

UNIVERSITÉ NATIONAL DU VIETNAM
INSTITUT FRANCOPHONE
INTERNATIONAL

Spécialité : Réseaux et Systèmes Communicants

Code : 8480201.01

Lesly ROC

APPRENTISSAGE AUTOMATIQUE APPLIQUE AUX TESTS LOGICIELS

ỨNG DỤNG HỌC MÁY TRONG KIỂM THỬ PHẦN MỀM

RESUME DU MEMOIRE DE FIN D'ETUDES DE
MASTER 2 EN INFORMATIQUE

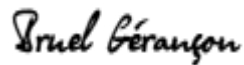
Hanoi - Decembre 2023

Travail réalisé à l’Institut Francophone International,
UNIVERSITÉ NATIONALE DU VIETNAM, HANOI

Sous la direction de :

Dr. Bruel GERANÇON

Chargé de cours au département d'informatique à l'Université du Québec À Montréal

A handwritten signature in black ink, reading "Bruel Gerançon".

Rapporteur 1 : ...

Rapporteur 2 : ...

Le mémoire est soutenu devant le jury à l’Institut Francophone International

Le mémoire est accessible :

— au Centre d’Informations et de Bibliothèque, Université Nationale du Vietnam, Hanoi à
l’Institut Francophone International, Université Nationale du Vietnam, Hanoi

ATTESTATION SUR L'HONNEUR

J'atteste sur l'honneur que ce mémoire a été réalisé par moi-même et que les données et les résultats qui y sont présentés sont exacts et n'ont jamais été publiés ailleurs. La source des informations citées dans ce mémoire a été bien précisée.

LỜI CAM ĐOAN

Tôi cam đoan đây là công trình nghiên cứu của riêng tôi. Các số liệu, kết quả nêu trong Luận văn là trung thực và chưa từng được ai công bố trong bất kỳ công trình nào khác. Các thông tin trích dẫn trong Luận văn đã được chỉ rõ nguồn gốc.

Signature de l'étudiant



Résumé

Le développement logiciel, un processus complexe avec des phases interdépendantes, requiert une identification précoce des erreurs pour minimiser les coûts de correction. L'ingénierie des exigences, traduisant les besoins en spécifications détaillées, est cruciale, avec 71% des échecs de projets logiciels attribués à des erreurs d'ingénierie des exigences, dont 50% surviennent pendant la phase de spécification. La détection tardive d'erreurs peut engendrer des coûts jusqu'à 100 fois plus élevés.

Dans le contexte du développement de systèmes critiques, la conformité aux normes internationales, notamment dans l'industrie ferroviaire, est essentielle. La rédaction des exigences en langage naturel expose les spécifications à des ambiguïtés, nécessitant des outils pour détecter précocement les défauts logiciels.

Ce mémoire propose un outil utilisant le traitement automatique des langues (NLP) pour détecter automatiquement les ambiguïtés dans les spécifications logicielles, telles que les cas d'utilisation, rédigées en langage naturel. L'outil, basé sur l'approche par triplets, évalue les techniques de rédaction de spécifications logicielles, décrivant leurs forces et lacunes.

Les résultats de la recherche indiquent que le modèle de triplets permet d'identifier les ambiguïtés et les défauts logiciels dans les documents de spécifications. Des taux de détection de 92,30% pour les documents d'exigences en français et de 100% pour les documents de spécifications en anglais ont été obtenus, démontrant la promesse de cette approche.

Mots clés : Traitement automatique des langues, exigences logicielles, triplets, défauts logiciels ambiguïtés

Abstract

Software development, a complex process with interdependent phases, requires early error identification to minimize correction costs. Requirements engineering, translating needs into detailed specifications, is crucial, with 71% of software project failures attributed to requirements engineering errors, 50% of which occur during the specification phase. Late error detection can result in costs up to 100 times higher.

In the context of critical systems development, especially in the railway industry, compliance with international standards is essential. Drafting requirements in natural language exposes specifications to ambiguities, necessitating tools to early detect software defects.

This research proposes a tool employing Natural Language Processing (NLP) to automatically detect ambiguities in software specifications, such as use cases, written in natural language. The tool, based on the triplets approach, assesses techniques for drafting software specifications, describing their strengths and weaknesses.

This research proposes a tool employing Natural Language Processing (NLP) to automatically detect ambiguities in software specifications, such as use cases, written in natural language. The tool, based on the triplets approach, assesses techniques for drafting software specifications, describing their strengths and weaknesses.

Keywords : Natural language processing, software requirements, triples, software defects ambiguities

CHAPITRE 1 : INTRODUCTION

Le développement logiciel représente une entreprise complexe, caractérisée par des phases interconnectées, dont la phase des exigences joue un rôle crucial. Les erreurs logicielles peuvent surgir à tout moment, engendrant des coûts considérables pour leur détection et correction si elles ne sont pas identifiées dès le début. La détection précoce des erreurs logicielles est donc essentielle, non seulement pour assurer la qualité finale du logiciel, mais aussi pour restreindre les coûts de réalisation des projets logiciels.

L'ingénierie logicielle des exigences est le pilier central de cette démarche, reposant sur une exploration minutieuse des besoins des parties prenantes convertis en spécifications détaillées. Ces spécifications jouent un rôle fondamental dans les phases de conception, de développement et de test du logiciel, définissant ce que le logiciel doit accomplir et comment il doit le réaliser.

Des statistiques, telles que celles du Standish Group [34], révèlent qu'une amélioration du taux d'échec des projets, actuellement d'environ 71%, est attribuée à des spécifications incomplètes, non claires ou ambiguës, ainsi qu'à d'autres facteurs tels que le manque d'implication des utilisateurs et les changements dans les exigences métier.

Les chercheurs, dont Yves Généaux & Ken Nathan, indiquent que 50% [2] des erreurs naissent lors de la phase de spécification, mais seulement 4% sont identifiées à ce stade initial. La détection précoce des erreurs est un objectif partagé par l'ingénierie système et logiciel, car la correction d'erreurs identifiées tardivement peut coûter jusqu'à 100 fois plus cher que si elles avaient été détectées plus tôt [2].

Ainsi, la planification d'efforts minutieux en amont, visant à identifier les erreurs dès le début, impacte la qualité intrinsèque du logiciel et la viabilité économique du projet dans son ensemble :

- 3 à 4 \$ si découverts au moment de la conception.
- 100 \$ à corriger, après déploiement [32], [33].

Dans le contexte du développement de systèmes critiques, notamment dans l'industrie ferroviaire, la conformité aux normes internationales revêt une importance cruciale. Ces normes exigent que les documents liés aux exigences soient exhaustifs, clairs, précis, dépourvus d'ambiguïté, vérifiables, sujets à des tests, aisément maintenables et réalisables.

Cependant, la majorité de ces documents, rédigés en langage naturel, peuvent être sujets aux ambiguïtés et aux erreurs malgré leur facilité de rédaction et de compréhension.

L'ingénierie des exigences est considérée comme le fondement essentiel du développement logiciel réussi. La détection précoce des erreurs à toutes les étapes du processus, de la spécification à la conception et au-delà, est une pratique cruciale pour assurer la qualité du produit final tout en limitant les coûts et les risques associés. La reconnaissance de l'impact significatif des erreurs de spécification sur le déroulement et les résultats des projets logiciels a conduit la communauté de l'ingénierie logicielle à concentrer ses efforts sur l'identification proactive de ces erreurs, dans le but d'aboutir à des solutions logicielles plus fiables, durables et économiquement viables.

1.1- Problématique Etudié

Dans le domaine du développement logiciel, le document de spécification des exigences logicielles (SRS) joue un rôle crucial en fournissant une description complète des exigences fonctionnelles et non fonctionnelles d'un système à développer [5]. Rédiger des spécifications claires et dépourvues d'ambiguïté demeure une priorité cruciale, car près de 87,7% des exigences logicielles sont documentées en langage naturel (NL) [8]. Cependant, la plupart des SRS NL contiennent des exigences ambiguës, ce qui peut impacter négativement le processus de développement logiciel.

