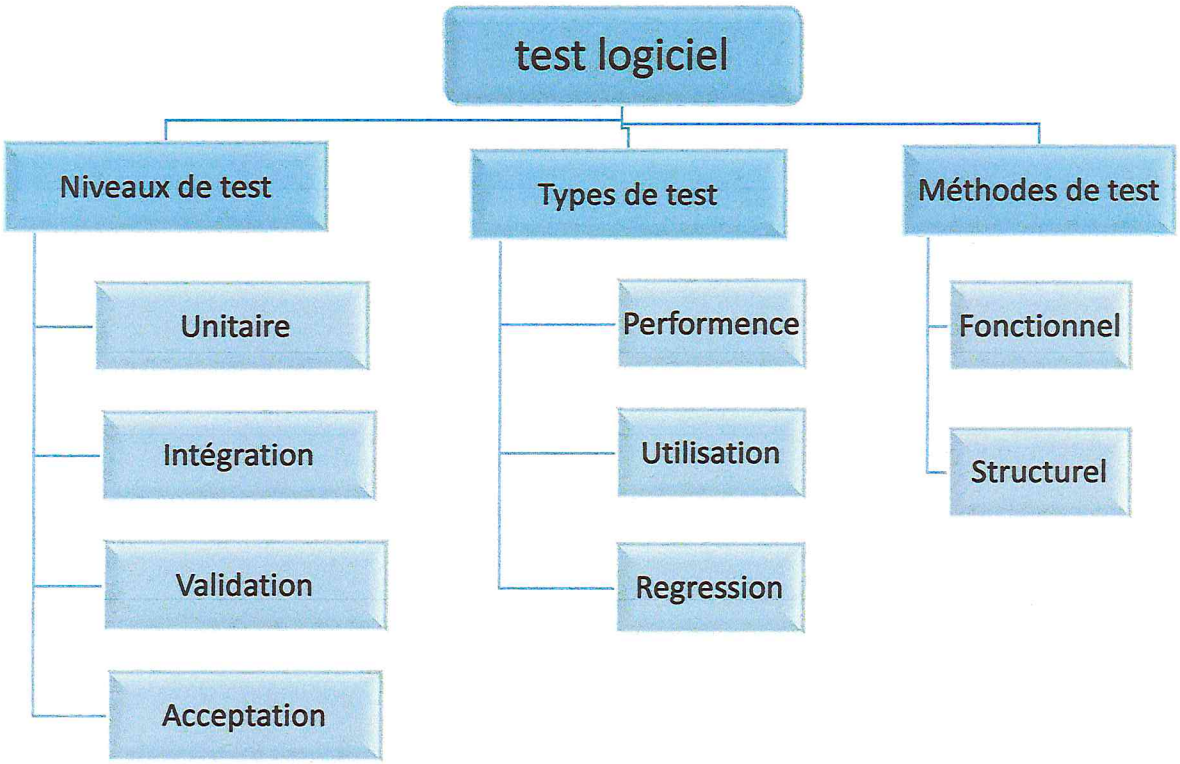


Figure 11: Tests logiciels

Conclusion

Les types de test de logiciel diffèrent des niveaux ou/et des méthodes. Contrairement aux types de test de logiciel répertoriés ci-dessus, les niveaux de test sont des tests effectués à différentes étapes du développement de logiciel et les méthodes de test sont effectuées lors de la réalisation de différents types de test de logiciel à différents niveaux de test de logiciel. Par exemple, nous pouvons effectuer un type de test lors d'un niveau de test à l'aide d'une méthode de test.

Partie II

La conception de l'outil de validation

Dans la deuxième partie, nous élaborons une approche de validation de logiciel avec laquelle nous allons vérifier la validité de l'outil DMSatisfaction, outil support au modèle de mesure de la satisfaction du décideur que nous avons pris comme cas d'étude et ce, par le biais d'un processus de validation que nous implémentons dans un outil support de validation de logiciel.

Chapitre 3

Approche de validation proposée

Dans ce chapitre, nous allons élaborer une approche de validation, dont laquelle une méthode de validation de logiciel est définit, ainsi que le processus suivi. Le processus de validation est inspiré du standard [IEEE 2012-1998].

3.1 Processus de validation

Un processus est défini pour représenter le déroulement de la validation d'un logiciel. Il commence par identifier le logiciel et se termine par la génération d'un rapport de validation décrivant l'identification de logiciel, les fonctionnalités et les cas de tests utilisés.

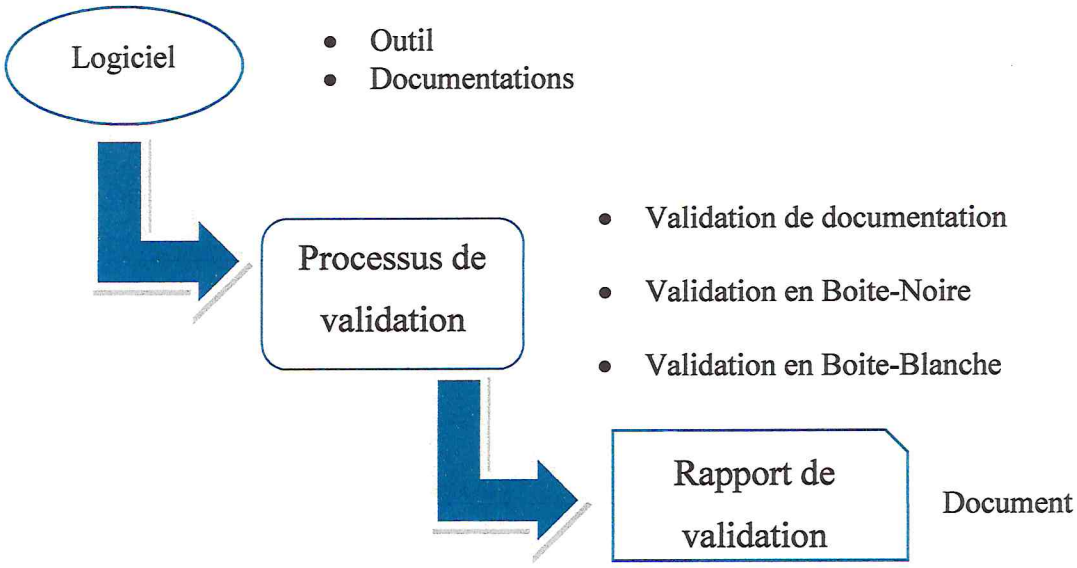


Figure 12: Déroulement de la Validation de logiciel

Selon le standard [IEEE 1998-2012], le processus de validation prend en charge six sous-processus, et dans chacun de ces sous-processus plusieurs tâches sont effectuées pour une validation logicielle ainsi que les techniques utilisées pour effectuer chaque tâche.

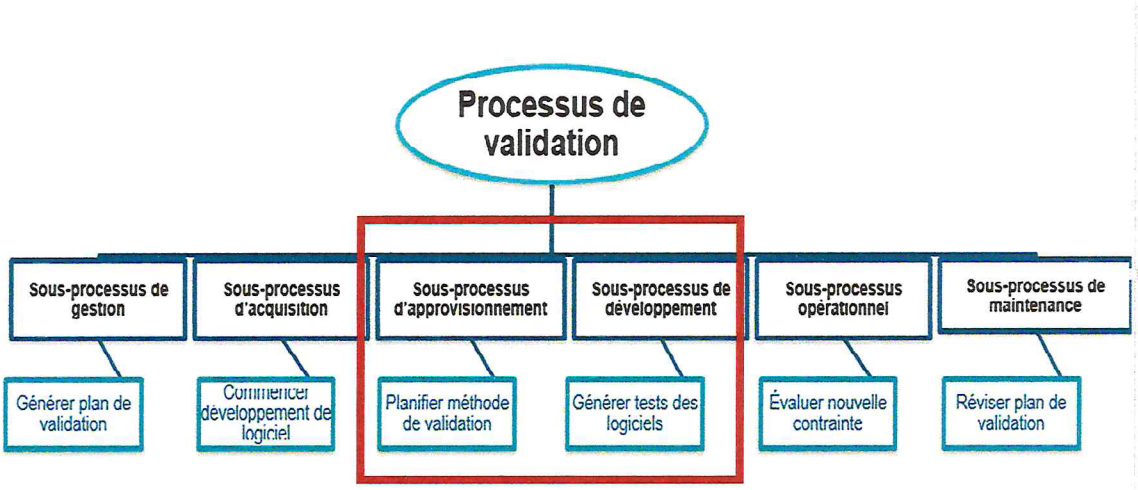


Figure 13: Sous-processus de validation retenus

3.2 Choix des sous-processus de validation

Le choix de ces deux processus se jusifie par la présence des méthodes et techniques appropirées à notre étude et nos besoins à savoir :

- La génération du plan de validation qui se trouve dans le processus de gestion
- la description de la méthode de validation qui se trouve dans le processus d'approvisionnement que nous jugeons nécessaire pour valider un logiciel.
- le panel des test et leur exécution que nous trouvons dans le processus de développement et dont nous avons besoin pour mener à bien notre travail.

Quant à l'élimination des autres sous process se justifie par l'absence de leur utilité, ils ne testent pas le cadre de notre travail qui consiste à valider un logiciel pour les raisons suivantes :

- Le développement de logiciel qu'on souhaite validé ce fait dans le sous processus d'acquisition,
- L'évaluation de l'utilisation final du logiciel par l'utilisateur dans un environnement opérationnel, ce que le processus opérationnel exploite
- L'évaluation du logiciel après une modification ou amélioration du système, ce qui déclenche le processus de maintenace

Sur ce fait, notre processus de validation contiendera les deux sous processus principaux, comme illustre la figure suivante :

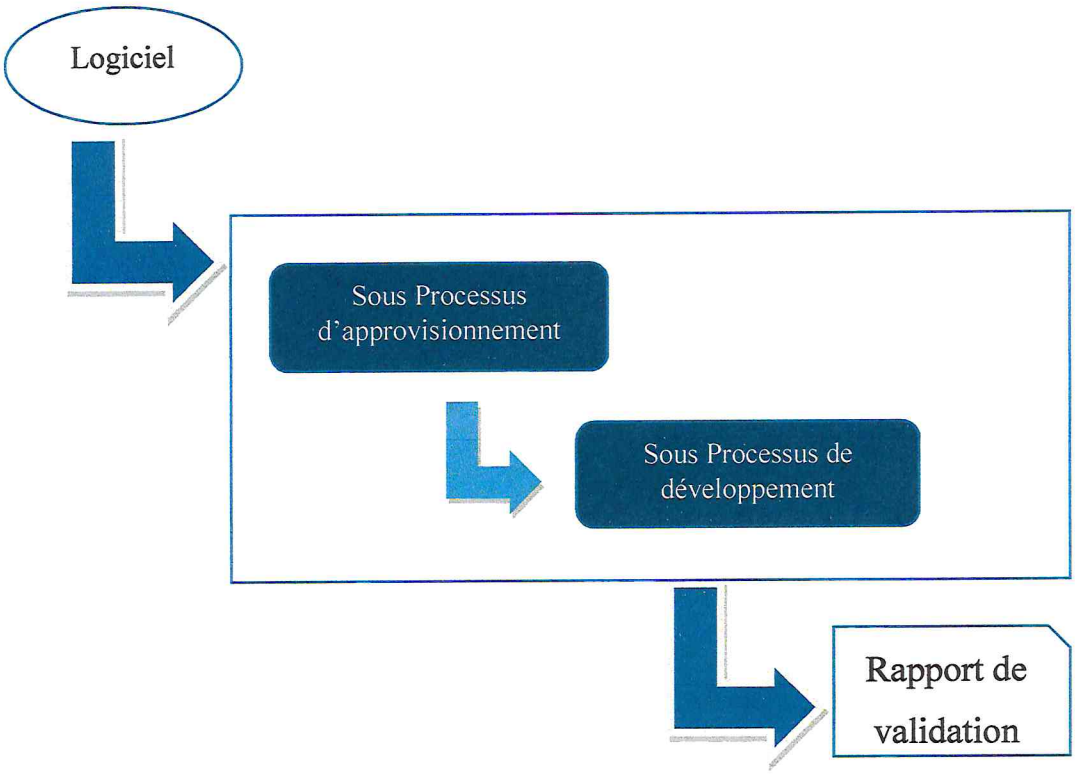


Figure 14: Vue détaillée du processus de validation

3.2.1 Sous-processus d'Approvisionnement

Au niveau du sous-processus Approvisionnement seulement une méthode de validation est planifié pour une meilleure validation. En s'inspirant du travail éffectué par [Jan et Benget 2004], nous proposons une validation qui se fait en trois phases :

- **Phase 1 :** Validation de Documentation en répondant à un questionnaire.
- **Phase 2 :** Validation en Boite noire qui est une validation globale de l'outil que nous souhaitons validé par une exécution successive de cet outil (introduction de jeux d'essai) et comparer le nombre de résultat positif par rapport au nombre de jeux d'essai effectué.

- **Phase 3 :** Validation en Boite blanche qui est validation du code source dans le cas de sa disponibilité.