L'objectif fondamental de cette recherche est de capitaliser sur les avancées en Traitement Automatique du Langage Naturel (NLP) pour concevoir des modèles permettant d'identifier précocement les ambiguïtés spécifiques aux spécifications logicielles. Comme illustre la figure 1 pour, Près de 50% des défauts logiciels, souvent dus à des erreurs d'ambiguïté, sont injectés dans la phase de spécification[32]. La problématique de cette recherche réside dans l'ambiguïté des exigences logicielles, entravant une interprétation uniforme par l'équipe de développement et les parties prenantes. Le projet vise à concevoir un outil de détection automatique des erreurs logicielles dès la phase de spécification, exploitant les techniques de traitement automatique des langues naturelles pour réduire/corriger les erreurs d'ambiguïté. La figure 1 illustre le pourcentage des coûts associés aux défauts logiciels à chaque phase du processus de développement logiciel.

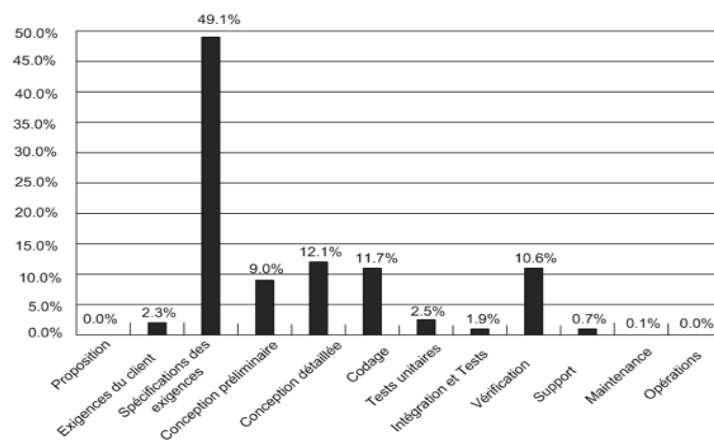


Figure 1.- Statistiques des de défauts injectés dans la phase de spécification[32]

1.3 Objectif de recherche

L'objectif de ce mémoire est de concevoir un outil, en exploitant les techniques de traitement automatique des langues, capable de détecter les ambiguïtés dans le document des spécifications d'exigences logicielles. Pour y parvenir, nous nous inspirons de l'approche par triplets proposée par Géranson [1], pour la spécification, rédaction et représentation des exigences logicielles. Cette approche consiste à écrire les exigences logicielles de façon simple et claire sous forme de triplets (sujet, prédicat, objet), en vue de réduire l'ambiguïté dans le document d'exigences logicielles. En d'autres termes, chaque exigence logicielle constitue un acteur qui déclenche une opération système sur un objet ou un composant logiciel [1].

CHAPITRE 2 : L'Ingénierie des Exigences

2.1- Introduction

Cette étude se concentre sur la phase cruciale du processus de développement logiciel, à savoir l'Analyse des Besoins, avec un accent particulier sur l'Ingénierie des Exigences (IE) et l'amélioration de la Spécification des Exigences Logicielles (SRS). La SRS joue un rôle prédominant dans le cycle de vie du développement logiciel, impactant la conception du système, la programmation et les tests, et la réussite d'un projet logiciel dépend en grande partie de la qualité de la SRS [21] [22].

Les SRS sont principalement rédigées en langage naturel en raison de la difficulté des parties prenantes à comprendre les langages formels. Cependant, le langage naturel est intrinsèquement sujet à l'ambiguïté, avec quatre types courants d'ambiguïté identifiés : lexicale, syntaxique, sémantique et pragmatique [23] [24] [25]. Cette étude se base sur des méthodes de Traitement du Langage Naturel (NLP) pour détecter et résoudre efficacement ces ambiguïtés dans les documents d'exigences, contribuant ainsi à améliorer la qualité et la fiabilité du processus de développement logiciel [20].

2.2- Les documents d'exigences et leur importance

Les documents d'exigences, selon l'International Organisation for Standardisation (ISO) [15], décrivent les fonctionnalités et les contraintes d'un système logiciel, servant à communiquer les besoins des utilisateurs aux développeurs et aux autres parties prenantes. Ces documents sont classifiés en deux catégories principales : les exigences fonctionnelles, qui décrivent les fonctionnalités du système [18], [15], [19], et les exigences non fonctionnelles, qui spécifient les propriétés du système telles que la performance, la sécurité et la fiabilité.

2.2.1 Ils sont importants pour plusieurs raisons :

- ❖ Ils aident à garantir que le système logiciel répond aux besoins des utilisateurs.
- ❖ Ils aident à éviter les changements de conception coûteux en cours de développement.
- ❖ Ils aident à améliorer la communication entre les utilisateurs, les développeurs et les autres parties prenantes.
- ❖ Ils peuvent être utilisés pour évaluer le système logiciel et garantir sa conformité aux exigences.
- ❖ Ils servent de base pour la conception, le développement et la validation du système ou du produit logiciel [17]

2.3 Le Processus d'ingénierie des Exigences : Analyse et Optimisation

Le processus d'ingénierie des exigences, également connu sous le nom de processus IE, représente la première phase du cycle de développement logiciel [28]. Il joue un rôle crucial en guidant la transformation des besoins informels des parties prenantes en une spécification initiale hautement abstraite, communément appelée spécification des besoins. Les étapes suivantes du processus impliquent un affinement progressif de cette spécification, conduisant à l'élaboration d'une spécification d'exigences complète et détaillée [29].

Une fois créées, les spécifications initiales sont consolidées dans le cahier des charges, document capital marquant le point d'entrée vers les phases ultérieures du développement. Il reflète les accords et les décisions convenus entre les parties prenantes, tout en constituant un document contractuel pour les acteurs industriels, définissant les rôles et les responsabilités des clients et des prestataires.

Cette démarche vise à établir un cadre clair et structuré pour la gestion des exigences tout au long du processus de développement logiciel, favorisant la communication et l'alignement

entre les parties prenantes. Elle garantit également la traçabilité des décisions prises à chaque étape du processus [41].

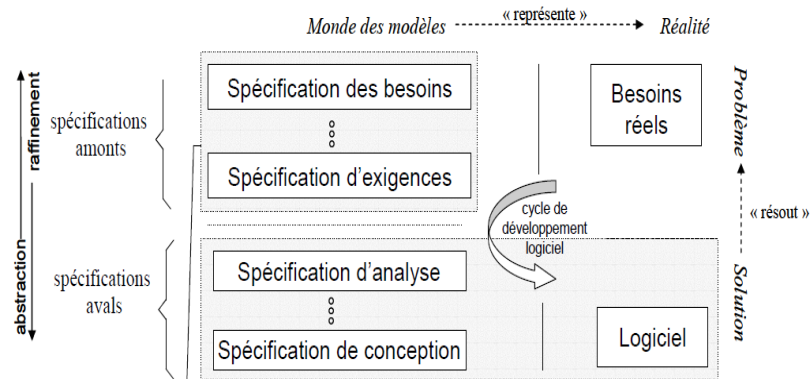


Figure 2 : Le processus d'ingénierie des exigences au sein du processus de développement logiciel source[41]

2.4 Étapes de l'ingénierie des exigences

Il y a quelques activités auxquelles nous sommes confrontés lorsque nous travaillons avec les exigences. Dans le cycle Ingénierie des Exigences, il y a cinq activités principales, à savoir,

1. **Recueil des exigences ou Elicitation** - C'est le processus d'examen, de documentation et de compréhension des besoins des parties prenantes et des utilisateurs pour la saison. En utilisant ces données, nous élicitons ensuite les exigences, procédant à l'analyse et à la négociation des besoins.
2. **Analyse des besoins et négociation** – l'analyse est le processus d'affinement des besoins et des contraintes de l'utilisateur sur la base des informations recueillies et obtenues. Ensuite, nous passons à l'activité de documentation.
3. **Documentation/spécification des exigences** – après avoir obtenu le cahier des charges, nous passons à la partie documentation. Nous documentons clairement et précisément les besoins et contraintes des utilisateurs.
4. **Validation des exigences** - enfin, dans l'activité de validation, nous insérons que les exigences de la saison sont complètes, concises et claires.