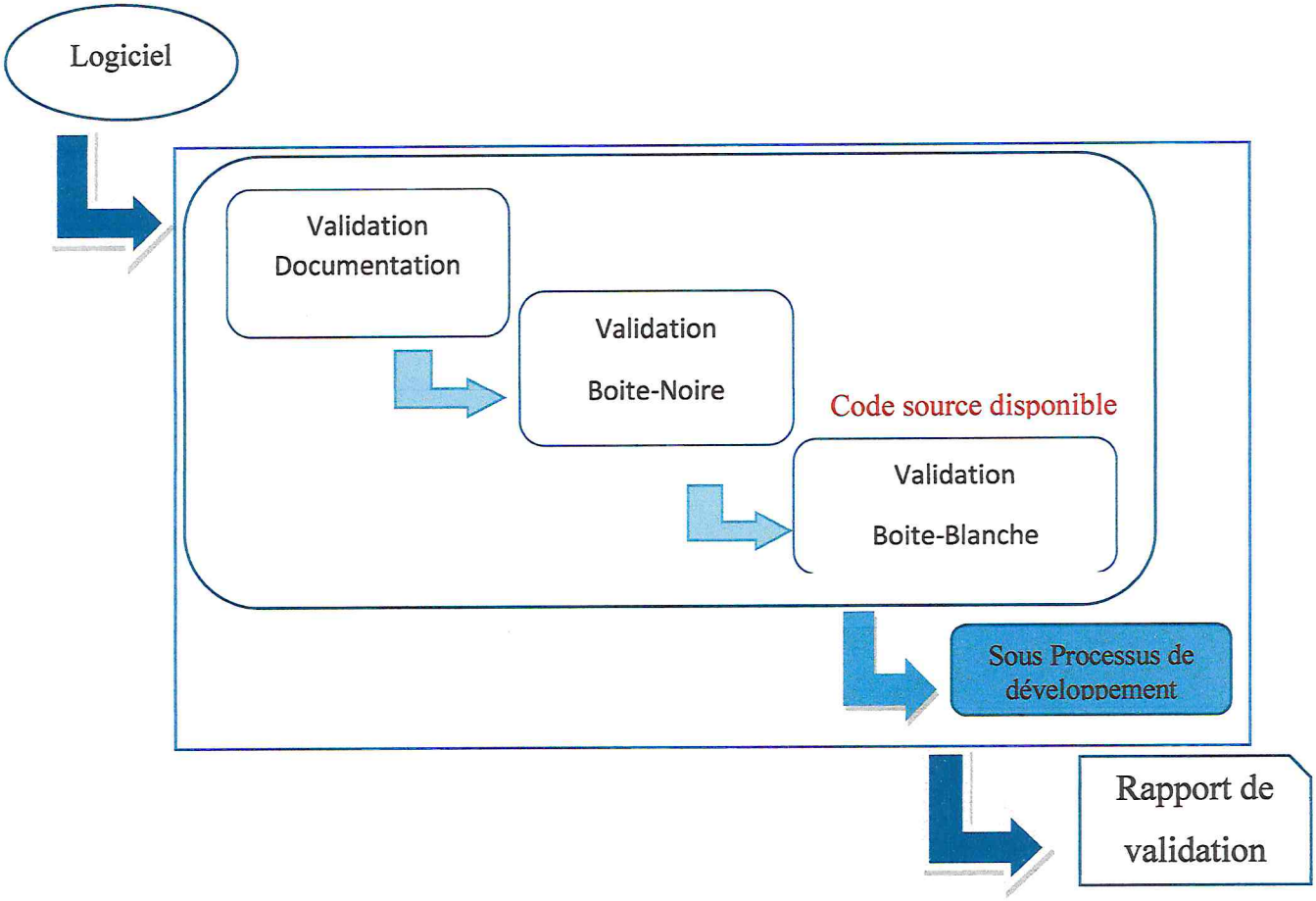


Figure 15 : Vue détaillée du processus Approvisionnement

3.2.2 Sous-processus Développement

Après la planification de la méthode de validation, nous passons à la deuxième étape qui consiste à planifier et organiser les tests logiciels que nous allons utiliser dans chaque phase de validation définit dans la section précédente

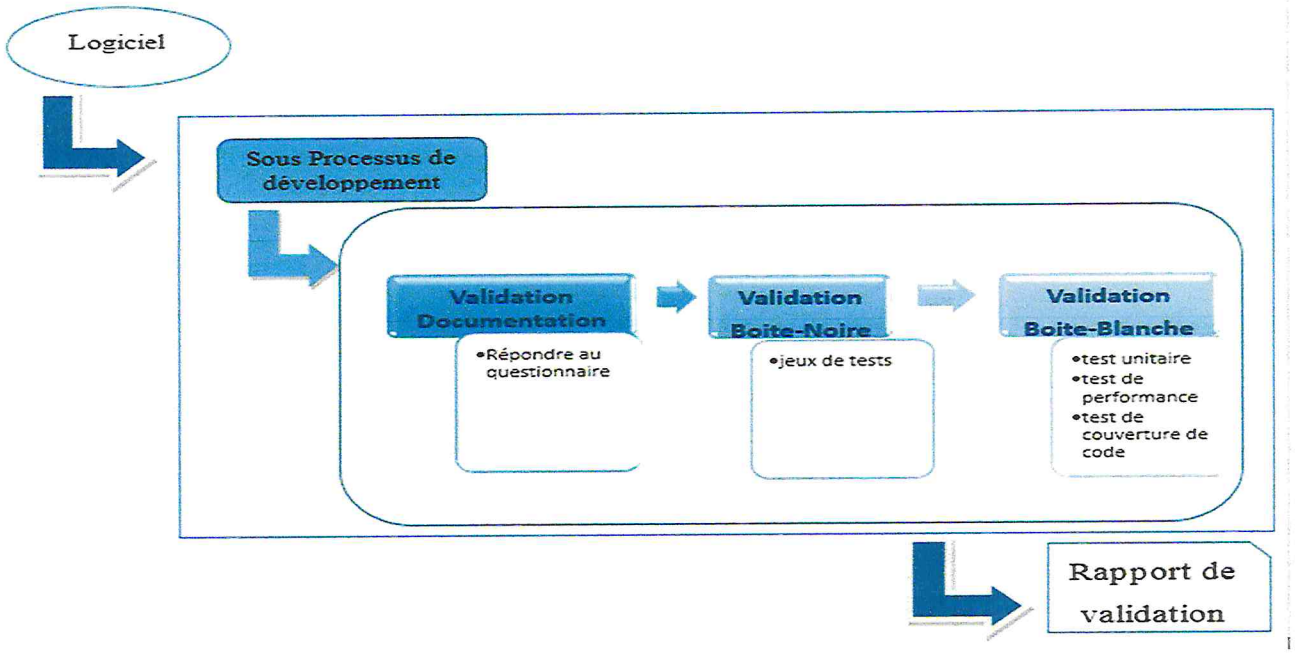


Figure 16 : Vue détaillée du sous-processus de développement

3.2.2.1 Phase 1 : validation de documentation

Le contrôle basé sur la documentation est la première phase pour la validation logicielle. Elle consiste en premier lieu à vérifier l'existence des différents documents décrivant l'outil support, tel que le manuel d'utilisation et le cahier des charges. Ce contrôle est fait d'une manière statique en répondant à une liste de questions (un questionnaire) inspiré des questionnaires élaboré par [Jan et Bengt, 2004].

Ainsi, le testeur doit vérifier d'une manière manuelle si toutes les fonctionnalités de l'application sont conformes aux exigences spécifiées dans le cahier des charges. La réponse à ces questions est faite par développeur de l'outil.

Un questionnaire extrait de la norme [ISO 17025] et appuyé par le travail effectué par [Jan et Benget, 2004], est utilisé pour supporter notre méthode de validation de documentation.

➢ **Le questionnaire :**

Question	Oui	Non	Commentaire
Des systèmes d'exploitation sont-ils utilisés ?			
Les outils de développement logiciel sont-elles été identifiés et documentés ?			
Existe-t-il un manuel d'utilisation ?			
Existe-t-il un manuel d'installation ?			
Existe-t-il des descriptions de la conception du logiciel ?			
Existe-t-il une documentation sur les fonctionnalités du logiciel ?			
Existe-t-il une description de la précision des algorithmes utilisés ?			
Existe-t-il un aperçu de l'aspect de sécurité, par exemple : les comptes des utilisateurs ?			
Toutes les exigences, sont-elles implémentées dans l'outil support ?			

Tableau 7: Questionnaire 1 « validation de documentation proposée »

Remarque

La réponse à ces questions est par OUI/NON, et le résultat final obtenu donne l'évaluation de cette phase de validation (valide/ non valide).

Selon le questionnaire précédant, nous concluons que le logiciel est dit valide dans cette première phase de validation après que le taux de réponses positives est supérieur aux taux de réponses négatif.

Si Taux des OUI > taux de NON ⟶ Valide

Sinon ⟶ invalide

3.2.2.2 Phase 2 : validation Boite-Noire

La validation Boite noire est la deuxième phase de validation de logiciel. Appeler aussi validation fonctionnelle car elle est basée sur des tests fonctionnels qui ont pour objectif de déterminer si l'unité fonctionnelle est conforme à ses exigences spécifier de logiciel.

Dans cette phase de validation le code source n'est pas connu par le testeur, c'est à dire qu'il contrôle le comportement externe du logiciel sans se préoccuper du design du code.

La validation est faite en introduisant un ensemble de jeux d'essai sur le logiciel, cela est fait en suivant ces trois étapes :

- Etape 1 : mettre le logiciel à son état initial,
- Etape 2 : exécuter le logiciel plusieurs fois et garder le résultat obtenu de chaque fonctionnalité,
- Etape 3 : comparer les résultats obtenus après chaque jeu d'essai de logiciel,
  Si : le résultat est le même après chaque exécution avec les mêmes variables d'entrées, alors le logiciel et valide dans cette phase.
  Sinon : un taux de validation est calculé en fonction de nombres d'échec et le nombre total d'essai.

3.2.2.3 Phase 3 : validation Boite-Blanche

La validation boite blanche est la troisième phase de validation, elle est basé sur la validation du code source de l'application en appliquant les différents tests logiciels sur chaque composant de logiciel puis l'ensemble des composants. Cette phase de validation se fait comme suit :

- Tester unitairement le bon déroulement de chaque fonction de logiciel que nous souhaitons validé, ce qu'on appelle « Test Unitaire » pour chaque fonctionnalité principale de logiciel. Ceci est assuré par un outil technique d'automatisation de ce test.
- Tester la performance de logiciel, ainsi que le temps de réponse de chaque requête de cet outil par un outil technique.

- Tester la couverture de code source de l'application et générer son rapport de validation par un outil automatique.

Le choix, la définition et la mise en pratique de ces outils techniques d'automatisations de test que nous avons utilisé dans cette phase de validation sont décrites dans le chapitre suivant.

Conclusion

Après le choix de la méthode de validation et ses différentes techniques, nous sommes passées à l'élaboration de l'approche de validation, pour ce faire nous avons choisis les deux sous processus d'Approvisionnement et de Développement inspiré du standard [IEEE 2012-1998] dont chaque sous processus est une succession de trois étapes de validation : documentation ensuite boite noire et enfin boite blanche pour l'implémentation de l'outil de validation

Chapitre 4

Architecture de l'outil de validation « ValidApp »

4.1 Introduction

Si un logiciel est validé, il peut assurer sa qualité et sa performance. Pour cela un système de validation de logiciel est conçu et implémenté. Il supporte un processus de validation extrait du Standard IEEE (software validation 2012). Dans ce présent travail nous développons un logiciel qui représente l'outil de validation logicielle qui va servir dans notre cas d'étude de valider l'outil « DMSatisfaction » l'outil support au modèle de mesure de la satisfaction du décideur par le biais de modèle multidimensionnel de D.Bouaissa et réalisé par les deux étudiantes H.Metat et F.Dahmani.

L'outil est une succession d'étapes par lesquelles passe un logiciel pour définir sa validité et pour enfin aboutir à un rapport de validation. Afin d'augmenter leurs chances de réussite, une méthode de développement de logiciel et des différents tests logiciels ont été définies, celles-ci permettent de mieux organiser et d'avoir une meilleure compréhension.

Pour réaliser notre outil nous présentons toutes les étapes de développements de l'outil qui sont : La spécification des besoins, l'analyse, la conception, l'implémentation, le test et la validation.

4.2 Choix de la méthode suivie

La représentation du processus de validation et la méthode choisie sont définis par une méthode de développement qui comprend [Muller, 1997] :

- Un processus de validation qui décrit les étapes à suivre pour valider un logiciel.
- Une notation qui assure le rendu visuel des éléments de modélisation.
- Des éléments de modélisation.

UML, la notation unifiée que nous avons utilisée, est basée sur les méthodes d'analyse et de conception. De ce fait, elle permet de couvrir le cycle de vie d'un logiciel depuis l'analyse du besoin jusqu'au test, ainsi que toutes les phases du développement :

- Les besoins des utilisateurs du système, exprimé à l'aide de diagrammes de cas d'utilisation.
- La spécification complète du système, sous forme de diagrammes.
- La conception globale et détaillée, jusqu'à un niveau proche du code en langage de programmation.
- Les suites de tests permettant de s'assurer qu'une implémentation candidate est effectivement conforme à la spécification élaborée dans la première phase.

Le cycle de vie en cascade [Royce, 1970] est le processus de développement que nous avons suivi pour la description générale des activités liées aux logiciels [Muller, 1997]. Il décrit le cycle de vie d'un logiciel par une suite de phase qui s'enchainent dans un déroulement linéaire (Figure 18), depuis l'analyse des besoins jusqu'à la maintenance.