Lorsque nous finalisons ces cinq activités, nous les répétons encore et encore jusqu'à ce que nous obtenions un ensemble de documents d'exigences convenus qui sont des spécifications formelles.

2.5 Exigences de Qualités

La définition des exigences de qualité est une priorité pour les gestionnaires, analystes et chercheurs spécialisés en ingénierie d'exigences. Selon la norme ISO 9000, les exigences de qualité sont des dispositions spécifiques permettant aux produits, services ou processus de satisfaire aux attentes des parties prenantes, soulignant l'importance de définir des exigences claires pour garantir la satisfaction des clients et l'amélioration continue.

Le Project Management Institute (PMI) considère les exigences de qualité comme les caractéristiques essentielles qu'un produit, service ou résultat de projet doit posséder pour répondre aux besoins des parties prenantes, soulignant l'importance de les définir précisément dès le début du projet pour assurer le succès.

L'American Society for Quality (ASQ) souligne que les exigences de qualité sont des caractéristiques spécifiques d'un produit ou d'un service répondant aux besoins des clients et aux attentes du marché. L'ASQ met en avant l'importance de la collecte, de l'analyse et de la documentation de ces exigences pour assurer la conformité des produits et services, tout en favorisant l'amélioration continue de la qualité.

Gilb et Graham (1993) proposent des critères de qualité pour les documents d'exigences logicielles, basés sur la clarté, la non-ambiguïté et l'unicité des exigences [35]. Ces perspectives mettent en évidence l'importance cruciale de définir des exigences de qualité dès le début d'un projet pour assurer le succès et la satisfaction des parties prenantes.

Critère de Qualité	Description
Clarté	Les exigences doivent être rédigées de manière claire et compréhensible, évitant toute ambiguïté.
Cohérence	Les exigences ne doivent pas entrer en conflit les unes avec les autres et doivent être alignées sur les objectifs du projet.
Complétude	Toutes les fonctionnalités et contraintes essentielles doivent être correctement spécifiées, laissant peu de place à l'interprétation.
Traçabilité	Chaque exigence doit être tracée de manière à pouvoir remonter jusqu'à son origine, et inversement.
Non-ambiguïté	Les exigences ne doivent pas prêter à confusion et leur signification doit être indiscutable.
Testabilité	Les exigences doivent être formulées de manière à permettre une vérification efficace lors des tests.
Pertinence	Les exigences doivent être directement liées aux besoins du client ou de l'utilisateur, sans ajout inutile.
Priorisation	Les exigences doivent être hiérarchisées en fonction de leur importance pour permettre une gestion efficace des ressources.
Stabilité	Les exigences doivent être relativement stables tout au long du projet pour minimiser les changements de dernière minute.
Cohérence temporelle	Les exigences doivent être cohérentes dans le temps, sans contradictions entre les besoins actuels et futurs.

Tableau 1.- Résumé des critères de qualités d'exigences logicielles

2.6- Ambiguïté d'exigences

L'ambiguïté d'exigences se réfère à une situation où les spécifications ou les exigences d'un projet, généralement exprimées dans la documentation, peuvent être interprétées de différentes manières. En d'autres termes, il y a une incertitude ou une confusion quant à la signification ou à l'intention réelle des exigences énoncées. L'ambiguïté peut provenir de termes vagues, de formulations ambiguës, d'expressions subjectives, ou d'une absence de détails suffisants pour une interprétation sans équivoque [36].

2.7- Ambiguïtés des exigences logicielles

Des exigences ambiguës engendrent de la confusion, un gaspillage d'efforts et des révisions supplémentaires. L'ambiguïté découle de deux sources principales : l'absence d'informations et les erreurs de communication. Les erreurs peuvent être classées en deux catégories distinctes. La première catégorie comprend les erreurs indépendantes du rédacteur, celles qui peuvent être résolues sans nécessiter de connaissances spécifiques au domaine, telles que les "erreurs grammaticales", par exemple. La seconde catégorie concerne les erreurs dépendantes de l'auteur, qui exigent une compréhension approfondie du domaine pour être corrigées, telles que le manque de détails, qui doit être rectifié pour corriger l'erreur [26, 27].

Il existe deux catégories principales d'ambiguïtés : les ambiguïtés linguistiques et les ambiguïtés en génie logiciel. Les ambiguïtés linguistiques sont celles qui peuvent être identifiées par n'importe quel lecteur ayant une bonne maîtrise de la langue. Le Manuel des Ambiguïtés [26] présente différents types d'ambiguïtés linguistiques. En revanche, les ambiguïtés en génie logiciel sont contextuelles et ne peuvent être repérées que par les lecteurs ayant une connaissance suffisante du domaine concerné.

Ambiguïtés	Type d'ambiguïté	Sous-type
Ambiguïté Linguistique	Ambiguïté lexicale	Ambiguïté d'homonymie Ambiguïté de polysémie
	Ambiguïté syntaxique	Ambiguïté Analytique Ambiguïté de l'Attachement Ambiguïté de la Coordination Ambiguïté Elliptique
Ambiguïté RE-Spécifique	Ambiguïté Sémantique	Ambiguïté de Portée
	Ambiguïté Pragmatique	Ambiguïté Référentielle
	Vaguité-Erreur de Langage-Généralité	Ambiguïté Déictique
	Ambiguïté du Document des Exigences	
	Ambiguïté du Domaine d'Application	
	Ambiguïté du Domaine du Système	
	Ambiguïté du Domaine de Développement	

Tableau 2.- Résumé des types d'Ambiguïtés[26]

La majeure partie des ambiguïtés dans le domaine de l'ingénierie logicielle résulte des particularités propres à ce domaine, tandis que les ambiguïtés strictement liées au langage jouent un rôle relativement mineur. Dans l'article [27], les auteurs ont identifié divers types d'ambiguïtés spécifiques à l'ingénierie des exigences (IE). Nous avons synthétisé ces deux catégories d'ambiguïtés, à savoir celles liées au langage et celles propres à la RE, dans le tableau 2. Une description détaillée ainsi que des exemples pertinents sont disponibles dans [26], qui traite de l'ambiguïté linguistique, et dans [27], qui se concentre sur l'ambiguïté spécifique à l'IE.

CHAPITRE 3 : Revue Littérature

3.1- Le traitement automatique du langage naturel

Le Traitement du Langage Naturel (TLN) est une discipline informatique qui vise à analyser et représenter le langage naturel à divers niveaux d'analyse linguistique dans le but de parvenir à un traitement du langage humain pour différentes applications [11]. Ces niveaux comprennent l'analyse phonétique, morphologique, lexicale, syntaxique, sémantique, discursive et pragmatique du langage [11], avec l'hypothèse que les humains utilisent généralement tous ces niveaux pour produire ou comprendre le langage [12]. Actuellement, les technologies TLN avancées se limitent principalement aux niveaux d'analyse lexicale et syntaxique pour l'anglais, avec des capacités sémantiques limitées [13].

Les approches en TLN sont classées en TLN symbolique et TLN statistique [11]. Le TLN symbolique, basé sur l'intelligence artificielle, utilise la représentation explicite des faits linguistiques et des algorithmes associés pour une analyse approfondie des phénomènes linguistiques. Cependant, ces approches manquent de flexibilité pour s'adapter dynamiquement à de nouveaux phénomènes linguistiques, car elles reposent sur des règles préalablement établies, ce qui peut les rendre difficiles à gérer. De plus, elles peuvent être fragiles face à des entrées non familières ou non grammaticales [11].