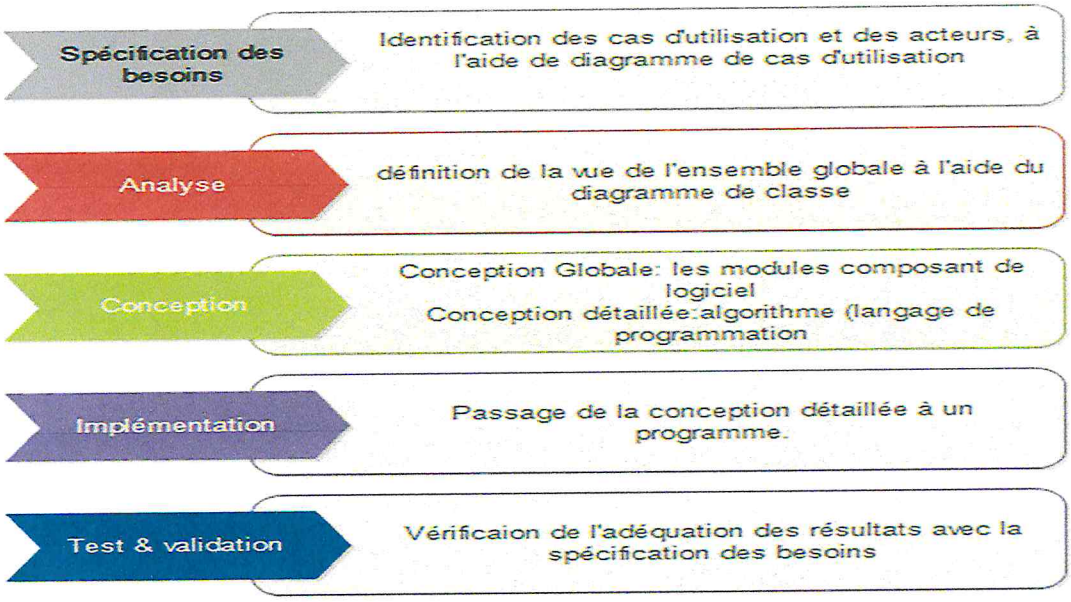


Figure 17: Cycle de vie de logiciel en cascade

4.3 Etape 1 : La spécification des besoins

La première étape du développement consiste à décrire en générale les fonctionnalités et les utilisateurs du système. Le comportement du système exprimé sous la forme des cas d'utilisation, le contexte du système et les acteurs.

4.3.1 Besoins fonctionnels

Les besoins fonctionnels s'agissent des fonctionnalités du système. Ce sont les besoins spécifiant un comportement d'entrée et de sortie de ce système pour que ce dernier soit opérationnel. Les principales fonctionnalités que couvre notre application sont les suivantes :

- Identifier le logiciel objet de la validation.
- Valider le logiciel.
- Analyser le résultat de la validation.

Les cas d'utilisations ci-dessous représentent ces besoins fonctionnels.

4.3.1.1 Identification des acteurs

Les utilisateurs de notre système sont :

- **Un testeur :** est un utilisateur qui est chargé d'effectuer les différents tests sur le logiciel que nous souhaitons valider, il est donc l'acteur principal du système.
- **Un administrateur :** est un acteur secondaire qui est chargé d'effectuer une maintenance sur notre système si nécessaire.

Le tableau ci-dessous définir les taches de chaque acteur du système :

Acteur	Testeur	Administrateur
Taches	• S'authentifier • Inspecter le cahier des charges (par un questionnaire) • Evaluer les fonctionnalités de l'outil support et les comparer avec le cahier des charges • Effectuer des jeux de tests sur l'outil • Tester la performance de l'outil • Tester la couverture de code • Tester le code source de l'outil • Communiquer les résultats avec le client (propriétaire de logiciel que nous allons valider)	• Intervient en cas de problème survenu lors de l'utilisation de l'application Mettre à jour les cas de test si nécessaire

Tableau 8: Les taches associées aux différents acteurs du système

4.3.1.2 Identification des messages échangés

La communication entre les différents acteurs du système est faite avec un échange de message entre eux. Un message définit un évènement ou une information envoyée à un acteur et provoquant en réponse le début d'une action associées à ce dernier.

Deux types de messages ont distingué pour assurer la communication, les messages entrants représentent les demandes qu'un acteur effectue et les messages sortants représentent la réponse du système à une demande.

Ces messages seront utilisés par la suite dans un diagramme d'activité.

Acteur	Messages entrants	Messages sortants
Testeur	Demande d'authentification	Formulaire d'enregistrement
	Authentification	Formulaire d'authentification
	Remplir formulaire d'identification de logiciel	Formulaire d'identification logiciel
	Tester et valider l'outil support	Trois différentes interfaces contenants les différents type de test pour valider l'outil support
	Evaluer le résultat de test	Interface contenant le résultat de tous les tests effectués
	Afficher le résultat de validation	Interface de « validation »
Administrateur	Résoudre les problèmes survenus lors de l'utilisation de l'application de validation	Accès au code source de l'application

Tableau 9: Messages échangés avec le système

4.3.1.3 Identification des cas d'utilisation

L'ensemble de cas d'utilisation est formé par les différentes fonctionnalités de notre outil. La structure des fonctionnalités que notre système offrira à ses utilisateurs est proposée par les digrammes *(« Use Case »)* UML, ainsi que Les interactions et les dialogues des utilisateurs (acteurs) effectués avec le système.

Le tableau ci-après décrit l'ensemble des cas d'utilisation du système à réaliser :

N°	Cas d'utilisation			Acteur
01	S'authentifier		Nom d'utilisateur et mot de passe	Testeur
02	Identifier logiciel		Remplir formulaire d'identification	Testeur
03	Valider Logiciel	Valider Documentation	Répondre au questionnaire présenté dans l'interface par oui/non	Testeur
		Valider Boite-Noir	• Jeux d'essai sur l'outil • Comparer nombre d'échec avec nombre de jeux d'essai	Testeur
		Valider Boite-Blanche	• Test unitaire de chaque fonction • Test de performance • Test de couverture de code	Testeur
04	Analyser le résultat		Analyser et communiquer le résultat des trois phases de validation	Testeur
05	Effectuer la maintenance		Résoudre tous les problèmes rencontrés par le testeur	Administrateur

Tableau 10 : Les cas d'utilisation associés au système à développer

4.3.1.4 Diagramme de cas d'utilisation

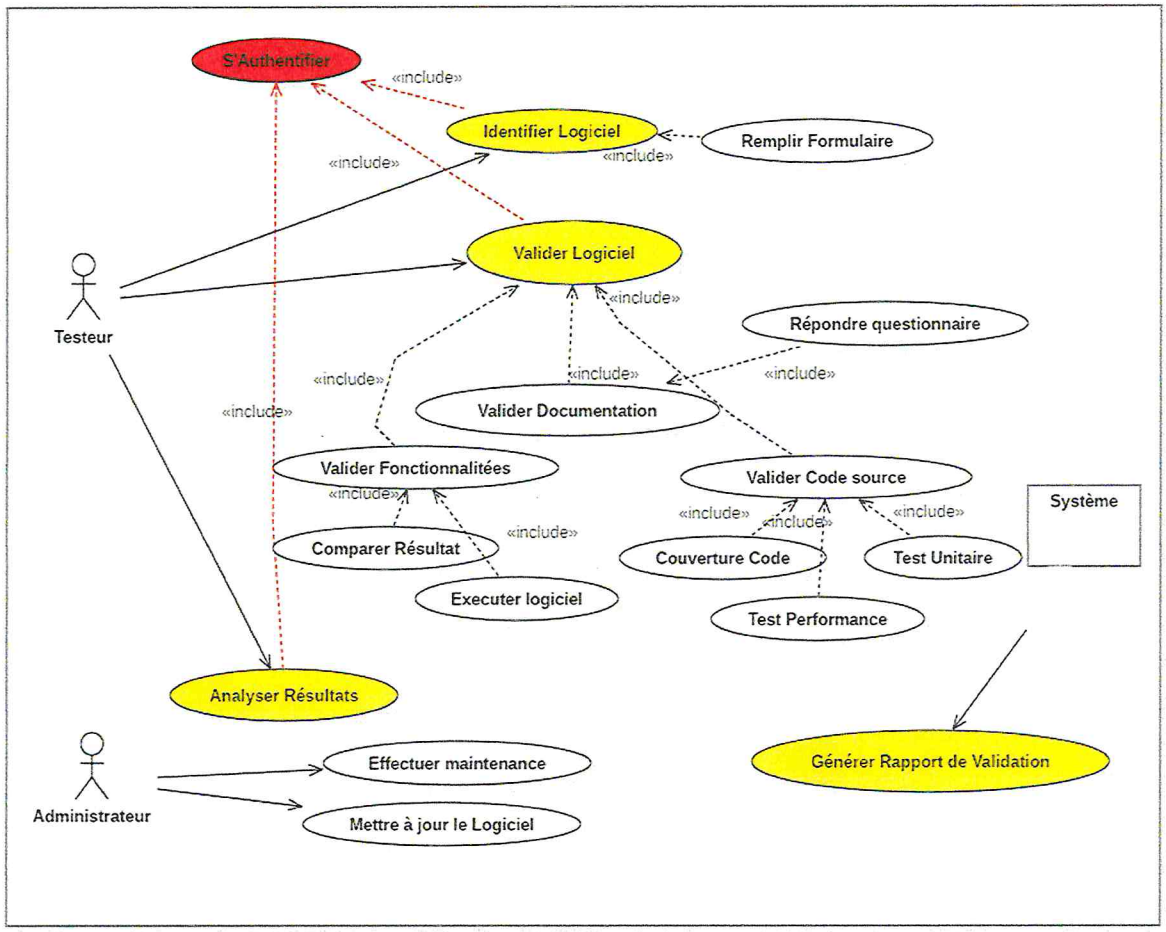


Figure 18: Diagramme de cas d'utilisation de l'outil de validation proposé

4.3.2 Besoins non fonctionnels

Les besoins non fonctionnels s'agissent des besoins qui caractérisent le système. Ce sont des besoins en matière de performance, de type de matériel ou le type de conception, et identifient les contraints internes et externes du système. Les points suivants résument les principaux besoins non fonctionnels de notre application :

- La sécurité : la confidentialité des données doit être respectée.

- L'ergonomie : une meilleure densité d'éléments sur l'écran et une interface simple à utiliser
- L'intégrité : la cohérence des données lors des modifications
- La clarté et la simplicité du code source pour des futures améliorations.

4.4 Etape 2 : L'analyse

La deuxième étape du développement consiste à recueillir et formaliser les besoins spécifiés dans l'étape précédente en un ensemble contrainte, ce qui donne une vue d'ensemble globale de notre système, ainsi que ses composants et les relations entre ces derniers. Ceci est exprimé par un diagramme de classe.

4.4.1 Diagramme de classe

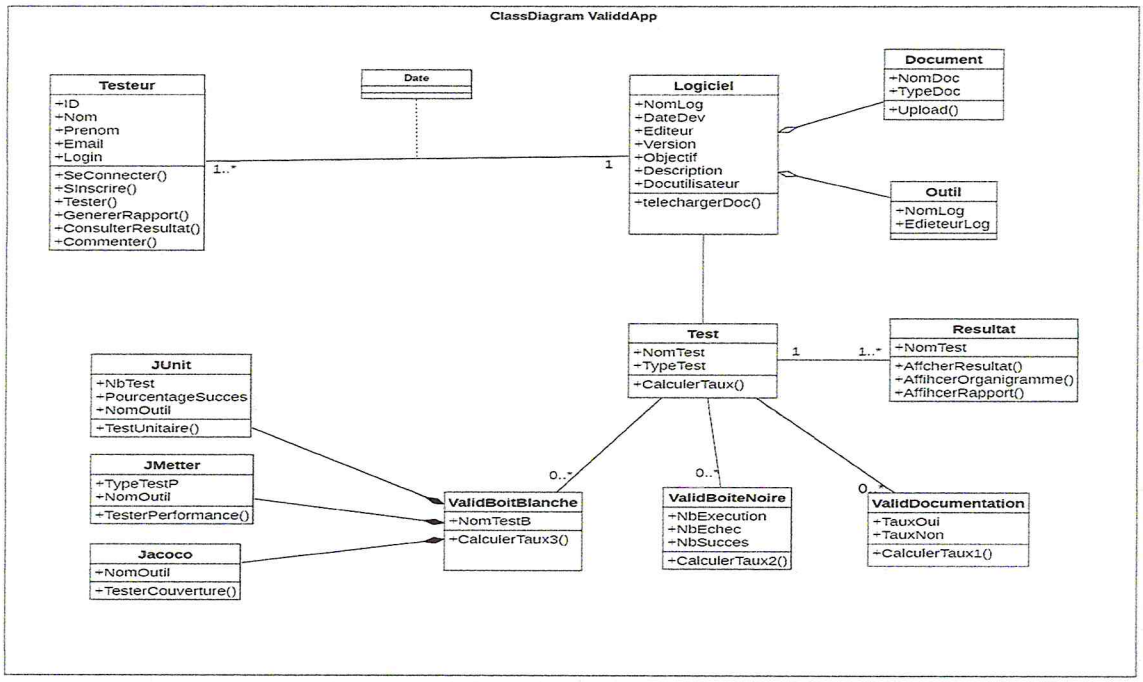


Figure 19: Diagramme de classe de l'outil proposé

4.5 Etape 3 : La conception

La conception de notre système se décompose en deux parties :

- La conception globale (architecturale) : un schéma général est élaboré pour définir le système.
- La conception détaillée : une description détaillée de chaque fonctionnalité du système.

4.5.1 La conception globale (architecturale)

Il s'agit de l'élaboration des spécifications de l'architecture générale de notre application, ce qui le montre la figure ci-dessous :

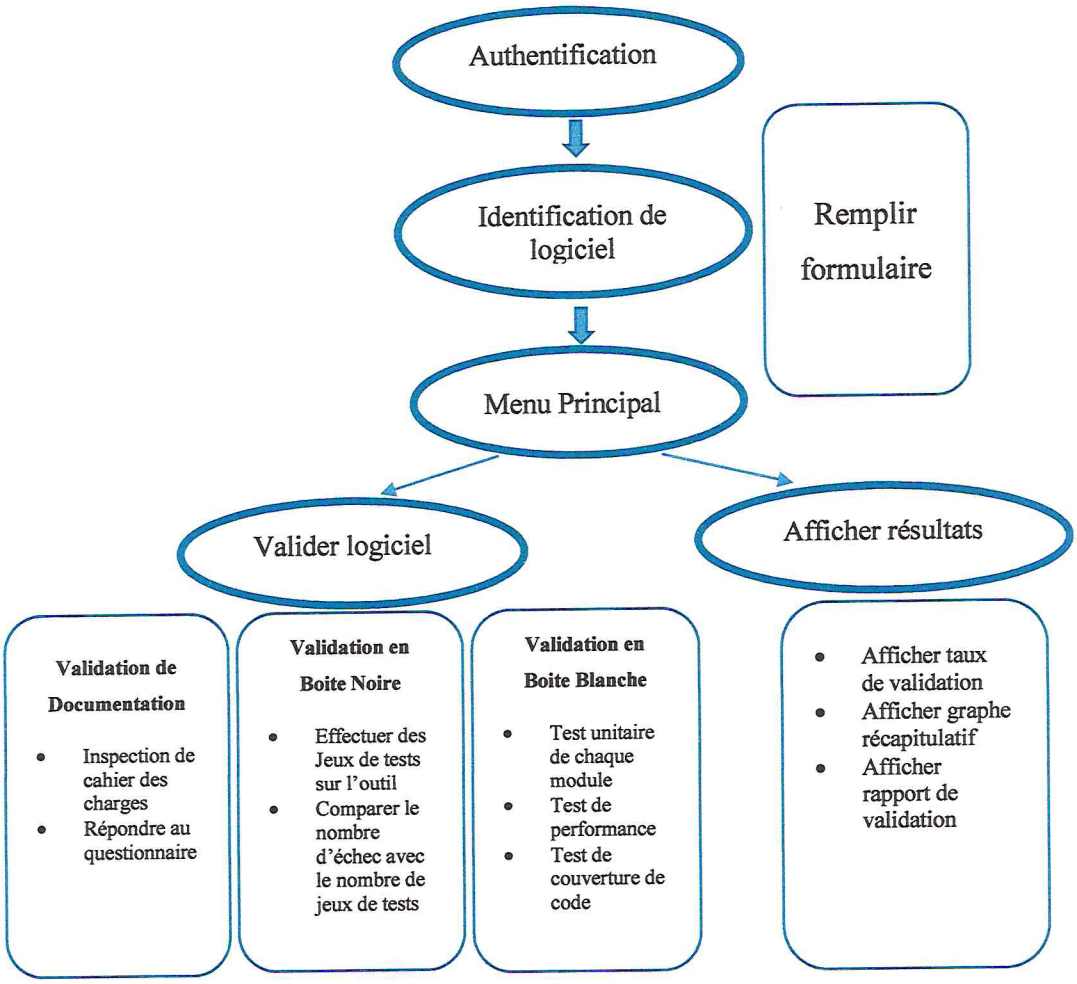


Figure 20: Conception globale de l'outil proposé

La clé du schéma ci-dessus représente les différents composants de notre système, à savoir :

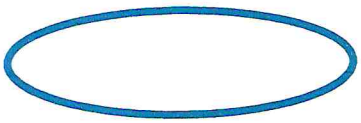


Figure 22: Module

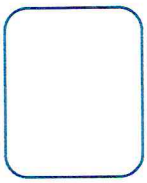


Figure 21: Sous Module

Le tableau ci-dessous représente les fonctionnalités de chaque module du schéma précédant :

Module	Sous module	Fonctionnalités
Authentification	Créer	Accéder à l'application (Obligatoire)
	Supprimer	
Identification du logiciel	Formulaire	Un formulaire à remplir contenant toutes les informations nécessaire concernant le logiciel
Menu principal	Valider Logiciel	Interface principale du système regroupant toutes les fonctionnalités
	Analyser Résultat	
Valider logiciel	Validation Documentation	**La fonction principale** : mesurer le taux de validation d'un logiciel après passer par les trois phases de validation
	Validation Boite-Noire	
	Validation Boite-Blanche	
Analyser résultat	Résultat final de validation	Procurer le graphique et le résultat final de la validation et générer le rapport de validation
	Graphique de résultat	
	Rapport de validation	

Tableau 11: Modules, sous modules et fonctionnalités du système proposé

4.5.2 La conception détaillée

Les fonctionnalités de chaque sous module de notre système est précisé et détaillé dans cette section, nous détaillons aussi le fonctionnement de la méthode principale de la validation de « l'outil support qui mesure la satisfaction du décideur ».

4.5.2.1 Fonctionnalités

Notre outil de validation est composé d'un ensemble de fonctionnalités, nous expliquons l'objectif de chacune par la suite et nous détaillons le déroulement de chaque cas d'utilisation avec une description textuelle.

➢ **Authentification** : un identifiant et un mot de passe sont utilisés pour accéder aux différents modules de notre système.

Cas d'utilisation N°1	S'authentifier
Résume	Vérification de l'identité des utilisateurs (Login et mot de passe)
Acteurs	Utilisateurs (testeur et administrateur)
Précondition	Utilisateur déjà enregistré dans la base de données
Scénario nominal	[début] 1. Accès à l'application ; 2. Le système affiche un formulaire d'authentification ; 3. L'utilisateur saisit le nom d'utilisateur et mot de passe ; Si les champs sont incomplets : Alors afficher un message d'erreur ; Si non Aller à (4) ; 4. Le système vérifie la validité des informations fournies ; S les champs sont incorrects : Alors afficher un message d'erreur ; Si non Aller à (5) ; 5. Le système donne l'accès à l'interface correspondante. [fin]
Alternative exceptionnelle	Le système affiche un message d'erreur et réaffiche la boite d'authentification et attend que l'utilisateur ressaisisse ses informations.

Tableau 12: Description du cas d'utilisation "S'authentifier"

➢ **Identification de logiciel** : un formulaire contenant toutes les informations que nous avons jugé nécessaire pour identifier un logiciel.

Cas d'utilisation N°2	Identification de logiciel
Résume	Remplir le formulaire d'identification de logiciel
Acteurs	Testeur
Précondition	Testeur authentifié
Scénario nominal	[début] 1. Accès à l'application ; 2. S'authentifier ; 3. Accéder au formulaire d'identification de logiciel ; 4. Le système affiche le formulaire ; 5. Le testeur remplit les champs du formulaire (nom, la version, date de création, le développeur, objectif et description de logiciel) ; 6. Le testeur enregistre les informations ; Sinon : Le testeur annule l'enregistrement ; aller à la page d'accueil ; Si les champs sont incomplets : Alors afficher un message d'erreur ; Si non Aller à (7) ; 7. Le système enregistre les informations fournies ; 8. Le système donne un identifiant à logiciel ; 9. Le système donne l'accès à l'interface suivante (menu principale). [fin]

Alternative **Exception**	Le système affiche un message d'erreur et réaffiche le formulaire et attend que l'utilisateur ressaisisse les informations.

Tableau 13: Description du cas d'utilisation "Identification de logiciel"

Formulaire : Identification de logiciel

L'identification de logiciel est l'étape initiale pour pouvoir accéder au menu principale.
Le formulaire contient les informations suivantes que nous avons jugées nécessaire pour une identification claire de logiciel que nous allons valider :

- Nom de logiciel.
- Date de création.
- Editeur.
- Développeur qui a créé cette application.
- Version de logiciel.
- Domaine d'application // informatique, biologie, économie…
- Objectif de l'outil.
- Description générale de l'outil.
- Documentation utilisateur (à cocher) : procédure d'installation, manuel utilisateur, cahier des charges, s
- Documentation techniques (à cocher) : modèle de cas d'utilisation, plan de validation,
- Code source spécifier de programmation : java, c++, c# …

Ensuite ces informations sur le logiciel seront enregistrées et affichées dans le rapport de validation que le système génère.

Ce module donne accès au «**Menu principal**» (Tableau 13) dans lequel nous retrouvons :

Cas d'utilisation N°3	Menu principal
Résume	Englobe les deux fonctionnalités du système (valider logiciel et analyser résultats)
Acteurs	Testeur
Précondition	Authentification requise
Scénario nominal	[début] 1. Accès à l'application ; 2. S'authentifier ; 3. Accéder au menu principal 4. L'utilisateur choisit une action souhaitée (valider logiciel, analyser résultats) ; 5. Accéder à l'interface choisis ; [fin]
Alternative exceptionnelle	Le système affiche un message en cas d'erreurs.

Tableau 14: Description du cas d'utilisation " Menu Principale"

- **Valider le logiciel** : c'est la fonction principale de notre système, qui consiste à valider un logiciel.

Sous module N°1	Validation de documentation
Résume	Répondre à un questionnaire
Acteurs	Testeur
Précondition	Identification de logiciel est déjà enregistrée dans la base de données et authentification requis.
Scénario nominal	[début]

	1. Accéder à la validation de documentation ; 2. Le système affiche un questionnaire de validation de documentation ; 3. Le testeur répond au questionnaire ; 4. Le testeur enregistre ses réponses ; Si les champs sont incomplets : Alors afficher un message d'erreur ; Si non Aller à (5) ; 5. Le système calcule le taux de validité de cette première phase de validation ; Si testeur annule : Alors aller à (6) Si non aller à (1) 6. Le système donne l'accès à l'interface suivante. [fin]
Alternative **Exception**	Le système affiche un message d'erreur et réaffiche le questionnaire et attend que l'utilisateur remplit le questionnaire à nouveau

Tableau 15: Description du cas d'utilisation " Validation Documentation"

Ce module est une succession de trois étapes de validation :

- **Validation de documentation** : le développeur donne le logiciel et le cahier des charges au testeur, ce dernier vérifie la conformité entre le logiciel et son cahier des charges en répondant à un questionnaire.

Sous module N°1	Validation de Documentation
Résume	Tester et valider en répondant à un questionnaire

Acteurs	Testeur
Précondition	Authentification requise, Logiciel déjà enregistré dans la base de données.
Scénario nominal	[début] 1. Accéder à la validation de documentation ; 2. Répondre au questionnaire affiché ; 3. Enregistrer les réponses ; 4. Passer à la phase suivante ; [fin]
Alternative Exception	Le système affiche un message d'erreur

Tableau 16: Description du cas d'utilisation "Validation de Documentation"

Méthode de calcul

Le taux de validation est le pourcentage de la division du nombre de *oui* par le nombre des questions.

- **Validation en Boite Noire** : cette deuxième phase consiste à effectuer des jeux de tests sur le logiciel et comparer le nombre de jeux d'essai avec le nombre d'échec.

Sous module N°2	Validation Boite-Noir
Résume	Tester et valider l'outil en effectuant plusieurs exécutions.
Acteurs	Testeur
Précondition	Authentification requise, Logiciel déjà enregistré dans la base de données, et passage obligatoire par la première phase.
Scénario nominal	[début]

	1. Accéder à la validation en boite noire ; 2. Suivre les étapes ; 3. Effectuer des jeux d'essai sur l'outil DMSatisfaction ; 4. introduire le nombre de jeux d'essai ; 5. introduire le nombre d'échec ; 6. Enregistrer les réponses ; 7. Passer à la phase suivante ; [fin]
Alternative Exception	Le système affiche un message d'erreur

Tableau 17: Description du cas d'utilisation "Validation Boite-Noire"

Méthode de calcul

Pour le calcul du taux de validation de cette phase, il nous faut connaitre si les résultats obtenus convient aux besoins. Si la réponse est « Oui » on obtient un taux de 100% ; si non le taux est calculé ainsi :

Le résultat de la division du produit de la différence entre le nombre total des exécutions et le nombre des exécutions échouées par 100 et le nombre total d'exécutions.

- **Validation en Boite Blanche :** le code source de logiciel est obligatoire dans cette troisième phase de validation.

Sous module N°3	Validation Boite-Blanche
Résume	Tester chaque unité puis l'ensemble des unités de logiciel, tester la performance ainsi que la couverture du code de logiciel
Acteurs	Testeur

Précondition	Authentification requise, Logiciel déjà enregistré dans la base de données, et passage obligatoire par les deux phases précédentes
Scénario nominal	[début] 1. Accéder à la validation Boite-Blanche ; 2. Test unitaire des classes de l'outil ; 2.1 tester 1ere unité : « Gestion d'utilisateur » ; -Ajouter -Modifier -Chercher -Supprimer 2.2 tester 2ème unité : « Gestion de la satisfaction » ; - méthode de la satisfaction - l'affichage des évaluations -choix du système 3. Test de performance « JMeter » ; 4. Test de couverture de code avec « JaCoCo » ; 5. Appuyer sur suivant pour passer à la phase de résultat ; [fin]
Alternative exceptionnelle	Le système affiche des messages d'erreurs

Tableau 18 : Description du cas d'utilisation " validation Boite-Blanche"

Fonctionnement du test de performance et de couverture de code

- Pour tester la performance du logiciel nous passons par « JMeter » suivant cet enchainement :

- Création d'un « Test Plan » est l'espace de travail où on va mettre les éléments qui constituent notre plan de test.
- Ajouter un « Thread Group » est l'élément de base pour un scénario.
- Ajouter d'un « Sampler » qui permet de définir le type de requête échangée avec le serveur (HTTP, FTP, JDBC, JAVA...) et de les émettre.
- Ajouter un « Listeners » qui permet de récupérer et d'afficher les résultats des tests (par exemple : fichiers, graphes).
- Analyser les résultats.