Le TLN statistique, en revanche, repose sur des modèles probabilistes appris à partir de grands ensembles de données linguistiques. Bien que plus flexible pour s'adapter à des données variées, il peut également présenter des défis, tels que la nécessité de grandes quantités de données d'entraînement et la difficulté à interpréter les modèles [11].

Pour atteindre ses objectifs, le TLN utilise diverses techniques et outils, tels que l'analyse lexicale et syntaxique, les modèles statistiques, et d'autres méthodes pour traiter le langage

naturel. Ces concepts clés du TLN sont pertinents pour l'analyse des exigences logicielles [11].

3.2 Tâches de traitement du langage naturel

Le langage humain est rempli d'ambiguïtés qui rendent incroyablement difficile l'écriture d'un logiciel capable de déterminer avec précision le sens d'un texte ou de données vocales. Les homonymes, les homophones, le sarcasme, les expressions idiomatiques, les métaphores, les exceptions de grammaire et d'usage, les variations dans la structure des phrases ne sont que quelques-unes des irrégularités du langage humain qui nécessitent de nombreuses années d'apprentissage pour l'homme, mais que les applications basées sur le langage naturel doivent apprendre à reconnaître et à comprendre avec précision dès le départ, si l'on veut qu'elles soient utiles.

Plusieurs tâches de traitement du langage naturel décomposent le texte humain et les données vocales de manière à aider l'ordinateur à donner un sens à ce qu'il ingère. Certaines de ces tâches comprennent [37] :

- La reconnaissance vocale
- Le marquage des parties du discours
- La désambiguïsation du sens d'un mot
- La reconnaissance des entités nommées
- La résolution de coréférence
- L'analyse du sentiment
- La génération de langage naturel

3.3- Présentation du GPT-3 l'approche LNP utilisé dans le cadre de cette recherche

GPT-3, développé par OpenAI en 2018, est un modèle de deep learning avec 175 milliards de paramètres, appartenant à la famille des GPT. Conçu pour des tâches de traitement automatique du langage naturel, il excelle dans les questions-réponses, la traduction, la composition de texte, la résolution de problèmes et la génération de code applicatif [37].

3.3.1- Fonctionnement

GPT-3, un Large Language Model, utilise une architecture de type transformer, plus rapide à entraîner que les RNN. Cela facilite le traitement du langage naturel en permettant une découpe des traitements et une parallélisations des calculs [37].

3.3.2- Pourquoi le choix du GPT-3 D'open AI

Le choix du modèle GP3-OpenAI pour la détection des ambiguïtés dans les exigences logicielles est justifié par sa puissance linguistique, sa capacité d'apprentissage continu et sa disponibilité en open source. Ce modèle offre une détection plus précise, rapide et efficace des ambiguïtés, particulièrement utile pour analyser des exigences longues, détecter les incohérences entre différentes sources et améliorer la qualité globale du processus de spécification [37].

CHAPITRE 4 : Technique de rédaction des Exigences

La documentation des exigences logicielles est cruciale pour la réussite des projets de développement. Elle sert de référence tout au long du processus, de la conception aux tests. La clarté et la simplicité dans la rédaction des exigences sont essentielles pour éviter l'ambiguïté dans le document de spécification [1], [41]. Ce chapitre explore deux techniques fondamentales pour cette rédaction, jouant un rôle clé dans l'articulation des besoins et des fonctionnalités du logiciel.

4.2- Les Cas d'utilisation (*Use Cases*)

Les Use Cases sont une méthodologie détaillée pour saisir les exigences logicielles en décrivant les interactions système-utilisateur. Chaque Use Case représente un scénario spécifique avec actions, réponses du système et résultats attendus, offrant une compréhension approfondie. Comparés aux User Stories, ils détaillent les séquences d'événements. Les Use Cases aident les équipes de développement à concevoir des fonctionnalités alignées sur les besoins des utilisateurs, fournissant un contexte détaillé pour définir les exigences fonctionnelles [1].

4.3- Les récits d'utilisation (*User Stories*)

Les User Stories, au cœur de l'approche Agile, sont des descriptions concises de fonctionnalités mettant en avant les besoins d'un utilisateur pour atteindre un objectif spécifique. Elles privilégient le langage naturel, évitant les termes techniques, et se concentrent sur l'expérience utilisateur. Cette approche assure que les fonctionnalités

développées répondent aux besoins réels des utilisateurs, facilitant ainsi un processus de développement plus précis et aligné sur les attentes des utilisateurs finaux [1].

4.4- Limites des approches, méthodes ou techniques de rédaction d'exigences du logiciel

L'ingénierie des exigences, une étape cruciale du développement logiciel selon l'IEEE 729-1990, est essentielle pour le succès des projets, comme observé par Wiegers [40]. Cependant, les approches traditionnelles, telles que les cas d'utilisation, peuvent présenter des ambiguïtés, et le risque d'échec est souligné par Ambler [43]. Une nouvelle approche innovante basée sur l'utilisation de triplets, proposée pour surmonter ces limitations, rend les exigences plus explicites et facilite l'automatisation de la mesure de la taille fonctionnelle des logiciels, offrant une perspective prometteuse pour l'avenir de l'ingénierie des exigences, comme évoqué par Nuseibeh et Easterbrook [44].

4.5- Spécification des exigences logicielles avec l'approche par triplets

L'approche par triplets pour rédiger les exigences logicielles [1] propose une structure claire avec un sujet (acteur), un prédicat (action), et un objet (composant). Cette méthode simplifie la rédaction, éliminant les détails superflus [1]. Elle favorise une communication directe et compréhensible entre les équipes de développement et les parties prenantes, réduisant l'ambiguïté [1]. De plus, elle améliore la traçabilité des exigences tout au long du cycle de vie du logiciel [1], [2], offrant une perspective prometteuse selon Gerançon [1].

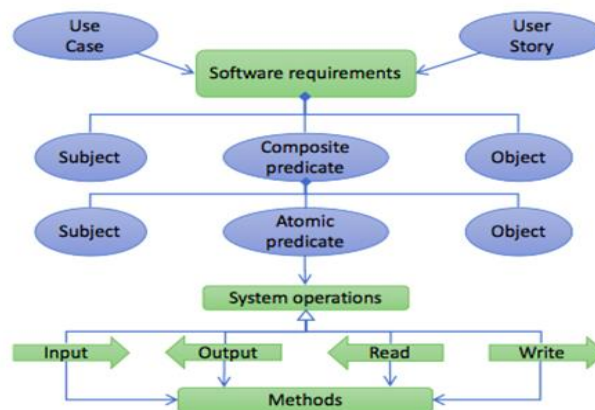


Figure 3 : Figure résumant l'approche par triplet source[1]

CHAPITRE 5 : Méthodologie

5.1- Présentation de la méthodologie adoptée

La recherche se concentre sur la détection des ambiguïtés dans les spécifications d'exigences logicielles (SEL). L'objectif est de concevoir un outil de détection précoce d'erreurs d'ambiguïté, utilisant la méthodologie "Design Science Research" de Hevner et al [45]. Les étapes comprennent l'analyse des techniques de rédaction d'exigences, la justification du modèle de triplets de Géranson [31] comme norme de spécification, la conception d'un outil de détection d'ambiguïté, et l'évaluation de l'outil sur des spécifications d'exigences logicielles en langage naturel [45].

5.1.1- Évaluation des techniques de rédaction des exigences logicielles

L'analyse des méthodes de rédaction d'exigences logicielles révèle des défis, notamment des limitations dans les outils d'automatisation et l'absence de détails techniques favorisant l'automatisation. Des chercheurs, tels que Wiegers et Beatty [42], ont identifié des faiblesses telles que l'incohérence, l'ambiguïté et la non-atOMICITÉ des exigences. Trudel et Boisvert [41] soulignent que de nombreuses exigences industrielles sont peu utiles, peu utilisables et peu exploitées, nécessitant une amélioration des approches de rédaction d'exigences logicielles [42][41].

5.1.2- Choix de la technique par triplets pour la conception de notre outil

L'approche par triplets, proposée par Géranson [1], consiste à structurer les exigences logicielles en triplets (sujet, prédicat, objet), simplifiant ainsi l'expression des besoins des utilisateurs, améliorant la clarté et facilitant l'automatisation du traitement des exigences logicielles. Cette méthode complète les approches traditionnelles comme les Use Cases et les User Stories, offrant une rédaction et une automatisation des exigences plus efficaces [1].

5.1.3- Conception d'un outil basé sur l'approche par triplet

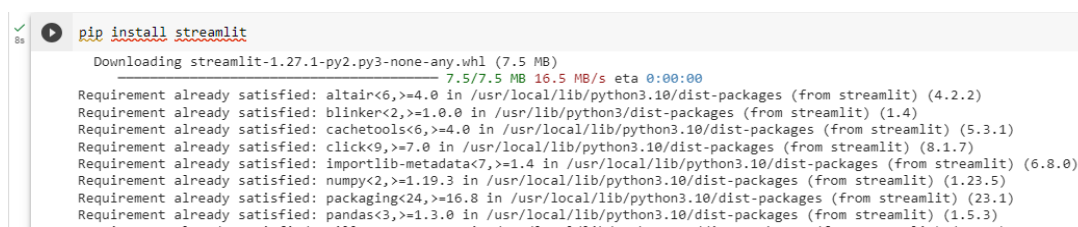
Notre outil innovant, basé sur l'approche par triplets et intégrant des techniques avancées de TALN, automatise la détection des ambiguïtés dans les spécifications logicielles. Il vise à simplifier la rédaction des exigences, améliorant ainsi la clarté et la cohérence. L'interface utilisateur conviviale facilite l'utilisation par les parties prenantes, offrant une solution complète pour la rédaction, la clarification et l'automatisation des exigences logicielles. Cette approche répond efficacement aux défis du processus de développement logiciel.

CHAPITRE 6 : Implémentation de la solution

6.1- Les différentes étapes de l'implémentation de la solution

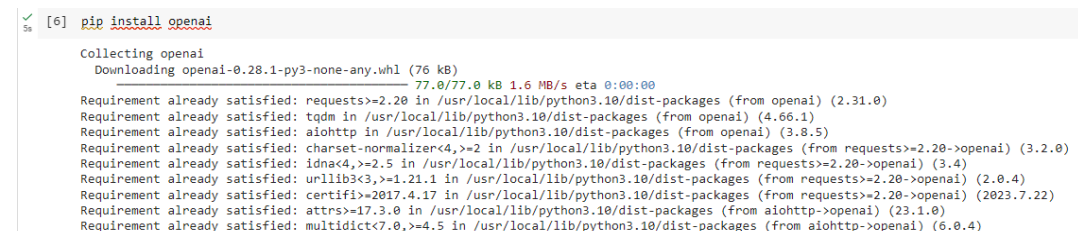
6.1.1- Installation et dépendances

Le projet débute par la mise en place de l'environnement de développement, ce qui implique l'installation de Streamlit, la bibliothèque Python d'Openai, ainsi que d'autres dépendances indispensables. Cette étape cruciale permet de préparer le terrain pour le développement ultérieur en garantissant que toutes les ressources nécessaires sont correctement configurées et prêtes à être utilisées. Elle inclut également la configuration des paramètres initiaux, la définition des chemins de fichiers, et la création des variables requises pour le bon fonctionnement du projet.



```
pip install streamlit
Downloading streamlit-1.27.1-py2.py3-none-any.whl (7.5 MB)
7.5/7.5 MB 16.5 MB/s eta 0:00:00
Requirement already satisfied: altair<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (4.2.2)
Requirement already satisfied: blinker<2,>=1.0.0 in /usr/lib/python3/dist-packages (from streamlit) (1.4)
Requirement already satisfied: cachetools<6,>=4.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (5.3.1)
Requirement already satisfied: click<9,>=7.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (8.1.7)
Requirement already satisfied: importlib-metadata<7,>=1.4 in /usr/local/lib/python3.10/dist-packages (from streamlit) (6.8.0)
Requirement already satisfied: numpy<2,>=1.19.3 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.23.5)
Requirement already satisfied: packaging<24,>=16.8 in /usr/local/lib/python3.10/dist-packages (from streamlit) (23.1)
Requirement already satisfied: pandas<3,>=1.3.0 in /usr/local/lib/python3.10/dist-packages (from streamlit) (1.5.3)
```

Figure 4 : Installation streamlit



```
[6] pip install openai
Collecting openai
Downloading openai-0.28.1-py3-none-any.whl (76 kB)
77.0/77.0 kB 1.6 MB/s eta 0:00:00
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai) (2.31.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai) (4.66.1)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from openai) (3.8.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.2.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (3.4)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2.0.4)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai) (2023.7.22)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (23.1.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai) (6.0.4)
```

Figure 5 : Installation d'Openai

6.1.1- Conception de l'interface utilisateur

Streamlit est utilisé pour concevoir l'interface utilisateur. Des éléments clés tels que le titre, le téléchargeur de fichiers, les boutons et les listes de sélection sont créés grâce à Streamlit. Cette plateforme simplifie considérablement la création d'une application web en nécessitant un minimum de code. Elle offre également la possibilité d'ajouter des fonctionnalités interactives et des visualisations de données pour rendre l'application conviviale et attrayante pour les utilisateurs.



Ambiguity Finder

Figure 6 : Interface de l'application

6.1.3- Génération de couleurs aléatoires

La fonction *random_color* est définie pour générer des couleurs aléatoires afin d'afficher les ambiguïtés avec différentes couleurs dans l'interface utilisateur. Cela ajoute de la variété visuelle aux résultats affichés.



Figure 7 : Affiche des ambiguïtés

6.1.4- Détection d'Ambiguïté

La fonction Find Ambiguities interagit avec le modèle GPT-3 d'OpenAI pour repérer les ambiguïtés dans le texte téléchargé. Elle envoie une requête au modèle, identifie les zones où la signification est incertaine, et renvoie les résultats. Cette détection d'ambiguïté est cruciale car elle clarifie et améliore la compréhension du texte, permettant de mettre en évidence visuellement les zones ambiguës dans l'interface utilisateur. Cela offre aux utilisateurs une compréhension approfondie du contenu, facilitant ainsi la prise de décisions éclairées en fonction du contexte fourni.

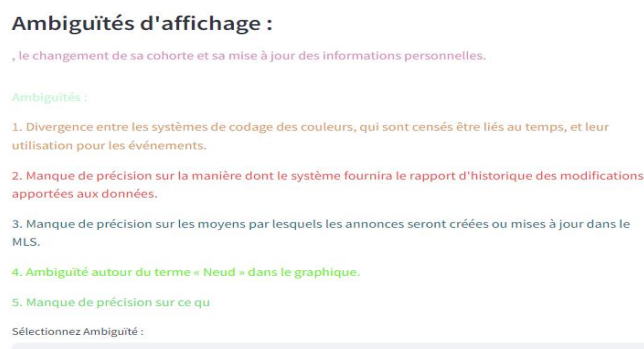


Figure 8 : Détection des ambiguïtés

6.1.5- Logique principale de l'application

Dans la fonction principale, la logique principale de l'outil est implémentée. Cela comprend la gestion des téléversement de fichiers, l'affichage du texte téléversé et la proposition d'options pour analyser les ambiguïtés. Les ambiguïtés sont affichées avec des couleurs aléatoires pour une distinction visuelle.

6.1.6- Exécution conditionnelle

Pour garantir que le code d'analyse des ambiguïtés s'exécute uniquement lorsqu'une ligne est sélectionnée, une vérification conditionnelle sur la variable *selected_line* est mise en place. Cela évite les exécutions inutiles lorsqu'aucune ligne n'est choisie.