➢ Pour tester la couverture de code du logiciel nous passons par « jaCoCo » suivant cet enchainement :

- Installer le plugin JaCoCo sur Netbeans.
- Exécuter le package de projet avec JaCoCo en appuyant sur le bouton droit et en choisissant « run with JaCoCo ».
- Une page Html s'affiche automatiquement contenant un tableau de résultats.
- Analyser le rapport des résultats de la page html.

Méthode de calcul

Le taux de validation des trois tests effectué est calculé automatiquement par les outils techniques choisis (JUnit pour les tests unitaires, JaCoCo pour la couverture de code et JMeter pour les tests de performance), ensuite le taux total de cette phase est la division du produit de la somme de taux des trois tests par 100 sur 3.

➢ **Consulter les résultats** : ce module procure le résultat de chaque phase de validation sous forme de pourcentage et de graphique récapitulatif, ainsi qu'un rapport de validation sera générer.

Cas d'utilisation N°5	Analyser les résultats
Résume	Analyser les résultats des trois phases de validation précédente
Acteurs	Testeur
Précondition	Les trois phases de validation enregistrée Le tester doit faire l'évaluation du résultat obtenue

Scénario nominal	[début] 1. Accéder aux résultats ; 2. Le système calcule le résultat des trois phases de validation ; 3. Le système affiche le taux de validation de chaque phase ; 4. Le système affiche le taux totale de validation en pourcentage ; 5. Le système génère le rapport de validation ; 6. Le testeur imprime de rapport de validation ; [fin]
Alternative **Exception**	Le système affiche un message d'erreur.

Tableau 19: Description du cas d'utilisation " Analyser Résultat"

Méthode de calcul

Le taux de total de validation de l'outil est la division de la somme des trois taux de chaque phase par 3.

Conclusion

Dans ce présent chapitre nous avons présenté les étapes de développement de notre outil, en utilisant une démarche dans un environnement orienté objet à savoir la modélisation et la conception en suivant le modèle en cascade.

Pour développer notre outil nous avons commencé par la spécification des besoins, qui décrit les fonctionnalités du système présentés par les diagrammes des cas d'utilisation et de séquence. Puis nous avons entamé l'étape d'analyse pour déterminer les éléments

intervenants dans notre système. Enfin la troisième étape est la phase la plus importante dans le développement de l'outil, elle commence par une conception globale qui décrit l'architecture du système suivie par une autre détaillée. Les deux dernières étapes du modèle en cascade, l'implémentation, test et validation se trouvent dans le chapitre suivant.

Chapitre 5

Implémentation de l'outil de validation « ValidApp »

Le dernier chapitre de notre mémoire est la quatrième étape du modèle en cascade. Il s'agit d'implémenter la solution retenue au niveau de la phase de conception du chapitre précédant, c'est la phase au cours de laquelle les structures et les algorithmes définis pendant la conception sont traduits dans un langage de programmation et une BDD. Ainsi que la mise en œuvre de l'outil support au modèle de mesure de la satisfaction du décideur.

5.1 Choix du langage de programmation

Dans la réalisation d'un logiciel, le choix du langage de programmation est une décision très importante. Pour l'implémentation de notre système, nous avons utilisé le langage de programmation orienté objet JAVA, car ce langage permet au programmeur d'avoir un niveau d'abstraction très haut par rapport à la machine, et d'avoir une application bien structurée, modulable, maintenable et efficace.

5.2 Outils techniques

5.2.1 Environnement de développement

Netbeans IDE « l'environnent de développement intégré, libre, extensible et universel, qui constitue une plateforme permettant le développement des applications et prenant en charge de divers langages de programmation», et est l'outil technique que nous avons utilisé pour implémenter notre application.

5.2.2 Outils de tests

Dans la troisième phase de validation d'un logiciel « Boite-Blanche », les outils de test que nous avons utilisé sont Junit, JMeter et JaCoCo.

A. JUnit

Définition

JUnit est un Framework open source pour le développement et l'exécution de tests unitaires automatisables et qui est intégré dans la plus part des IDE. Junit ne fait pas partie de la librairie standard de Java, mais plutôt une partie d'un ensemble plus grand nommé **XUnit** qui désigne tous les Framework de test répondant à certains critères pour une multitude de langage. Le principal intérêt est de s'assurer que le code répond toujours aux besoins même après d'éventuelles modifications. Plus généralement, ce type de tests est appelé tests unitaires de non-régression.

JUnit propose :

- Un Framework pour le développement des tests unitaires reposant sur des assertions qui testent les résultats attendus.
- Des applications permettant l'exécution des tests et l'affichage des résultats.

Mise en pratique

Pour simplifier la maintenance du code et le packaging de notre logiciel, nous allons créer deux packages principaux : **main** et **test**.

Dans la classe **main**, nous intégrons toutes nos classes du logiciel et dans la classe **test** nous intégrons toutes les classes de test.

Le fonctionnement de JUnit est expliqué dans l'exemple ci-dessous, à savoir :

1. Dans un premier temps, nous allons créer la classe à tester « Calculate » :

```java
package com.javacodegeeks.junit;

public class Calculate {

    public int sum(int var1, int var2) {
        System.out.println("Adding values: " + var1 + " + " + var2);
        return var1 + var2;
    }

}
```

Dans le code source ci-dessus, nous pouvons remarquer que la classe Calculate a une méthode publique nommée *somme ()*, qui a en entrées deux entiers et renvoie en sortie un entier. Donc, nous allons tester cette méthode.

2. Ensuite, nous allons créer la classe test « CalculateTest » :

La classe de test « CalculateTest » contient les méthodes permettront de tester chacune des méthodes de la classe précédente (dans ce cas, nous avons une seule méthode à tester *somme()*).

```java
package com.javacodegeeks.junit;

import static org.junit.Assert.*;

import org.junit.Test;

public class CalculateTest {

    Calculate calculation = new Calculate();
    int sum = calculation.sum(2, 5);
    int testSum = 7;

    @Test
    public void testSum() {
        System.out.println("@Test sum(): " + sum + " = " + testSum);
        assertEquals(sum, testSum);
    }

}
```

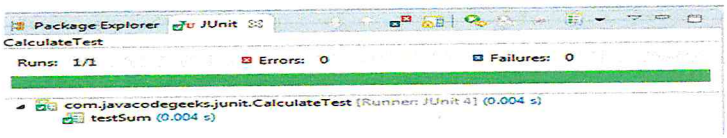
Explication du déroulement du test

Tout d'abord, nous pouvons voir qu'il y a une annotation **@Test**, Cette annotation indique que la méthode *public* à laquelle elle est attaché peut être exécuté comme un cas de test. Par conséquent, la méthode ***testSum ()*** est la méthode qui permettra de tester la méthode *somme ()*.

Nous avons aussi une autre méthode appelée ***assertEquals (somme, testsum)*** qui prend en entrée deux objets et affirme que les deux objets sont égaux.

Le lancement du test est fait par un clic droit sur la classe **Run As -> Junit Test,** et le résultat est affiché comme suit :

```
1 Adding values: 2 + 5
2 @Test sum(): 7 = 7
```



B. JMetter

Définition

L'application Apache JMeter est un logiciel open source, une application Java à 100 % pour charger le comportement fonctionnel des tests et mesurer les performances, ainsi que le temps de réponse de chaque requête et produire leurs statistiques

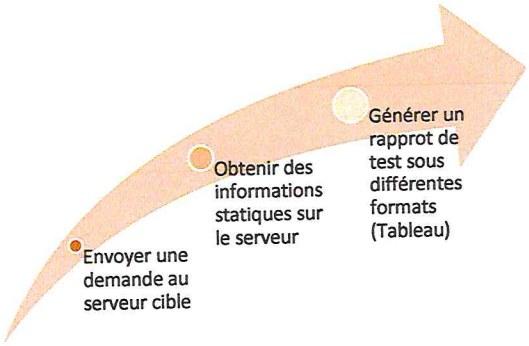
Les fonctionnalités d'Apache JMeter incluent :

- Capacité à charger et à tester les performances de plusieurs types d'application y compris Java comme dans notre cas d'étude,
- IDE de test complet qui permet un enregistrement, un développement et un débogage rapides,
- portabilité complète et pureté 100% Java,
- La structure multithreading complète permet l'échantillonnage simultané de plusieurs threads et l'échantillonnage simultané de différentes fonctions par des groupes de threads distincts.

Mise en pratique

JMeter simule un groupe d'utilisateurs qui envoient des demandes à un serveur cible et renvoie les informations statistiques du serveur cible via un tableau récapitulatif.

La figure ci-dessous explique le fonctionnement JMeter :



C. JaCoCo

Définition

La couverture de code est une métrique logicielle utilisée pour le nombre de ligne de notre code exécuté lors de test automatisés, pour cela nous avons utilisé JaCoCo qui est un générateur de rapports de couverture de code pour les projets Java.

Mise en pratique

Afin de démarrer avec JaCoCo, nous devons déclarer le plugin *Maven* dans un fichier *pom.xml,* Ensuite, tout ce dont nous avons besoin, c'est d'un simple test JUnit sur une fonction d'une application. Ce test JUnit activera automatiquement l'agent JaCoCo et créera un rapport de couverture de code sous forme d'un tableau :

Element	Missed Instructions	Cov.	Missed Branches	Cov.	Missed	Cxty	Missed	Lines	Missed	Methods	Missed	Classes
Palindrome		21%		17%	3	5	4	7	0	2	0	1
Total	30 of 38	21%	5 of 6	17%	3	5	4	7	0	2	0	1

Nous pouvons mêmes explorer une vue plus détaillée pour chaque classe Java :

```
1.  package com.baeldung.testing.jacoco;
2.
3.  public class Palindrome {
4.
5.      public boolean isPalindrome(String inputString) {
6.          if (inputString.length() == 0) {
7.              return true;
8.          } else {
9.              char firstChar = inputString.charAt(0);
10.             char lastChar = inputString.charAt(inputString.length() - 1);
11.             String mid = inputString.substring(1, inputString.length() - 1);
12.             return (firstChar == lastChar) && isPalindrome(mid);
13.         }
14.     }
15. }
```

Analyse de rapport

Le rapport JaCoCo nous aide à analyser visuellement la couverture de code en utilisant des losanges avec des couleurs (figure précédente) pour les branches et des couleurs d'arrière-plan pour les lignes :

- **Le losange rouge** signifie qu'aucune branche n'a été exercée pendant la phase de test.
- **Le losange jaune** indique que le code est partiellement couvert - certaines branches n'ont pas été exercées.
- **Le losange vert** signifie que toutes les branches ont été exercées au cours de l'essai.

JaCoCo fournit principalement trois mesures importantes :

- **La couverture des lignes** reflète la quantité de code qui a été exercée en fonction du nombre d'instructions de code en octets Java appelées par les tests.
- **La couverture des branches** indique le pourcentage de branches exercées dans le code - généralement lié aux instructions *if / else* et *switch*.
- **La complexité cyclomatique** reflète la complexité du code en donnant le nombre de chemins nécessaires pour couvrir tous les chemins possibles dans un code par combinaison linéaire.

5.3 Création de la base de données

Le moteur de base de données que nous avons utilisé est Apache Derby, qui est un moteur pratique, portable dans les systèmes et ne nécessite aucune administration.

Les tables de notre base de données sont les suivantes :

- **LOGIN :** (correspond à la table Testeur dans le diagramme de classe) contient le nom d'utilisateur et le mot de passe

- **FORMULAIRE :** (correspond à la table Logiciel dans le diagramme de classe) contient le nom du logiciel, sa version, sa description et son objectif ainsi que le nom du développeur et la date de création.

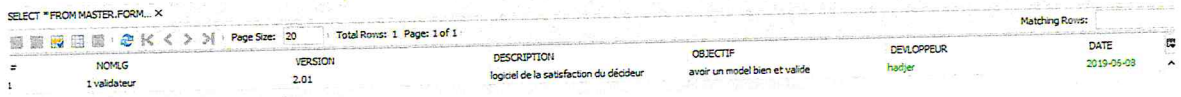
- **QUESTIONNAIRE :** (correspond à la table ValidationDocumentation dans le diagramme de classe) deux champs (oui / non) qui contient le nombre total des réponses sur le questionnaire de la première phase de validation.

- **RESULTAT :** (correspond à la table Résultat dans le diagramme de classe)trois champs pour le taux de chaque de chaque phase de validation et un quatrième champ pour le taux total de la validation final du logiciel.

5.3.1 Description des tables

- La table « LOGIN »

- La table « FORMULAIRE »



- La table « QUESTIONNAIRE »

- La table « RESULTAT »

5.4 Etape 5 : Test et validation

Cette étape consiste à définir l'outil support de mesure de la satisfaction du décideur ainsi que son fonctionnement, ensuite nous allons décrire les tests effectués sur chaque module de cet outil pour le valider.