6.1.7- Aperçu de l'algorithme

L'algorithme principal utilisé dans cet outil est l'utilisation du modèle GPT-3 d'Open API pour des tâches de traitement du langage naturel. L'algorithme implique l'envoi de requêtes au modèle, qui génère des réponses en fonction de ses capacités de compréhension du langage. Les requêtes sont soigneusement conçues pour demander au modèle d'identifier les ambiguïtés et de fournir des informations détaillées à leur sujet.

```
import spacy

# Chargez le modèle linguistique en français (ou la langue de votre choix)
nlp = spacy.load("fr_core_news_sm")

def is_triplet_structure(sentence):
    # Utilisez spacy pour analyser la structure grammaticale de la phrase
    doc = nlp(sentence)

    # Vérifiez si la phrase suit la structure de triplet "sujet, verbe, complément"
    has_subject, has_verb, has_object = False, False, False
    for token in doc:
        if token.dep_ == "nsubj":
            has_subject = True
        elif token.dep_ == "ROOT" and token.pos_ == "VERB":
            has_verb = True
        elif token.dep_ in ["dobj", "attr"]:
            has_object = True

    return has_subject and has_verb and has_object

@st.cache_resource
def find_ambiguities(text):
    openai.api_key = "sk-Wvg01dcPwz81JrmwQivt3DlnKFJW8qIT1K1aUt6U9ccwYF"

    response = openai.Completion.create(
        engine="text-davinci-002",
        prompt=f"Identify ambiguous lines in the given text: {text}",
        max_tokens=200 # You can adjust this as needed
    )

    ambiguous_lines = response.choices[0].text.strip().split('\n')

    # Filtrer les lignes ambiguës pour celles qui suivent la structure de triplet
    triplet_ambiguous_lines = [line for line in ambiguous_lines if is_triplet_structure(line)]

    return triplet_ambiguous_lines

def analyze_ambiguity(line, text):
    openai.api_key = "sk-Wvg01dcPwz81JrmwQivt3DlnKFJW8qIT1K1aUt6U9ccwYF"
```

Figure 10 : Aperçu de l'algorithme

L'algorithme de détection d'ambiguïté combine habilement l'analyse grammaticale via spaCy et l'intelligence artificielle de GPT-3. Utilisant Streamlit pour une interface conviviale, le programme assure une détection précise des ambiguïtés dans les exigences logicielles. En respectant une structure grammaticale spécifique, il interagit avec GPT-3 pour filtrer les suggestions et fournit des détails contextuels sur chaque ambiguïté. Cette convergence offre

une solution sophistiquée pour renforcer la clarté et la compréhension des spécifications logicielles.

6.1.8- Ensemble de données

L'outil Ambiguity Finder analyse les données textuelles fournies par les utilisateurs. Ensemble de données est dynamique, dépendant des fichiers texte téléchargés. Les points clés incluent :

- **Source des données** : Composée de documents texte téléchargés ou écrits par les utilisateurs, l'ensemble peut varier en contenu, style et domaine.
- **Prétraitement des données** : Un prétraitement minimal assure la compatibilité avec les capacités de traitement du langage naturel du modèle GPT-3, impliquant un nettoyage de base et une tokenisation.
- **Taille des données** : La taille dépend des fichiers téléversés, variant de quelques phrases à des documents étendus.
- **Diversité des données** : La force de l'outil réside dans la diversité de l'ensemble, permettant son application dans différents domaines et contextes où la détection d'ambiguïtés est pertinente.

6.1.9- Création et Entraînement du Modèle

Le cœur des capacités de traitement du langage naturel de l'outil Ambiguity Finder repose sur le modèle GPT-3 d'Openai. Voici les principaux aspects de l'utilisation du modèle :

- **Sélection du Modèle** : GPT-3 d'Openai a été choisi pour ses capacités exceptionnelles de compréhension du langage, le rendant adapté à la détection des ambiguïtés.
- **Données d'Entraînement** : GPT-3 est un modèle pré-entraîné et ne nécessite pas de données d'entraînement spécifiques. Il a été peaufiné par Openai sur un ensemble de données vaste et diversifié couvrant différents domaines.
- **Hyper paramètres** : L'outil *The Roc Ambiguity Detect* interagit avec le modèle GPT-3 en utilisant des paramètres spécifiques tels que le moteur et max tokens pour obtenir des réponses pertinentes.

- **Processus d'Entraînement** : L'outil Ambiguity Finder utilise les capacités pré-entraînées de GPT-3 pour effectuer la détection et l'analyse des ambiguïtés, sans nécessiter d'entraînement supplémentaire pour le modèle.

6.2- Architecture du Système

L'architecture de l'outil *The Roc Ambiguity Detect* englobe à la fois l'interface utilisateur et les composants en arrière-plan :

- **Interface Frontale** : L'interface conviviale est construite à l'aide de Streamlit, offrant des éléments tels que le téléversement de fichiers, des boutons et des cases de sélection. Elle permet aux utilisateurs de téléverser des fichiers texte et d'interagir avec l'outil facilement.
- **Composants en Arrière-Plan** : Le backend gère la logique principale de la détection et de l'analyse des ambiguïtés. Il envoie des requêtes au modèle GPT-3 et traite les réponses. Le système est conçu pour gérer les demandes des utilisateurs et fournir des retours en temps réel.
- **Points d'Intégration** : L'outil s'intègre à GPT-3 d'Openai via son API. Il envoie des requêtes au moteur GPT-3, reçoit des réponses et formate les informations pour les présenter à l'utilisateur.
- **Scalabilité et Performances** : Bien que l'implémentation actuelle de l'outil Ambiguity Finder convienne aux utilisateurs individuels, il peut être déployé sur une infrastructure évolutive pour accueillir une base d'utilisateurs plus importante et gérer efficacement des fichiers texte plus étendus.

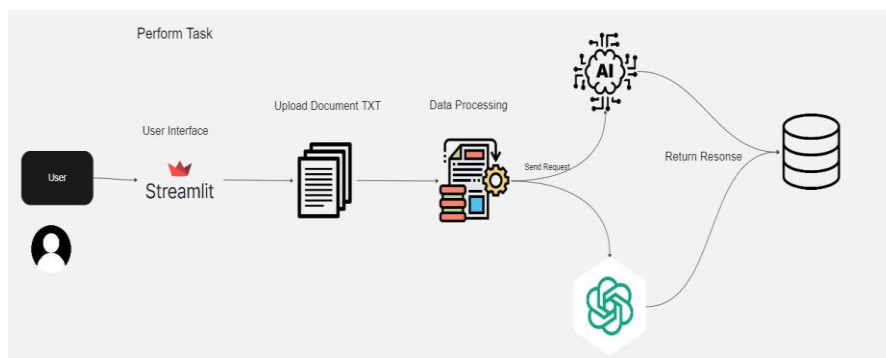


Figure 10 : Architecture de l'application

CHAPITRE 7 : Evaluation et Analyse Résultats

7.1- Présentation de l'outil logiciel "The Roc Ambiguity Detect"

Il est évident que les exigences formulées en langage naturel peuvent souvent être sujettes à l'ambiguïté. Ainsi, une étape préalable de traitement du document de spécification des exigences est nécessaire. L'objectif principal de ce processus est de repérer les phrases ambiguës dans un document de spécification d'exigences logicielles (SEL) afin de faciliter la compréhension de leur nature ambiguë par les utilisateurs. Cette identification permet ensuite de les corriger, contribuant ainsi à améliorer la qualité globale du SRS (Software Requirements Specification). Dans ce chapitre, une approche méthodologique détaillée est présentée pour atteindre cet objectif ambitieux de détection des ambiguïtés dans les exigences logicielles, en utilisant l'outil appelé "*The Roc Ambiguity Detect*". Cette section est cruciale pour comprendre en détail la démarche de recherche et de développement entreprise.

7.2- Objectifs de l'outil

L'outil "*The Roc Ambiguity Détection*" a été conçu avec des objectifs spécifiques visant à améliorer la qualité et la clarté des documents de spécifications logiciels. Les objectifs principaux de cet outil sont les suivants :

7.2.1- Détection Automatisée des Ambiguïté

L'objectif principal de "*The Roc Ambiguity Detect*" est de fournir une solution automatisée pour la détection des ambiguïtés dans les documents de spécifications logiciels. Il vise à identifier les zones de texte ambiguës ou sujettes à des interprétations multiples, ce qui peut entraîner des malentendus ou des erreurs dans le développement de logiciels.

7.2.2- Amélioration de la Compréhension

En détectant les ambiguïtés, l'outil vise à améliorer la compréhension des spécifications logicielles par les parties prenantes, y compris les développeurs, les testeurs et les clients. En clarifiant le langage utilisé dans les documents, il contribue à réduire les risques de malentendus et d'interprétations incorrectes.

7.2.3- Augmentation de la Qualité des Spécifications

Un autre objectif clé est d'améliorer la qualité générale des spécifications logicielles en éliminant les ambiguïtés. Des spécifications plus claires et mieux définies facilitent la conception, le développement et les tests du logiciel, ce qui peut entraîner une réduction des erreurs et des retards.

7.2.4- Gain de Temps et d'Efforts

En identifiant les ambiguïtés de manière automatisée, l'outil "*The Roc Ambiguity Detect*" vise à économiser du temps et des efforts précieux pour les équipes de développement. Plutôt que de dépendre d'examen manuels intensifs des spécifications, les équipes peuvent se concentrer sur des zones identifiées comme potentiellement problématiques.

7.2.5- Support pour la Conformité aux Normes

Dans de nombreux domaines, notamment dans l'industrie de la sécurité, la conformité aux normes strictes est essentielle. L'outil "*The Roc Ambiguity Detect*" peut aider à garantir que les spécifications logicielles respectent les exigences de conformité en identifiant et en corrigeant les ambiguïtés qui pourraient entraîner des problèmes de conformité.

7.3- Problèmes Résolus par "*The Roc Ambiguity Detect*"

L'outil "*The Roc Ambiguity Detect*" s'attaque à plusieurs problèmes courants dans la détection des ambiguïtés dans les documents de spécifications logiciels, notamment :

7.3.1- Ambiguïtés Linguistiques

Les spécifications logicielles peuvent contenir des termes ambigus, des phrases mal construites ou des formulations imprécises. "*The Roc Ambiguity Detect*" identifie ces ambiguïtés linguistiques et les signale.

7.3.2- Interprétations Multiples

Les documents de spécifications peuvent être sujets à des interprétations multiples par différentes personnes. L'outil vise à aligner l'interprétation des spécifications entre les membres de l'équipe.

7.3.3- Incohérences

Les incohérences dans les spécifications, telles que des informations contradictoires, sont identifiées par l'outil pour garantir que toutes les parties des spécifications sont cohérentes entre elles.

7.4.- Fonctionnalités de l'outil

L'outil offre un ensemble de fonctionnalités puissantes conçues pour faciliter la détection et la gestion des ambiguïtés dans les documents de spécifications logiciels. Les principales fonctionnalités de cet outil sont les suivantes :

7.4.1- Importer un Document ou Écrire un Texte

Notre interface conviviale comporte un bouton de parcours de fichiers qui vous permet de sélectionner facilement le document d'exigences dans lequel vous souhaitez détecter les ambiguïtés et une zone dans laquelle vous pouvez écrire le texte que vous voulez analyser. Vous pouvez ainsi travailler sur le document pertinent en toute simplicité.

7.4.2- Analyse des Ambiguïtés

Une fois que vous avez téléchargé le document en question ou écrit le texte, il vous suffit de cliquer sur le bouton "Analyser les Ambiguïtés". Notre système puissant s'attelle alors à analyser le texte à la recherche d'ambiguïtés potentielles.

7.4.3- Résultats Interactifs

Lorsque l'analyse est terminée, la liste des ambiguïtés détectées vous est présentée de manière interactive. Chaque ambiguïté est mise en évidence avec une couleur correspondant à son type. Cela facilite grandement la visualisation et la compréhension des problèmes.

7.4.4- Sélection Personnalisée

Vous avez la possibilité de sélectionner n'importe quelle des ambiguïtés détectées dans la liste. En cliquant sur une ambiguïté spécifique, vous accédez à des détails importants pour la clarification.

7.4.5-Détails des Ambiguïtés

Les détails concernant chaque ambiguïté sont clairement présentés, notamment le type d'ambiguïté détecté et des suggestions de clarification. Vous disposez ainsi de toutes les informations nécessaires pour prendre des mesures correctives efficaces.

7.5- Évaluation de l'outil

Dans cette section, nous allons explorer en détail comment nous avons évalué l'efficacité et les performances de notre outil, connu sous le nom de “*The Roc Ambiguity Detect*”.

7.5.1- Présentation des résultats Obtenus des deux méthodes utilisées

Dans cette section, nous exposons les résultats obtenus à travers l'application de deux méthodes distinctes d'évaluation de notre outil. La première méthode repose sur une analyse manuelle, où un expert en spécifications logicielles passe en revue les documents pour identifier les ambiguïtés. La seconde méthode consiste en une analyse automatique, au cours de laquelle notre système spécialisé examine les mêmes documents en quête d'ambiguïtés. Cette approche double permet d'établir une comparaison significative entre les performances humaines et celles de notre outil. Les résultats obtenus à la fin de cette section nous fourniront un aperçu de l'efficacité de notre application dans la détection des ambiguïtés, soulignant ainsi ses avantages potentiels dans le domaine des spécifications logicielles.

7.5.2 - Collecte de données

La principale collecte de données pour ce projet s'est concentrée sur les documents SRS (Software Requirements Specifications), marquant une première étape dans l'exploration de la détection automatisée des exigences ambiguës dans les SRS rédigés en français et en anglais. Quatre (3) documents SRS ont été méticuleusement sélectionnés, chacun provenant de projets de développement logiciel distincts trouvés en ligne. Ces documents comprenaient une version en français, une version en anglais, et un document élaboré par notre équipe selon des critères définis dans le cadre de l'approche par triplet. Issus de projets variés en ligne, ils ont apporté une diversité linguistique. Un total de 56 spécifications de besoins logiciels a été extrait de ces documents, soigneusement choisi pour constituer un ensemble de données représentatif et adéquat pour les analyses.

A- Cas 1

Projet SRG : Système de gestion de réservation de terrain

Description : Le "Système de gestion de réservation de terrain" est une plateforme en ligne facilitant la réservation d'espaces sportifs. Il permet une réservation aisée d'emplacements pour des activités sportives ou récréatives, offrant une interface conviviale pour visualiser les disponibilités, choisir des plages horaires, et effectuer des réservations en temps réel. Le document de spécifications logicielles, rédigé en français, a fourni 7 phrases de spécifications. Le Dr. Bruel Gerançon, mon encadrant spécialiste, a identifié trois spécifications incomplètes ou ambiguës parmi ces 7 phrases.

Analyse du jeu de données SRG à l'aide de notre outil "The Roc Ambiguity Detect"

The Roc Ambiguity Detect

Your text:

Un membre réserve un terrain à une heure donnée
Le gérant crée les terrains
Le gérant établit les horaires et la disponibilité des terrains
Fixer le prix de location des terrains
Payer une réservation
Réserver un terrain en permanence
Sous louer une plage horaire en réservation permanente

Total Sentences: 7

Ambiguities Detected: 2

Ambiguity Percentage: 28.57%

Display Ambiguities:

"Réserver un terrain à une heure donnée"

Figure 9 : Résultat de l'analyse du jeu de données SRG

B- Cas 2

Marvel Electronics and Home Entertainment

Description : Marvel Electronics and Home Entertainment est une boutique en ligne spécialisée dans la vente d'accessoires électroniques et d'appareils électroniques pour la maison. Cette plateforme propose une variété d'équipements électroniques pour la maison, des accessoires high-tech, et des appareils électroniques de divertissement.