Etude de cas : Outil support au modèle du décideur

L'objet de cette évaluation technique s'inscrit dans le cadre de l'évaluation des systèmes d'aide à la décision (SIAD). Il permet de mesurer la satisfaction du décideur.

L'instrument a été implémenté et soumis à une vérification technique afin de s'assurer de son bon fonctionnement et de sa validité.

Nous présentons et décrivons en détail cette application ci-dessous.

Informations concernant l'outil

Date de création : 15/06/2019

Nom de l'outil : DMSatisfaction

Version : 4.0

Développeur : Fella Dahmani et Hadjer Metat

Objectif de l'outil développé : Evaluer la satisfaction du décideur.

Description de l'application

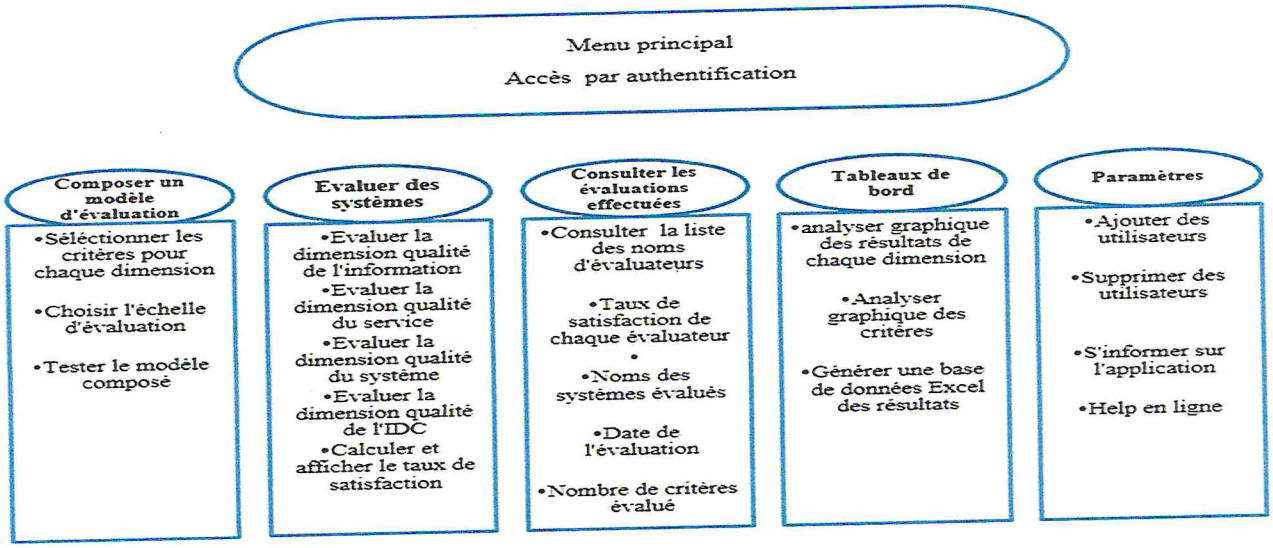


Figure 23: Architecture global de l'outil DMSatisfaction

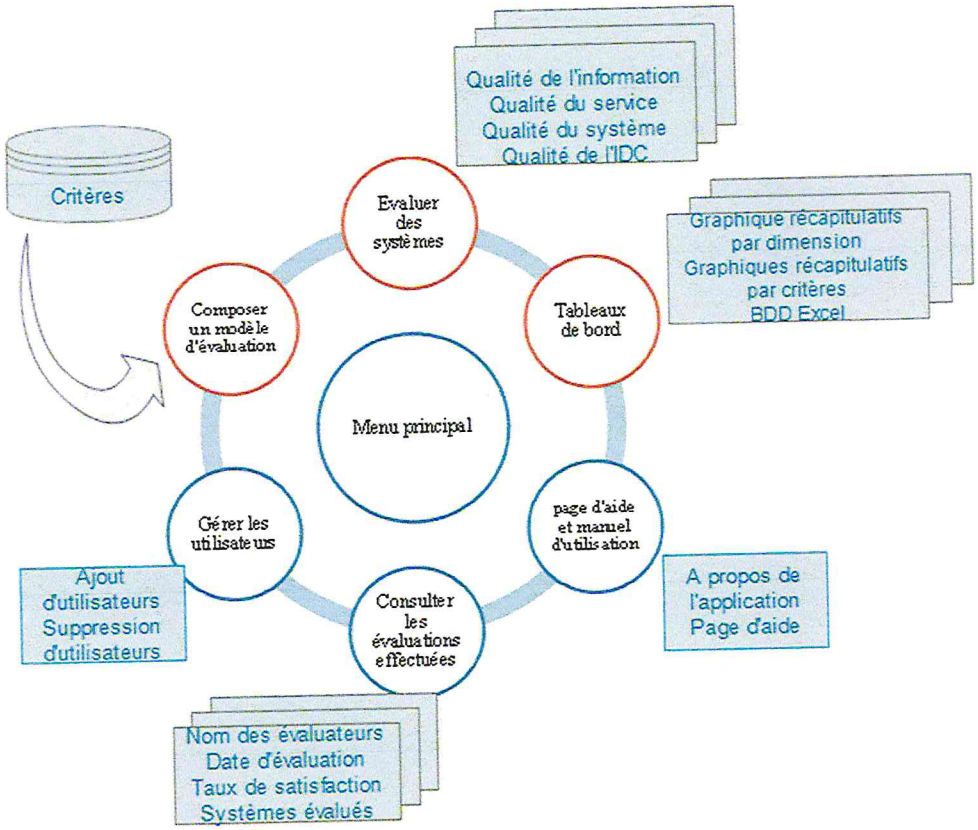


Figure 24: Architecture détaillée de l'outil DMSatisfaction.

L'application se compose de deux interfaces principales

➢ Interface Login

Cette interface est le premier pas pour l'accès au corps de l'application et aux fonctionnalités. Nous y retrouvons le moyen de s'authentifier par le biais d'un identifiant et mot de passe. Une fois ces deux derniers valides et existants dans la base de données l'accès est autorisé.

➢ Menu principal

Cette interface contient toutes les fonctionnalités de l'application, on les retrouve en bas de l'interface avec des images descriptives du rôle des fonctionnalités. Nous allons ainsi citer et expliquer les fonctionnalités.

- **Afficher la liste des évaluations**

Donne comme résultat un tableau récapitulatif de toutes les évaluations effectuées regroupées par : Nom d'utilisateur, Nom du système évalué, Taux de satisfaction obtenu et le nombre de critères évalués.

- **Evaluer**

Se fait par le parcours du nom du logiciel à évaluer
Une fois cette étape confirmée l'utilisateur sera dirigé vers une autre fenêtre pour commencer l'évaluation dimension par dimension (une dimension n'est débloquée que si les réponses ont validées par l'utilisateur).
A la fin, la main est donnée pour calculer la satisfaction et afficher le taux dans une barre de progression.

- **Composer un nouveau modèle**

Ce module permet de créer un nouveau modèle d'évaluation, en sélectionnant les critères à évaluer selon le besoin de l'utilisateur.

- **Gérer les utilisateurs**

Permet d'ajouter un utilisateur en entrant des informations le concernant (nom, prénom, identifiant, mot de passe, poste occupé, adresse et numéro de téléphone)
Ou en supprimer un en entrant son identifiant.

- **Tableaux de bord**

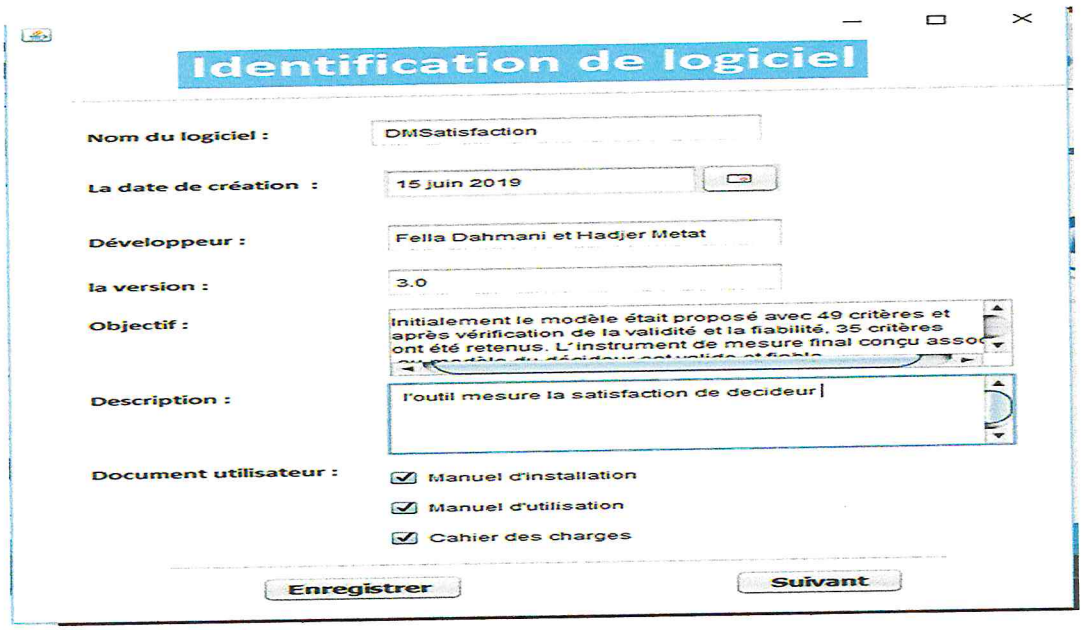
Cette fonctionnalité permet de générer des graphes de résultat obtenu après l'évaluation d'un logiciel donné faite par un utilisateur donné soit par : dimensions ou par critères.

- **A propos**

  Une rubrique « à propos de l'application » a été intégrée comme aide aux utilisateurs ayant des difficultés d'utilisation, elle contient des instructions et un manuel d'utilisation.

5.4.1 Test

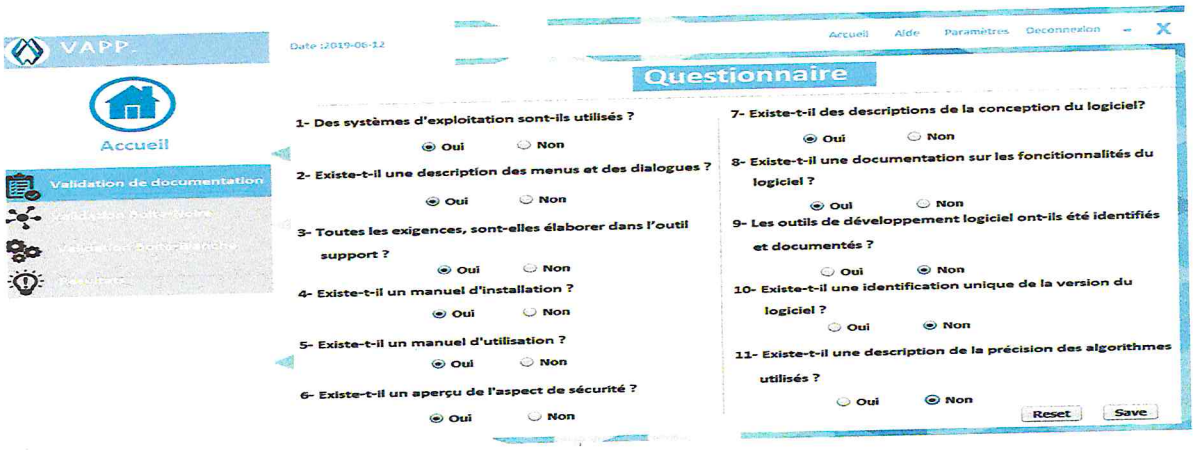
La validation de l'outil support au modèle du décideur commence après l'identification de cet outil (voir Annexe A) en remplissant le formulaire suivant :



Le système enregistre les informations de l'outil, et donne l'accès au menu principal pour commencer la validation

5.4.1.1 Phase I : validation de Documentation

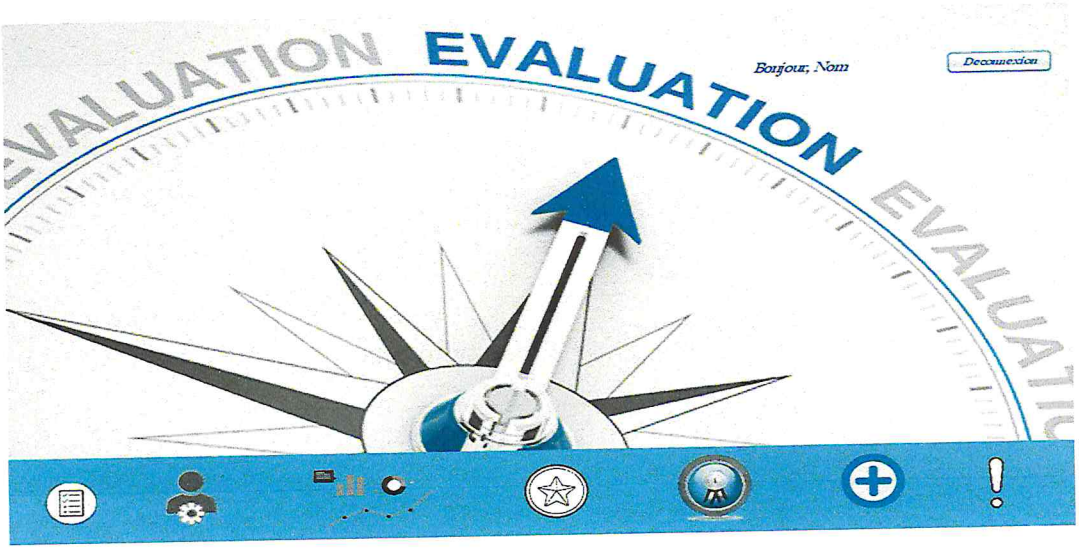
Le questionnaire est l'outil de test utilisé pour cette première phase de validation ;



5.4.1.2 Phase II : validation Boite-Noire

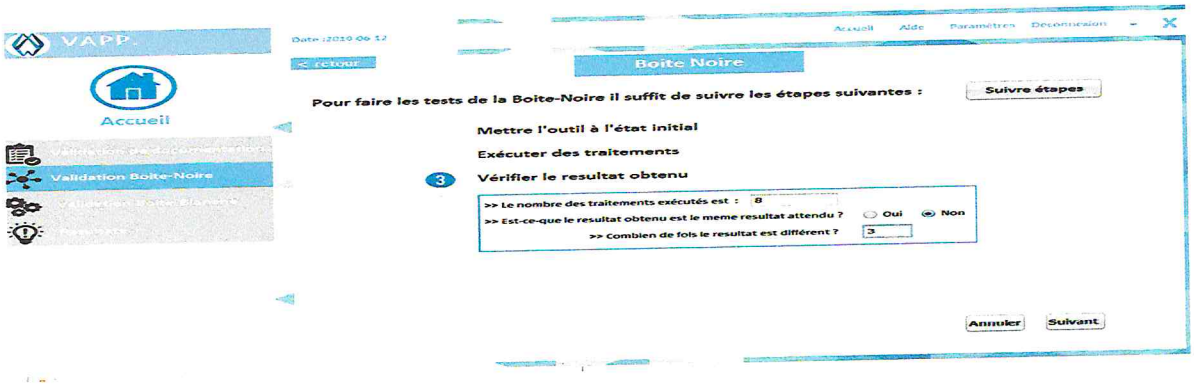
Cette deuxième phase de validation se fait en suivant trois étapes :

1. Mettre l'outil à l'état initial ;



2. Exécuter les traitements de l'outil plusieurs fois ;
3. Comparer les résultats après chaque exécution ;

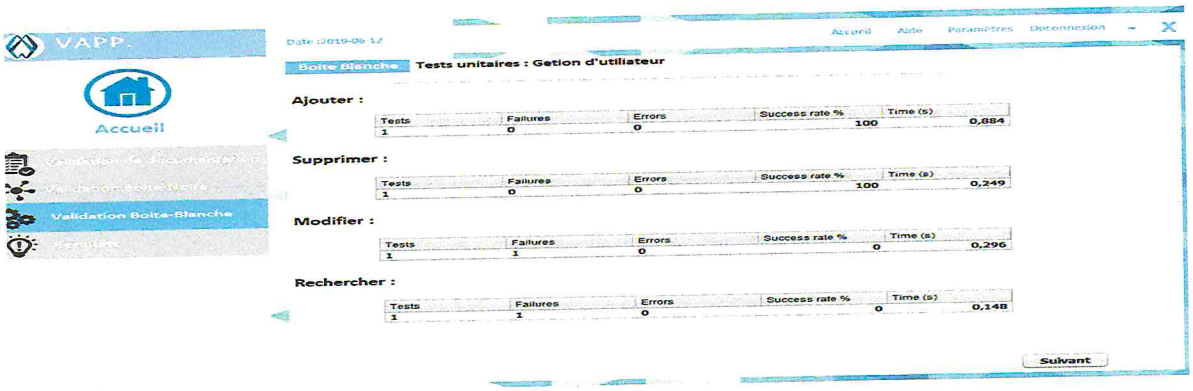
Cela se fait de la manière suivante :



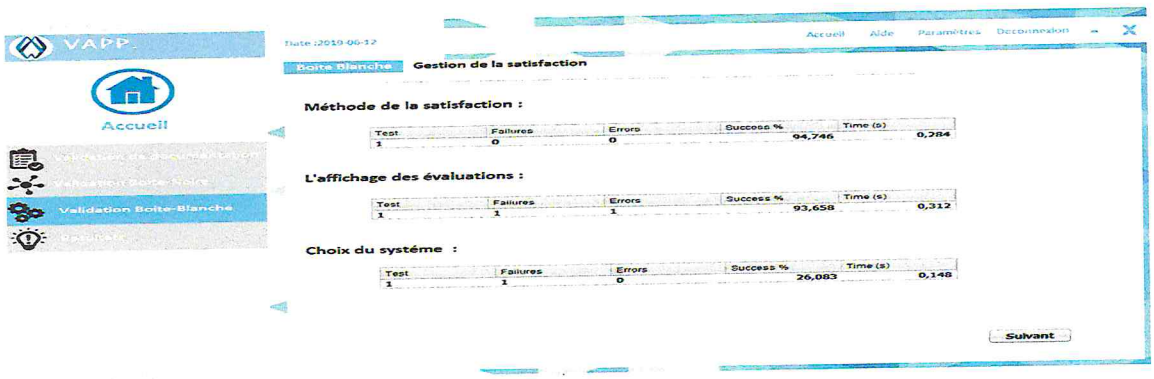
5.4.1.3 Phase III : validation Boite-Blanche

La troisième phase nécessite le code source de l'outil pour tester chaque méthode de chaque module de l'outil ;

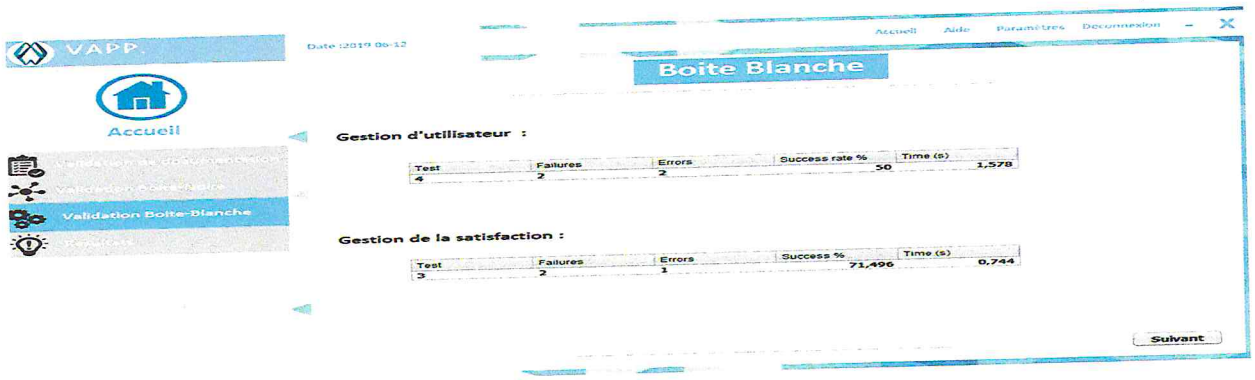
1. Test unitaire de chaque méthode du module « Gérer utilisateur »



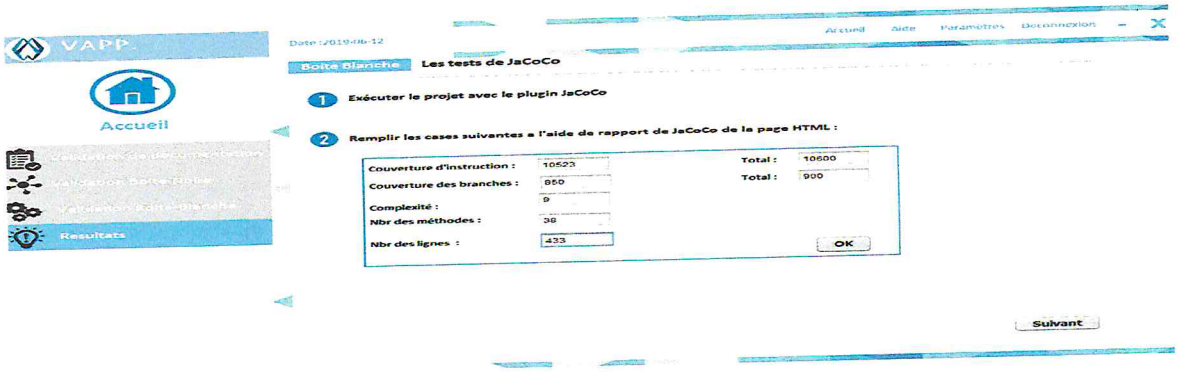
2. Test unitaire de chaque méthode du module « Gestion de la satisfaction »



Après le résultat final de chaque module est calculé automatiquement avec l'outil JUnit, puis le résultat est affiché comme suit :



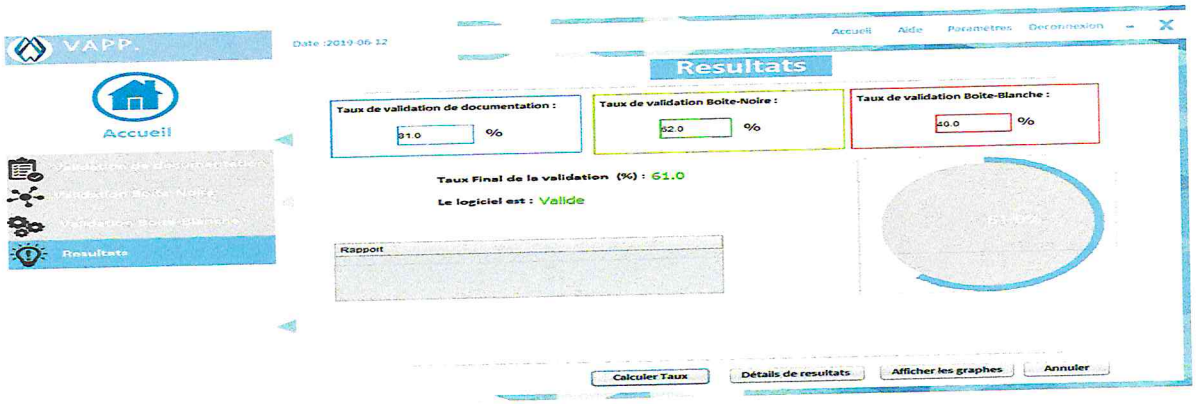
Ensuite, la deuxième étape consiste à tester la couverture du code de l'outil en utilisant le Plugin JaCoCo est le rapport de ce test sera affiché automatiquement dans une page web, puis le testeur rempli les valeurs obtenue comme suit :

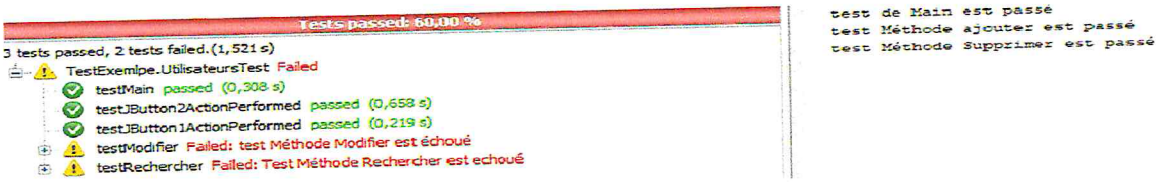


5.4.2 Validation

À la fin de la troisième phase de validation, la première activité du processus de validation se termine, ensuite le taux de chaque phase est calculé selon les formules définit dans le chapitre précédant.

1. Les taux de validation de chaque phase ainsi que le taux totale de validation de l'outil.





2. Un graphe récapitulatif de taux de validation.

3. Le système génère un rapport de validation final contenant toutes les informations de l'outil et les résultats des tests effectués.

Conclusion

La dernière étape du développement de notre outil a pour but d'implémenter notre solution. Quant à l'étape de test et de validation a pour objectif de mettre en œuvre l'outil DMSatisfaction, de tester unitairement toutes ses principales fonctionnalités et le valider.

Conclusion

Ce projet de fin d'étude avait pour ambition **d'élaborer une approche de validation de logiciel**, en se demandant si le besoin réel existait ou pas et faire face à l'absence de la validation de logiciel avant sa mise en place. Cela est fait en répondant à la question « Avons-nous construit le bon logiciel ? »

Ce mémoire mentionne toutes les étapes pour arriver au résultat attendu et à valider le logiciel. Il a fallu dans un premier temps recenser les différents besoins existants et les informations pertinentes du domaine tels que le processus de validation ainsi que les méthodes et les techniques utilisés. Pour ce faire, nous avons pu aussi donner un contexte général à notre projet et identifier les différentes exigences du futur système.

Nous avons établi ensuite une approche de validation basée sur le processus de validation extrait du standard [IEEE 2012-1998] qui est une succession de deux sous processus d'Approvisionnement et de Développement, dont une méthode de validation y intégré. Cette dernière est une suite de trois de phases : validation de documentation, validation en Boite-Noire et validation en Boite-Blanche, qui nous permettent de connaitre les fonctionnalités primordiales.

Notre projet d'étude atteint sa fin et pour conclure le dernier chapitre nous avons entamé la phase d'analyse et de conception du système, ensuite nous avons abordé notre environnement de travail. Par la suite, nous avons expliqué notre architecture d'application « ValidApp » afin de présenter finalement les différentes principales parties d'implémentation de notre application réalisée.