Le document de spécifications logicielles de ce projet est rédigé en anglais. Un total de 19 phrases a été extrait du document pour être analysé en vue de détecter des ambiguïtés. Après analyse, il s'est avéré qu'une seule phrase est ambiguë.

Analyse de ces phrases à l'aide de notre outil "The Roc Ambiguity Détection"

The Roc Ambiguity Detect

Your text:

The system shall display all the products that can be configured.
The system shall allow user to select the product to configure.
The system shall display all the available components of the product to configure.
The system shall enable user to add one or more component to the configuration.
The system shall notify the user about any conflict in the current configuration.
The system shall allow user to update the configuration to resolve conflict in the
The system shall allow user to confirm the completion of current configuration.
The system shall display detailed information of the selected products.
The system shall provide browsing options to see product details.
The system shall display detailed product categorization to the user.
The system shall enable user to select multiple options on the screen to search.
The system shall display all the matching products based on the search.
The system shall display only 10 matching results on the current screen.
The system shall enable user to navigate between the search results.
The system shall notify the user when no matching product is found on the search.
The system shall display different shipping options provided by shipping department.
The system shall enable user to select the shipping method during payment process.
The system shall display the shipping charges.
The system shall display tentative duration for shipping.

Total Sentences: 19

Ambiguities Detected: 1

Ambiguity Percentage: 5.26%

Display Ambiguities:

"The system shall enable user to select multiple options on the screen to search."

Figure 10 : Résultat de l'analyse du jeu de données Marvel Electronics and Home Entertainment

C- Cas 3

Création d'un jeu données sur mesure

Pour ce cas, nous développons un jeu de données d'exigences logicielles en fonction des critères d'écriture d'exigences pour ce projet, à savoir la structure en triplets. Ce jeu de données contient un total de 40 phrases, dont 10 sont ambiguës et les 30 autres sont complètes.

Total des phrases : 40
Ambiguïtés détectées : 10
Pourcentage d'ambiguïté : 25,00 %
Ambiguïtés d'affichage :
L'application doit être conviviale.
L'application doit être sécurisée.
L'application doit être performante.
L'application doit être compatible avec tous les systèmes d'exploitation.
L'application doit être compatible avec tous les navigateurs web.
L'application doit être compatible avec tous les appareils mobiles.
L'application doit être accessible aux personnes handicapées.
L'application doit être conforme aux normes.

Figure 11 : Résultat de l'analyse du jeu de données du 3e cas

7.6.- Synthèse des résultats

Nous avons analysé un total de 56 phrases réparties en deux projets : un document de spécifications rédigé en anglais, un autre en français, et un document de spécifications que nous avons rédigé selon les critères de la méthode adoptée pour détecter les ambiguïtés dans le cadre de ce projet, à savoir l'approche par triplets. Les résultats obtenus avec notre outil, 'The Roc Ambiguity Détection', sont très satisfaisants. En ce qui concerne les phrases françaises, le taux de détection des phrases ambiguës est plutôt bon, soit 92,30. Quant aux phrases anglaises, l'outil a obtenu un taux de détection de 100 %, correspondant au résultat de la méthode non automatisée. Le tableau 3 présente une synthèse des résultats des différents documents de spécifications, évaluant la précision des mesures manuelles et automatisées, ainsi que les écarts.

Projet	Quantité de phrases	Quantité de Phrase ambiguës	Quantités détectés	Ecart	Ecart(%)	Précision(%)
SRG	7	3	2	+1	33,33%	66.67%
Data test	40	10	10	0	0.00%	100%
Sous-total français	47	13	12	+1	7.69%	92.30%

E-store	19	1	1	0	0,00%	100
Sous-total anglais	19	1	1	0	0.00%	100
Total	56	14	13	2	7.14%	92.85

Tableau 3 : Synthèses des résultats de méthode non automatisé et automatisé

Conclusion, Contribution, Perspectives et limites de Recherche

Conclusion

Cette thèse se concentre sur le développement d'un outil logiciel novateur dédié à la détection des ambiguïtés dans les documents de spécifications logiciels. L'objectif principal de la recherche est d'améliorer significativement la qualité des spécifications logicielles en identifiant et corrigeant les ambiguïtés dès les premières phases du développement. L'outil utilise une approche par triplets 'sujet-verbe-prédicat', basée sur les principes d'écriture des exigences préconisés par Gerançon. Cette méthode, caractérisée par sa simplicité, sa clarté, et sa précision, vise à perfectionner les documents d'exigences en assurant une qualité optimale. Chaque exigence qui ne suit pas cette structure spécifique est identifiée comme ambiguë, renforçant ainsi la qualité du processus de rédaction des spécifications. L'approche triplet favorise la détection proactive des ambiguïtés et contribue à la création d'un langage clair et cohérent, élément essentiel pour des spécifications logicielles de haute qualité.

Contribution

La contribution principale de ce travail réside dans la création d'un outil logiciel convivial et simple d'utilisation capable de repérer les ambiguïtés dans les documents de spécifications. L'outil repose sur une approche basée sur le Machine Learning, qui lui permet d'analyser les spécifications logiciels en utilisant une méthodologie basée sur la structure de triplets "sujet, verbe, prédicat". Cette méthodologie est en grande partie inspirée du travail de Gerançon et Bruel. Plus spécifiquement, les contributions majeures de ce mémoire sont les suivantes :

- **Développement de l'outil logiciel** : La création d'un outil logiciel qui offre une interface graphique intuitive, permettant aux utilisateurs de charger leurs documents

de spécifications logiciels et de lancer le processus de détection d'ambiguïtés. Cela simplifie considérablement le processus de vérification des spécifications.

- **Utilisation de l'apprentissage automatique** : L'intégration de techniques d'apprentissage automatique pour analyser le texte des spécifications logiciels en utilisant la structure de triplets "sujet, verbe, prédicat". Cette approche permet une détection plus précise des ambiguïtés, tout en réduisant la nécessité d'une intervention manuelle intensive.
- **Détection proactive des exigences ambiguës** : L'outil est conçu pour repérer les exigences ambiguës dès leur rédaction, ce qui facilite la correction immédiate et contribue à la prévention des erreurs coûteuses à des étapes ultérieures du projet.

En somme, cette recherche vise à simplifier et à améliorer la phase de spécification des exigences logicielles en offrant un outil novateur qui s'appuie sur les avancées de l'apprentissage automatique. Cette contribution est particulièrement pertinente dans le contexte actuel du développement logiciel, où la précision des spécifications est cruciale pour le succès du projet et la satisfaction du client.

Perspectives et limites de recherche

Malgré les progrès notables dans le développement de l'outil de détection des ambiguïtés dans les documents de spécifications logiciels, des perspectives et des limites exigent une analyse approfondie. Une limite importante est l'incapacité actuelle de l'outil à repérer les défauts d'ambiguïtés au niveau des exigences non fonctionnelles, en raison de sa focalisation exclusive sur les exigences fonctionnelles au cours de cette étude.

Pour améliorer la couverture, une direction prometteuse serait l'intégration d'un module additionnel capable de détecter les erreurs d'ambiguïtés liées aux exigences non fonctionnelles. Cela renforcerait la capacité de l'outil à identifier les ambiguïtés dans l'ensemble des spécifications logicielles, accroissant ainsi sa pertinence dans des contextes fonctionnels et non fonctionnels.

Malgré ses performances, l'outil actuel ne parvient pas à englober certaines nuances et éléments spécifiques, et il est crucial de reconnaître ces lacunes pour maintenir la crédibilité de la recherche. L'identification claire de ces aspects non pris en compte constitue le point de départ pour des recherches futures visant à perfectionner l'outil, le rendant plus robuste dans

sa capacité à détecter et corriger toutes les formes d'ambiguïtés dans les documents de spécifications logiciels. En résumé, cette étude offre une base solide pour des développements ultérieurs visant à élargir la couverture de détection et à accroître la pertinence pratique de l'outil dans divers contextes de développement logiciel.