Ce travail nous a été très formateur, puisqu'il a nous perms de découvrir le domaine de la validation de logiciel et connaitre les différents techniques de tests de validation, et nous a permis également de se confronter à plusieurs contraintes à la fois : contraintes de temps, contraintes d'expérience et de technologie et aussi l'automatisation des méthodes de validation. En outre, ce projet nous a permis d'approfondir nos connaissances dans les bonnes pratiques de l'ingénierie génie logiciel.

Perspective

Notre réalisation est encore d'actualité et ne s'arrête pas à ce niveau, en effet plusieurs perspectives s'offrent à ce projet :

- S'approfondir dans la validation en Boite-noire
- Utiliser d'autres bibliothèques que le « JUnit » pour les tests unitaires comme « MSest » de Microsoft et « NUnit »
- Utiliser les tests de régression dans le cas de modification du code source
- Utiliser d'autres techniques de test pour la validation en boite noire, nous proposons d'utiliser « SolverSDK ».
- Supporter tous les langages de programmation.

Finalement, vu l'accomplissement de projet, nous souhaitons très fortement qu'il soit le fruit du progrès, de l'évolution et qu'il reste à la hauteur des exigences de notre promotrice.

Bibliographie

A

[ABPMP, SD] « GUIDE DE GESTION DES PROCESSUS METIER ABPMP V3, GLOSSARY INDEX »

[ALIANI, 2006] « ALANI. T, INTRODUCTION AU DIAGNOSTIC DES DEFAILLANCES, LABORATOIRE A2SI-ESIEE-PARIS, SOUTENU LE 01/10/2006, T.ALANI@ESIEE.FR »

[A. OFFUTT, 1988] « A. J. OFFUTT VI, "AUTOMATIC TEST DATA GENERATION," GEORGIA INSTITUTE OF TECHNOLOGY, ATLANTA, GA, USA, 1988. »

[ARNAUD, 2005] « ARNAUD BAILLY, THESE DE DOCTORAT « TEST ET VALIDATION DE COMPOSANTS LOGICIELS, LILLE1, 2005 »

[AURELIE, SD] « ANALYSE DE RISQUES, METHODES QUALITATIVES D'ANALYSE DE RISQUES »

B

[B. BEIZER, 1990] « B. BEIZER, SOFTWARE TESTING TECHNIQUES (2ND ED.). NEW YORK, NY, USA : VAN NOSTRAND REINHOLD CO., 1990 »

C

[CARLE, 2003] « METHOD OF SOFTWARE VALIDATION, NORDTEST REPORT. »

D

[DEMILLO R A AND PERLIS 1979] "SOCIAL PROCESSES AND PROOFS OF THEOREMS AND PROGRAMS", P 271-280, 1979

[DICTIONNAIRE INFORMATIQUE, URL://WWW.COUR-INFORMATIQUE.FR]

E

[E, DIJKSTRA, 1972] « EDSGER DIJKSTRA, THE HUMBLE PROGRAMMER, 1972

F

[FAUCHER, SD] « FAUCHER. J, PRATIQUE DE L'AMDEC (ASSUREZ LA QUALITE ET LA SURETE DE FONCTIONNEMENT DE VOS PRODUITS, EQUIPEMENTS ET PROCEDES), SERIE PERFORMANCE INDUSTRIELLE, WWW.DUNOD.COM. »

[FDA, CLAUSE 3.1.2] « GENERAL PRINCIPLES OF SOFTWARE VALIDATION, FINAL GUIDANCE FOR INDUSTRY AND FOOD AND DRUG ADMINISTRATION STAFF, 11 JANVIER, 2002 »

[FLOYD R W 1967] « ASSIGNING MEANING TO PROGRAMS, VOL 19, AMERICAN MATHEMATICS SOCIETY, P 19-32, 1967 »

[futura-sciences, 2019] « www.futura-science.com, 2001-2019 »

[F. X. FOURNARI, 2011] « INTRODUCTION AUX TESTS DU LOGICIEL, 2011

G

[GSI] "GESTION DES SYSTEMES D'INFORMATIQUES", IFTS14

H

[HERGON ET AL] « HERGON. E, CRESPEAU. H, ROUGER. PH, MODES DE DEFAILLANCE DU PROCESSUS TRANSFUSIONNEL. INTERET DE L'ANALYSE PREVISIONNELLE DE SURETE DE FONCTIONNEMENT, INSTITUT NATIONAL DE LA TRANSFUSION SANGUINE, PARIS. »

[HOREAU] « HOAREAU LLP- L.DESCHAMPS LJF, 'DIALOGUE RESEAU ET ENCAPSULATION, PRINCIPES FONDAMENTAUX »

I

[IEC 61508] « INTERNATIONAL STANDARD IEC, FAULT TREE ANALYSIS, WWW.IEC.CH »

[IEC 61025 :1990] «ANALYSE PAR ARBRE DE PANNE »

[IEC 61508-7, CLAUSE B.6.6.3] « INTERNATIONAL STANDARD IEC 61508, PART 1- 7, "FUNCTIONAL SAFETY OF ELECTRICAL/ELECTRONIC/PROGRAMMABLE ELECTRONIC SAFETY-RELATED SYSTEMS »

[IEEE STANDARD 2012-1998] « IEEE STANDARD FOR SOFTWARE VERIFICATION AND VALIDATION, P 9-25, 2012-1998 »

[IEEE 729, 2002] «Glossaire standard IEEE de la terminologie du génie logiciel »

[I.OLIVIER, 2014] « OLIVIER IDDIR , ETUDES DES DANGERS : ARBRE DE DEFAILLANCES (METHODE D'ANALYSE DETAILLEE DES RISQUES ADR, FICHE PRATIQUE, 21 MAI 2014 »

[ISO 17025] « EXIGENCES GNERALES CONCERNANT LA COMPETENCE DES LABORATOIRES D'ETALONNAGES ET D'ESSAIS, L'EVALUATION DE LA CONFORMITE »

[ISO STANDARD] « IEEE STANDARD FOR SOFTWARE VALIDATION, 2005 »

J

[JAN ET BENGT, 2004] "JAN JACOBSON, BENGT JOHANSSON, METHOD OF VALIDATION AND TESTING SOFTWARE, MID SOFTWARE, REVISION 24 SEPTEMBER 2004

[J, BADRIKIAN, 2002] « J, BADRIKIAN. CODES CORRECTEURS. ELLIPSES, 2002 »

L

[LAMBERT LOYE MONDJO, ARTICLE 27/10/2016]

[LAURENT] « UML2, DIAGRAMME D'ETAT DE TRANSITION, LAURENT AUDIBERT, WWW. DEVELOPPEZ.COM »

[LUC, 2014] « LUC BERSON. J, 1290 COMPRENDRE L'AMDEC, CONSULTANT ET FORMATEUR EN MANAGEMENT DES SYSTEMES, AUDITEUR QMS CERTIFIE IRCA, BASE DOCUMENTAIRE : PILOTER ET ANIMER LA QUALITE, DELIVRE LE : 24/10/2014 »

[L. MORELL, 1983] « L. J. MORELL, "A THEORY OF ERROR-BASED TESTING," UNIVERSITY OF MARYLAND AT COLLEGE PARK, COLLEGE PARK, MD, USA, 1983. »

M

[MORTUREUX, 2005] « MORTUREUX Y. LA SURETE DE FONCTIONNEMENT : METHODE POUR MAITRISER LES RISQUES, TECHNIQUES DE L'INGENIEUR, N° BM 5008. PARIS : 2005, P17) »

N

[NATHALIE, 2019] « NATHALIE POUILLARD, LE TEST DE PERFORMANCE, 30 JANVIER 2019 »

O

[OIQ, 2014] «L'ORDRE DES INGENIEURS DU QUEBEC', LES PRINCIPALES METHODES D'IDENTIFICATION DES DANGERS ET D'ANALYSE DES RISQUES, ANALYSE PAR ARBRE DES EVENEMENTS', MAI 2011, MISE A JOUR EN DECEMBRE 2014 »

[OLIVIER, 2014] « ÉTUDES DES DANGERS : ARBRE DE DEFAILLANCES, 21 MAI 2014 » FICHE PRATIQUE

P

[P.Muller, 2000] « Pierre-Alain Muller, *Modélisation objet avec UML*, Eyrolles, deuxième édition 2000, p.231 (les composants) »

R

[RIDOUX, 1999] « RIDOUX. M, AG4220 AMDEC – MOYEN, BASE DOCUMENTAIRE METHODES DE PRODUCTION DANS LE THEME : CONCEPTION ET PRODUCTION ET DANS L'UNIVERS GENIE INDUSTRIEL, DATE DE PUBLICATION : 10/07/1999.

S

[S. KANGOYE, 2017] « SEKOU KANGOYE, VALIDATION DE LOGICIEL EMBARQUE AUTOMOBILE, BASE SUR LA GENERATION AUTOMATIQUE DE CAS DE TEST, 11 MAI 2017 »

T

[THESOFTWAREEXPERTS] « PLATEFORME : WWW.THE-SOFTWARE-EXPERTS.COM, TESTING OF SOFTWARE MODULES »

[T. IQBAL, 2017] « RESEARCH PAPER, SOFTWARE TESTING , P 3-9, NOVEMBER 2017 »

[TODD, 2004] « TODD STOKER, LES GENERATEURS D'IMPULSIONS/ MODELES APPORTENT DES FONNALITES DE CONTROLES SUR LES SIGNAUX QU'ILS SONT CAPABLES DE CREER" » ARTICLE, FEVRIER 2004.

W

[WELMEC, 1999] « SOFTWAR REQUIREMENTS ON THE BASIS OF THE MESURING INSTRUMENTS DIRECTIVE, OCTOBER 1999, WELMEC GUIDE 7.1 , DOWNLOADFROM WWW.WELMEC.ORG »

[W.RICHARDS ADRION, MATHA A.BRANSTAD AND JOHN C.CHERNIAVSKY, 2013] « ASSIGNING MEANING TO PROGRAMS", VOL 19, AMERICAN MATHEMATICS SOCIETY »

Y

[Y, BOUKHOUCHI, 2017] "YOUNESS BOUKHOUCHI", PROFESSOR ENSA AGADIR, ARTICLE, 2017

Annexe A

Cahier des charges de l'outil de mesure de la satisfaction du décideur

« DMSatisfaction »

Ce cahier de charge a pour but de présenter l'outil support au modèle de mesure de la satisfaction du décideur ainsi que les fonctionnalités qui le composent afin de le valider.

1. Contexte et définition du problème

1.1. Définition

L'objectif du logiciel et de mesurer la satisfaction du décideur, et ce en implémentant l'ensemble des critères impactant la satisfaction du décideur

1.2. Problématique

Les problématiques liées à cet outil mesurant la satisfaction du décideur sont les suivantes :

- La satisfaction étant une attitude, il est donc difficile de la mesure avec objectivité.
- Trouver un moyen simple d'exposé l'ensemble des critères.
- La difficulté de connaitre le degré de satisfaction du décideur.
- La multiplicité des critères mesurant la satisfaction, la rend difficile à calculer.

2. Objectifs projetés

La performance, l'exactitude de calcul et la précision de la mesure est l'objectif principal du logiciel. Le résultat procuré par le logiciel est un taux il doit ainsi être précis et exacte.

3. Description fonctionnel

Fonction	Authentification
Objectif	Limiter l'utilisation de l'application à des personnes précises
Description	Une interface de login.
Contraintes et règles de gestion	Les utilisateurs doivent posséder un identifiant.
Niveau de priorité	Moyen

Fonction	Evaluer des systèmes
Objectif	Calculer la valeur de la satisfaction
Description	Authentification, description de chaque critère sur les échelles de chaque pair d'adjectif, calcul du résultat.
Contraintes et règles de gestion	Décideur.
Niveau de priorité	Haut

Fonction	Consulter les évaluations effectuées
Objectif	Consulter un récapitulatif de toutes les évaluations effectués et le taux de satisfaction.

Description	retrouvons un récapitulatif de toutes les évaluations effectuées accompagnées du nom de la personne l'ayant faite, le taux de satisfaction obtenu, la date de l'évaluation et le nom du système évalué ainsi que le nombre de critères évalués.
Contraintes et règles de gestion	Le responsable/manager du décideur.
Niveau de priorité	Haut

Fonction	Composer un modèle d'évaluation
Objectif	Sélectionner un ensemble de critères et une échelle.
Description	Personnaliser le modèle et évaluer un système.
Contraintes et règles de gestion	Le responsable/manager du décideur.
Niveau de priorité	Haut

4. Périmètre

Nous souhaitons que notre outil « logiciel » de mesure de satisfaction du décideur soit utile pour les individus qui ont une relation avec le domaine du décisionnel, à savoir : les entreprises et organisations possédant un outil d'aide à la décision dans le but de mesurer la satisfaction de leurs employés/ utilisateurs du système.

5. Enveloppe budgétaire

5.1. Ressources humaines

Les deux étudiantes en 2éme année master « Metat Hadjer et Dahmani Fella »

Sous la supervision de Mme D.Bouaissa.

5.2. Ressources logicielles

L'environnement de développement : NetbeansIDE

Langage de programmation : Java

Base de données : Apache Derby

6. Délai

Notre projet de fin d'études pour l'obtention du diplôme 'master 2' en informatique ingénierie des logiciels qui consiste à « tester et valider un modèle multidimensionnel de mesure de la satisfaction du décideur » doit être fini au plus tard le mois de juillet 2019.